

# KD Chart 2 Reference Manual

[rev.2.0]

Generated by Doxygen 1.5.1

Mon Sep 17 16:16:49 2007



# Contents

<b>1</b>	<b>KD Chart 2 Directory Hierarchy</b>	<b>1</b>
1.1	KD Chart 2 Directories . . . . .	1
<b>2</b>	<b>KD Chart 2 Namespace Index</b>	<b>3</b>
2.1	KD Chart 2 Namespace List . . . . .	3
<b>3</b>	<b>KD Chart 2 Hierarchical Index</b>	<b>5</b>
3.1	KD Chart 2 Class Hierarchy . . . . .	5
<b>4</b>	<b>KD Chart 2 Class Index</b>	<b>9</b>
4.1	KD Chart 2 Class List . . . . .	9
<b>5</b>	<b>KD Chart 2 File Index</b>	<b>13</b>
5.1	KD Chart 2 File List . . . . .	13
<b>6</b>	<b>KD Chart 2 Page Index</b>	<b>17</b>
6.1	KD Chart 2 Related Pages . . . . .	17
<b>7</b>	<b>KD Chart 2 Directory Documentation</b>	<b>19</b>
7.1	src/PrerenderedElements/ Directory Reference . . . . .	19
7.2	src/Scenery/ Directory Reference . . . . .	20
7.3	src/ Directory Reference . . . . .	21
7.4	src/Ternary/ Directory Reference . . . . .	25
<b>8</b>	<b>KD Chart 2 Namespace Documentation</b>	<b>27</b>
8.1	KDChart Namespace Reference . . . . .	27
<b>9</b>	<b>KD Chart 2 Class Documentation</b>	<b>35</b>
9.1	KDChart::AbstractArea Class Reference . . . . .	35
9.2	KDChart::AbstractAreaBase Class Reference . . . . .	48
9.3	KDChart::AbstractAreaWidget Class Reference . . . . .	55

9.4	<a href="#">KDChart::AbstractAxis Class Reference</a>	66
9.5	<a href="#">KDChart::AbstractCartesianDiagram Class Reference</a>	86
9.6	<a href="#">KDChart::AbstractCoordinatePlane Class Reference</a>	134
9.7	<a href="#">KDChart::AbstractDiagram Class Reference</a>	167
9.8	<a href="#">KDChart::AbstractLayoutItem Class Reference</a>	209
9.9	<a href="#">KDChart::AbstractPieDiagram Class Reference</a>	213
9.10	<a href="#">KDChart::AbstractPolarDiagram Class Reference</a>	261
9.11	<a href="#">KDChart::AbstractProxyModel Class Reference</a>	305
9.12	<a href="#">KDChart::AbstractTernaryDiagram Class Reference</a>	308
9.13	<a href="#">KDChart::AbstractThreeDAttributes Class Reference</a>	350
9.14	<a href="#">KDChart::AttributesModel Class Reference</a>	353
9.15	<a href="#">KDChart::AutoSpacerLayoutItem Class Reference</a>	371
9.16	<a href="#">KDChart::BackgroundAttributes Class Reference</a>	378
9.17	<a href="#">KDChart::BarAttributes Class Reference</a>	383
9.18	<a href="#">KDChart::BarDiagram Class Reference</a>	390
9.19	<a href="#">KDChart::CartesianAxis Class Reference</a>	447
9.20	<a href="#">KDChart::CartesianCoordinatePlane Class Reference</a>	487
9.21	<a href="#">KDChart::Chart Class Reference</a>	543
9.22	<a href="#">KDChart::ChartGraphicsItem Class Reference</a>	565
9.23	<a href="#">KDChart::DataDimension Class Reference</a>	567
9.24	<a href="#">KDChart::DatasetProxyModel Class Reference</a>	571
9.25	<a href="#">KDChart::DatasetSelectorWidget Class Reference</a>	579
9.26	<a href="#">KDChart::DataValueAttributes Class Reference</a>	581
9.27	<a href="#">KDChart::DiagramObserver Class Reference</a>	593
9.28	<a href="#">KDChart::FrameAttributes Class Reference</a>	596
9.29	<a href="#">KDChart::GlobalMeasureScaling Class Reference</a>	600
9.30	<a href="#">KDChart::GridAttributes Class Reference</a>	603
9.31	<a href="#">KDChart::HeaderFooter Class Reference</a>	611
9.32	<a href="#">KDChart::HorizontalLineLayoutItem Class Reference</a>	631
9.33	<a href="#">KDChart::Legend Class Reference</a>	637
9.34	<a href="#">KDChart::LineAttributes Class Reference</a>	670
9.35	<a href="#">KDChart::LineDiagram Class Reference</a>	674
9.36	<a href="#">KDChart::LineLayoutItem Class Reference</a>	733
9.37	<a href="#">KDChart::LineWithMarkerLayoutItem Class Reference</a>	739
9.38	<a href="#">KDChart::MarkerAttributes Class Reference</a>	745
9.39	<a href="#">KDChart::MarkerLayoutItem Class Reference</a>	751



9.40	KDChart::Measure Class Reference . . . . .	758
9.41	KDChart::PaintContext Class Reference . . . . .	765
9.42	KDChart::Palette Class Reference . . . . .	768
9.43	KDChart::PieAttributes Class Reference . . . . .	773
9.44	KDChart::PieDiagram Class Reference . . . . .	777
9.45	KDChart::Plotter Class Reference . . . . .	830
9.46	KDChart::PolarCoordinatePlane Class Reference . . . . .	887
9.47	KDChart::PolarDiagram Class Reference . . . . .	927
9.48	KDChart::Position Class Reference . . . . .	976
9.49	KDChart::PositionPoints Class Reference . . . . .	985
9.50	KDChart::PrivateAttributesModel Class Reference . . . . .	990
9.51	KDChart::RelativePosition Class Reference . . . . .	1008
9.52	KDChart::ReverseMapper Class Reference . . . . .	1017
9.53	KDChart::RingDiagram Class Reference . . . . .	1022
9.54	KDChart::SignalCompressor Class Reference . . . . .	1073
9.55	KDChart::TernaryAxis Class Reference . . . . .	1075
9.56	KDChart::TernaryCoordinatePlane Class Reference . . . . .	1099
9.57	KDChart::TernaryLineDiagram Class Reference . . . . .	1134
9.58	KDChart::TernaryPointDiagram Class Reference . . . . .	1179
9.59	KDChart::TextArea Class Reference . . . . .	1223
9.60	KDChart::TextAttributes Class Reference . . . . .	1240
9.61	KDChart::TextLayoutItem Class Reference . . . . .	1250
9.62	KDChart::ThreeDBarAttributes Class Reference . . . . .	1261
9.63	KDChart::ThreeDLineAttributes Class Reference . . . . .	1266
9.64	KDChart::ThreeDPieAttributes Class Reference . . . . .	1271
9.65	KDChart::ValueTrackerAttributes Class Reference . . . . .	1276
9.66	KDChart::VerticalLineLayoutItem Class Reference . . . . .	1281
9.67	KDChart::Widget Class Reference . . . . .	1287
9.68	KDChart::ZoomParameters Class Reference . . . . .	1305
9.69	KDChartEnums Class Reference . . . . .	1308
9.70	KDTextDocument Class Reference . . . . .	1319
9.71	PrerenderedElement Class Reference . . . . .	1322
9.72	PrerenderedLabel Class Reference . . . . .	1325
9.73	QAbstractItemView Class Reference . . . . .	1333
9.74	QAbstractProxyModel Class Reference . . . . .	1334
9.75	QFrame Class Reference . . . . .	1335

9.76	<a href="#">QGraphicsPolygonItem Class Reference</a>	1336
9.77	<a href="#">QLayoutItem Class Reference</a>	1337
9.78	<a href="#">QObject Class Reference</a>	1338
9.79	<a href="#">QSortFilterProxyModel Class Reference</a>	1339
9.80	<a href="#">QTextDocument Class Reference</a>	1340
9.81	<a href="#">QWidget Class Reference</a>	1341
9.82	<a href="#">TernaryPoint Class Reference</a>	1342
<b>10</b>	<b>KD Chart 2 File Documentation</b>	<b>1345</b>
10.1	<a href="#">ChartGraphicsItem.cpp File Reference</a>	1345
10.2	<a href="#">ChartGraphicsItem.h File Reference</a>	1346
10.3	<a href="#">KDChartAbstractArea.cpp File Reference</a>	1347
10.4	<a href="#">KDChartAbstractArea.h File Reference</a>	1353
10.5	<a href="#">KDChartAbstractAreaBase.cpp File Reference</a>	1354
10.6	<a href="#">KDChartAbstractAreaBase.h File Reference</a>	1355
10.7	<a href="#">KDChartAbstractAreaWidget.cpp File Reference</a>	1357
10.8	<a href="#">KDChartAbstractAreaWidget.h File Reference</a>	1358
10.9	<a href="#">KDChartAbstractAxis.cpp File Reference</a>	1359
10.10	<a href="#">KDChartAbstractAxis.h File Reference</a>	1360
10.11	<a href="#">KDChartAbstractCartesianDiagram.cpp File Reference</a>	1361
10.12	<a href="#">KDChartAbstractCartesianDiagram.h File Reference</a>	1362
10.13	<a href="#">KDChartAbstractCoordinatePlane.cpp File Reference</a>	1364
10.14	<a href="#">KDChartAbstractCoordinatePlane.h File Reference</a>	1365
10.15	<a href="#">KDChartAbstractDiagram.cpp File Reference</a>	1367
10.16	<a href="#">KDChartAbstractDiagram.h File Reference</a>	1369
10.17	<a href="#">KDChartAbstractPieDiagram.cpp File Reference</a>	1371
10.18	<a href="#">KDChartAbstractPieDiagram.h File Reference</a>	1372
10.19	<a href="#">KDChartAbstractPolarDiagram.cpp File Reference</a>	1373
10.20	<a href="#">KDChartAbstractPolarDiagram.h File Reference</a>	1374
10.21	<a href="#">KDChartAbstractProxyModel.cpp File Reference</a>	1375
10.22	<a href="#">KDChartAbstractProxyModel.h File Reference</a>	1376
10.23	<a href="#">KDChartAbstractTernaryDiagram.cpp File Reference</a>	1377
10.24	<a href="#">KDChartAbstractTernaryDiagram.h File Reference</a>	1378
10.25	<a href="#">KDChartAbstractThreeDAttributes.cpp File Reference</a>	1379
10.26	<a href="#">KDChartAbstractThreeDAttributes.h File Reference</a>	1380
10.27	<a href="#">KDChartAttributesModel.cpp File Reference</a>	1382
10.28	<a href="#">KDChartAttributesModel.h File Reference</a>	1384

10.29KDChartBackgroundAttributes.cpp File Reference . . . . .	1386
10.30KDChartBackgroundAttributes.h File Reference . . . . .	1387
10.31KDChartBarAttributes.cpp File Reference . . . . .	1389
10.32KDChartBarAttributes.h File Reference . . . . .	1390
10.33KDChartBarDiagram.cpp File Reference . . . . .	1391
10.34KDChartBarDiagram.h File Reference . . . . .	1393
10.35KDChartBarDiagram_p.cpp File Reference . . . . .	1394
10.36KDChartCartesianAxis.cpp File Reference . . . . .	1395
10.37KDChartCartesianAxis.h File Reference . . . . .	1399
10.38KDChartCartesianCoordinatePlane.cpp File Reference . . . . .	1400
10.39KDChartCartesianCoordinatePlane.h File Reference . . . . .	1402
10.40KDChartCartesianDiagramDataCompressor_p.cpp File Reference . . . . .	1403
10.41 KDChartChart.cpp File Reference . . . . .	1404
10.42KDChartChart.h File Reference . . . . .	1408
10.43KDChartDatasetProxyModel.cpp File Reference . . . . .	1410
10.44KDChartDatasetProxyModel.h File Reference . . . . .	1411
10.45KDChartDatasetSelector.cpp File Reference . . . . .	1412
10.46KDChartDatasetSelector.h File Reference . . . . .	1413
10.47KDChartDataValueAttributes.cpp File Reference . . . . .	1414
10.48KDChartDataValueAttributes.h File Reference . . . . .	1416
10.49KDChartDiagramObserver.cpp File Reference . . . . .	1418
10.50KDChartDiagramObserver.h File Reference . . . . .	1419
10.51 KDChartEnums.h File Reference . . . . .	1420
10.52KDChartFrameAttributes.cpp File Reference . . . . .	1422
10.53KDChartFrameAttributes.h File Reference . . . . .	1423
10.54KDChartGlobal.h File Reference . . . . .	1425
10.55KDChartGridAttributes.cpp File Reference . . . . .	1432
10.56KDChartGridAttributes.h File Reference . . . . .	1433
10.57KDChartHeaderFooter.cpp File Reference . . . . .	1435
10.58KDChartHeaderFooter.h File Reference . . . . .	1437
10.59KDChartLayoutItems.cpp File Reference . . . . .	1438
10.60KDChartLayoutItems.h File Reference . . . . .	1442
10.61 KDChartLegend.cpp File Reference . . . . .	1444
10.62KDChartLegend.h File Reference . . . . .	1446
10.63KDChartLineAttributes.cpp File Reference . . . . .	1447
10.64KDChartLineAttributes.h File Reference . . . . .	1448

10.65KDChartLineDiagram.cpp File Reference . . . . .	1450
10.66KDChartLineDiagram.h File Reference . . . . .	1452
10.67KDChartLineDiagram_p.cpp File Reference . . . . .	1453
10.68KDChartMarkerAttributes.cpp File Reference . . . . .	1454
10.69KDChartMarkerAttributes.h File Reference . . . . .	1456
10.70KDChartMeasure.cpp File Reference . . . . .	1458
10.71KDChartMeasure.h File Reference . . . . .	1460
10.72KDChartNormalBarDiagram_p.cpp File Reference . . . . .	1462
10.73KDChartNormalLineDiagram_p.cpp File Reference . . . . .	1463
10.74KDChartNormalPlotter_p.cpp File Reference . . . . .	1464
10.75KDChartPaintContext.cpp File Reference . . . . .	1465
10.76KDChartPaintContext.h File Reference . . . . .	1466
10.77KDChartPalette.cpp File Reference . . . . .	1467
10.78KDChartPalette.h File Reference . . . . .	1470
10.79KDChartPercentBarDiagram_p.cpp File Reference . . . . .	1471
10.80KDChartPercentLineDiagram_p.cpp File Reference . . . . .	1472
10.81KDChartPieAttributes.cpp File Reference . . . . .	1473
10.82KDChartPieAttributes.h File Reference . . . . .	1474
10.83KDChartPieDiagram.cpp File Reference . . . . .	1476
10.84KDChartPieDiagram.h File Reference . . . . .	1478
10.85KDChartPlotter.cpp File Reference . . . . .	1479
10.86KDChartPlotter.h File Reference . . . . .	1480
10.87KDChartPlotter_p.cpp File Reference . . . . .	1481
10.88KDChartPolarCoordinatePlane.cpp File Reference . . . . .	1482
10.89KDChartPolarCoordinatePlane.h File Reference . . . . .	1484
10.90KDChartPolarDiagram.cpp File Reference . . . . .	1485
10.91KDChartPolarDiagram.h File Reference . . . . .	1486
10.92KDChartPosition.cpp File Reference . . . . .	1487
10.93KDChartPosition.h File Reference . . . . .	1490
10.94KDChartRelativePosition.cpp File Reference . . . . .	1492
10.95KDChartRelativePosition.h File Reference . . . . .	1494
10.96KDChartRingDiagram.cpp File Reference . . . . .	1496
10.97KDChartRingDiagram.h File Reference . . . . .	1497
10.98KDChartSignalCompressor.cpp File Reference . . . . .	1498
10.99KDChartSignalCompressor.h File Reference . . . . .	1499
10.100KDChartStackedBarDiagram_p.cpp File Reference . . . . .	1500

10.10KDChartStackedLineDiagram_p.cpp File Reference . . . . .	1501
10.10KDChartTernaryAxis.cpp File Reference . . . . .	1502
10.10KDChartTernaryAxis.h File Reference . . . . .	1503
10.10KDChartTernaryCoordinatePlane.cpp File Reference . . . . .	1504
10.10KDChartTernaryCoordinatePlane.h File Reference . . . . .	1505
10.10KDChartTernaryLineDiagram.cpp File Reference . . . . .	1506
10.10KDChartTernaryLineDiagram.h File Reference . . . . .	1507
10.10KDChartTernaryPointDiagram.cpp File Reference . . . . .	1508
10.10KDChartTernaryPointDiagram.h File Reference . . . . .	1509
10.11KDChartTextArea.cpp File Reference . . . . .	1510
10.11KDChartTextArea.h File Reference . . . . .	1511
10.11KDChartTextAttributes.cpp File Reference . . . . .	1512
10.11KDChartTextAttributes.h File Reference . . . . .	1514
10.11KDChartTextLabelCache.cpp File Reference . . . . .	1516
10.11KDChartTextLabelCache.h File Reference . . . . .	1518
10.11KDChartThreeDBarAttributes.cpp File Reference . . . . .	1519
10.11KDChartThreeDBarAttributes.h File Reference . . . . .	1520
10.11KDChartThreeDLineAttributes.cpp File Reference . . . . .	1522
10.11KDChartThreeDLineAttributes.h File Reference . . . . .	1523
10.12KDChartThreeDPieAttributes.cpp File Reference . . . . .	1525
10.12KDChartThreeDPieAttributes.h File Reference . . . . .	1526
10.12KDChartValueTrackerAttributes.cpp File Reference . . . . .	1528
10.12KDChartValueTrackerAttributes.h File Reference . . . . .	1529
10.12KDChartWidget.cpp File Reference . . . . .	1531
10.12KDChartWidget.h File Reference . . . . .	1534
10.12KDChartZoomParameters.h File Reference . . . . .	1535
10.12KDTextDocument.cpp File Reference . . . . .	1536
10.12KDTextDocument.h File Reference . . . . .	1537
10.12ReverseMapper.cpp File Reference . . . . .	1538
10.12ReverseMapper.h File Reference . . . . .	1539
10.13TernaryConstants.cpp File Reference . . . . .	1540
10.13TernaryConstants.h File Reference . . . . .	1542
10.13TernaryPoint.cpp File Reference . . . . .	1544
10.13TernaryPoint.h File Reference . . . . .	1546
<b>11 KD Chart 2 Page Documentation</b>	<b>1549</b>
11.1 Deprecated List . . . . .	1549



# Chapter 1

## KD Chart 2 Directory Hierarchy

### 1.1 KD Chart 2 Directories

This directory hierarchy is sorted roughly, but not completely, alphabetically:

src . . . . .	21
PrerenderedElements . . . . .	19
Scenery . . . . .	20
Ternary . . . . .	25





# Chapter 2

## KD Chart 2 Namespace Index

### 2.1 KD Chart 2 Namespace List

Here is a list of all namespaces with brief descriptions:

<a href="#">KDChart</a>	.....	27
-------------------------	-------	----



## Chapter 3

# KD Chart 2 Hierarchical Index

### 3.1 KD Chart 2 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

KDChart::AbstractAreaBase . . . . .	48
KDChart::AbstractArea . . . . .	35
KDChart::AbstractAxis . . . . .	66
KDChart::CartesianAxis . . . . .	447
KDChart::TernaryAxis . . . . .	1075
KDChart::AbstractCoordinatePlane . . . . .	134
KDChart::CartesianCoordinatePlane . . . . .	487
KDChart::PolarCoordinatePlane . . . . .	887
KDChart::TernaryCoordinatePlane . . . . .	1099
KDChart::AbstractAreaWidget . . . . .	55
KDChart::Legend . . . . .	637
KDChart::TextArea . . . . .	1223
KDChart::HeaderFooter . . . . .	611
KDChart::AbstractThreeDAttributes . . . . .	350
KDChart::ThreeDBarAttributes . . . . .	1261
KDChart::ThreeDLineAttributes . . . . .	1266
KDChart::ThreeDPieAttributes . . . . .	1271
KDChart::BackgroundAttributes . . . . .	378
KDChart::BarAttributes . . . . .	383
KDChart::DataDimension . . . . .	567
KDChart::DataValueAttributes . . . . .	581
KDChart::FrameAttributes . . . . .	596
KDChart::GlobalMeasureScaling . . . . .	600
KDChart::GridAttributes . . . . .	603
KDChart::LineAttributes . . . . .	670
KDChart::MarkerAttributes . . . . .	745
KDChart::Measure . . . . .	758
KDChart::PaintContext . . . . .	765
KDChart::PieAttributes . . . . .	773
KDChart::Position . . . . .	976
KDChart::PositionPoints . . . . .	985
KDChart::RelativePosition . . . . .	1008

KDChart::ReverseMapper . . . . .	1017
KDChart::TextAttributes . . . . .	1240
KDChart::ValueTrackerAttributes . . . . .	1276
KDChart::ZoomParameters . . . . .	1305
PrerenderedElement . . . . .	1322
PrerenderedLabel . . . . .	1325
QAbstractItemView . . . . .	1333
KDChart::AbstractDiagram . . . . .	167
KDChart::AbstractCartesianDiagram . . . . .	86
KDChart::BarDiagram . . . . .	390
KDChart::LineDiagram . . . . .	674
KDChart::Plotter . . . . .	830
KDChart::AbstractPolarDiagram . . . . .	261
KDChart::AbstractPieDiagram . . . . .	213
KDChart::PieDiagram . . . . .	777
KDChart::RingDiagram . . . . .	1022
KDChart::PolarDiagram . . . . .	927
KDChart::AbstractTernaryDiagram . . . . .	308
KDChart::TernaryLineDiagram . . . . .	1134
KDChart::TernaryPointDiagram . . . . .	1179
QAbstractProxyModel . . . . .	1334
KDChart::AbstractProxyModel . . . . .	305
KDChart::AttributesModel . . . . .	353
KDChart::PrivateAttributesModel . . . . .	990
QFrame . . . . .	1335
KDChart::DatasetSelectorWidget . . . . .	579
QGraphicsPolygonItem . . . . .	1336
KDChart::ChartGraphicsItem . . . . .	565
QLayoutItem . . . . .	1337
KDChart::AbstractLayoutItem . . . . .	209
KDChart::AbstractArea . . . . .	35
KDChart::AutoSpacerLayoutItem . . . . .	371
KDChart::HorizontalLineLayoutItem . . . . .	631
KDChart::LineLayoutItem . . . . .	733
KDChart::LineWithMarkerLayoutItem . . . . .	739
KDChart::MarkerLayoutItem . . . . .	751
KDChart::TextLayoutItem . . . . .	1250
KDChart::TextArea . . . . .	1223
KDChart::VerticalLineLayoutItem . . . . .	1281
QObject . . . . .	1338
KDChart::AbstractArea . . . . .	35
KDChart::DiagramObserver . . . . .	593
KDChart::Palette . . . . .	768
KDChart::SignalCompressor . . . . .	1073
KDChart::TextArea . . . . .	1223
KDChartEnums . . . . .	1308
QSortFilterProxyModel . . . . .	1339
KDChart::DatasetProxyModel . . . . .	571
QTextDocument . . . . .	1340
KDTextDocument . . . . .	1319

---

QWidget . . . . .	1341
KDChart::AbstractAreaWidget . . . . .	55
KDChart::Chart . . . . .	543
KDChart::Widget . . . . .	1287
TernaryPoint . . . . .	1342



## Chapter 4

# KD Chart 2 Class Index

### 4.1 KD Chart 2 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#">KDChart::AbstractArea</a> (An area in the chart with a background, a frame, etc ) . . . . .	35
<a href="#">KDChart::AbstractAreaBase</a> (Base class for <a href="#">AbstractArea</a> and <a href="#">AbstractAreaWidget</a> : An area in the chart with a background, a frame, etc ) . . . . .	48
<a href="#">KDChart::AbstractAreaWidget</a> (An area in the chart with a background, a frame, etc ) . . . . .	55
<a href="#">KDChart::AbstractAxis</a> (The base class for axes ) . . . . .	66
<a href="#">KDChart::AbstractCartesianDiagram</a> (Base class for diagrams based on a cartesian coordinate system ) . . . . .	86
<a href="#">KDChart::AbstractCoordinatePlane</a> (Base class common for all coordinate planes, <a href="#">CartesianCoordinatePlane</a> , <a href="#">PolarCoordinatePlane</a> , <a href="#">TernaryCoordinatePlane</a> ) . . . . .	134
<a href="#">KDChart::AbstractDiagram</a> ( <a href="#">AbstractDiagram</a> defines the interface for diagram classes ) . . . . .	167
<a href="#">KDChart::AbstractLayoutItem</a> (Base class for all layout items of <a href="#">KD Chart</a> ) . . . . .	209
<a href="#">KDChart::AbstractPieDiagram</a> (Base class for any diagram type ) . . . . .	213
<a href="#">KDChart::AbstractPolarDiagram</a> (Base class for diagrams based on a polar coordinate system ) . . . . .	261
<a href="#">KDChart::AbstractProxyModel</a> (Base class for all proxy models used inside <a href="#">KD Chart</a> ) . . . . .	305
<a href="#">KDChart::AbstractTernaryDiagram</a> (Base class for diagrams based on a ternary coordinate plane ) . . . . .	308
<a href="#">KDChart::AbstractThreeDAttributes</a> (Base class for 3D attributes ) . . . . .	350
<a href="#">KDChart::AttributesModel</a> (A proxy model used for storing attributes ) . . . . .	353
<a href="#">KDChart::AutoSpacerLayoutItem</a> (An empty layout item ) . . . . .	371
<a href="#">KDChart::BackgroundAttributes</a> (Set of attributes usable for background pixmaps ) . . . . .	378
<a href="#">KDChart::BarAttributes</a> (Set of attributes for changing the appearance of bar charts ) . . . . .	383
<a href="#">KDChart::BarDiagram</a> ( <a href="#">BarDiagram</a> defines a common bar diagram ) . . . . .	390
<a href="#">KDChart::CartesianAxis</a> (The class for cartesian axes ) . . . . .	447
<a href="#">KDChart::CartesianCoordinatePlane</a> (Cartesian coordinate plane ) . . . . .	487
<a href="#">KDChart::Chart</a> (A chart with one or more diagrams ) . . . . .	543
<a href="#">KDChart::ChartGraphicsItem</a> (Graphics item used inside of the <a href="#">ReverseMapper</a> ) . . . . .	565
<a href="#">KDChart::DataDimension</a> (Helper class for one dimension of data, e.g ) . . . . .	567
<a href="#">KDChart::DatasetProxyModel</a> ( <a href="#">DatasetProxyModel</a> takes a <a href="#">KDChart</a> dataset configuration and translates it into a filtering proxy model ) . . . . .	571
<a href="#">KDChart::DatasetSelectorWidget</a> . . . . .	579
<a href="#">KDChart::DataValueAttributes</a> (Diagram attributes dealing with data value labels ) . . . . .	581
<a href="#">KDChart::DiagramObserver</a> (A <a href="#">DiagramObserver</a> watches the associated diagram for changes and deletion and emits corresponding signals ) . . . . .	593
<a href="#">KDChart::FrameAttributes</a> (A set of attributes for frames around items ) . . . . .	596

<a href="#">KDChart::GlobalMeasureScaling</a> (Auxiliary class used by the <a href="#">KDChart::Measure</a> and <a href="#">KDChart::Chart</a> class ) . . . . .	600
<a href="#">KDChart::GridAttributes</a> (A set of attributes controlling the appearance of grids ) . . . . .	603
<a href="#">KDChart::HeaderFooter</a> (A header or even footer displaying text above or below charts ) . . . . .	611
<a href="#">KDChart::HorizontalLineLayoutItem</a> (Layout item showing a horizontal line ) . . . . .	631
<a href="#">KDChart::Legend</a> ( <a href="#">Legend</a> defines the interface for the legend drawing class ) . . . . .	637
<a href="#">KDChart::LineAttributes</a> (Set of attributes for changing the appearance of line charts ) . . . . .	670
<a href="#">KDChart::LineDiagram</a> ( <a href="#">LineDiagram</a> defines a common line diagram ) . . . . .	674
<a href="#">KDChart::LineLayoutItem</a> (Layout item showing a coloured line ) . . . . .	733
<a href="#">KDChart::LineWithMarkerLayoutItem</a> (Layout item showing a coloured line and a data point marker ) . . . . .	739
<a href="#">KDChart::MarkerAttributes</a> (A set of attributes controlling the appearance of data set markers ) . . . . .	745
<a href="#">KDChart::MarkerLayoutItem</a> (Layout item showing a data point marker ) . . . . .	751
<a href="#">KDChart::Measure</a> ( <a href="#">Measure</a> is used to specify all relative and/or absolute measures in <a href="#">KDChart</a> , e.g ) . . . . .	758
<a href="#">KDChart::PaintContext</a> (Stores information about painting diagrams ) . . . . .	765
<a href="#">KDChart::Palette</a> (A <a href="#">Palette</a> is a set of brushes (or colors) to be used for painting data sets ) . . . . .	768
<a href="#">KDChart::PieAttributes</a> (A set of attributes controlling the appearance of pie charts ) . . . . .	773
<a href="#">KDChart::PieDiagram</a> ( <a href="#">PieDiagram</a> defines a common pie diagram ) . . . . .	777
<a href="#">KDChart::Plotter</a> ( <a href="#">Plotter</a> defines a diagram type plotting two-dimensional data ) . . . . .	830
<a href="#">KDChart::PolarCoordinatePlane</a> (Polar coordinate plane ) . . . . .	887
<a href="#">KDChart::PolarDiagram</a> ( <a href="#">PolarDiagram</a> defines a common polar diagram ) . . . . .	927
<a href="#">KDChart::Position</a> (Defines a position, using compass terminology ) . . . . .	976
<a href="#">KDChart::PositionPoints</a> (Stores the absolute target points of a <a href="#">Position</a> ) . . . . .	985
<a href="#">KDChart::PrivateAttributesModel</a> (Internally used class just adding a special constructor used by <a href="#">AbstractDiagram</a> ) . . . . .	990
<a href="#">KDChart::RelativePosition</a> (Defines relative position information: reference area, position in this area, horizontal / vertical padding, and rotating ) . . . . .	1008
<a href="#">KDChart::ReverseMapper</a> (The <a href="#">ReverseMapper</a> stores information about objects on a chart and their respective model indexes ) . . . . .	1017
<a href="#">KDChart::RingDiagram</a> ( <a href="#">RingDiagram</a> defines a common ring diagram ) . . . . .	1022
<a href="#">KDChart::SignalCompressor</a> ( <a href="#">SignalCompressor</a> compresses signals where the same signal needs to be emitted by several pieces of the code, but only one of the signals should be received at the end ) . . . . .	1073
<a href="#">KDChart::TernaryAxis</a> (The class for ternary axes ) . . . . .	1075
<a href="#">KDChart::TernaryCoordinatePlane</a> (Ternary coordinate plane ) . . . . .	1099
<a href="#">KDChart::TernaryLineDiagram</a> (A <a href="#">TernaryLineDiagram</a> is a line diagram with a ternary coordinate plane ) . . . . .	1134
<a href="#">KDChart::TernaryPointDiagram</a> (A <a href="#">TernaryPointDiagram</a> is a point diagram within a ternary coordinate plane ) . . . . .	1179
<a href="#">KDChart::TextArea</a> (A text area in the chart with a background, a frame, etc ) . . . . .	1223
<a href="#">KDChart::TextAttributes</a> (A set of text attributes ) . . . . .	1240
<a href="#">KDChart::TextLayoutItem</a> (Layout item showing a text ) . . . . .	1250
<a href="#">KDChart::ThreeDBarAttributes</a> (A set of 3D bar attributes ) . . . . .	1261
<a href="#">KDChart::ThreeDLineAttributes</a> (A set of 3D line attributes ) . . . . .	1266
<a href="#">KDChart::ThreeDPieAttributes</a> (A set of 3D pie attributes ) . . . . .	1271
<a href="#">KDChart::ValueTrackerAttributes</a> (Cell-specific attributes regarding value tracking ) . . . . .	1276
<a href="#">KDChart::VerticalLineLayoutItem</a> (Layout item showing a vertical line ) . . . . .	1281
<a href="#">KDChart::Widget</a> (The <a href="#">KD Chart</a> widget for usage without Model/View ) . . . . .	1287
<a href="#">KDChart::ZoomParameters</a> ( <a href="#">ZoomParameters</a> stores the center and the factor of zooming internally ) . . . . .	1305
<a href="#">KDChartEnums</a> (Project global class providing some enums needed both by <a href="#">KDChartParams</a> and by <a href="#">KDChartCustomBox</a> ) . . . . .	1308
<a href="#">KDTextDocument</a> ( <a href="#">KDTextDocument</a> is an internally used enhanced <a href="#">QTextDocument</a> ) . . . . .	1319



<a href="#">PrerenderedElement</a> (Base class for prerendered elements like labels, pixmaps, markers, etc ) . .	1322
<a href="#">PrerenderedLabel</a> ( <a href="#">PrerenderedLabel</a> is an internal <a href="#">KDChart</a> class that simplifies creation and caching of cached text labels ) . . . . .	1325
<a href="#">QAbstractItemView</a> (Class only listed here to document inheritance of some <a href="#">KDChart</a> classes ) .	1333
<a href="#">QAbstractProxyModel</a> (Class only listed here to document inheritance of some <a href="#">KDChart</a> classes )	1334
<a href="#">QFrame</a> (Class only listed here to document inheritance of some <a href="#">KDChart</a> classes ) . . . . .	1335
<a href="#">QGraphicsPolygonItem</a> (Class only listed here to document inheritance of some <a href="#">KDChart</a> classes )	1336
<a href="#">QLayoutItem</a> (Class only listed here to document inheritance of some <a href="#">KDChart</a> classes ) . . . .	1337
<a href="#">QObject</a> (Class only listed here to document inheritance of some <a href="#">KDChart</a> classes ) . . . . .	1338
<a href="#">QSortFilterProxyModel</a> (Class only listed here to document inheritance of some <a href="#">KDChart</a> classes )	1339
<a href="#">QTextDocument</a> (Class only listed here to document inheritance of some <a href="#">KDChart</a> classes ) . . .	1340
<a href="#">QWidget</a> (Class only listed here to document inheritance of some <a href="#">KDChart</a> classes ) . . . . .	1341
<a href="#">TernaryPoint</a> ( <a href="#">TernaryPoint</a> defines a point within a ternary coordinate plane ) . . . . .	1342



# Chapter 5

## KD Chart 2 File Index

### 5.1 KD Chart 2 File List

Here is a list of all files with brief descriptions:

<a href="#">ChartGraphicsItem.cpp</a>	1345
<a href="#">ChartGraphicsItem.h</a>	1346
<a href="#">KDChartAbstractArea.cpp</a>	1347
<a href="#">KDChartAbstractArea.h</a>	1353
<a href="#">KDChartAbstractAreaBase.cpp</a>	1354
<a href="#">KDChartAbstractAreaBase.h</a>	1355
<a href="#">KDChartAbstractAreaWidget.cpp</a>	1357
<a href="#">KDChartAbstractAreaWidget.h</a>	1358
<a href="#">KDChartAbstractAxis.cpp</a>	1359
<a href="#">KDChartAbstractAxis.h</a>	1360
<a href="#">KDChartAbstractCartesianDiagram.cpp</a>	1361
<a href="#">KDChartAbstractCartesianDiagram.h</a>	1362
<a href="#">KDChartAbstractCoordinatePlane.cpp</a>	1364
<a href="#">KDChartAbstractCoordinatePlane.h</a>	1365
<a href="#">KDChartAbstractDiagram.cpp</a>	1367
<a href="#">KDChartAbstractDiagram.h</a>	1369
<a href="#">KDChartAbstractPieDiagram.cpp</a>	1371
<a href="#">KDChartAbstractPieDiagram.h</a>	1372
<a href="#">KDChartAbstractPolarDiagram.cpp</a>	1373
<a href="#">KDChartAbstractPolarDiagram.h</a>	1374
<a href="#">KDChartAbstractProxyModel.cpp</a>	1375
<a href="#">KDChartAbstractProxyModel.h</a>	1376
<a href="#">KDChartAbstractTernaryDiagram.cpp</a>	1377
<a href="#">KDChartAbstractTernaryDiagram.h</a>	1378
<a href="#">KDChartAbstractThreeDAttributes.cpp</a>	1379
<a href="#">KDChartAbstractThreeDAttributes.h</a>	1380
<a href="#">KDChartAttributesModel.cpp</a>	1382
<a href="#">KDChartAttributesModel.h</a>	1384
<a href="#">KDChartBackgroundAttributes.cpp</a>	1386
<a href="#">KDChartBackgroundAttributes.h</a>	1387
<a href="#">KDChartBarAttributes.cpp</a>	1389
<a href="#">KDChartBarAttributes.h</a>	1390
<a href="#">KDChartBarDiagram.cpp</a>	1391

KDChartBarDiagram.h	1393
KDChartBarDiagram_p.cpp	1394
KDChartCartesianAxis.cpp	1395
KDChartCartesianAxis.h	1399
KDChartCartesianCoordinatePlane.cpp	1400
KDChartCartesianCoordinatePlane.h	1402
KDChartCartesianDiagramDataCompressor_p.cpp	1403
KDChartChart.cpp	1404
KDChartChart.h (Declaring the class <code>KDChart::Chart</code> )	1408
KDChartDatasetProxyModel.cpp	1410
KDChartDatasetProxyModel.h	1411
KDChartDatasetSelector.cpp	1412
KDChartDatasetSelector.h	1413
KDChartDataValueAttributes.cpp	1414
KDChartDataValueAttributes.h (Declaring the class <code>KDChart::DataValueAttributes</code> )	1416
KDChartDiagramObserver.cpp	1418
KDChartDiagramObserver.h	1419
KDChartEnums.h (Definition of global enums)	1420
KDChartFrameAttributes.cpp	1422
KDChartFrameAttributes.h	1423
KDChartGlobal.h	1425
KDChartGridAttributes.cpp	1432
KDChartGridAttributes.h	1433
KDChartHeaderFooter.cpp	1435
KDChartHeaderFooter.h	1437
KDChartLayoutItems.cpp	1438
KDChartLayoutItems.h	1442
KDChartLegend.cpp	1444
KDChartLegend.h	1446
KDChartLineAttributes.cpp	1447
KDChartLineAttributes.h	1448
KDChartLineDiagram.cpp	1450
KDChartLineDiagram.h	1452
KDChartLineDiagram_p.cpp	1453
KDChartMarkerAttributes.cpp	1454
KDChartMarkerAttributes.h	1456
KDChartMeasure.cpp	1458
KDChartMeasure.h (Declaring the class <code>KDChart::Measure</code> )	1460
KDChartNormalBarDiagram_p.cpp	1462
KDChartNormalLineDiagram_p.cpp	1463
KDChartNormalPlotter_p.cpp	1464
KDChartPaintContext.cpp	1465
KDChartPaintContext.h	1466
KDChartPalette.cpp	1467
KDChartPalette.h	1470
KDChartPercentBarDiagram_p.cpp	1471
KDChartPercentLineDiagram_p.cpp	1472
KDChartPieAttributes.cpp	1473
KDChartPieAttributes.h	1474
KDChartPieDiagram.cpp	1476
KDChartPieDiagram.h	1478
KDChartPlotter.cpp	1479
KDChartPlotter.h	1480
KDChartPlotter_p.cpp	1481

KDChartPolarCoordinatePlane.cpp	1482
KDChartPolarCoordinatePlane.h	1484
KDChartPolarDiagram.cpp	1485
KDChartPolarDiagram.h	1486
KDChartPosition.cpp	1487
KDChartPosition.h	1490
KDChartRelativePosition.cpp	1492
KDChartRelativePosition.h	1494
KDChartRingDiagram.cpp	1496
KDChartRingDiagram.h	1497
KDChartSignalCompressor.cpp	1498
KDChartSignalCompressor.h	1499
KDChartStackedBarDiagram_p.cpp	1500
KDChartStackedLineDiagram_p.cpp	1501
KDChartTernaryAxis.cpp	1502
KDChartTernaryAxis.h	1503
KDChartTernaryCoordinatePlane.cpp	1504
KDChartTernaryCoordinatePlane.h	1505
KDChartTernaryLineDiagram.cpp	1506
KDChartTernaryLineDiagram.h	1507
KDChartTernaryPointDiagram.cpp	1508
KDChartTernaryPointDiagram.h	1509
KDChartTextArea.cpp	1510
KDChartTextArea.h	1511
KDChartTextAttributes.cpp	1512
KDChartTextAttributes.h	1514
KDChartTextLabelCache.cpp	1516
KDChartTextLabelCache.h	1518
KDChartThreeDBarAttributes.cpp	1519
KDChartThreeDBarAttributes.h	1520
KDChartThreeDLineAttributes.cpp	1522
KDChartThreeDLineAttributes.h	1523
KDChartThreeDPieAttributes.cpp	1525
KDChartThreeDPieAttributes.h	1526
KDChartValueTrackerAttributes.cpp	1528
KDChartValueTrackerAttributes.h	1529
KDChartWidget.cpp	1531
KDChartWidget.h	1534
KDChartZoomParameters.h	1535
KDTextDocument.cpp	1536
KDTextDocument.h	1537
ReverseMapper.cpp	1538
ReverseMapper.h	1539
TernaryConstants.cpp	1540
TernaryConstants.h	1542
TernaryPoint.cpp	1544
TernaryPoint.h	1546



# Chapter 6

## KD Chart 2 Page Index

### 6.1 KD Chart 2 Related Pages

Here is a list of all related documentation pages:

Deprecated List . . . . .	<a href="#">1549</a>
---------------------------	----------------------

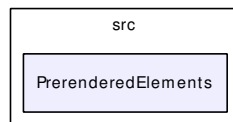




## Chapter 7

# KD Chart 2 Directory Documentation

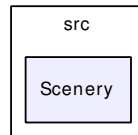
### 7.1 src/PrerenderedElements/ Directory Reference



#### Files

- file [KDChartTextLabelCache.cpp](#)
- file [KDChartTextLabelCache.h](#)

## 7.2 src/Scenery/ Directory Reference



### Files

- file [ChartGraphicsItem.cpp](#)
- file [ChartGraphicsItem.h](#)
- file [ReverseMapper.cpp](#)
- file [ReverseMapper.h](#)

## 7.3 src/ Directory Reference

### 7.3.1 Detailed Description

Implementation directory of [KDChart](#).

This directory contains the header files and the source files of both, the private and the public classes.

**Note:**

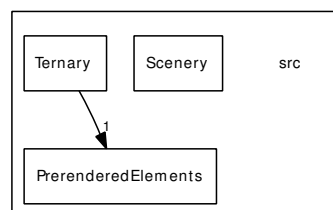
Only classes that have an include wrapper in the `$KDCHARTDIR/include` directory are part of the supported API. All other classes are to be considered as implementation details, they could be changed in future versions of [KDChart](#) without notice.

In other words: No class that is not mentioned in the `$KDCHARTDIR/include` directory may be directly used by your application.

The recommended way to include classes of the [KDChart](#) API is including them by class name, so instead of including [KDChartChart.h](#) you would say:

```
#include <KDChartChart>
```

When following this there is no reason to include the `$KDCHARTDIR/src` directory, it is sufficient to include `$KDCHARTDIR/include`



### Directories

- directory [PrerenderedElements](#)
- directory [Scenery](#)
- directory [Ternary](#)

### Files

- file [KDChartAbstractArea.cpp](#)
- file [KDChartAbstractArea.h](#)
- file [KDChartAbstractAreaBase.cpp](#)
- file [KDChartAbstractAreaBase.h](#)
- file [KDChartAbstractAreaWidget.cpp](#)
- file [KDChartAbstractAreaWidget.h](#)
- file [KDChartAbstractAxis.cpp](#)
- file [KDChartAbstractAxis.h](#)

- file [KDChartAbstractCartesianDiagram.cpp](#)
- file [KDChartAbstractCartesianDiagram.h](#)
- file [KDChartAbstractCoordinatePlane.cpp](#)
- file [KDChartAbstractCoordinatePlane.h](#)
- file [KDChartAbstractDiagram.cpp](#)
- file [KDChartAbstractDiagram.h](#)
- file [KDChartAbstractPieDiagram.cpp](#)
- file [KDChartAbstractPieDiagram.h](#)
- file [KDChartAbstractPolarDiagram.cpp](#)
- file [KDChartAbstractPolarDiagram.h](#)
- file [KDChartAbstractProxyModel.cpp](#)
- file [KDChartAbstractProxyModel.h](#)
- file [KDChartAbstractThreeDAttributes.cpp](#)
- file [KDChartAbstractThreeDAttributes.h](#)
- file [KDChartAttributesModel.cpp](#)
- file [KDChartAttributesModel.h](#)
- file [KDChartBackgroundAttributes.cpp](#)
- file [KDChartBackgroundAttributes.h](#)
- file [KDChartBarAttributes.cpp](#)
- file [KDChartBarAttributes.h](#)
- file [KDChartBarDiagram.cpp](#)
- file [KDChartBarDiagram.h](#)
- file [KDChartBarDiagram\\_p.cpp](#)
- file [KDChartCartesianAxis.cpp](#)
- file [KDChartCartesianAxis.h](#)
- file [KDChartCartesianCoordinatePlane.cpp](#)
- file [KDChartCartesianCoordinatePlane.h](#)
- file [KDChartCartesianDiagramDataCompressor\\_p.cpp](#)
- file [KDChartChart.cpp](#)
- file [KDChartChart.h](#)

*Declaring the class `KDChart::Chart`.*

- file [KDChartDatasetProxyModel.cpp](#)
- file [KDChartDatasetProxyModel.h](#)
- file [KDChartDatasetSelector.cpp](#)
- file [KDChartDatasetSelector.h](#)
- file [KDChartDataValueAttributes.cpp](#)
- file [KDChartDataValueAttributes.h](#)

*Declaring the class `KDChart::DataValueAttributes`.*

- file [KDChartDiagramObserver.cpp](#)
- file [KDChartDiagramObserver.h](#)
- file [KDChartEnums.h](#)

*Definition of global enums.*

- file [KDChartFrameAttributes.cpp](#)
- file [KDChartFrameAttributes.h](#)
- file [KDChartGlobal.h](#)
- file [KDChartGridAttributes.cpp](#)
- file [KDChartGridAttributes.h](#)

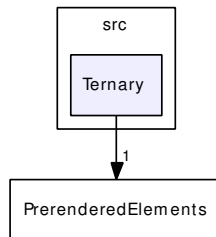
- file [KDChartHeaderFooter.cpp](#)
- file [KDChartHeaderFooter.h](#)
- file [KDChartLayoutItems.cpp](#)
- file [KDChartLayoutItems.h](#)
- file [KDChartLegend.cpp](#)
- file [KDChartLegend.h](#)
- file [KDChartLineAttributes.cpp](#)
- file [KDChartLineAttributes.h](#)
- file [KDChartLineDiagram.cpp](#)
- file [KDChartLineDiagram.h](#)
- file [KDChartLineDiagram\\_p.cpp](#)
- file [KDChartMarkerAttributes.cpp](#)
- file [KDChartMarkerAttributes.h](#)
- file [KDChartMeasure.cpp](#)
- file [KDChartMeasure.h](#)

*Declaring the class `KDChart::Measure`.*

- file [KDChartNormalBarDiagram\\_p.cpp](#)
- file [KDChartNormalLineDiagram\\_p.cpp](#)
- file [KDChartNormalPlotter\\_p.cpp](#)
- file [KDChartPaintContext.cpp](#)
- file [KDChartPaintContext.h](#)
- file [KDChartPalette.cpp](#)
- file [KDChartPalette.h](#)
- file [KDChartPercentBarDiagram\\_p.cpp](#)
- file [KDChartPercentLineDiagram\\_p.cpp](#)
- file [KDChartPieAttributes.cpp](#)
- file [KDChartPieAttributes.h](#)
- file [KDChartPieDiagram.cpp](#)
- file [KDChartPieDiagram.h](#)
- file [KDChartPlotter.cpp](#)
- file [KDChartPlotter.h](#)
- file [KDChartPlotter\\_p.cpp](#)
- file [KDChartPolarCoordinatePlane.cpp](#)
- file [KDChartPolarCoordinatePlane.h](#)
- file [KDChartPolarDiagram.cpp](#)
- file [KDChartPolarDiagram.h](#)
- file [KDChartPosition.cpp](#)
- file [KDChartPosition.h](#)
- file [KDChartRelativePosition.cpp](#)
- file [KDChartRelativePosition.h](#)
- file [KDChartRingDiagram.cpp](#)
- file [KDChartRingDiagram.h](#)
- file [KDChartSignalCompressor.cpp](#)
- file [KDChartSignalCompressor.h](#)
- file [KDChartStackedBarDiagram\\_p.cpp](#)
- file [KDChartStackedLineDiagram\\_p.cpp](#)
- file [KDChartTextArea.cpp](#)
- file [KDChartTextArea.h](#)
- file [KDChartTextAttributes.cpp](#)

- file [KDChartTextAttributes.h](#)
- file [KDChartThreeDBarAttributes.cpp](#)
- file [KDChartThreeDBarAttributes.h](#)
- file [KDChartThreeDLineAttributes.cpp](#)
- file [KDChartThreeDLineAttributes.h](#)
- file [KDChartThreeDPieAttributes.cpp](#)
- file [KDChartThreeDPieAttributes.h](#)
- file [KDChartValueTrackerAttributes.cpp](#)
- file [KDChartValueTrackerAttributes.h](#)
- file [KDChartWidget.cpp](#)
- file [KDChartWidget.h](#)
- file [KDChartZoomParameters.h](#)
- file [KDTextDocument.cpp](#)
- file [KDTextDocument.h](#)

## 7.4 src/Ternary/ Directory Reference



### Files

- file [KDChartAbstractTernaryDiagram.cpp](#)
- file [KDChartAbstractTernaryDiagram.h](#)
- file [KDChartTernaryAxis.cpp](#)
- file [KDChartTernaryAxis.h](#)
- file [KDChartTernaryCoordinatePlane.cpp](#)
- file [KDChartTernaryCoordinatePlane.h](#)
- file [KDChartTernaryLineDiagram.cpp](#)
- file [KDChartTernaryLineDiagram.h](#)
- file [KDChartTernaryPointDiagram.cpp](#)
- file [KDChartTernaryPointDiagram.h](#)
- file [TernaryConstants.cpp](#)
- file [TernaryConstants.h](#)
- file [TernaryPoint.cpp](#)
- file [TernaryPoint.h](#)





# Chapter 8

## KD Chart 2 Namespace Documentation

### 8.1 KDChart Namespace Reference

#### Classes

- class [AbstractArea](#)  
*An area in the chart with a background, a frame, etc.*
- class [AbstractAreaBase](#)  
*Base class for [AbstractArea](#) and [AbstractAreaWidget](#): An area in the chart with a background, a frame, etc.*
- class [AbstractAreaWidget](#)  
*An area in the chart with a background, a frame, etc.*
- class [AbstractAxis](#)  
*The base class for axes.*
- class [AbstractCartesianDiagram](#)  
*Base class for diagrams based on a cartesian coordinate system.*
- class [AbstractCoordinatePlane](#)  
*Base class common for all coordinate planes, [CartesianCoordinatePlane](#), [PolarCoordinatePlane](#), [TernaryCoordinatePlane](#).*
- class [AbstractDiagram](#)  
*[AbstractDiagram](#) defines the interface for diagram classes.*
- class [AbstractLayoutItem](#)  
*Base class for all layout items of KD Chart.*
- class [AbstractPieDiagram](#)  
*Base class for any diagram type.*
- class [AbstractPolarDiagram](#)  
*Base class for diagrams based on a polar coordinate system.*

- class [AbstractProxyModel](#)  
*Base class for all proxy models used inside KD [Chart](#).*
- class [AbstractTernaryDiagram](#)  
*Base class for diagrams based on a ternary coordinate plane.*
- class [AbstractThreeDAttributes](#)  
*Base class for 3D attributes.*
- class [AttributesModel](#)  
*A proxy model used for storing attributes.*
- class [AutoSpacerLayoutItem](#)  
*An empty layout item.*
- class [BackgroundAttributes](#)  
*Set of attributes usable for background pixmaps.*
- class [BarAttributes](#)  
*Set of attributes for changing the appearance of bar charts.*
- class [BarDiagram](#)  
*[BarDiagram](#) defines a common bar diagram.*
- class [CartesianAxis](#)  
*The class for cartesian axes.*
- class [CartesianCoordinatePlane](#)  
*Cartesian coordinate plane.*
- class [Chart](#)  
*A chart with one or more diagrams.*
- class [ChartGraphicsItem](#)  
*Graphics item used inside of the [ReverseMapper](#).*
- class [DataDimension](#)  
*Helper class for one dimension of data, e.g.*
- class [DatasetProxyModel](#)  
*[DatasetProxyModel](#) takes a [KDChart](#) dataset configuration and translates it into a filtering proxy model.*
- class [DatasetSelectorWidget](#)
- class [DataValueAttributes](#)  
*Diagram attributes dealing with data value labels.*
- class [DiagramObserver](#)  
*A [DiagramObserver](#) watches the associated diagram for changes and deletion and emits corresponding signals.*

- class [FrameAttributes](#)  
*A set of attributes for frames around items.*
- class [GlobalMeasureScaling](#)  
*Auxiliary class used by the [KDChart::Measure](#) and [KDChart::Chart](#) class.*
- class [GridAttributes](#)  
*A set of attributes controlling the appearance of grids.*
- class [HeaderFooter](#)  
*A header or even footer displaying text above or below charts.*
- class [HorizontalLineLayoutItem](#)  
*Layout item showing a horizontal line.*
- class [Legend](#)  
*[Legend](#) defines the interface for the legend drawing class.*
- class [LineAttributes](#)  
*Set of attributes for changing the appearance of line charts.*
- class [LineDiagram](#)  
*[LineDiagram](#) defines a common line diagram.*
- class [LineLayoutItem](#)  
*Layout item showing a coloured line.*
- class [LineWithMarkerLayoutItem](#)  
*Layout item showing a coloured line and a data point marker.*
- class [MarkerAttributes](#)  
*A set of attributes controlling the appearance of data set markers.*
- class [MarkerLayoutItem](#)  
*Layout item showing a data point marker.*
- class [Measure](#)  
*[Measure](#) is used to specify all relative and/or absolute measures in [KDChart](#), e.g.*
- class [PaintContext](#)  
*Stores information about painting diagrams.*
- class [Palette](#)  
*A [Palette](#) is a set of brushes (or colors) to be used for painting data sets.*
- class [PieAttributes](#)  
*A set of attributes controlling the appearance of pie charts.*
- class [PieDiagram](#)

*PieDiagram* defines a common pie diagram.

- class [Plotter](#)  
*Plotter* defines a diagram type plotting two-dimensional data.
- class [PolarCoordinatePlane](#)  
*Polar coordinate plane.*
- class [PolarDiagram](#)  
*PolarDiagram* defines a common polar diagram.
- class [Position](#)  
*Defines a position, using compass terminology.*
- class [PositionPoints](#)  
*Stores the absolute target points of a [Position](#).*
- class [PrivateAttributesModel](#)  
*Internally used class just adding a special constructor used by [AbstractDiagram](#).*
- class [RelativePosition](#)  
*Defines relative position information: reference area, position in this area, horizontal / vertical padding, and rotating.*
- class [ReverseMapper](#)  
*The [ReverseMapper](#) stores information about objects on a chart and their respective model indexes.*
- class [RingDiagram](#)  
*RingDiagram* defines a common ring diagram.
- class [SignalCompressor](#)  
*SignalCompressor* compresses signals where the same signal needs to be emitted by several pieces of the code, but only one of the signals should be received at the end.
- class [TernaryAxis](#)  
*The class for ternary axes.*
- class [TernaryCoordinatePlane](#)  
*Ternary coordinate plane.*
- class [TernaryLineDiagram](#)  
*A [TernaryLineDiagram](#) is a line diagram with a ternary coordinate plane.*
- class [TernaryPointDiagram](#)  
*A [TernaryPointDiagram](#) is a point diagram within a ternary coordinate plane.*
- class [TextArea](#)  
*A text area in the chart with a background, a frame, etc.*
- class [TextAttributes](#)

*A set of text attributes.*

- class [TextLayoutItem](#)  
*Layout item showing a text.*
- class [ThreeDBarAttributes](#)  
*A set of 3D bar attributes.*
- class [ThreeDLineAttributes](#)  
*A set of 3D line attributes.*
- class [ThreeDPieAttributes](#)  
*A set of 3D pie attributes.*
- class [ValueTrackerAttributes](#)  
*Cell-specific attributes regarding value tracking.*
- class [VerticalLineLayoutItem](#)  
*Layout item showing a vertical line.*
- class [Widget](#)  
*The KD [Chart](#) widget for usage without Model/View.*
- class [ZoomParameters](#)  
*[ZoomParameters](#) stores the center and the factor of zooming internally.*

## Typedefs

- typedef QList< [AbstractDiagram](#) \* > [AbstractDiagramList](#)
- typedef QList< [CartesianAxis](#) \* > [CartesianAxisList](#)
- typedef QList< const [AbstractDiagram](#) \* > [ConstAbstractDiagramList](#)
- typedef QList< const [AbstractDiagram](#) \* > [ConstDiagramList](#)
- typedef QList< [AbstractCoordinatePlane](#) \* > [CoordinatePlaneList](#)
- typedef QList< [DataDimension](#) > [DataDimensionsList](#)
- typedef QVector< int > [DatasetDescriptionVector](#)
- typedef QList< [AbstractDiagram](#) \* > [DiagramList](#)
- typedef QList< [HeaderFooter](#) \* > [HeaderFooterList](#)
- typedef QList< [Legend](#) \* > [LegendList](#)
- typedef QList< [TernaryAxis](#) \* > [TernaryAxisList](#)

## Enumerations

- enum [DisplayRoles](#) {  
    [DatasetPenRole](#) = 0x0A79EF95,  
    [DatasetBrushRole](#),  
    [DataValueLabelAttributesRole](#),  
    [ThreeDAttributesRole](#),  
};

```

    LineAttributesRole,
    ThreeDLineAttributesRole,
    BarAttributesRole,
    ThreeDBarAttributesRole,
    PieAttributesRole,
    ThreeDPieAttributesRole,
    DataHiddenRole,
    ValueTrackerAttributesRole }

```

## Functions

- QDebug [operator<<](#) (QDebug stream, const [DataDimension](#) &r)

### 8.1.1 Typedef Documentation

**8.1.1.1** `typedef QList<AbstractDiagram\*> KDChart::AbstractDiagramList`

Definition at line 650 of file `KDChartAbstractDiagram.h`.

**8.1.1.2** `typedef QList<CartesianAxis\*> KDChart::CartesianAxisList`

Definition at line 135 of file `KDChartCartesianAxis.h`.

**8.1.1.3** `typedef QList<const AbstractDiagram\*> KDChart::ConstAbstractDiagramList`

Definition at line 651 of file `KDChartAbstractDiagram.h`.

**8.1.1.4** `typedef QList<const AbstractDiagram\*> KDChart::ConstDiagramList`

Definition at line 43 of file `KDChartLegend.h`.

**8.1.1.5** `typedef QList<AbstractCoordinatePlane\*> KDChart::CoordinatePlaneList`

Definition at line 48 of file `KDChartChart.h`.

**8.1.1.6** `typedef QList<DataDimension> KDChart::DataDimensionsList`

Definition at line 40 of file `KDChartAbstractCoordinatePlane.h`.

**8.1.1.7** `typedef QVector<int> KDChart::DatasetDescriptionVector`

Definition at line 36 of file `KDChartDatasetProxyModel.h`.

**8.1.1.8 typedef QList<[AbstractDiagram\\*](#)> [KDChart::DiagramList](#)**

Definition at line 41 of file KDChartLegend.h.

**8.1.1.9 typedef QList<[HeaderFooter\\*](#)> [KDChart::HeaderFooterList](#)**

Definition at line 51 of file KDChartChart.h.

**8.1.1.10 typedef QList<[Legend\\*](#)> [KDChart::LegendList](#)**

Definition at line 52 of file KDChartChart.h.

**8.1.1.11 typedef QList<[TernaryAxis\\*](#)> [KDChart::TernaryAxisList](#)**

Definition at line 100 of file KDChartTernaryAxis.h.

**8.1.2 Enumeration Type Documentation****8.1.2.1 enum [KDChart::DisplayRoles](#)**

Enumerator:

*DatasetPenRole*  
*DatasetBrushRole*  
*DataValueLabelAttributesRole*  
*ThreeDAttributesRole*  
*LineAttributesRole*  
*ThreeDLineAttributesRole*  
*BarAttributesRole*  
*ThreeDBarAttributesRole*  
*PieAttributesRole*  
*ThreeDPieAttributesRole*  
*DataHiddenRole*  
*ValueTrackerAttributesRole*

Definition at line 245 of file KDChartGlobal.h.

```

245         {
246     DatasetPenRole = 0x0A79EF95,
247     DatasetBrushRole,
248     DataValueLabelAttributesRole,
249     ThreeDAttributesRole,
250     LineAttributesRole,
251     ThreeDLineAttributesRole,
252     BarAttributesRole,
253     ThreeDBarAttributesRole,
254     PieAttributesRole,
255     ThreeDPieAttributesRole,
256     DataHiddenRole,
257     ValueTrackerAttributesRole
258 };

```

### 8.1.3 Function Documentation

#### 8.1.3.1 QDebug KDChart::operator<< (QDebug *stream*, const [DataDimension](#) & *r*)

Definition at line 417 of file KDChartAbstractCoordinatePlane.cpp.

References [KDChart::DataDimension::calcMode](#), [KDChart::DataDimension::end](#), [KDChartEnums::granularitySequenceToString\(\)](#), [KDChart::DataDimension::isCalculated](#), [KDChart::AbstractCoordinatePlane::Logarithmic](#), [KDChart::DataDimension::sequence](#), [KDChart::DataDimension::start](#), [KDChart::DataDimension::stepWidth](#), and [KDChart::DataDimension::subStepWidth](#).

```
418 {  
419     stream << "DataDimension("  
420         << " start=" << r.start  
421         << " end=" << r.end  
422         << " sequence=" << KDChartEnums::granularitySequenceToString( r.sequence )  
423         << " isCalculated=" << r.isCalculated  
424         << " calcMode=" << ( r.calcMode == AbstractCoordinatePlane::Logarithmic ? "Logarithmic" : "  
425         << " stepWidth=" << r.stepWidth  
426         << " subStepWidth=" << r.subStepWidth  
427         << " )";  
428     return stream;  
429 }
```



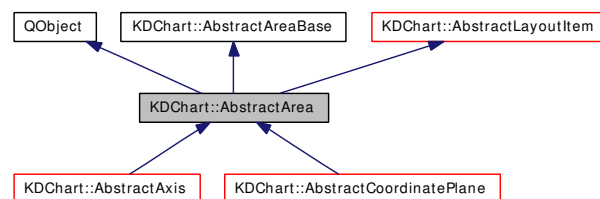
## Chapter 9

# KD Chart 2 Class Documentation

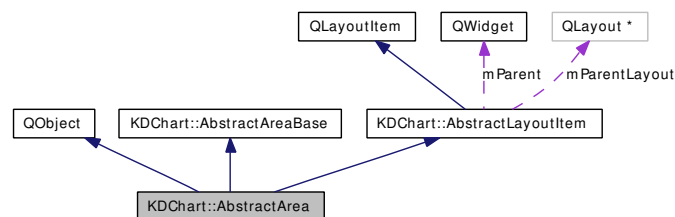
### 9.1 KDChart::AbstractArea Class Reference

```
#include <KDChartAbstractArea.h>
```

Inheritance diagram for KDChart::AbstractArea:



Collaboration diagram for KDChart::AbstractArea:



#### 9.1.1 Detailed Description

An area in the chart with a background, a frame, etc.

[AbstractArea](#) is the base class for all non-widget chart elements that have a set of background attributes and frame attributes, such as coordinate planes or axes.

**Note:**

This class inherits from [AbstractAreaBase](#), [AbstractLayoutItem](#), [QObject](#). The reason for this tripple inheritance is that neither [AbstractAreaBase](#) nor [AbstractLayoutItem](#) are [QObject](#).

Definition at line 54 of file KDChartAbstractArea.h.

## Signals

- void [positionChanged](#) ([AbstractArea](#) \*)

## Public Member Functions

- void [alignToReferencePoint](#) (const [RelativePosition](#) &position)
- [BackgroundAttributes](#) [backgroundAttributes](#) () const
- virtual int [bottomOverlap](#) (bool doNotRecalculate=false) const  
*This is called at layout time by [KDChart::AutoSpacerLayoutItem::sizeHint\(\)](#).*
- bool [compare](#) (const [AbstractAreaBase](#) \*other) const  
*Returns true if both areas have the same settings.*
- [FrameAttributes](#) [frameAttributes](#) () const
- void [getFrameLeadings](#) (int &left, int &top, int &right, int &bottom) const
- virtual int [leftOverlap](#) (bool doNotRecalculate=false) const  
*This is called at layout time by [KDChart::AutoSpacerLayoutItem::sizeHint\(\)](#).*
- virtual void [paint](#) (QPainter \*)=0
- virtual void [paintAll](#) (QPainter &painter)  
*Call [paintAll](#), if you want the background and the frame to be drawn before the normal [paint\(\)](#) is invoked automatically.*
- virtual void [paintBackground](#) (QPainter &painter, const QRect &rectangle)
- virtual void [paintCtx](#) ([PaintContext](#) \*context)  
*Default impl: Paint the complete item using its layouted position and size.*
- virtual void [paintFrame](#) (QPainter &painter, const QRect &rectangle)
- virtual void [paintIntoRect](#) (QPainter &painter, const QRect &rect)  
*Draws the background and frame, then calls [paint\(\)](#).*
- QLayout \* [parentLayout](#) ()
- void [removeFromParentLayout](#) ()
- virtual int [rightOverlap](#) (bool doNotRecalculate=false) const  
*This is called at layout time by [KDChart::AutoSpacerLayoutItem::sizeHint\(\)](#).*
- void [setBackgroundAttributes](#) (const [BackgroundAttributes](#) &a)
- void [setFrameAttributes](#) (const [FrameAttributes](#) &a)
- void [setParentLayout](#) (QLayout \*lay)
- virtual void [setParentWidget](#) (QWidget \*widget)  
*Inform the item about its widget: This enables the item, to trigger that widget's update, whenever the size of the item's contents has changed.*
- virtual void [sizeHintChanged](#) () const  
*Report changed size hint: ask the parent widget to recalculate the layout.*

- virtual int [topOverlap](#) (bool doNotRecalculate=false) const  
*This is called at layout time by [KDChart::AutoSpacerLayoutItem::sizeHint\(\)](#).*
- virtual [~AbstractArea](#) ()

## Static Public Member Functions

- static void [paintBackgroundAttributes](#) (QPainter &painter, const QRect &rectangle, const [KDChart::BackgroundAttributes](#) &attributes)
- static void [paintFrameAttributes](#) (QPainter &painter, const QRect &rectangle, const [KDChart::FrameAttributes](#) &attributes)

## Protected Member Functions

- [AbstractArea](#) ()
- virtual QRect [areaGeometry](#) () const
- QRect [innerRect](#) () const
- virtual void [positionHasChanged](#) ()

## Protected Attributes

- [QWidget](#) \* [mParent](#)
- [QLayout](#) \* [mParentLayout](#)

## 9.1.2 Constructor & Destructor Documentation

### 9.1.2.1 AbstractArea::~~AbstractArea () [virtual]

Definition at line 62 of file KDChartAbstractArea.cpp.

```
63 {
64     // this bloc left empty intentionally
65 }
```

### 9.1.2.2 AbstractArea::AbstractArea () [protected]

Definition at line 54 of file KDChartAbstractArea.cpp.

```
55     : QObject()
56     , KDChart::AbstractAreaBase()
57     , KDChart::AbstractLayoutItem()
58 {
59     init();
60 }
```

### 9.1.3 Member Function Documentation

#### 9.1.3.1 void AbstractAreaBase::alignToReferencePoint (const [RelativePosition](#) & *position*) [inherited]

Definition at line 90 of file KDChartAbstractAreaBase.cpp.

```

91 {
92     Q_UNUSED( position );
93     // PENDING(kalle) FIXME
94     qWarning( "Sorry, not implemented: void AbstractAreaBase::alignToReferencePoint( const RelativePosi
95 }
```

#### 9.1.3.2 [QRect](#) AbstractArea::areaGeometry () const [protected, virtual]

Implements [KDChart::AbstractAreaBase](#).

Definition at line 150 of file KDChartAbstractArea.cpp.

Referenced by KDChart::CartesianCoordinatePlane::drawingArea(), KDChart::TernaryCoordinatePlane::layoutDiagrams(), KDChart::PolarCoordinatePlane::layoutDiagrams(), KDChart::TernaryCoordinatePlane::paint(), KDChart::CartesianAxis::paint(), paintAll(), and KDChart::CartesianAxis::paintCtx().

```

151 {
152     return geometry();
153 }
```

#### 9.1.3.3 [BackgroundAttributes](#) AbstractAreaBase::backgroundAttributes () const [inherited]

Definition at line 120 of file KDChartAbstractAreaBase.cpp.

References d.

Referenced by KDChart::AbstractAreaBase::compare(), and updateCommonBrush().

```

121 {
122     return d->backgroundAttributes;
123 }
```

#### 9.1.3.4 int AbstractArea::bottomOverlap (bool *doNotRecalculate* = false) const [virtual]

This is called at layout time by KDChart::AutoSpacerLayoutItem::sizeHint().

The method triggers AbstractArea::sizeHint() to find out the amount of overlap at the bottom edge of the area.

#### Note:

The default implementation is not using any caching, it might make sense to implement a more sophisticated solution for derived classes that have complex work to do in sizeHint(). All we have here is a primitive flag to be set by the caller if it is sure that no sizeHint() needs to be called.

Definition at line 101 of file KDChartAbstractArea.cpp.

References d.

Referenced by KDChart::CartesianAxis::maximumSize().

```

102 {
103     // Re-calculate the sizes,
104     // so we also get the amountOf..Overlap members set newly:
105     if( ! doNotRecalculate )
106         sizeHint();
107     return d->amountOfBottomOverlap;
108 }
```

### 9.1.3.5 bool AbstractAreaBase::compare (const [AbstractAreaBase](#) \* *other*) const [inherited]

Returns true if both areas have the same settings.

Definition at line 75 of file KDChartAbstractAreaBase.cpp.

References KDChart::AbstractAreaBase::backgroundAttributes(), and KDChart::AbstractAreaBase::frameAttributes().

```

76 {
77     if( other == this ) return true;
78     if( ! other ){
79         //qDebug() << "CartesianAxis::compare() cannot compare to Null pointer";
80         return false;
81     }
82     /*
83     qDebug() << "AbstractAreaBase:" << (frameAttributes() == other->frameAttributes())
84         << (backgroundAttributes() == other->backgroundAttributes()) << "\n";
85     */
86     return (frameAttributes() == other->frameAttributes()) &&
87         (backgroundAttributes() == other->backgroundAttributes());
88 }
```

### 9.1.3.6 [FrameAttributes](#) AbstractAreaBase::frameAttributes () const [inherited]

Definition at line 106 of file KDChartAbstractAreaBase.cpp.

References d.

Referenced by KDChart::Legend::clone(), KDChart::AbstractAreaBase::compare(), and updateCommonBrush().

```

107 {
108     return d->frameAttributes;
109 }
```

### 9.1.3.7 void AbstractAreaBase::getFrameLeadings (int & *left*, int & *top*, int & *right*, int & *bottom*) const [inherited]

Definition at line 212 of file KDChartAbstractAreaBase.cpp.

References d.

Referenced by KDChart::AbstractAreaBase::innerRect(), and KDChart::AbstractAreaWidget::paintAll().

```

213 {
214     if( d && d->frameAttributes.isVisible() ){
215         const int padding = qMax( d->frameAttributes.padding(), 0 );
216         left    = padding;
217         top     = padding;
218         right   = padding;
219         bottom  = padding;
220     }else{
221         left    = 0;
222         top     = 0;
223         right   = 0;
224         bottom  = 0;
225     }
226 }

```

### 9.1.3.8 QRect AbstractAreaBase::innerRect () const [protected, inherited]

Definition at line 228 of file KDChartAbstractAreaBase.cpp.

References [KDChart::AbstractAreaBase::areaGeometry\(\)](#), and [KDChart::AbstractAreaBase::getFrameLeadings\(\)](#).

Referenced by [KDChart::TextArea::paintAll\(\)](#), and [paintAll\(\)](#).

```

229 {
230     int left;
231     int top;
232     int right;
233     int bottom;
234     getFrameLeadings( left, top, right, bottom );
235     return
236         QRect( QPoint(0,0), areaGeometry().size() )
237             .adjusted( left, top, -right, -bottom );
238 }

```

### 9.1.3.9 int AbstractArea::leftOverlap (bool *doNotRecalculate* = false) const [virtual]

This is called at layout time by [KDChart::AutoSpacerLayoutItem::sizeHint\(\)](#).

The method triggers [AbstractArea::sizeHint\(\)](#) to find out the amount of overlap at the left edge of the area.

#### Note:

The default implementation is not using any caching, it might make sense to implement a more sophisticated solution for derived classes that have complex work to do in [sizeHint\(\)](#). All we have here is a primitive flag to be set by the caller if it is sure that no [sizeHint\(\)](#) needs to be called.

Definition at line 77 of file KDChartAbstractArea.cpp.

References [d](#).

Referenced by [KDChart::CartesianAxis::maximumSize\(\)](#).

```

78 {
79     // Re-calculate the sizes,
80     // so we also get the amountOf..Overlap members set newly:
81     if( ! doNotRecalculate )
82         sizeHint();
83     return d->amountOfLeftOverlap;
84 }

```

### 9.1.3.10 virtual void KDChart::AbstractLayoutItem::paint (QPainter \*) [pure virtual, inherited]

Implemented in [KDChart::CartesianAxis](#), [KDChart::CartesianCoordinatePlane](#), [KDChart::TextLayoutItem](#), [KDChart::MarkerLayoutItem](#), [KDChart::LineLayoutItem](#), [KDChart::LineWithMarkerLayoutItem](#), [KDChart::HorizontalLineLayoutItem](#), [KDChart::VerticalLineLayoutItem](#), [KDChart::AutoSpacerLayoutItem](#), [KDChart::PolarCoordinatePlane](#), [KDChart::TernaryAxis](#), and [KDChart::TernaryCoordinatePlane](#).

Referenced by [KDChart::Legend::paint\(\)](#), [KDChart::AbstractLayoutItem::paintAll\(\)](#), [paintAll\(\)](#), and [KDChart::AbstractLayoutItem::paintCtx\(\)](#).

### 9.1.3.11 void AbstractArea::paintAll (QPainter & painter) [virtual]

Call [paintAll](#), if you want the background and the frame to be drawn before the normal [paint\(\)](#) is invoked automatically.

Reimplemented from [KDChart::AbstractLayoutItem](#).

Reimplemented in [KDChart::TernaryAxis](#).

Definition at line 123 of file [KDChartAbstractArea.cpp](#).

References [areaGeometry\(\)](#), [d](#), [KDChart::AbstractAreaBase::innerRect\(\)](#), [KDChart::AbstractLayoutItem::paint\(\)](#), [KDChart::AbstractAreaBase::paintBackground\(\)](#), and [KDChart::AbstractAreaBase::paintFrame\(\)](#).

Referenced by [paintIntoRect\(\)](#).

```

124 {
125     // Paint the background and frame
126     const QRect overlappingArea( geometry().adjusted(
127         -d->amountOfLeftOverlap,
128         -d->amountOfTopOverlap,
129         d->amountOfRightOverlap,
130         d->amountOfBottomOverlap ) );
131     paintBackground( painter, overlappingArea );
132     paintFrame(      painter, overlappingArea );
133
134     // temporarily adjust the widget size, to be sure all content gets calculated
135     // to fit into the inner rectangle
136     const QRect oldGeometry( areaGeometry() );
137     QRect inner( innerRect() );
138     inner.moveTo(
139         oldGeometry.left() + inner.left(),
140         oldGeometry.top()  + inner.top() );
141     const bool needAdjustGeometry = oldGeometry != inner;
142     if( needAdjustGeometry )
143         setGeometry( inner );
144     paint( &painter );
145     if( needAdjustGeometry )
146         setGeometry( oldGeometry );
147     //qDebug() << "AbstractAreaWidget::paintAll() done.";
148 }
```

### 9.1.3.12 void AbstractAreaBase::paintBackground (QPainter & painter, const QRect & rectangle) [virtual, inherited]

Definition at line 196 of file [KDChartAbstractAreaBase.cpp](#).

References [d](#), and [KDChart::AbstractAreaBase::paintBackgroundAttributes\(\)](#).

Referenced by `KDChart::TextArea::paintAll()`, `KDChart::AbstractAreaWidget::paintAll()`, and `paintAll()`.

```

197 {
198     Q_ASSERT_X ( d != 0, "AbstractAreaBase::paintBackground()",
199                 "Private class was not initialized!" );
200     paintBackgroundAttributes( painter, rect, d->backgroundAttributes );
201 }
```

### 9.1.3.13 void AbstractAreaBase::paintBackgroundAttributes (QPainter & *painter*, const QRect & *rectangle*, const [KDChart::BackgroundAttributes](#) & *attributes*) [static, inherited]

Definition at line 127 of file `KDChartAbstractAreaBase.cpp`.

References `KDChart::BackgroundAttributes::BackgroundPixmapModeCentered`, `KDChart::BackgroundAttributes::BackgroundPixmapModeNone`, `KDChart::BackgroundAttributes::BackgroundPixmapModeScaled`, `KDChart::BackgroundAttributes::BackgroundPixmapModeStretched`, `KDChart::BackgroundAttributes::brush()`, `KDChart::BackgroundAttributes::isVisible()`, `KDChart::BackgroundAttributes::pixmap()`, and `KDChart::BackgroundAttributes::pixmapMode()`.

Referenced by `KDChart::AbstractAreaBase::paintBackground()`.

```

129 {
130     if( !attributes.isVisible() ) return;
131
132     /* first draw the brush (may contain a pixmap)*/
133     if( Qt::NoBrush != attributes.brush().style() ) {
134         KDChart::PainterSaver painterSaver( &painter );
135         painter.setPen( Qt::NoPen );
136         const QPointF newTopLeft( painter.deviceMatrix().map( rect.topLeft() ) );
137         painter.setBrushOrigin( newTopLeft );
138         painter.setBrush( attributes.brush() );
139         painter.drawRect( rect.adjusted( 0, 0, -1, -1 ) );
140     }
141     /* next draw the backPixmap over the brush */
142     if( !attributes.pixmap().isNull() &&
143         attributes.pixmapMode() != BackgroundAttributes::BackgroundPixmapModeNone ) {
144         QPointF ol = rect.topLeft();
145         if( BackgroundAttributes::BackgroundPixmapModeCentered == attributes.pixmapMode() )
146         {
147             ol.setX( rect.center().x() - attributes.pixmap().width() / 2 );
148             ol.setY( rect.center().y() - attributes.pixmap().height() / 2 );
149             painter.drawPixmap( ol, attributes.pixmap() );
150         } else {
151             QMatrix m;
152             double zW = (double)rect.width() / (double)attributes.pixmap().width();
153             double zH = (double)rect.height() / (double)attributes.pixmap().height();
154             switch( attributes.pixmapMode() ) {
155                 case BackgroundAttributes::BackgroundPixmapModeScaled:
156                 {
157                     double z;
158                     z = qMin( zW, zH );
159                     m.scale( z, z );
160                 }
161                 break;
162                 case BackgroundAttributes::BackgroundPixmapModeStretched:
163                 {
164                     m.scale( zW, zH );
165                     break;
166                 }
167                 default:
168                 {
169                     ; // Cannot happen, previously checked
170                 }
171             }
172             QPixmap pm = attributes.pixmap().transformed( m );

```



```

169         ol.setX( rect.center().x() - pm.width() / 2 );
170         ol.setY( rect.center().y() - pm.height() / 2 );
171         painter.drawPixmap( ol, pm );
172     }
173 }
174 }
```

#### 9.1.3.14 void KDChart::AbstractLayoutItem::paintCtx ([PaintContext](#) \* *context*) [virtual, inherited]

Default impl: Paint the complete item using its layouted position and size.

Reimplemented in [KDChart::CartesianAxis](#), and [KDChart::TernaryAxis](#).

Definition at line 77 of file KDChartLayoutItems.cpp.

References [KDChart::AbstractLayoutItem::paint\(\)](#), and [KDChart::PaintContext::painter\(\)](#).

```

78 {
79     if( context )
80         paint( context->painter() );
81 }
```

#### 9.1.3.15 void AbstractAreaBase::paintFrame (QPainter & *painter*, const QRect & *rectangle*) [virtual, inherited]

Definition at line 204 of file KDChartAbstractAreaBase.cpp.

References [d](#), and [KDChart::AbstractAreaBase::paintFrameAttributes\(\)](#).

Referenced by [KDChart::TextArea::paintAll\(\)](#), [KDChart::AbstractAreaWidget::paintAll\(\)](#), and [paintAll\(\)](#).

```

205 {
206     Q_ASSERT_X ( d != 0, "AbstractAreaBase::paintFrame()",
207                 "Private class was not initialized!" );
208     paintFrameAttributes( painter, rect, d->frameAttributes );
209 }
```

#### 9.1.3.16 void AbstractAreaBase::paintFrameAttributes (QPainter & *painter*, const QRect & *rectangle*, const [KDChart::FrameAttributes](#) & *attributes*) [static, inherited]

Definition at line 177 of file KDChartAbstractAreaBase.cpp.

References [KDChart::FrameAttributes::isVisible\(\)](#), and [KDChart::FrameAttributes::pen\(\)](#).

Referenced by [KDChart::AbstractAreaBase::paintFrame\(\)](#).

```

179 {
180
181     if( !attributes.isVisible() ) return;
182
183     // Note: We set the brush to NoBrush explicitly here.
184     //       Otherwise we might get a filled rectangle, so any
185     //       previously drawn background would be overwritten by that area.
186
187     const QPen oldPen( painter.pen() );
188     const QBrush oldBrush( painter.brush() );
```

```

189     painter.setPen(    attributes.pen() );
190     painter.setBrush( Qt::NoBrush );
191     painter.drawRect( rect.adjusted( 0, 0, -1, -1 ) );
192     painter.setBrush( oldBrush );
193     painter.setPen(    oldPen );
194 }

```

#### 9.1.3.17 void AbstractArea::paintIntoRect (QPainter & *painter*, const QRect & *rect*) [virtual]

Draws the background and frame, then calls [paint\(\)](#).

In most cases there is no need to overwrite this method in a derived class, but you would overwrite [AbstractLayoutItem::paint\(\)](#) instead.

Definition at line 111 of file KDChartAbstractArea.cpp.

References [paintAll\(\)](#).

```

112 {
113     const QRect oldGeometry( geometry() );
114     if( oldGeometry != rect )
115         setGeometry( rect );
116     painter.translate( rect.left(), rect.top() );
117     paintAll( painter );
118     painter.translate( -rect.left(), -rect.top() );
119     if( oldGeometry != rect )
120         setGeometry( oldGeometry );
121 }

```

#### 9.1.3.18 QLayout\* KDChart::AbstractLayoutItem::parentLayout () [inherited]

Definition at line 76 of file KDChartLayoutItems.h.

```

77     {
78         return mParentLayout;
79     }

```

#### 9.1.3.19 void KDChart::AbstractArea::positionChanged ([AbstractArea](#) \*) [signal]

Referenced by [positionHasChanged\(\)](#).

#### 9.1.3.20 void AbstractArea::positionHasChanged () [protected, virtual]

Reimplemented from [KDChart::AbstractAreaBase](#).

Definition at line 155 of file KDChartAbstractArea.cpp.

References [positionChanged\(\)](#).

```

156 {
157     emit positionChanged( this );
158 }

```

**9.1.3.21 void KDChart::AbstractLayoutItem::removeFromParentLayout ()** [inherited]

Definition at line 80 of file KDChartLayoutItems.h.

Referenced by KDChart::Chart::takeCoordinatePlane().

```

81         {
82             if( mParentLayout ){
83                 if( widget() )
84                     mParentLayout->removeWidget( widget() );
85                 else
86                     mParentLayout->removeItem( this );
87             }
88         }

```

**9.1.3.22 int AbstractArea::rightOverlap (bool *doNotRecalculate* = false) const** [virtual]

This is called at layout time by [KDChart::AutoSpacerLayoutItem::sizeHint\(\)](#).

The method triggers AbstractArea::sizeHint() to find out the amount of overlap at the right edge of the area.

**Note:**

The default implementation is not using any caching, it might make sense to implement a more sophisticated solution for derived classes that have complex work to do in sizeHint(). All we have here is a primitive flag to be set by the caller if it is sure that no sizeHint() needs to be called.

Definition at line 85 of file KDChartAbstractArea.cpp.

References d.

Referenced by KDChart::CartesianAxis::maximumSize().

```

86 {
87     // Re-calculate the sizes,
88     // so we also get the amountOf..Overlap members set newly:
89     if( ! doNotRecalculate )
90         sizeHint();
91     return d->amountOfRightOverlap;
92 }

```

**9.1.3.23 void AbstractAreaBase::setBackgroundAttributes (const [BackgroundAttributes](#) & a)**  
[inherited]

Definition at line 111 of file KDChartAbstractAreaBase.cpp.

References d, and KDChart::AbstractAreaBase::positionHasChanged().

```

112 {
113     if( d->backgroundAttributes == a )
114         return;
115
116     d->backgroundAttributes = a;
117     positionHasChanged();
118 }

```

### 9.1.3.24 void AbstractAreaBase::setFrameAttributes (const [FrameAttributes](#) & a) [inherited]

Definition at line 97 of file KDChartAbstractAreaBase.cpp.

References d, and KDChart::AbstractAreaBase::positionHasChanged().

Referenced by KDChart::Legend::clone().

```

98 {
99     if( d->frameAttributes == a )
100         return;
101
102     d->frameAttributes = a;
103     positionHasChanged();
104 }
```

### 9.1.3.25 void KDChart::AbstractLayoutItem::setParentLayout (QLayout \* lay) [inherited]

Definition at line 72 of file KDChartLayoutItems.h.

```

73     {
74         mParentLayout = lay;
75     }
```

### 9.1.3.26 void KDChart::AbstractLayoutItem::setParentWidget ([QWidget](#) \* widget) [virtual, inherited]

Inform the item about its widget: This enables the item, to trigger that widget's update, whenever the size of the item's contents has changed.

Thus, you need to call setParentWidget on every item, that has a non-fixed size.

Definition at line 64 of file KDChartLayoutItems.cpp.

References KDChart::AbstractLayoutItem::mParent.

Referenced by KDChart::HeaderFooter::setParent(), and KDChart::AbstractCartesianDiagram::takeAxis().

```

65 {
66     mParent = widget;
67 }
```

### 9.1.3.27 void KDChart::AbstractLayoutItem::sizeHintChanged () const [virtual, inherited]

Report changed size hint: ask the parent widget to recalculate the layout.

Definition at line 86 of file KDChartLayoutItems.cpp.

References KDChart::AbstractLayoutItem::mParent.

Referenced by KDChart::TextLayoutItem::sizeHint().

```

87 {
88     // This is exactly like what QWidget::updateGeometry does.
89     // qDebug("KDChart::AbstractLayoutItem::sizeHintChanged() called");
90     if( mParent ) {
91         if ( mParent->layout() )
92             mParent->layout()->invalidate();
93         else
94             QApplication::postEvent( mParent, new QEvent( QEvent::LayoutRequest ) );
95     }
96 }

```

### 9.1.3.28 int AbstractArea::topOverlap (bool *doNotRecalculate* = false) const [virtual]

This is called at layout time by [KDChart::AutoSpacerLayoutItem::sizeHint\(\)](#).

The method triggers AbstractArea::sizeHint() to find out the amount of overlap at the top edge of the area.

#### Note:

The default implementation is not using any caching, it might make sense to implement a more sophisticated solution for derived classes that have complex work to do in sizeHint(). All we have here is a primitive flag to be set by the caller if it is sure that no sizeHint() needs to be called.

Definition at line 93 of file KDChartAbstractArea.cpp.

References d.

Referenced by KDChart::CartesianAxis::maximumSize().

```

94 {
95     // Re-calculate the sizes,
96     // so we also get the amountOf..Overlap members set newly:
97     if( ! doNotRecalculate )
98         sizeHint();
99     return d->amountOfTopOverlap;
100 }

```

## 9.1.4 Member Data Documentation

### 9.1.4.1 QWidget\* KDChart::AbstractLayoutItem::mParent [protected, inherited]

Definition at line 90 of file KDChartLayoutItems.h.

Referenced by KDChart::AbstractLayoutItem::setParentWidget(), KDChart::TextLayoutItem::setText(), KDChart::TextLayoutItem::setTextAttributes(), and KDChart::AbstractLayoutItem::sizeHintChanged().

### 9.1.4.2 QLayout\* KDChart::AbstractLayoutItem::mParentLayout [protected, inherited]

Definition at line 91 of file KDChartLayoutItems.h.

Referenced by KDChart::AutoSpacerLayoutItem::paint().

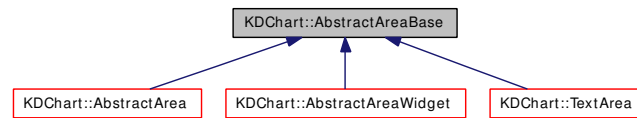
The documentation for this class was generated from the following files:

- [KDChartAbstractArea.h](#)
- [KDChartAbstractArea.cpp](#)

## 9.2 KDChart::AbstractAreaBase Class Reference

```
#include <KDChartAbstractAreaBase.h>
```

Inheritance diagram for KDChart::AbstractAreaBase:



### 9.2.1 Detailed Description

Base class for [AbstractArea](#) and [AbstractAreaWidget](#): An area in the chart with a background, a frame, etc.

[AbstractAreaBase](#) is the base class for all chart elements that have a set of background attributes and frame attributes, such as legends or axes.

#### Note:

Normally you should not use [AbstractAreaBase](#) directly, but derive your classes from [AbstractArea](#) or [AbstractAreaWidget](#).

This class is not a [QObject](#), so it is easier to inherit from it, if you are inheriting from a [QObject](#) too like [AbstractAreaWidget](#) does it.

#### See also:

[AbstractArea](#), [AbstractAreaWidget](#)

Definition at line 69 of file `KDChartAbstractAreaBase.h`.

### Public Member Functions

- void [alignToReferencePoint](#) (const [RelativePosition](#) &position)
- [BackgroundAttributes](#) [backgroundAttributes](#) () const
- bool [compare](#) (const [AbstractAreaBase](#) \*other) const  
*Returns true if both areas have the same settings.*
- [FrameAttributes](#) [frameAttributes](#) () const
- void [getFrameLeadings](#) (int &left, int &top, int &right, int &bottom) const
- virtual void [paintBackground](#) (QPainter &painter, const QRect &rectangle)
- virtual void [paintFrame](#) (QPainter &painter, const QRect &rectangle)
- void [setBackgroundAttributes](#) (const [BackgroundAttributes](#) &a)
- void [setFrameAttributes](#) (const [FrameAttributes](#) &a)

### Static Public Member Functions

- static void [paintBackgroundAttributes](#) (QPainter &painter, const QRect &rectangle, const [KDChart::BackgroundAttributes](#) &attributes)
- static void [paintFrameAttributes](#) (QPainter &painter, const QRect &rectangle, const [KDChart::FrameAttributes](#) &attributes)

## Protected Member Functions

- [AbstractAreaBase \(\)](#)
- virtual QRect [areaGeometry \(\)](#) const=0
- QRect [innerRect \(\)](#) const
- virtual void [positionHasChanged \(\)](#)
- virtual [~AbstractAreaBase \(\)](#)

## 9.2.2 Constructor & Destructor Documentation

### 9.2.2.1 AbstractAreaBase::AbstractAreaBase () [protected]

Definition at line 57 of file KDChartAbstractAreaBase.cpp.

```
57                                     :
58     _d( new Private() )
59 {
60 }
```

### 9.2.2.2 AbstractAreaBase::~AbstractAreaBase () [protected, virtual]

Definition at line 62 of file KDChartAbstractAreaBase.cpp.

```
63 {
64     delete _d; _d = 0;
65 }
```

## 9.2.3 Member Function Documentation

### 9.2.3.1 void AbstractAreaBase::alignToReferencePoint (const [RelativePosition](#) & *position*)

Definition at line 90 of file KDChartAbstractAreaBase.cpp.

```
91 {
92     Q_UNUSED( position );
93     // PENDING(kalle) FIXME
94     qWarning( "Sorry, not implemented: void AbstractAreaBase::alignToReferencePoint( const RelativePosi
95 }
```

### 9.2.3.2 virtual QRect KDChart::AbstractAreaBase::areaGeometry () const [protected, pure virtual]

Implemented in [KDChart::AbstractArea](#), [KDChart::AbstractAreaWidget](#), and [KDChart::TextArea](#).

Referenced by [innerRect\(\)](#).

### 9.2.3.3 [BackgroundAttributes](#) AbstractAreaBase::backgroundAttributes () const

Definition at line 120 of file KDChartAbstractAreaBase.cpp.

References d.

Referenced by compare(), and updateCommonBrush().

```
121 {
122     return d->backgroundAttributes;
123 }
```

### 9.2.3.4 bool AbstractAreaBase::compare (const [AbstractAreaBase](#) \* other) const

Returns true if both areas have the same settings.

Definition at line 75 of file KDChartAbstractAreaBase.cpp.

References backgroundAttributes(), and frameAttributes().

```
76 {
77     if( other == this ) return true;
78     if( ! other ){
79         //QDebug() << "CartesianAxis::compare() cannot compare to Null pointer";
80         return false;
81     }
82     /*
83     qDebug() << "AbstractAreaBase:" << (frameAttributes() == other->frameAttributes())
84         << (backgroundAttributes() == other->backgroundAttributes()) << "\n";
85     */
86     return (frameAttributes() == other->frameAttributes()) &&
87         (backgroundAttributes() == other->backgroundAttributes());
88 }
```

### 9.2.3.5 [FrameAttributes](#) AbstractAreaBase::frameAttributes () const

Definition at line 106 of file KDChartAbstractAreaBase.cpp.

References d.

Referenced by KDChart::Legend::clone(), compare(), and updateCommonBrush().

```
107 {
108     return d->frameAttributes;
109 }
```

### 9.2.3.6 void AbstractAreaBase::getFrameLeadings (int & left, int & top, int & right, int & bottom) const

Definition at line 212 of file KDChartAbstractAreaBase.cpp.

References d.

Referenced by innerRect(), and KDChart::AbstractAreaWidget::paintAll().

```
213 {
214     if( d && d->frameAttributes.isVisible() ){
```



```

215         const int padding = qMax( d->frameAttributes.padding(), 0 );
216         left  = padding;
217         top   = padding;
218         right = padding;
219         bottom = padding;
220     }else{
221         left  = 0;
222         top   = 0;
223         right = 0;
224         bottom = 0;
225     }
226 }

```

### 9.2.3.7 QRect AbstractAreaBase::innerRect () const [protected]

Definition at line 228 of file KDChartAbstractAreaBase.cpp.

References `areaGeometry()`, and `getFrameLeadings()`.

Referenced by `KDChart::TextArea::paintAll()`, and `KDChart::AbstractArea::paintAll()`.

```

229 {
230     int left;
231     int top;
232     int right;
233     int bottom;
234     getFrameLeadings( left, top, right, bottom );
235     return
236         QRect( QPoint(0,0), areaGeometry().size() )
237             .adjusted( left, top, -right, -bottom );
238 }

```

### 9.2.3.8 void AbstractAreaBase::paintBackground (QPainter & painter, const QRect & rectangle) [virtual]

Definition at line 196 of file KDChartAbstractAreaBase.cpp.

References `d`, and `paintBackgroundAttributes()`.

Referenced by `KDChart::TextArea::paintAll()`, `KDChart::AbstractAreaWidget::paintAll()`, and `KDChart::AbstractArea::paintAll()`.

```

197 {
198     Q_ASSERT_X ( d != 0, "AbstractAreaBase::paintBackground()",
199                 "Private class was not initialized!" );
200     paintBackgroundAttributes( painter, rect, d->backgroundAttributes );
201 }

```

### 9.2.3.9 void AbstractAreaBase::paintBackgroundAttributes (QPainter & painter, const QRect & rectangle, const KDChart::BackgroundAttributes & attributes) [static]

Definition at line 127 of file KDChartAbstractAreaBase.cpp.

References `KDChart::BackgroundAttributes::BackgroundPixmapModeCentered`, `KDChart::BackgroundAttributes::BackgroundPixmapModeNone`, `KDChart::BackgroundAttributes::BackgroundPixmapModeScaled`, `KDChart::BackgroundAttributes::BackgroundPixmapModeStretched`, `KDChart::BackgroundAttributes::brush()`, `KDChart::BackgroundAttributes::isVisible()`, `KDChart::BackgroundAttributes::pixmap()`, and `KDChart::BackgroundAttributes::pixmapMode()`.

Referenced by `paintBackground()`.

```

129 {
130     if( !attributes.isVisible() ) return;
131
132     /* first draw the brush (may contain a pixmap)*/
133     if( Qt::NoBrush != attributes.brush().style() ) {
134         KDChart::PainterSaver painterSaver( &painter );
135         painter.setPen( Qt::NoPen );
136         const QPointF newTopLeft( painter.deviceMatrix().map( rect.topLeft() ) );
137         painter.setBrushOrigin( newTopLeft );
138         painter.setBrush( attributes.brush() );
139         painter.drawRect( rect.adjusted( 0, 0, -1, -1 ) );
140     }
141     /* next draw the backPixmap over the brush */
142     if( !attributes.pixmap().isNull() &&
143         attributes.pixmapMode() != BackgroundAttributes::BackgroundPixmapModeNone ) {
144         QPointF ol = rect.topLeft();
145         if( BackgroundAttributes::BackgroundPixmapModeCentered == attributes.pixmapMode() )
146         {
147             ol.setX( rect.center().x() - attributes.pixmap().width() / 2 );
148             ol.setY( rect.center().y() - attributes.pixmap().height() / 2 );
149             painter.drawPixmap( ol, attributes.pixmap() );
150         } else {
151             QMatrix m;
152             double zW = (double)rect.width() / (double)attributes.pixmap().width();
153             double zH = (double)rect.height() / (double)attributes.pixmap().height();
154             switch( attributes.pixmapMode() ) {
155                 case BackgroundAttributes::BackgroundPixmapModeScaled:
156                 {
157                     double z;
158                     z = qMin( zW, zH );
159                     m.scale( z, z );
160                 }
161                 break;
162                 case BackgroundAttributes::BackgroundPixmapModeStretched:
163                 {
164                     m.scale( zW, zH );
165                     break;
166                 }
167                 default:
168                 {
169                     ; // Cannot happen, previously checked
170                 }
171             }
172             QPixmap pm = attributes.pixmap().transformed( m );
173             ol.setX( rect.center().x() - pm.width() / 2 );
174             ol.setY( rect.center().y() - pm.height() / 2 );
175             painter.drawPixmap( ol, pm );
176         }
177     }
178 }

```

### 9.2.3.10 void AbstractAreaBase::paintFrame (QPainter & painter, const QRect & rectangle) [virtual]

Definition at line 204 of file `KDChartAbstractAreaBase.cpp`.

References `d`, and `paintFrameAttributes()`.

Referenced by `KDChart::TextArea::paintAll()`, `KDChart::AbstractAreaWidget::paintAll()`, and `KDChart::AbstractArea::paintAll()`.

```

205 {
206     Q_ASSERT_X ( d != 0, "AbstractAreaBase::paintFrame()",
207                 "Private class was not initialized!" );
208     paintFrameAttributes( painter, rect, d->frameAttributes );
209 }

```

### 9.2.3.11 void AbstractAreaBase::paintFrameAttributes (QPainter & *painter*, const QRect & *rectangle*, const [KDChart::FrameAttributes](#) & *attributes*) [static]

Definition at line 177 of file KDChartAbstractAreaBase.cpp.

References [KDChart::FrameAttributes::isVisible\(\)](#), and [KDChart::FrameAttributes::pen\(\)](#).

Referenced by [paintFrame\(\)](#).

```

179 {
180
181     if( !attributes.isVisible() ) return;
182
183     // Note: We set the brush to NoBrush explicitly here.
184     //       Otherwise we might get a filled rectangle, so any
185     //       previously drawn background would be overwritten by that area.
186
187     const QPen    oldPen(    painter.pen() );
188     const QBrush  oldBrush( painter.brush() );
189     painter.setPen(    attributes.pen() );
190     painter.setBrush( Qt::NoBrush );
191     painter.drawRect( rect.adjusted( 0, 0, -1, -1 ) );
192     painter.setBrush( oldBrush );
193     painter.setPen(    oldPen );
194 }
```

### 9.2.3.12 void AbstractAreaBase::positionHasChanged () [protected, virtual]

Reimplemented in [KDChart::AbstractArea](#), [KDChart::AbstractAreaWidget](#), and [KDChart::TextArea](#).

Definition at line 240 of file KDChartAbstractAreaBase.cpp.

Referenced by [setBackgroundAttributes\(\)](#), and [setFrameAttributes\(\)](#).

```

241 {
242     // this bloc left empty intentionally
243 }
```

### 9.2.3.13 void AbstractAreaBase::setBackgroundAttributes (const [BackgroundAttributes](#) & *a*)

Definition at line 111 of file KDChartAbstractAreaBase.cpp.

References [d](#), and [positionHasChanged\(\)](#).

```

112 {
113     if( d->backgroundAttributes == a )
114         return;
115
116     d->backgroundAttributes = a;
117     positionHasChanged();
118 }
```

### 9.2.3.14 void AbstractAreaBase::setFrameAttributes (const [FrameAttributes](#) & *a*)

Definition at line 97 of file KDChartAbstractAreaBase.cpp.

References [d](#), and [positionHasChanged\(\)](#).

Referenced by [KDChart::Legend::clone\(\)](#).

```
98 {  
99     if( d->frameAttributes == a )  
100         return;  
101  
102     d->frameAttributes = a;  
103     positionHasChanged();  
104 }
```

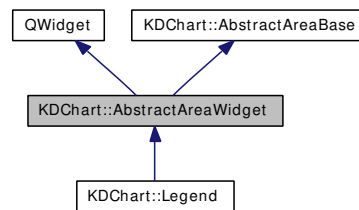
The documentation for this class was generated from the following files:

- [KDChartAbstractAreaBase.h](#)
- [KDChartAbstractAreaBase.cpp](#)

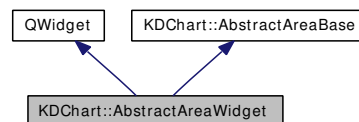
## 9.3 KDChart::AbstractAreaWidget Class Reference

```
#include <KDChartAbstractAreaWidget.h>
```

Inheritance diagram for KDChart::AbstractAreaWidget:



Collaboration diagram for KDChart::AbstractAreaWidget:



### 9.3.1 Detailed Description

An area in the chart with a background, a frame, etc.

[AbstractAreaWidget](#) is the base for all widget classes that have a set of background attributes and frame attributes, such as [KDChart::Chart](#) and [KDChart::Legend](#).

Definition at line 51 of file [KDChartAbstractAreaWidget.h](#).

### Signals

- void [positionChanged](#) ([AbstractAreaWidget](#) \*)

### Public Member Functions

- [AbstractAreaWidget](#) ([QWidget](#) \*parent=0)
- void [alignToReferencePoint](#) (const [RelativePosition](#) &position)
- [BackgroundAttributes](#) [backgroundAttributes](#) () const
- bool [compare](#) (const [AbstractAreaBase](#) \*other) const  
*Returns true if both areas have the same settings.*
- virtual void [forceRebuild](#) ()  
*Call this to trigger an unconditional re-building of the widget's internals.*
- [FrameAttributes](#) [frameAttributes](#) () const
- void [getFrameLeadings](#) (int &left, int &top, int &right, int &bottom) const
- virtual void [needSizeHint](#) ()  
*Call this to trigger an conditional re-building of the widget's internals.*

- virtual void [paint](#) (QPainter \*painter)=0  
*Overwrite this to paint the inner contents of your widget.*
- void [paintAll](#) (QPainter &painter)  
*Call paintAll, if you want the background and the frame to be drawn before the normal [paint\(\)](#) is invoked automatically.*
- virtual void [paintBackground](#) (QPainter &painter, const QRect &rectangle)
- virtual void [paintEvent](#) (QPaintEvent \*event)  
*Draws the background and frame, then calls [paint\(\)](#).*
- virtual void [paintFrame](#) (QPainter &painter, const QRect &rectangle)
- virtual void [paintIntoRect](#) (QPainter &painter, const QRect &rect)  
*Draws the background and frame, then calls [paint\(\)](#).*
- virtual void [resizeLayout](#) (const QSize &)
- void [setBackgroundAttributes](#) (const [BackgroundAttributes](#) &a)
- void [setFrameAttributes](#) (const [FrameAttributes](#) &a)

## Static Public Member Functions

- static void [paintBackgroundAttributes](#) (QPainter &painter, const QRect &rectangle, const [KDChart::BackgroundAttributes](#) &attributes)
- static void [paintFrameAttributes](#) (QPainter &painter, const QRect &rectangle, const [KDChart::FrameAttributes](#) &attributes)

## Protected Member Functions

- virtual QRect [areaGeometry](#) () const
- QRect [innerRect](#) () const
- virtual void [positionHasChanged](#) ()
- virtual [~AbstractAreaWidget](#) ()

## 9.3.2 Constructor & Destructor Documentation

### 9.3.2.1 AbstractAreaWidget::AbstractAreaWidget ([QWidget](#) \*parent = 0) [explicit]

Definition at line 69 of file KDChartAbstractAreaWidget.cpp.

```

70     : QWidget( parent )
71     , AbstractAreaBase( new Private() )
72 {
73     init();
74 }
```

### 9.3.2.2 AbstractAreaWidget::~~AbstractAreaWidget () [protected, virtual]

Definition at line 76 of file KDChartAbstractAreaWidget.cpp.

```
77 {  
78     // this block left empty intentionally  
79 }
```

## 9.3.3 Member Function Documentation

### 9.3.3.1 void AbstractAreaBase::alignToReferencePoint (const [RelativePosition](#) & position) [inherited]

Definition at line 90 of file KDChartAbstractAreaBase.cpp.

```
91 {  
92     Q_UNUSED( position );  
93     // PENDING(kalle) FIXME  
94     qWarning( "Sorry, not implemented: void AbstractAreaBase::alignToReferencePoint( const RelativePosi  
95 }
```

### 9.3.3.2 QRect AbstractAreaWidget::areaGeometry () const [protected, virtual]

Implements [KDChart::AbstractAreaBase](#).

Definition at line 186 of file KDChartAbstractAreaWidget.cpp.

```
187 {  
188     return geometry();  
189 }
```

### 9.3.3.3 [BackgroundAttributes](#) AbstractAreaBase::backgroundAttributes () const [inherited]

Definition at line 120 of file KDChartAbstractAreaBase.cpp.

References [d](#).

Referenced by [KDChart::AbstractAreaBase::compare\(\)](#), and [updateCommonBrush\(\)](#).

```
121 {  
122     return d->backgroundAttributes;  
123 }
```

### 9.3.3.4 bool AbstractAreaBase::compare (const [AbstractAreaBase](#) \* other) const [inherited]

Returns true if both areas have the same settings.

Definition at line 75 of file KDChartAbstractAreaBase.cpp.

References [KDChart::AbstractAreaBase::backgroundAttributes\(\)](#), and [KDChart::AbstractAreaBase::frameAttributes\(\)](#).

```

76 {
77     if( other == this ) return true;
78     if( ! other ){
79         qDebug() << "CartesianAxis::compare() cannot compare to Null pointer";
80         return false;
81     }
82     /*
83     qDebug() << "AbstractAreaBase:" << (frameAttributes() == other->frameAttributes())
84         << (backgroundAttributes() == other->backgroundAttributes()) << "\n";
85     */
86     return (frameAttributes() == other->frameAttributes()) &&
87         (backgroundAttributes() == other->backgroundAttributes());
88 }

```

### 9.3.3.5 void AbstractAreaWidget::forceRebuild () [virtual]

Call this to trigger an unconditional re-building of the widget's internals.

Reimplemented in [KDChart::Legend](#).

Definition at line 140 of file KDChartAbstractAreaWidget.cpp.

```

141 {
142     //bloc left empty intentionally
143 }

```

### 9.3.3.6 [FrameAttributes](#) AbstractAreaBase::frameAttributes () const [inherited]

Definition at line 106 of file KDChartAbstractAreaBase.cpp.

References d.

Referenced by KDChart::Legend::clone(), KDChart::AbstractAreaBase::compare(), and updateCommonBrush().

```

107 {
108     return d->frameAttributes;
109 }

```

### 9.3.3.7 void AbstractAreaBase::getFrameLeadings (int & left, int & top, int & right, int & bottom) const [inherited]

Definition at line 212 of file KDChartAbstractAreaBase.cpp.

References d.

Referenced by KDChart::AbstractAreaBase::innerRect(), and paintAll().

```

213 {
214     if( d && d->frameAttributes.isVisible() ){
215         const int padding = qMax( d->frameAttributes.padding(), 0 );
216         left   = padding;
217         top    = padding;
218         right  = padding;
219         bottom = padding;
220     }else{
221         left   = 0;

```



```

222         top    = 0;
223         right   = 0;
224         bottom  = 0;
225     }
226 }

```

### 9.3.3.8 QRect AbstractAreaBase::innerRect () const [protected, inherited]

Definition at line 228 of file KDChartAbstractAreaBase.cpp.

References KDChart::AbstractAreaBase::areaGeometry(), and KDChart::AbstractAreaBase::getFrameLeadings().

Referenced by KDChart::TextArea::paintAll(), and KDChart::AbstractArea::paintAll().

```

229 {
230     int left;
231     int top;
232     int right;
233     int bottom;
234     getFrameLeadings( left, top, right, bottom );
235     return
236         QRect( QPoint(0,0), areaGeometry().size() )
237             .adjusted( left, top, -right, -bottom );
238 }

```

### 9.3.3.9 void AbstractAreaWidget::needSizeHint () [virtual]

Call this to trigger an conditional re-building of the widget's internals.

e.g. [AbstractAreaWidget](#) call this, before calling layout()->setGeometry()

Reimplemented in [KDChart::Legend](#).

Definition at line 86 of file KDChartAbstractAreaWidget.cpp.

```

87 {
88     // this block left empty intentionally
89 }

```

### 9.3.3.10 virtual void KDChart::AbstractAreaWidget::paint (QPainter \* *painter*) [pure virtual]

Overwrite this to paint the inner contents of your widget.

#### Note:

When overriding this method, please let your widget draw itself at the top/left corner of the painter. You should call rect() (or width(), height(), resp.) to find the drawable area's size: While the [paint\(\)](#) method is being executed the frame of the widget is outside of its rect(), so you can use all of rect() for your custom drawing!

#### See also:

[paint](#), [paintIntoRect](#)

Implemented in [KDChart::Legend](#).

Referenced by paintAll().

### 9.3.3.11 void AbstractAreaWidget::paintAll (QPainter & painter)

Call `paintAll`, if you want the background and the frame to be drawn before the normal `paint()` is invoked automatically.

Definition at line 145 of file `KDChartAbstractAreaWidget.cpp`.

References `KDChart::AbstractAreaBase::getFrameLeadings()`, `paint()`, `KDChart::AbstractAreaBase::paintBackground()`, and `KDChart::AbstractAreaBase::paintFrame()`.

Referenced by `paintEvent()`, and `paintIntoRect()`.

```

146 {
147     //qDebug() << "AbstractAreaWidget::paintAll() called";
148
149     // Paint the background and frame
150     paintBackground( painter, QRect(QPoint(0, 0), size() ) );
151     paintFrame(      painter, QRect(QPoint(0, 0), size() ) );
152
153     /*
154     we do not call setContentsMargins() now,
155     but we call resizeLayout() whenever the size or the frame has changed
156
157     // adjust the widget's content margins,
158     // to be sure all content gets calculated
159     // to fit into the inner rectangle
160     const QRect oldGeometry( areaGeometry() );
161     const QRect inner( innerRect() );
162     //qDebug() << "areaGeometry():" << oldGeometry
163     //      << "contentsRect():" << contentsRect() << " inner:" << inner;
164     if( contentsRect() != inner ){
165         //qDebug() << "old contentsRect():" << contentsRect() << " new innerRect:" << inner;
166         setContentsMargins(
167             inner.left(),
168             inner.top(),
169             oldGeometry.width() -inner.width()-1,
170             oldGeometry.height()-inner.height()-1 );
171         //forceRebuild();
172     }
173     */
174     int left;
175     int top;
176     int right;
177     int bottom;
178     getFrameLeadings( left, top, right, bottom );
179     const QPoint translation( left, top );
180     painter.translate( translation );
181     paint( &painter );
182     painter.translate( -translation.x(), -translation.y() );
183     //qDebug() << "AbstractAreaWidget::paintAll() done.";
184 }
```

### 9.3.3.12 void AbstractAreaBase::paintBackground (QPainter & painter, const QRect & rectangle) [virtual, inherited]

Definition at line 196 of file `KDChartAbstractAreaBase.cpp`.

References `d`, and `KDChart::AbstractAreaBase::paintBackgroundAttributes()`.

Referenced by `KDChart::TextArea::paintAll()`, `paintAll()`, and `KDChart::AbstractArea::paintAll()`.

```

197 {
198     Q_ASSERT_X ( d != 0, "AbstractAreaBase::paintBackground()",
```

```

199         "Private class was not initialized!" );
200     paintBackgroundAttributes( painter, rect, d->backgroundAttributes );
201 }

```

### 9.3.3.13 void AbstractAreaBase::paintBackgroundAttributes (QPainter & painter, const QRect & rectangle, const KDChart::BackgroundAttributes & attributes) [static, inherited]

Definition at line 127 of file KDChartAbstractAreaBase.cpp.

References KDChart::BackgroundAttributes::BackgroundPixmapModeCentered, KDChart::BackgroundAttributes::BackgroundPixmapModeNone, KDChart::BackgroundAttributes::BackgroundPixmapModeScaled, KDChart::BackgroundAttributes::BackgroundPixmapModeStretched, KDChart::BackgroundAttributes::brush(), KDChart::BackgroundAttributes::isVisible(), KDChart::BackgroundAttributes::pixmap(), and KDChart::BackgroundAttributes::pixmapMode().

Referenced by KDChart::AbstractAreaBase::paintBackground().

```

129 {
130     if( !attributes.isVisible() ) return;
131
132     /* first draw the brush (may contain a pixmap)*/
133     if( Qt::NoBrush != attributes.brush().style() ) {
134         KDChart::PainterSaver painterSaver( &painter );
135         painter.setPen( Qt::NoPen );
136         const QPointF newTopLeft( painter.deviceMatrix().map( rect.topLeft() ) );
137         painter.setBrushOrigin( newTopLeft );
138         painter.setBrush( attributes.brush() );
139         painter.drawRect( rect.adjusted( 0, 0, -1, -1 ) );
140     }
141     /* next draw the backPixmap over the brush */
142     if( !attributes.pixmap().isNull() &&
143         attributes.pixmapMode() != BackgroundAttributes::BackgroundPixmapModeNone ) {
144         QPointF ol = rect.topLeft();
145         if( BackgroundAttributes::BackgroundPixmapModeCentered == attributes.pixmapMode() )
146         {
147             ol.setX( rect.center().x() - attributes.pixmap().width() / 2 );
148             ol.setY( rect.center().y() - attributes.pixmap().height() / 2 );
149             painter.drawPixmap( ol, attributes.pixmap() );
150         } else {
151             QMatrix m;
152             double zW = (double)rect.width() / (double)attributes.pixmap().width();
153             double zH = (double)rect.height() / (double)attributes.pixmap().height();
154             switch( attributes.pixmapMode() ) {
155                 case BackgroundAttributes::BackgroundPixmapModeScaled:
156                 {
157                     double z;
158                     z = qMin( zW, zH );
159                     m.scale( z, z );
160                 }
161                 break;
162                 case BackgroundAttributes::BackgroundPixmapModeStretched:
163                 {
164                     m.scale( zW, zH );
165                     break;
166                 }
167                 default:
168                 {
169                     ; // Cannot happen, previously checked
170                 }
171             }
172             QPixmap pm = attributes.pixmap().transformed( m );
173             ol.setX( rect.center().x() - pm.width() / 2 );
174             ol.setY( rect.center().y() - pm.height() / 2 );
175             painter.drawPixmap( ol, pm );
176         }
177     }
178 }

```

```
174 }
```

#### 9.3.3.14 void AbstractAreaWidget::paintEvent (QPaintEvent \* event) [virtual]

Draws the background and frame, then calls [paint\(\)](#).

In most cases there is no need to overwrite this method in a derived class, but you would overwrite [paint\(\)](#) instead.

**See also:**

[paint](#)

Definition at line 99 of file KDChartAbstractAreaWidget.cpp.

References [d](#), and [paintAll\(\)](#).

```
100 {
101     Q_UNUSED( event );
102     QPainter painter( this );
103     if( size() != d->currentLayoutSize ){
104         d->resizeLayout( this, size() );
105     }
106     paintAll( painter );
107 }
```

#### 9.3.3.15 void AbstractAreaBase::paintFrame (QPainter & painter, const QRect & rectangle) [virtual, inherited]

Definition at line 204 of file KDChartAbstractAreaBase.cpp.

References [d](#), and [KDChart::AbstractAreaBase::paintFrameAttributes\(\)](#).

Referenced by [KDChart::TextArea::paintAll\(\)](#), [paintAll\(\)](#), and [KDChart::AbstractArea::paintAll\(\)](#).

```
205 {
206     Q_ASSERT_X ( d != 0, "AbstractAreaBase::paintFrame()",
207                 "Private class was not initialized!" );
208     paintFrameAttributes( painter, rect, d->frameAttributes );
209 }
```

#### 9.3.3.16 void AbstractAreaBase::paintFrameAttributes (QPainter & painter, const QRect & rectangle, const [KDChart::FrameAttributes](#) & attributes) [static, inherited]

Definition at line 177 of file KDChartAbstractAreaBase.cpp.

References [KDChart::FrameAttributes::isVisible\(\)](#), and [KDChart::FrameAttributes::pen\(\)](#).

Referenced by [KDChart::AbstractAreaBase::paintFrame\(\)](#).

```
179 {
180
181     if( !attributes.isVisible() ) return;
182
183     // Note: We set the brush to NoBrush explicitly here.
184     //       Otherwise we might get a filled rectangle, so any
```

```

185     //          previously drawn background would be overwritten by that area.
186
187     const QPen    oldPen(    painter.pen() );
188     const QBrush oldBrush( painter.brush() );
189     painter.setPen(    attributes.pen() );
190     painter.setBrush( Qt::NoBrush );
191     painter.drawRect( rect.adjusted( 0, 0, -1, -1 ) );
192     painter.setBrush( oldBrush );
193     painter.setPen(    oldPen );
194 }

```

### 9.3.3.17 void AbstractAreaWidget::paintIntoRect (QPainter & *painter*, const QRect & *rect*) [virtual]

Draws the background and frame, then calls [paint\(\)](#).

In most cases there is no need to overwrite this method in a derived class, but you would overwrite [paint\(\)](#) instead.

Definition at line 109 of file KDChartAbstractAreaWidget.cpp.

References [d](#), and [paintAll\(\)](#).

Referenced by [KDChart::Chart::paint\(\)](#).

```

110 {
111     //qDebug() << "AbstractAreaWidget::paintIntoRect() called rect=" << rect;
112
113     if( rect.isEmpty() ) return;
114
115     d->resizeLayout( this, rect.size() );
116
117     const QPoint translation( rect.topLeft() );
118     painter.translate( translation );
119     paintAll( painter );
120     painter.translate( -translation.x(), -translation.y() );
121
122     /*
123     // make sure, the contents of the widget have been set up,
124     // so we get a usefull geometry:
125     needSizeHint();
126
127     const QRect oldGeometry( layout()->geometry() );
128     const QRect newGeo( QPoint(0,0), rect.size() );
129     const bool mustChangeGeo = layout() && oldGeometry != newGeo;
130     if( mustChangeGeo )
131         layout()->setGeometry( newGeo );
132     painter.translate( rect.left(), rect.top() );
133     paintAll( painter );
134     painter.translate( -rect.left(), -rect.top() );
135     if( mustChangeGeo )
136         layout()->setGeometry( oldGeometry );
137     */
138 }

```

### 9.3.3.18 void KDChart::AbstractAreaWidget::positionChanged ([AbstractAreaWidget \\*](#)) [signal]

Referenced by [positionHasChanged\(\)](#).

**9.3.3.19 void AbstractAreaWidget::positionHasChanged ()** [protected, virtual]

Reimplemented from [KDChart::AbstractAreaBase](#).

Definition at line 191 of file KDChartAbstractAreaWidget.cpp.

References [positionChanged\(\)](#).

```
192 {  
193     emit positionChanged( this );  
194 }
```

**9.3.3.20 void AbstractAreaWidget::resizeLayout (const QSize &)** [virtual]

Reimplemented in [KDChart::Legend](#).

Definition at line 93 of file KDChartAbstractAreaWidget.cpp.

```
94 {  
95     Q_UNUSED( size );  
96     // this block left empty intentionally  
97 }
```

**9.3.3.21 void AbstractAreaBase::setBackgroundAttributes (const [BackgroundAttributes](#) & a)**  
[inherited]

Definition at line 111 of file KDChartAbstractAreaBase.cpp.

References [d](#), and [KDChart::AbstractAreaBase::positionHasChanged\(\)](#).

```
112 {  
113     if( d->backgroundAttributes == a )  
114         return;  
115  
116     d->backgroundAttributes = a;  
117     positionHasChanged();  
118 }
```

**9.3.3.22 void AbstractAreaBase::setFrameAttributes (const [FrameAttributes](#) & a)**  
[inherited]

Definition at line 97 of file KDChartAbstractAreaBase.cpp.

References [d](#), and [KDChart::AbstractAreaBase::positionHasChanged\(\)](#).

Referenced by [KDChart::Legend::clone\(\)](#).

```
98 {  
99     if( d->frameAttributes == a )  
100         return;  
101  
102     d->frameAttributes = a;  
103     positionHasChanged();  
104 }
```

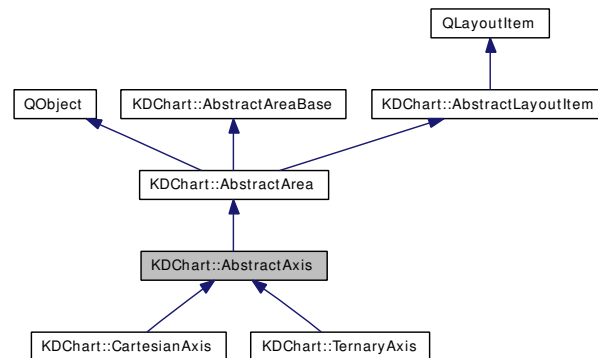
The documentation for this class was generated from the following files:

- [KDChartAbstractAreaWidget.h](#)
- [KDChartAbstractAreaWidget.cpp](#)

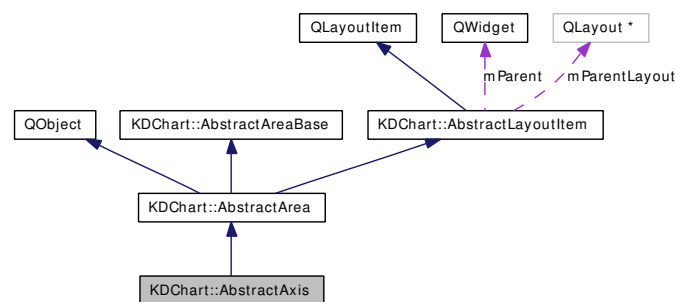
## 9.4 KDChart::AbstractAxis Class Reference

```
#include <KDChartAbstractAxis.h>
```

Inheritance diagram for KDChart::AbstractAxis:



Collaboration diagram for KDChart::AbstractAxis:



### 9.4.1 Detailed Description

The base class for axes.

For being useful, axes need to be assigned to a diagram, see [AbstractCartesianDiagram::addAxis](#) and [AbstractCartesianDiagram::takeAxis](#).

See also:

PolarAxis, [AbstractCartesianDiagram](#)

Definition at line 63 of file KDChartAbstractAxis.h.

### Public Slots

- void [update](#) ()

### Signals

- void [positionChanged](#) ([AbstractArea](#) \*)



## Public Member Functions

- [AbstractAxis](#) ([AbstractDiagram](#) \*diagram=0)
- void [alignToReferencePoint](#) (const [RelativePosition](#) &position)
- [BackgroundAttributes](#) [backgroundAttributes](#) () const
- virtual int [bottomOverlap](#) (bool doNotRecalculate=false) const  
*This is called at layout time by [KDChart::AutoSpacerLayoutItem::sizeHint\(\)](#).*
- bool [compare](#) (const [AbstractAreaBase](#) \*other) const  
*Returns true if both areas have the same settings.*
- bool [compare](#) (const [AbstractAxis](#) \*other) const  
*Returns true if both axes have the same settings.*
- virtual void [connectSignals](#) ()  
*Wiring the signal/slot connections.*
- const [AbstractCoordinatePlane](#) \* [coordinatePlane](#) () const  
*Convenience function, returns the coordinate plane, in which this axis is used.*
- void [createObserver](#) ([AbstractDiagram](#) \*diagram)
- virtual const QString [customizedLabel](#) (const QString &label) const  
*Implement this method if you want to adjust axis labels before they are printed.*
- void [deleteObserver](#) ([AbstractDiagram](#) \*diagram)
- const [AbstractDiagram](#) \* [diagram](#) () const
- [FrameAttributes](#) [frameAttributes](#) () const
- virtual QRect [geometry](#) () const=0
- void [getFrameLeadings](#) (int &left, int &top, int &right, int &bottom) const
- QStringList [labels](#) () const  
*Returns a list of strings, that are used as axis labels, as set via [setLabels](#).*
- virtual int [leftOverlap](#) (bool doNotRecalculate=false) const  
*This is called at layout time by [KDChart::AutoSpacerLayoutItem::sizeHint\(\)](#).*
- bool [observedBy](#) ([AbstractDiagram](#) \*diagram) const
- virtual void [paint](#) (QPainter \*)=0
- virtual void [paintAll](#) (QPainter &painter)  
*Call [paintAll](#), if you want the background and the frame to be drawn before the normal [paint\(\)](#) is invoked automatically.*
- virtual void [paintBackground](#) (QPainter &painter, const QRect &rectangle)
- virtual void [paintCtx](#) ([PaintContext](#) \*context)  
*Default impl: Paint the complete item using its layouted position and size.*
- virtual void [paintFrame](#) (QPainter &painter, const QRect &rectangle)
- virtual void [paintIntoRect](#) (QPainter &painter, const QRect &rect)  
*Draws the background and frame, then calls [paint\(\)](#).*
- QLayout \* [parentLayout](#) ()

- void [removeFromParentLayout](#) ()
- virtual int [rightOverlap](#) (bool doNotRecalculate=false) const  
*This is called at layout time by [KDChart::AutoSpacerLayoutItem::sizeHint\(\)](#).*
- void [setBackgroundAttributes](#) (const [BackgroundAttributes](#) &a)
- void [setFrameAttributes](#) (const [FrameAttributes](#) &a)
- virtual void [setGeometry](#) (const QRect &rect)=0
- void [setLabels](#) (const QStringList &list)  
*Use this to specify your own set of strings, to be used as axis labels.*
- void [setParentLayout](#) (QLayout \*lay)
- virtual void [setParentWidget](#) (QWidget \*widget)  
*Inform the item about its widget: This enables the item, to trigger that widget's update, whenever the size of the item's contents has changed.*
- void [setShortLabels](#) (const QStringList &list)  
*Use this to specify your own set of strings, to be used as axis labels, in case the normal labels are too long.*
- void [setTextAttributes](#) (const [TextAttributes](#) &a)  
*Use this to specify the text attributes to be used for axis labels.*
- QStringList [shortLabels](#) () const  
*Returns a list of strings, that are used as axis labels, as set via [setShortLabels](#).*
- virtual void [sizeHintChanged](#) () const  
*Report changed size hint: ask the parent widget to recalculate the layout.*
- [TextAttributes](#) [textAttributes](#) () const  
*Returns the text attributes to be used for axis labels.*
- virtual int [topOverlap](#) (bool doNotRecalculate=false) const  
*This is called at layout time by [KDChart::AutoSpacerLayoutItem::sizeHint\(\)](#).*
- virtual [~AbstractAxis](#) ()

## Static Public Member Functions

- static void [paintBackgroundAttributes](#) (QPainter &painter, const QRect &rectangle, const [KDChart::BackgroundAttributes](#) &attributes)
- static void [paintFrameAttributes](#) (QPainter &painter, const QRect &rectangle, const [KDChart::FrameAttributes](#) &attributes)

## Protected Slots

- virtual void [delayedInit](#) ()  
*called for initializing after the c'tor has completed*

## Protected Member Functions

- virtual QRect [areaGeometry](#) () const
- QRect [innerRect](#) () const
- virtual void [positionHasChanged](#) ()

## Protected Attributes

- QWidget \* [mParent](#)
- QLayout \* [mParentLayout](#)

## 9.4.2 Constructor & Destructor Documentation

### 9.4.2.1 AbstractAxis::AbstractAxis ([AbstractDiagram](#) \* *diagram* = 0) [explicit]

Definition at line 108 of file KDChartAbstractAxis.cpp.

References [delayedInit\(\)](#).

```
109      : AbstractArea( new Private( diagram, this ) )
110  {
111      init();
112      QTimer::singleShot(0, this, SLOT(delayedInit()));
113  }
```

### 9.4.2.2 AbstractAxis::~~AbstractAxis () [virtual]

Definition at line 115 of file KDChartAbstractAxis.cpp.

References [d](#).

```
116  {
117      d->mDiagram = 0;
118      d->secondaryDiagrams.clear();
119  }
```

## 9.4.3 Member Function Documentation

### 9.4.3.1 void AbstractAreaBase::alignToReferencePoint (const [RelativePosition](#) & *position*) [inherited]

Definition at line 90 of file KDChartAbstractAreaBase.cpp.

```
91  {
92      Q_UNUSED( position );
93      // PENDING(kalle) FIXME
94      qWarning( "Sorry, not implemented: void AbstractAreaBase::alignToReferencePoint( const RelativePosi
95  }
```

#### 9.4.3.2 **QRect AbstractArea::areaGeometry () const** [protected, virtual, inherited]

Implements [KDChart::AbstractAreaBase](#).

Definition at line 150 of file KDChartAbstractArea.cpp.

Referenced by [KDChart::CartesianCoordinatePlane::drawingArea\(\)](#), [KDChart::TernaryCoordinatePlane::layoutDiagrams\(\)](#), [KDChart::PolarCoordinatePlane::layoutDiagrams\(\)](#), [KDChart::TernaryCoordinatePlane::paint\(\)](#), [KDChart::CartesianAxis::paint\(\)](#), [KDChart::AbstractArea::paintAll\(\)](#), and [KDChart::CartesianAxis::paintCtx\(\)](#).

```
151 {
152     return geometry();
153 }
```

#### 9.4.3.3 **BackgroundAttributes AbstractAreaBase::backgroundAttributes () const** [inherited]

Definition at line 120 of file KDChartAbstractAreaBase.cpp.

References [d](#).

Referenced by [KDChart::AbstractAreaBase::compare\(\)](#), and [updateCommonBrush\(\)](#).

```
121 {
122     return d->backgroundAttributes;
123 }
```

#### 9.4.3.4 **int AbstractArea::bottomOverlap (bool *doNotRecalculate* = false) const** [virtual, inherited]

This is called at layout time by [KDChart::AutoSpacerLayoutItem::sizeHint\(\)](#).

The method triggers [AbstractArea::sizeHint\(\)](#) to find out the amount of overlap at the bottom edge of the area.

##### **Note:**

The default implementation is not using any caching, it might make sense to implement a more sophisticated solution for derived classes that have complex work to do in [sizeHint\(\)](#). All we have here is a primitive flag to be set by the caller if it is sure that no [sizeHint\(\)](#) needs to be called.

Definition at line 101 of file KDChartAbstractArea.cpp.

References [d](#).

Referenced by [KDChart::CartesianAxis::maximumSize\(\)](#).

```
102 {
103     // Re-calculate the sizes,
104     // so we also get the amountOf..Overlap members set newly:
105     if( ! doNotRecalculate )
106         sizeHint();
107     return d->amountOfBottomOverlap;
108 }
```

**9.4.3.5 bool AbstractAreaBase::compare (const [AbstractAreaBase](#) \* *other*) const** [inherited]

Returns true if both areas have the same settings.

Definition at line 75 of file KDChartAbstractAreaBase.cpp.

References [KDChart::AbstractAreaBase::backgroundAttributes\(\)](#), and [KDChart::AbstractAreaBase::frameAttributes\(\)](#).

```

76 {
77     if( other == this ) return true;
78     if( ! other ){
79         //qDebug() << "CartesianAxis::compare() cannot compare to Null pointer";
80         return false;
81     }
82     /*
83     qDebug() << "AbstractAreaBase:" << (frameAttributes() == other->frameAttributes())
84         << (backgroundAttributes() == other->backgroundAttributes()) << "\n";
85     */
86     return (frameAttributes() == other->frameAttributes()) &&
87         (backgroundAttributes() == other->backgroundAttributes());
88 }
```

**9.4.3.6 bool AbstractAxis::compare (const [AbstractAxis](#) \* *other*) const**

Returns true if both axes have the same settings.

Definition at line 142 of file KDChartAbstractAxis.cpp.

References [labels\(\)](#), [shortLabels\(\)](#), and [textAttributes\(\)](#).

```

143 {
144     if( other == this ) return true;
145     if( ! other ){
146         //qDebug() << "CartesianAxis::compare() cannot compare to Null pointer";
147         return false;
148     }
149     /*
150     qDebug() << (textAttributes() == other->textAttributes());
151     qDebug() << (labels() == other->labels());
152     qDebug() << (shortLabels() == other->shortLabels());
153     */
154     return ( static_cast<const AbstractAreaBase*>(this)->compare( other ) ) &&
155         (textAttributes() == other->textAttributes()) &&
156         (labels() == other->labels()) &&
157         (shortLabels() == other->shortLabels());
158 }
```

**9.4.3.7 void AbstractAxis::connectSignals ()** [virtual]

Wiring the signal/slot connections.

This method gets called automatically, each time, when you assign the axis to a diagram, either by passing a diagram\* to the c'tor, or by calling the diagram's setAxis method, resp.

If overwriting this method in derived classes, make sure to call this base method [AbstractAxis::connectSignals\(\)](#), so your axis gets connected to the diagram's built-in signals.

See also:

[AbstractCartesianDiagram::addAxis\(\)](#)

Definition at line 211 of file KDChartAbstractAxis.cpp.

References `d`, and `update()`.

Referenced by `createObserver()`.

```

212 {
213     if( d->observer ){
214         connect( d->observer, SIGNAL( diagramDataChanged( AbstractDiagram *) ),
215                 this, SLOT( update() ) );
216     }
217 }
```

#### 9.4.3.8 `const AbstractCoordinatePlane * AbstractAxis::coordinatePlane () const`

Convenience function, returns the coordinate plane, in which this axis is used.

If the axis is not used in a coordinate plane, the return value is Zero.

Definition at line 324 of file KDChartAbstractAxis.cpp.

References `d`.

Referenced by `KDChart::CartesianCoordinatePlane::sharedAxisMasterPlane()`.

```

325 {
326     if( d->diagram() )
327         return d->diagram()->coordinatePlane();
328     return 0;
329 }
```

#### 9.4.3.9 `void AbstractAxis::createObserver (AbstractDiagram * diagram)`

Definition at line 177 of file KDChartAbstractAxis.cpp.

References `connectSignals()`, `d`, and `diagram()`.

```

178 {
179     if( d->setDiagram( diagram ) )
180         connectSignals();
181 }
```

#### 9.4.3.10 `const QString AbstractAxis::customizedLabel (const QString & label) const` [virtual]

Implement this method if you want to adjust axis labels before they are printed.

KD [Chart](#) is calling this method immediately before drawing the text, this means: What you return here will be drawn without further modifications.

##### Parameters:

*label* The text of the label as KD [Chart](#) has calculated it automatically (or as it was taken from a QStringList provided by you, resp.)

##### Returns:

The text to be drawn. By default this is the same as `label`.

Definition at line 161 of file KDChartAbstractAxis.cpp.

Referenced by KDChart::CartesianAxis::maximumSize(), and KDChart::CartesianAxis::paintCtx().

```
162 {  
163     return label;  
164 }
```

#### 9.4.3.11 void AbstractAxis::delayedInit () [protected, virtual, slot]

called for initializing after the c'tor has completed

Definition at line 134 of file KDChartAbstractAxis.cpp.

References d.

Referenced by AbstractAxis().

```
135 {  
136     // We call setDiagram() here, because the c'tor of Private  
137     // only has stored the pointers, but it did not call setDiagram().  
138     if( d )  
139         d->setDiagram( 0, true /* delayedInit */ );  
140 }
```

#### 9.4.3.12 void AbstractAxis::deleteObserver ([AbstractDiagram](#) \* *diagram*)

Definition at line 193 of file KDChartAbstractAxis.cpp.

References d, and diagram().

Referenced by KDChart::AbstractCartesianDiagram::takeAxis(), and KDChart::AbstractCartesianDiagram::~~AbstractCartesianDiagram().

```
194 {  
195     d->unsetDiagram( diagram );  
196 }
```

#### 9.4.3.13 const [AbstractDiagram](#) \* KDChart::AbstractAxis::diagram () const

Definition at line 331 of file KDChartAbstractAxis.cpp.

References d.

Referenced by createObserver(), deleteObserver(), observedBy(), KDChart::CartesianAxis::paintCtx(), KDChart::TernaryAxis::TernaryAxis(), and KDChart::CartesianAxis::~~CartesianAxis().

```
332 {  
333     return d->diagram();  
334 }
```

#### 9.4.3.14 **FrameAttributes** AbstractAreaBase::frameAttributes () const [inherited]

Definition at line 106 of file KDChartAbstractAreaBase.cpp.

References d.

Referenced by KDChart::Legend::clone(), KDChart::AbstractAreaBase::compare(), and updateCommonBrush().

```
107 {
108     return d->frameAttributes;
109 }
```

#### 9.4.3.15 **virtual QRect** KDChart::AbstractAxis::geometry () const [pure virtual]

Implemented in [KDChart::CartesianAxis](#), and [KDChart::TernaryAxis](#).

#### 9.4.3.16 **void** AbstractAreaBase::getFrameLeadings (int & *left*, int & *top*, int & *right*, int & *bottom*) const [inherited]

Definition at line 212 of file KDChartAbstractAreaBase.cpp.

References d.

Referenced by KDChart::AbstractAreaBase::innerRect(), and KDChart::AbstractAreaWidget::paintAll().

```
213 {
214     if( d && d->frameAttributes.isVisible() ){
215         const int padding = qMax( d->frameAttributes.padding(), 0 );
216         left    = padding;
217         top     = padding;
218         right   = padding;
219         bottom  = padding;
220     }else{
221         left    = 0;
222         top     = 0;
223         right   = 0;
224         bottom  = 0;
225     }
226 }
```

#### 9.4.3.17 **QRect** AbstractAreaBase::innerRect () const [protected, inherited]

Definition at line 228 of file KDChartAbstractAreaBase.cpp.

References KDChart::AbstractAreaBase::areaGeometry(), and KDChart::AbstractAreaBase::getFrameLeadings().

Referenced by KDChart::TextArea::paintAll(), and KDChart::AbstractArea::paintAll().

```
229 {
230     int left;
231     int top;
232     int right;
233     int bottom;
234     getFrameLeadings( left, top, right, bottom );
235     return
```



```

236         QRect( QPoint(0,0), areaGeometry().size() )
237         .adjusted( left, top, -right, -bottom );
238     }

```

#### 9.4.3.18 QStringList AbstractAxis::labels () const

Returns a list of strings, that are used as axis labels, as set via setLabels.

**See also:**

[setLabels](#)

Definition at line 281 of file KDChartAbstractAxis.cpp.

References [d](#).

Referenced by [compare\(\)](#), [KDChart::CartesianAxis::maximumSize\(\)](#), [KDChart::TernaryAxis::paintCtx\(\)](#), and [KDChart::CartesianAxis::paintCtx\(\)](#).

```

282 {
283     return d->hardLabels;
284 }

```

#### 9.4.3.19 int AbstractArea::leftOverlap (bool *doNotRecalculate* = false) const [virtual, inherited]

This is called at layout time by [KDChart::AutoSpacerLayoutItem::sizeHint\(\)](#).

The method triggers AbstractArea::sizeHint() to find out the amount of overlap at the left edge of the area.

**Note:**

The default implementation is not using any caching, it might make sense to implement a more sophisticated solution for derived classes that have complex work to do in sizeHint(). All we have here is a primitive flag to be set by the caller if it is sure that no sizeHint() needs to be called.

Definition at line 77 of file KDChartAbstractArea.cpp.

References [d](#).

Referenced by [KDChart::CartesianAxis::maximumSize\(\)](#).

```

78 {
79     // Re-calculate the sizes,
80     // so we also get the amountOf..Overlap members set newly:
81     if( ! doNotRecalculate )
82         sizeHint();
83     return d->amountOfLeftOverlap;
84 }

```

#### 9.4.3.20 bool KDChart::AbstractAxis::observedBy ([AbstractDiagram](#) \* *diagram*) const

Definition at line 336 of file KDChartAbstractAxis.cpp.

References [d](#), and [diagram\(\)](#).

```

337 {
338     return d->hasDiagram( diagram );
339 }

```

#### 9.4.3.21 virtual void KDChart::AbstractLayoutItem::paint (QPainter \*) [pure virtual, inherited]

Implemented in [KDChart::CartesianAxis](#), [KDChart::CartesianCoordinatePlane](#), [KDChart::TextLayoutItem](#), [KDChart::MarkerLayoutItem](#), [KDChart::LineLayoutItem](#), [KDChart::LineWithMarkerLayoutItem](#), [KDChart::HorizontalLineLayoutItem](#), [KDChart::VerticalLineLayoutItem](#), [KDChart::AutoSpacerLayoutItem](#), [KDChart::PolarCoordinatePlane](#), [KDChart::TernaryAxis](#), and [KDChart::TernaryCoordinatePlane](#).

Referenced by [KDChart::Legend::paint\(\)](#), [KDChart::AbstractLayoutItem::paintAll\(\)](#), [KDChart::AbstractArea::paintAll\(\)](#), and [KDChart::AbstractLayoutItem::paintCtx\(\)](#).

#### 9.4.3.22 void AbstractArea::paintAll (QPainter & painter) [virtual, inherited]

Call `paintAll`, if you want the background and the frame to be drawn before the normal `paint()` is invoked automatically.

Reimplemented from [KDChart::AbstractLayoutItem](#).

Reimplemented in [KDChart::TernaryAxis](#).

Definition at line 123 of file `KDChartAbstractArea.cpp`.

References [KDChart::AbstractArea::areaGeometry\(\)](#), `d`, [KDChart::AbstractAreaBase::innerRect\(\)](#), [KDChart::AbstractLayoutItem::paint\(\)](#), [KDChart::AbstractAreaBase::paintBackground\(\)](#), and [KDChart::AbstractAreaBase::paintFrame\(\)](#).

Referenced by [KDChart::AbstractArea::paintIntoRect\(\)](#).

```

124 {
125     // Paint the background and frame
126     const QRect overlappingArea( geometry().adjusted(
127         -d->amountOfLeftOverlap,
128         -d->amountOfTopOverlap,
129         d->amountOfRightOverlap,
130         d->amountOfBottomOverlap ) );
131     paintBackground( painter, overlappingArea );
132     paintFrame( painter, overlappingArea );
133
134     // temporarily adjust the widget size, to be sure all content gets calculated
135     // to fit into the inner rectangle
136     const QRect oldGeometry( areaGeometry() );
137     QRect inner( innerRect() );
138     inner.moveTo(
139         oldGeometry.left() + inner.left(),
140         oldGeometry.top() + inner.top() );
141     const bool needAdjustGeometry = oldGeometry != inner;
142     if( needAdjustGeometry )
143         setGeometry( inner );
144     paint( &painter );
145     if( needAdjustGeometry )
146         setGeometry( oldGeometry );
147     //qDebug() << "AbstractAreaWidget::paintAll() done.";
148 }

```

### 9.4.3.23 void AbstractAreaBase::paintBackground (QPainter & painter, const QRect & rectangle) [virtual, inherited]

Definition at line 196 of file KDCartAbstractAreaBase.cpp.

References d, and KDCart::AbstractAreaBase::paintBackgroundAttributes().

Referenced by KDCart::TextArea::paintAll(), KDCart::AbstractAreaWidget::paintAll(), and KDCart::AbstractArea::paintAll().

```
197 {
198     Q_ASSERT_X ( d != 0, "AbstractAreaBase::paintBackground()",
199                 "Private class was not initialized!" );
200     paintBackgroundAttributes( painter, rect, d->backgroundAttributes );
201 }
```

### 9.4.3.24 void AbstractAreaBase::paintBackgroundAttributes (QPainter & painter, const QRect & rectangle, const KDCart::BackgroundAttributes & attributes) [static, inherited]

Definition at line 127 of file KDCartAbstractAreaBase.cpp.

References KDCart::BackgroundAttributes::BackgroundPixmapModeCentered, KDCart::BackgroundAttributes::BackgroundPixmapModeNone, KDCart::BackgroundAttributes::BackgroundPixmapModeScaled, KDCart::BackgroundAttributes::BackgroundPixmapModeStretched, KDCart::BackgroundAttributes::brush(), KDCart::BackgroundAttributes::isVisible(), KDCart::BackgroundAttributes::pixmap(), and KDCart::BackgroundAttributes::pixmapMode().

Referenced by KDCart::AbstractAreaBase::paintBackground().

```
129 {
130     if( !attributes.isVisible() ) return;
131
132     /* first draw the brush (may contain a pixmap)*/
133     if( Qt::NoBrush != attributes.brush().style() ) {
134         KDCart::PainterSaver painterSaver( &painter );
135         painter.setPen( Qt::NoPen );
136         const QPointF newTopLeft( painter.deviceMatrix().map( rect.topLeft() ) );
137         painter.setBrushOrigin( newTopLeft );
138         painter.setBrush( attributes.brush() );
139         painter.drawRect( rect.adjusted( 0, 0, -1, -1 ) );
140     }
141     /* next draw the backPixmap over the brush */
142     if( !attributes.pixmap().isNull() &&
143         attributes.pixmapMode() != BackgroundAttributes::BackgroundPixmapModeNone ) {
144         QPointF ol = rect.topLeft();
145         if( BackgroundAttributes::BackgroundPixmapModeCentered == attributes.pixmapMode() )
146         {
147             ol.setX( rect.center().x() - attributes.pixmap().width() / 2 );
148             ol.setY( rect.center().y() - attributes.pixmap().height() / 2 );
149             painter.drawPixmap( ol, attributes.pixmap() );
150         } else {
151             QMatrix m;
152             double zW = (double)rect.width() / (double)attributes.pixmap().width();
153             double zH = (double)rect.height() / (double)attributes.pixmap().height();
154             switch( attributes.pixmapMode() ) {
155             case BackgroundAttributes::BackgroundPixmapModeScaled:
156                 {
157                     double z;
158                     z = qMin( zW, zH );
159                     m.scale( z, z );
160                 }
```

```

161         break;
162         case BackgroundAttributes::BackgroundPixmapModeStretched:
163             m.scale( zW, zH );
164             break;
165         default:
166             ; // Cannot happen, previously checked
167     }
168     QPixmap pm = attributes.pixmap().transformed( m );
169     ol.setX( rect.center().x() - pm.width() / 2 );
170     ol.setY( rect.center().y() - pm.height() / 2 );
171     painter.drawPixmap( ol, pm );
172 }
173 }
174 }

```

#### 9.4.3.25 void KDChart::AbstractLayoutItem::paintCtx ([PaintContext](#) \* *context*) [virtual, inherited]

Default impl: Paint the complete item using its layouted position and size.

Reimplemented in [KDChart::CartesianAxis](#), and [KDChart::TernaryAxis](#).

Definition at line 77 of file KDChartLayoutItems.cpp.

References [KDChart::AbstractLayoutItem::paint\(\)](#), and [KDChart::PaintContext::painter\(\)](#).

```

78 {
79     if( context )
80         paint( context->painter() );
81 }

```

#### 9.4.3.26 void AbstractAreaBase::paintFrame (QPainter & *painter*, const QRect & *rectangle*) [virtual, inherited]

Definition at line 204 of file KDChartAbstractAreaBase.cpp.

References [d](#), and [KDChart::AbstractAreaBase::paintFrameAttributes\(\)](#).

Referenced by [KDChart::TextArea::paintAll\(\)](#), [KDChart::AbstractAreaWidget::paintAll\(\)](#), and [KDChart::AbstractArea::paintAll\(\)](#).

```

205 {
206     Q_ASSERT_X ( d != 0, "AbstractAreaBase::paintFrame()",
207                 "Private class was not initialized!" );
208     paintFrameAttributes( painter, rect, d->frameAttributes );
209 }

```

#### 9.4.3.27 void AbstractAreaBase::paintFrameAttributes (QPainter & *painter*, const QRect & *rectangle*, const [KDChart::FrameAttributes](#) & *attributes*) [static, inherited]

Definition at line 177 of file KDChartAbstractAreaBase.cpp.

References [KDChart::FrameAttributes::isVisible\(\)](#), and [KDChart::FrameAttributes::pen\(\)](#).

Referenced by [KDChart::AbstractAreaBase::paintFrame\(\)](#).

```

179 {
180
181     if( !attributes.isVisible() ) return;
182
183     // Note: We set the brush to NoBrush explicitly here.
184     //       Otherwise we might get a filled rectangle, so any
185     //       previously drawn background would be overwritten by that area.
186
187     const QPen    oldPen( painter.pen() );
188     const QBrush  oldBrush( painter.brush() );
189     painter.setPen( attributes.pen() );
190     painter.setBrush( Qt::NoBrush );
191     painter.drawRect( rect.adjusted( 0, 0, -1, -1 ) );
192     painter.setBrush( oldBrush );
193     painter.setPen( oldPen );
194 }

```

#### 9.4.3.28 void AbstractArea::paintIntoRect (QPainter & *painter*, const QRect & *rect*) [virtual, inherited]

Draws the background and frame, then calls [paint\(\)](#).

In most cases there is no need to overwrite this method in a derived class, but you would overwrite [AbstractLayoutItem::paint\(\)](#) instead.

Definition at line 111 of file KDChartAbstractArea.cpp.

References KDChart::AbstractArea::paintAll().

```

112 {
113     const QRect oldGeometry( geometry() );
114     if( oldGeometry != rect )
115         setGeometry( rect );
116     painter.translate( rect.left(), rect.top() );
117     paintAll( painter );
118     painter.translate( -rect.left(), -rect.top() );
119     if( oldGeometry != rect )
120         setGeometry( oldGeometry );
121 }

```

#### 9.4.3.29 QLayout\* KDChart::AbstractLayoutItem::parentLayout () [inherited]

Definition at line 76 of file KDChartLayoutItems.h.

```

77     {
78         return mParentLayout;
79     }

```

#### 9.4.3.30 void KDChart::AbstractArea::positionChanged ([AbstractArea](#) \*) [signal, inherited]

Referenced by KDChart::AbstractArea::positionHasChanged().

#### 9.4.3.31 void AbstractArea::positionHasChanged () [protected, virtual, inherited]

Reimplemented from [KDChart::AbstractAreaBase](#).

Definition at line 155 of file KDChartAbstractArea.cpp.

References KDChart::AbstractArea::positionChanged().

```
156 {
157     emit positionChanged( this );
158 }
```

#### 9.4.3.32 void KDChart::AbstractLayoutItem::removeFromParentLayout () [inherited]

Definition at line 80 of file KDChartLayoutItems.h.

Referenced by KDChart::Chart::takeCoordinatePlane().

```
81     {
82         if( mParentLayout ){
83             if( widget() )
84                 mParentLayout->removeWidget( widget() );
85             else
86                 mParentLayout->removeItem( this );
87         }
88     }
```

#### 9.4.3.33 int AbstractArea::rightOverlap (bool *doNotRecalculate* = false) const [virtual, inherited]

This is called at layout time by [KDChart::AutoSpacerLayoutItem::sizeHint\(\)](#).

The method triggers AbstractArea::sizeHint() to find out the amount of overlap at the right edge of the area.

##### Note:

The default implementation is not using any caching, it might make sense to implement a more sophisticated solution for derived classes that have complex work to do in sizeHint(). All we have here is a primitive flag to be set by the caller if it is sure that no sizeHint() needs to be called.

Definition at line 85 of file KDChartAbstractArea.cpp.

References d.

Referenced by KDChart::CartesianAxis::maximumSize().

```
86 {
87     // Re-calculate the sizes,
88     // so we also get the amountOf..Overlap members set newly:
89     if( ! doNotRecalculate )
90         sizeHint();
91     return d->amountOfRightOverlap;
92 }
```

#### 9.4.3.34 void AbstractAreaBase::setBackgroundAttributes (const [BackgroundAttributes](#) & a) [inherited]

Definition at line 111 of file KDChartAbstractAreaBase.cpp.

References d, and KDChart::AbstractAreaBase::positionHasChanged().

```
112 {
113     if( d->backgroundAttributes == a )
114         return;
115
116     d->backgroundAttributes = a;
117     positionHasChanged();
118 }
```

#### 9.4.3.35 void AbstractAreaBase::setFrameAttributes (const [FrameAttributes](#) & a) [inherited]

Definition at line 97 of file `KDChartAbstractAreaBase.cpp`.

References `d`, and `KDChart::AbstractAreaBase::positionHasChanged()`.

Referenced by `KDChart::Legend::clone()`.

```
98 {
99     if( d->frameAttributes == a )
100         return;
101
102     d->frameAttributes = a;
103     positionHasChanged();
104 }
```

#### 9.4.3.36 virtual void KDChart::AbstractAxis::setGeometry (const `QRect` & rect) [pure virtual]

Implemented in [KDChart::CartesianAxis](#), and [KDChart::TernaryAxis](#).

#### 9.4.3.37 void AbstractAxis::setLabels (const `QStringList` & list)

Use this to specify your own set of strings, to be used as axis labels.

Labels specified via `setLabels` take precedence: If a non-empty list is passed, `KD Chart` will use these strings as axis labels, instead of calculating them.

If you a smaller number of strings than the number of labels drawn at this axis, `KD Chart` will iterate over the list, repeating the strings, until all labels are drawn. As an example you could specify the seven days of the week as abscissa labels, which would be repeatedly used then.

By passing an empty `QStringList` you can reset the default behaviour.

**See also:**

[labels](#), [setShortLabels](#)

Definition at line 267 of file `KDChartAbstractAxis.cpp`.

References `d`, and `update()`.

```
268 {
269     if( d->hardLabels == list )
270         return;
271
272     d->hardLabels = list;
273     update();
274 }
```

**9.4.3.38 void KDChart::AbstractLayoutItem::setParentLayout (QLayout \* *lay*)** [inherited]

Definition at line 72 of file KDChartLayoutItems.h.

```

73     {
74         mParentLayout = lay;
75     }

```

**9.4.3.39 void KDChart::AbstractLayoutItem::setParentWidget (QWidget \* *widget*)** [virtual, inherited]

Inform the item about its widget: This enables the item, to trigger that widget's update, whenever the size of the item's contents has changed.

Thus, you need to call setParentWidget on every item, that has a non-fixed size.

Definition at line 64 of file KDChartLayoutItems.cpp.

References KDChart::AbstractLayoutItem::mParent.

Referenced by KDChart::HeaderFooter::setParent(), and KDChart::AbstractCartesianDiagram::takeAxis().

```

65 {
66     mParent = widget;
67 }

```

**9.4.3.40 void AbstractAxis::setShortLabels (const QStringList & *list*)**

Use this to specify your own set of strings, to be used as axis labels, in case the normal labels are too long.

**Note:**

Setting done via setShortLabels will be ignored, if you did not pass a non-empty string list via setLabels too!

By passing an empty QStringList you can reset the default behaviour.

**See also:**

[shortLabels](#), [setLabels](#)

Definition at line 297 of file KDChartAbstractAxis.cpp.

References d, and update().

```

298 {
299     if( d->hardShortLabels == list )
300         return;
301
302     d->hardShortLabels = list;
303     update();
304 }

```



**9.4.3.41 void AbstractAxis::setTextAttributes (const [TextAttributes](#) & a)**

Use this to specify the text attributes to be used for axis labels.

By default, the reference area will be set at painting time. It will be the then-valid coordinate plane's parent widget, so normally, it will be the [KDChart::Chart](#). Thus the labels of all of your axes in all of your diagrams within that [Chart](#) will be drawn in same font size, by default.

**See also:**

[textAttributes](#), [setLabels](#)

Definition at line 231 of file KDChartAbstractAxis.cpp.

References d, and update().

```
232 {  
233     if( d->textAttributes == a )  
234         return;  
235  
236     d->textAttributes = a;  
237     update();  
238 }
```

**9.4.3.42 QStringList AbstractAxis::shortLabels () const**

Returns a list of strings, that are used as axis labels, as set via setShortLabels.

**Note:**

Setting done via setShortLabels will be ignored, if you did not pass a non-empty string list via setLabels too!

**See also:**

[setShortLabels](#)

Definition at line 314 of file KDChartAbstractAxis.cpp.

References d.

Referenced by compare(), and KDChart::CartesianAxis::paintCtx().

```
315 {  
316     return d->hardShortLabels;  
317 }
```

**9.4.3.43 void KDChart::AbstractLayoutItem::sizeHintChanged () const** [virtual, inherited]

Report changed size hint: ask the parent widget to recalculate the layout.

Definition at line 86 of file KDChartLayoutItems.cpp.

References KDChart::AbstractLayoutItem::mParent.

Referenced by KDChart::TextLayoutItem::sizeHint().

```

87 {
88     // This is exactly like what QWidget::updateGeometry does.
89     // qDebug("KDChart::AbstractLayoutItem::sizeHintChanged() called");
90     if( mParent ) {
91         if ( mParent->layout() )
92             mParent->layout()->invalidate();
93         else
94             QApplication::postEvent( mParent, new QEvent( QEvent::LayoutRequest ) );
95     }
96 }

```

#### 9.4.3.44 [TextAttributes](#) AbstractAxis::textAttributes () const

Returns the text attributes to be used for axis labels.

See also:

[setTextAttributes](#)

Definition at line 245 of file KDChartAbstractAxis.cpp.

References d.

Referenced by [compare\(\)](#), [KDChart::CartesianAxis::maximumSize\(\)](#), [KDChart::CartesianAxis::paintCtx\(\)](#), and [KDChart::CartesianAxis::titleTextAttributes\(\)](#).

```

246 {
247     return d->textAttributes;
248 }

```

#### 9.4.3.45 [int](#) AbstractArea::topOverlap (bool *doNotRecalculate* = false) const [virtual, inherited]

This is called at layout time by [KDChart::AutoSpacerLayoutItem::sizeHint\(\)](#).

The method triggers [AbstractArea::sizeHint\(\)](#) to find out the amount of overlap at the top edge of the area.

**Note:**

The default implementation is not using any caching, it might make sense to implement a more sophisticated solution for derived classes that have complex work to do in [sizeHint\(\)](#). All we have here is a primitive flag to be set by the caller if it is sure that no [sizeHint\(\)](#) needs to be called.

Definition at line 93 of file KDChartAbstractArea.cpp.

References d.

Referenced by [KDChart::CartesianAxis::maximumSize\(\)](#).

```

94 {
95     // Re-calculate the sizes,
96     // so we also get the amountOf..Overlap members set newly:
97     if( ! doNotRecalculate )
98         sizeHint();
99     return d->amountOfTopOverlap;
100 }

```

**9.4.3.46 void KDChart::AbstractAxis::update ()** [slot]

Definition at line 341 of file KDChartAbstractAxis.cpp.

References d.

Referenced by connectSignals(), setLabels(), setShortLabels(), and setTextAttributes().

```
342 {  
343     if( d->diagram() )  
344         d->diagram()->update();  
345 }
```

**9.4.4 Member Data Documentation****9.4.4.1 QWidget\* KDChart::AbstractLayoutItem::mParent** [protected, inherited]

Definition at line 90 of file KDChartLayoutItems.h.

Referenced by KDChart::AbstractLayoutItem::setParentWidget(), KDChart::TextLayoutItem::setText(), KDChart::TextLayoutItem::setTextAttributes(), and KDChart::AbstractLayoutItem::sizeHintChanged().

**9.4.4.2 QLayout\* KDChart::AbstractLayoutItem::mParentLayout** [protected, inherited]

Definition at line 91 of file KDChartLayoutItems.h.

Referenced by KDChart::AutoSpacerLayoutItem::paint().

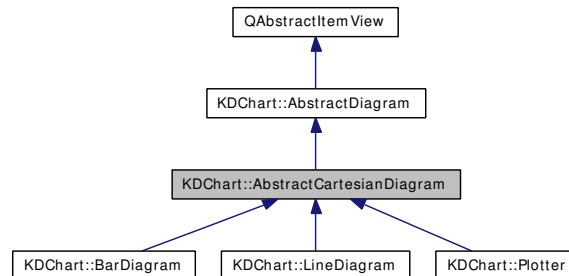
The documentation for this class was generated from the following files:

- [KDChartAbstractAxis.h](#)
- [KDChartAbstractAxis.cpp](#)

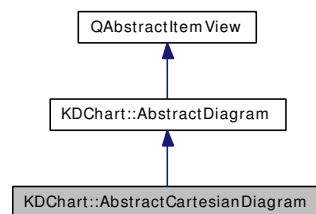
## 9.5 KDChart::AbstractCartesianDiagram Class Reference

```
#include <KDChartAbstractCartesianDiagram.h>
```

Inheritance diagram for KDChart::AbstractCartesianDiagram:



Collaboration diagram for KDChart::AbstractCartesianDiagram:



### 9.5.1 Detailed Description

Base class for diagrams based on a cartesian coordinate system.

The [AbstractCartesianDiagram](#) interface adds those elements that are specific to diagrams based on a cartesian coordinate system to the basic [AbstractDiagram](#) interface.

Definition at line 45 of file KDChartAbstractCartesianDiagram.h.

### Signals

- void [dataHidden](#) ()  
*This signal is emitted, when the hidden status of at least one data cell was (un)set.*
- void [layoutChanged](#) ([AbstractDiagram](#) \*)  
*Diagrams are supposed to emit this signal, when the layout of one of their element changes.*
- void [modelsChanged](#) ()  
*This signal is emitted, when either the model or the [AttributesModel](#) is replaced.*
- void [propertiesChanged](#) ()  
*Emitted upon change of a property of the Diagram.*

## Public Member Functions

- [AbstractCartesianDiagram](#) ([QWidget](#) \*parent=0, [CartesianCoordinatePlane](#) \*plane=0)
- virtual void [addAxis](#) ([CartesianAxis](#) \*axis)  
*Add the axis to the diagram.*
- bool [allowOverlappingDataValueTexts](#) () const  
**Returns:**  
*Whether data value labels are allowed to overlap.*
- bool [antiAliasing](#) () const  
**Returns:**  
*Whether anti-aliasing is to be used for rendering this diagram.*
- virtual [AttributesModel](#) \* [attributesModel](#) () const  
*Returns the [AttributesModel](#), that is used by this diagram.*
- virtual [KDChart::CartesianAxisList](#) [axes](#) () const  
**Returns:**  
*a list of all axes added to the diagram*
- [QBrush](#) [brush](#) (const [QModelIndex](#) &index) const  
*Retrieve the brush to be used, for painting the datapoint at the given index in the model.*
- [QBrush](#) [brush](#) (int dataset) const  
*Retrieve the brush to be used for the given dataset.*
- [QBrush](#) [brush](#) () const  
*Retrieve the brush to be used for painting datapoints globally.*
- bool [compare](#) (const [AbstractDiagram](#) \*other) const  
*Returns true if both diagrams have the same settings.*
- bool [compare](#) (const [AbstractCartesianDiagram](#) \*other) const  
*Returns true if both diagrams have the same settings.*
- [AbstractCoordinatePlane](#) \* [coordinatePlane](#) () const  
*The coordinate plane associated with the diagram.*
- const [QPair](#)< [QPointF](#), [QPointF](#) > [dataBoundaries](#) () const  
*Return the bottom left and top right data point, that the diagram will display (unless the grid adjusts these values).*
- virtual void [dataChanged](#) (const [QModelIndex](#) &topLeft, const [QModelIndex](#) &bottomRight)  
*[reimplemented]*
- [QList](#)< [QBrush](#) > [datasetBrushes](#) () const  
*The set of dataset brushes currently used, for use in legends, etc.*
- int [datasetDimension](#) () const

*The dataset dimension of a diagram determines, how many value dimensions it expects each datapoint to have.*

- QStringList [datasetLabels](#) () const  
*The set of dataset labels currently displayed, for use in legends, etc.*
- QList< [MarkerAttributes](#) > [datasetMarkers](#) () const  
*The set of dataset markers currently used, for use in legends, etc.*
- QList< QPen > [datasetPens](#) () const  
*The set of dataset pens currently used, for use in legends, etc.*
- [DataValueAttributes](#) [dataValueAttributes](#) (const QModelIndex &index) const  
*Retrieve the [DataValueAttributes](#) for the given index.*
- [DataValueAttributes](#) [dataValueAttributes](#) (int column) const  
*Retrieve the [DataValueAttributes](#) for the given dataset.*
- [DataValueAttributes](#) [dataValueAttributes](#) () const  
*Retrieve the [DataValueAttributes](#) specified globally.*
- virtual void [doItemsLayout](#) ()  
*[reimplemented]*
- virtual int [horizontalOffset](#) () const  
*[reimplemented]*
- virtual QModelIndex [indexAt](#) (const QPoint &point) const  
*[reimplemented]*
- QModelIndexList [indexesAt](#) (const QPoint &point) const  
*This method is added alongside with [indexAt](#) from QAIM, since in kdchart multiple indexes can be displayed at the same spot.*
- bool [isHidden](#) (const QModelIndex &index) const  
*Retrieve the hidden status for the given index.*
- bool [isHidden](#) (int column) const  
*Retrieve the hidden status for the given dataset.*
- bool [isHidden](#) () const  
*Retrieve the hidden status specified globally.*
- virtual bool [isIndexHidden](#) (const QModelIndex &index) const  
*[reimplemented]*
- QStringList [itemRowLabels](#) () const  
*The set of item row labels currently displayed, for use in Abscissa axes, etc.*
- virtual void [layoutPlanes](#) ()

*Triggers layouting of all coordinate planes on the current chart.*

- virtual QModelIndex [moveCursor](#) (CursorAction cursorAction, Qt::KeyboardModifiers modifiers)  
*[reimplemented]*
- virtual const int [numberOfAbscissaSegments](#) () const=0
- virtual const int [numberOfOrdinateSegments](#) () const=0
- virtual void [paint](#) (PaintContext \*paintContext)=0  
*Draw the diagram contents to the rectangle and painter, that are passed in as part of the paint context.*
- void [paintDataValueText](#) (QPainter \*painter, const QModelIndex &index, const QPointF &pos, double value)
- void [paintMarker](#) (QPainter \*painter, const QModelIndex &index, const QPointF &pos)
- virtual void [paintMarker](#) (QPainter \*painter, const MarkerAttributes &markerAttributes, const QBrush &brush, const QPen &, const QPointF &point, const QSizeF &size)
- QPen [pen](#) (const QModelIndex &index) const  
*Retrieve the pen to be used, for painting the datapoint at the given index in the model.*
- QPen [pen](#) (int dataset) const  
*Retrieve the pen to be used for the given dataset.*
- QPen [pen](#) () const  
*Retrieve the pen to be used for painting datapoints globally.*
- bool [percentMode](#) () const
- virtual AbstractCartesianDiagram \* [referenceDiagram](#) () const  
**Returns:**  
*this diagram's reference diagram*
- virtual QPointF [referenceDiagramOffset](#) () const  
**Returns:**  
*the relative offset of this diagram's reference diagram*
- virtual void [resize](#) (const QSizeF &area)=0  
*Called by the widget's sizeEvent.*
- virtual void [scrollTo](#) (const QModelIndex &index, ScrollHint hint=Ensure Visible)  
*[reimplemented]*
- void [setAllowOverlappingDataValueTexts](#) (bool allow)  
*Set whether data value labels are allowed to overlap.*
- void [setAntiAliasing](#) (bool enabled)  
*Set whether anti-aliasing is to be used while rendering this diagram.*
- void [setAttributesModel](#) (AttributesModel \*model)  
*Associate an AttributesModel with this diagram.*
- void [setBrush](#) (const QBrush &brush)  
*Set the brush to be used, for painting all datasets in the model.*

- void [setBrush](#) (int dataset, const QBrush &brush)  
*Set the brush to be used, for painting the given dataset.*
- void [setBrush](#) (const QModelIndex &index, const QBrush &brush)  
*Set the brush to be used, for painting the datapoint at the given index.*
- virtual void [setCoordinatePlane](#) ([AbstractCoordinatePlane](#) \*plane)  
*[reimplemented]*
- void [setDatasetDimension](#) (int dimension)  
*Sets the dataset dimension of the diagram.*
- void [setDataValueAttributes](#) (const [DataValueAttributes](#) &a)  
*Set the [DataValueAttributes](#) for all datapoints in the model.*
- void [setDataValueAttributes](#) (int dataset, const [DataValueAttributes](#) &a)  
*Set the [DataValueAttributes](#) for the given dataset.*
- void [setDataValueAttributes](#) (const QModelIndex &index, const [DataValueAttributes](#) &a)  
*Set the [DataValueAttributes](#) for the given index.*
- void [setHidden](#) (bool hidden)  
*Hide (or unhide, resp.*
- void [setHidden](#) (int column, bool hidden)  
*Hide (or unhide, resp.*
- void [setHidden](#) (const QModelIndex &index, bool hidden)  
*Hide (or unhide, resp.*
- void [setModel](#) (QAbstractItemModel \*model)  
*Associate a model with the diagram.*
- void [setPen](#) (const QPen &pen)  
*Set the pen to be used, for painting all datasets in the model.*
- void [setPen](#) (int dataset, const QPen &pen)  
*Set the pen to be used, for painting the given dataset.*
- void [setPen](#) (const QModelIndex &index, const QPen &pen)  
*Set the pen to be used, for painting the datapoint at the given index.*
- void [setPercentMode](#) (bool percent)
- virtual void [setReferenceDiagram](#) ([AbstractCartesianDiagram](#) \*diagram, const QPointF &offset=QPointF())  
*Makes this diagram use another diagram diagram as reference diagram with relative offset offset.*
- void [setRootIndex](#) (const QModelIndex &index)  
*Set the root index in the model, where the diagram starts referencing data for display.*



- virtual void [setSelection](#) (const QRect &rect, QItemSelectionModel::SelectionFlags command)  
*[reimplemented]*
- void [setUnitPrefix](#) (const QString &prefix, Qt::Orientation orientation)  
*Sets the unit prefix for all values.*
- void [setUnitPrefix](#) (const QString &prefix, int column, Qt::Orientation orientation)  
*Sets the unit prefix for one value.*
- void [setUnitSuffix](#) (const QString &suffix, Qt::Orientation orientation)  
*Sets the unit suffix for all values.*
- void [setUnitSuffix](#) (const QString &suffix, int column, Qt::Orientation orientation)  
*Sets the unit suffix for one value.*
- virtual void [takeAxis](#) ([CartesianAxis](#) \*axis)  
*Removes the axis from the diagram, without deleting it.*
- QString [unitPrefix](#) (Qt::Orientation orientation) const  
*Returns the global unit prefix.*
- QString [unitPrefix](#) (int column, Qt::Orientation orientation, bool fallback=false) const  
*Returns the unit prefix for a special value.*
- QString [unitSuffix](#) (Qt::Orientation orientation) const  
*Returns the global unit suffix.*
- QString [unitSuffix](#) (int column, Qt::Orientation orientation, bool fallback=false) const  
*Returns the unit suffix for a special value.*
- void [update](#) () const
- void [useDefaultColors](#) ()  
*Set the palette to be used, for painting datasets to the default palette.*
- void [useRainbowColors](#) ()  
*Set the palette to be used, for painting datasets to the rainbow palette.*
- virtual bool [usesExternalAttributesModel](#) () const  
*Returns whether the diagram is using its own built-in attributes model or an attributes model that was set via [setAttributesModel](#).*
- void [useSubduedColors](#) ()  
*Set the palette to be used, for painting datasets to the subdued palette.*
- virtual int [verticalOffset](#) () const  
*[reimplemented]*
- virtual QRect [visualRect](#) (const QModelIndex &index) const  
*[reimplemented]*

- virtual QRegion [visualRegionForSelection](#) (const QItemSelection &selection) const  
*[reimplemented]*
- virtual [~AbstractCartesianDiagram](#) ()

## Protected Member Functions

- QModelIndex [attributesModelRootIndex](#) () const
- virtual const QPair< QPointF, QPointF > [calculateDataBoundaries](#) () const=0
- virtual bool [checkInvariants](#) (bool justReturnTheStatus=false) const
- QModelIndex [columnToIndex](#) (int column) const
- virtual void [paintDataValueTexts](#) (QPainter \*painter)
- virtual void [paintMarkers](#) (QPainter \*painter)
- void [setAttributesModelRootIndex](#) (const QModelIndex &)
- void [setDataBoundariesDirty](#) () const
- virtual double [threeDItemDepth](#) (int column) const=0

### Returns:

*the 3D item depth of the data set column*

- virtual double [threeDItemDepth](#) (const QModelIndex &index) const =0

### Returns:

*the 3D item depth of the model index index*

- double [valueForCell](#) (int row, int column) const  
*Helper method, retrieving the data value (DisplayRole) for a given row and column.*

## 9.5.2 Constructor & Destructor Documentation

### 9.5.2.1 AbstractCartesianDiagram::AbstractCartesianDiagram (QWidget \*parent = 0, CartesianCoordinatePlane \*plane = 0) [explicit]

Definition at line 70 of file KDChartAbstractCartesianDiagram.cpp.

```
71      : AbstractDiagram ( new Private(), parent, plane )
72 {
73     init();
74 }
```

### 9.5.2.2 KDChart::AbstractCartesianDiagram::~~AbstractCartesianDiagram () [virtual]

Definition at line 76 of file KDChartAbstractCartesianDiagram.cpp.

References [d](#), and [KDChart::AbstractAxis::deleteObserver\(\)](#).

```
77 {
78     Q_FOREACH( CartesianAxis* axis, d->axesList ) {
79         axis->deleteObserver( this );
80     }
81     d->axesList.clear();
82 }
```

### 9.5.3 Member Function Documentation

#### 9.5.3.1 void AbstractCartesianDiagram::addAxis (CartesianAxis \* axis) [virtual]

Add the axis to the diagram.

The diagram takes ownership of the axis and will delete it.

To gain back ownership (e.g. for assigning the axis to another diagram) use the takeAxis method, before calling addAxis on the other diagram.

**See also:**

[takeAxis](#)

Definition at line 91 of file KDChartAbstractCartesianDiagram.cpp.

References [d](#), and [layoutPlanes\(\)](#).

```
92 {
93     if ( !d->axesList.contains( axis ) ) {
94         d->axesList.append( axis );
95         axis->createObserver( this );
96         layoutPlanes();
97     }
98 }
```

#### 9.5.3.2 bool AbstractDiagram::allowOverlappingDataValueTexts () const [inherited]

**Returns:**

Whether data value labels are allowed to overlap.

Definition at line 440 of file KDChartAbstractDiagram.cpp.

References [d](#).

Referenced by [KDChart::AbstractDiagram::compare\(\)](#).

```
441 {
442     return d->allowOverlappingDataValueTexts;
443 }
```

#### 9.5.3.3 bool AbstractDiagram::antiAliasing () const [inherited]

**Returns:**

Whether anti-aliasing is to be used for rendering this diagram.

Definition at line 451 of file KDChartAbstractDiagram.cpp.

References [d](#).

Referenced by [KDChart::AbstractDiagram::compare\(\)](#).

```
452 {
453     return d->antiAliasing;
454 }
```

#### 9.5.3.4 **AttributesModel** \* **AbstractDiagram::attributesModel () const** [virtual, inherited]

Returns the [AttributesModel](#), that is used by this diagram.

By default each diagram owns its own [AttributesModel](#), which should never be deleted. Only if a user-supplied [AttributesModel](#) has been set does the pointer returned here not belong to the diagram.

##### Returns:

The [AttributesModel](#) associated with the diagram.

##### See also:

[setAttributesModel](#)

Definition at line 280 of file `KDChartAbstractDiagram.cpp`.

References d.

Referenced by `KDChart::AbstractDiagram::brush()`, `KDChart::AbstractDiagram::compare()`, `KDChart::AbstractDiagram::datasetBrushes()`, `KDChart::AbstractDiagram::datasetLabels()`, `KDChart::AbstractDiagram::datasetMarkers()`, `KDChart::AbstractDiagram::datasetPens()`, `KDChart::AbstractDiagram::dataValueAttributes()`, `KDChart::AbstractDiagram::isHidden()`, `KDChart::AbstractDiagram::itemRowLabels()`, `KDChart::RingDiagram::paint()`, `KDChart::PolarDiagram::paint()`, `KDChart::AbstractDiagram::pen()`, `setAttributesModel()`, `KDChart::BarDiagram::setBarAttributes()`, `KDChart::AbstractDiagram::setBrush()`, `setModel()`, `KDChart::AbstractDiagram::setPen()`, and `KDChart::TernaryLineDiagram::TernaryLineDiagram()`.

```
281 {
282     return d->attributesModel;
283 }
```

#### 9.5.3.5 **QModelIndex** **AbstractDiagram::attributesModelRootIndex () const** [protected, inherited]

returns a `QModelIndex` pointing into the [AttributesModel](#) that corresponds to the root index of the diagram.

Definition at line 302 of file `KDChartAbstractDiagram.cpp`.

References d.

Referenced by `KDChart::AbstractDiagram::datasetBrushes()`, `KDChart::AbstractDiagram::datasetLabels()`, `KDChart::AbstractDiagram::datasetMarkers()`, `KDChart::AbstractDiagram::datasetPens()`, `KDChart::AbstractDiagram::itemRowLabels()`, `KDChart::Plotter::numberOfAbscissaSegments()`, `KDChart::LineDiagram::numberOfAbscissaSegments()`, `KDChart::BarDiagram::numberOfAbscissaSegments()`, `KDChart::Plotter::numberOfOrdinateSegments()`, `KDChart::LineDiagram::numberOfOrdinateSegments()`, `KDChart::BarDiagram::numberOfOrdinateSegments()`, `KDChart::AbstractDiagram::valueForCell()`, and `KDChart::LineDiagram::valueForCellTesting()`.

```
303 {
304     if ( !d->attributesModelRootIndex.isValid() )
305         d->attributesModelRootIndex = d->attributesModel->mapFromSource( rootIndex() );
306     return d->attributesModelRootIndex;
307 }
```

**9.5.3.6 KDChart::CartesianAxisList AbstractCartesianDiagram::axes () const** [virtual]**Returns:**

a list of all axes added to the diagram

Definition at line 110 of file KDChartAbstractCartesianDiagram.cpp.

References d.

Referenced by KDChart::Widget::setType(), and KDChart::CartesianCoordinatePlane::sharedAxisMasterPlane().

```
111 {
112     return d->axesList;
113 }
```

**9.5.3.7 QBrush AbstractDiagram::brush (const QModelIndex & index) const** [inherited]

Retrieve the brush to be used, for painting the datapoint at the given index in the model.

**Parameters:**

*index* The index of the datapoint in the model.

**Returns:**

The brush to use for painting.

Definition at line 838 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::brush(), KDChart::AttributesModel::data(), KDChart::DatasetBrushRole, and KDChart::AbstractDiagram::datasetDimension().

```
839 {
840     if( datasetDimension() > 1 )
841         return brush( index.column() );
842     return qVariantValue<QBrush>(
843         attributesModel()->data(
844             attributesModel()->mapFromSource( index ),
845             DatasetBrushRole ) );
846 }
```

**9.5.3.8 QBrush AbstractDiagram::brush (int dataset) const** [inherited]

Retrieve the brush to be used for the given dataset.

This will fall back automatically to what was set at model level, if there are no dataset specific settings.

**Parameters:**

*dataset* The dataset to retrieve the brush for.

**Returns:**

The brush to use for painting.

Definition at line 828 of file KDChartAbstractDiagram.cpp.

References [KDChart::AbstractDiagram::attributesModel\(\)](#), [KDChart::AbstractDiagram::columnToIndex\(\)](#), [KDChart::AttributesModel::data\(\)](#), [KDChart::DatasetBrushRole](#), and [KDChart::AbstractDiagram::datasetDimension\(\)](#).

```

829 {
830     if( datasetDimension() > 1 )
831         dataset /= datasetDimension();
832     return qVariantValue<QBrush>(
833         attributesModel()->data(
834             attributesModel()->mapFromSource( columnToIndex( dataset ) ),
835             DatasetBrushRole ) );
836 }
```

### 9.5.3.9 QBrush AbstractDiagram::brush () const [inherited]

Retrieve the brush to be used for painting datapoints globally.

This will fall back automatically to the default settings, if there are no specific settings.

#### Returns:

The brush to use for painting.

Definition at line 822 of file KDChartAbstractDiagram.cpp.

References [KDChart::AbstractDiagram::attributesModel\(\)](#), [KDChart::AttributesModel::data\(\)](#), and [KDChart::DatasetBrushRole](#).

Referenced by [KDChart::AbstractDiagram::brush\(\)](#), [KDChart::TernaryPointDiagram::paint\(\)](#), [KDChart::TernaryLineDiagram::paint\(\)](#), [KDChart::RingDiagram::paint\(\)](#), [KDChart::PolarDiagram::paint\(\)](#), and [KDChart::AbstractDiagram::paintMarker\(\)](#).

```

823 {
824     return qVariantValue<QBrush>(
825         attributesModel()->data( DatasetBrushRole ) );
826 }
```

### 9.5.3.10 virtual const QPair<QPointF, QPointF> KDChart::AbstractDiagram::calculateDataBoundaries () const [protected, pure virtual, inherited]

Implemented in [KDChart::BarDiagram](#), [KDChart::LineDiagram](#), [KDChart::PieDiagram](#), [KDChart::Plotter](#), [KDChart::PolarDiagram](#), [KDChart::RingDiagram](#), [KDChart::AbstractTernaryDiagram](#), [KDChart::TernaryLineDiagram](#), and [KDChart::TernaryPointDiagram](#).

Referenced by [KDChart::AbstractDiagram::dataBoundaries\(\)](#).

### 9.5.3.11 bool AbstractDiagram::checkInvariants (bool *justReturnTheStatus* = false) const [protected, virtual, inherited]

Definition at line 1060 of file KDChartAbstractDiagram.cpp.

References [KDChart::AbstractDiagram::coordinatePlane\(\)](#).

Referenced by KDChart::RingDiagram::calculateDataBoundaries(), KDChart::PolarDiagram::calculateDataBoundaries(), KDChart::Plotter::calculateDataBoundaries(), KDChart::PieDiagram::calculateDataBoundaries(), KDChart::LineDiagram::calculateDataBoundaries(), KDChart::BarDiagram::calculateDataBoundaries(), KDChart::RingDiagram::paint(), KDChart::PolarDiagram::paint(), KDChart::Plotter::paint(), KDChart::PieDiagram::paint(), KDChart::LineDiagram::paint(), KDChart::BarDiagram::paint(), KDChart::AbstractDiagram::paintDataValueTexts(), KDChart::AbstractDiagram::paintMarker(), and KDChart::AbstractDiagram::paintMarkers().

```

1061 {
1062     if( ! justReturnTheStatus ){
1063         Q_ASSERT_X ( model(), "AbstractDiagram::checkInvariants()",
1064                     "There is no usable model set, for the diagram." );
1065
1066         Q_ASSERT_X ( coordinatePlane(), "AbstractDiagram::checkInvariants()",
1067                     "There is no usable coordinate plane set, for the diagram." );
1068     }
1069     return model() && coordinatePlane();
1070 }

```

#### 9.5.3.12 QModelIndex AbstractDiagram::columnToIndex (int *column*) const [protected, inherited]

Definition at line 309 of file KDChartAbstractDiagram.cpp.

Referenced by KDChart::BarDiagram::barAttributes(), KDChart::AbstractDiagram::brush(), KDChart::AbstractDiagram::dataValueAttributes(), KDChart::AbstractDiagram::isHidden(), KDChart::Plotter::lineAttributes(), KDChart::LineDiagram::lineAttributes(), KDChart::AbstractDiagram::pen(), KDChart::AbstractPieDiagram::pieAttributes(), KDChart::BarDiagram::threeDBarAttributes(), KDChart::Plotter::threeDLineAttributes(), KDChart::LineDiagram::threeDLineAttributes(), and KDChart::AbstractPieDiagram::threeDPieAttributes().

```

310 { // FIXME (Mirko): shouldn't this be headerData? instead of the index for the first row?
311     if( model() )
312         return QModelIndex( model()->index( 0, column, rootIndex() ) );
313     return QModelIndex();
314 }

```

#### 9.5.3.13 bool AbstractDiagram::compare (const [AbstractDiagram](#) \* *other*) const [inherited]

Returns true if both diagrams have the same settings.

Definition at line 129 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::allowOverlappingDataValueTexts(), KDChart::AbstractDiagram::antiAliasing(), KDChart::AbstractDiagram::attributesModel(), KDChart::AttributesModel::compare(), KDChart::AbstractDiagram::datasetDimension(), and KDChart::AbstractDiagram::percentMode().

```

130 {
131     if( other == this ) return true;
132     if( ! other ){
133         //qDebug() << "AbstractDiagram::compare() cannot compare to Null pointer";
134         return false;
135     }
136     /*
137     qDebug() << "\n                AbstractDiagram::compare() QAbstractScrollArea:";

```

```

138         // compare QAbstractScrollArea properties
139     qDebug() <<
140         ((horizontalScrollBarPolicy() == other->horizontalScrollBarPolicy()) &&
141         (verticalScrollBarPolicy() == other->verticalScrollBarPolicy()));
142     qDebug() << "AbstractDiagram::compare() QFrame:";
143     // compare QFrame properties
144     qDebug() <<
145         ((frameShadow() == other->frameShadow()) &&
146         (frameShape() == other->frameShape()) &&
147         (frameWidth() == other->frameWidth()) &&
148         (lineWidth() == other->lineWidth()) &&
149         (midLineWidth() == other->midLineWidth()));
150     qDebug() << "AbstractDiagram::compare() QAbstractItemView:";
151     // compare QAbstractItemView properties
152     qDebug() <<
153         ((alternatingRowColors() == other->alternatingRowColors()) &&
154         (hasAutoScroll() == other->hasAutoScroll()) &&
155     #if QT_VERSION > 0x040199
156         (dragDropMode() == other->dragDropMode()) &&
157         (dragDropOverwriteMode() == other->dragDropOverwriteMode()) &&
158         (horizontalScrollMode() == other->horizontalScrollMode()) &&
159         (verticalScrollMode() == other->verticalScrollMode()) &&
160     #endif
161         (dragEnabled() == other->dragEnabled()) &&
162         (editTriggers() == other->editTriggers()) &&
163         (iconSize() == other->iconSize()) &&
164         (selectionBehavior() == other->selectionBehavior()) &&
165         (selectionMode() == other->selectionMode()) &&
166         (showDropIndicator() == other->showDropIndicator()) &&
167         (tabKeyNavigation() == other->tabKeyNavigation()) &&
168         (textElideMode() == other->textElideMode()));
169     qDebug() << "AbstractDiagram::compare() AttributesModel: ";
170     // compare all of the properties stored in the attributes model
171     qDebug() << attributesModel()->compare( other->attributesModel() );
172     qDebug() << "AbstractDiagram::compare() own:";
173     // compare own properties
174     qDebug() <<
175         ((rootIndex().column() == other->rootIndex().column()) &&
176         (rootIndex().row() == other->rootIndex().row()) &&
177         (allowOverlappingDataValueTexts() == other->allowOverlappingDataValueTexts()) &&
178         (antiAliasing() == other->antiAliasing()) &&
179         (percentMode() == other->percentMode()) &&
180         (datasetDimension() == other->datasetDimension()));
181     */
182     return // compare QAbstractScrollArea properties
183         (horizontalScrollBarPolicy() == other->horizontalScrollBarPolicy()) &&
184         (verticalScrollBarPolicy() == other->verticalScrollBarPolicy()) &&
185         // compare QFrame properties
186         (frameShadow() == other->frameShadow()) &&
187         (frameShape() == other->frameShape()) &&
188         // frameWidth is a read-only property defined by the style, it should not be in here:
189         // (frameWidth() == other->frameWidth()) &&
190         (lineWidth() == other->lineWidth()) &&
191         (midLineWidth() == other->midLineWidth()) &&
192         // compare QAbstractItemView properties
193         (alternatingRowColors() == other->alternatingRowColors()) &&
194         (hasAutoScroll() == other->hasAutoScroll()) &&
195     #if QT_VERSION > 0x040199
196         (dragDropMode() == other->dragDropMode()) &&
197         (dragDropOverwriteMode() == other->dragDropOverwriteMode()) &&
198         (horizontalScrollMode() == other->horizontalScrollMode()) &&
199         (verticalScrollMode() == other->verticalScrollMode()) &&
200     #endif
201         (dragEnabled() == other->dragEnabled()) &&
202         (editTriggers() == other->editTriggers()) &&
203         (iconSize() == other->iconSize()) &&
204         (selectionBehavior() == other->selectionBehavior()) &&

```



```

205         (selectionMode() == other->selectionMode()) &&
206         (showDropIndicator() == other->showDropIndicator()) &&
207         (tabKeyNavigation() == other->tabKeyNavigation()) &&
208         (textElideMode() == other->textElideMode()) &&
209         // compare all of the properties stored in the attributes model
210         attributesModel()->compare( other->attributesModel() ) &&
211         // compare own properties
212         (rootIndex().column() == other->rootIndex().column()) &&
213         (rootIndex().row() == other->rootIndex().row()) &&
214         (allowOverlappingDataValueTexts() == other->allowOverlappingDataValueTexts()) &&
215         (antiAliasing() == other->antiAliasing()) &&
216         (percentMode() == other->percentMode()) &&
217         (datasetDimension() == other->datasetDimension());
218 }

```

#### 9.5.3.14 bool AbstractCartesianDiagram::compare (const [AbstractCartesianDiagram](#) \* other) const

Returns true if both diagrams have the same settings.

Definition at line 46 of file KDChartAbstractCartesianDiagram.cpp.

References [referenceDiagram\(\)](#), and [referenceDiagramOffset\(\)](#).

```

47 {
48     if( other == this ) return true;
49     if( ! other ){
50         //qDebug() << "AbstractCartesianDiagram::compare() cannot compare to Null pointer";
51         return false;
52     }
53     /*
54     qDebug() << "\n          AbstractCartesianDiagram::compare() : ";
55     // compare own properties
56     qDebug() <<
57         ((referenceDiagram() == other->referenceDiagram()) &&
58          (!! referenceDiagram()) || (referenceDiagramOffset() == other->referenceDiagramOffset()));
59     */
60     return // compare the base class
61         ( static_cast<const AbstractDiagram*>(this)->compare( other ) ) &&
62         // compare own properties
63         (referenceDiagram() == other->referenceDiagram()) &&
64         (!! referenceDiagram()) || (referenceDiagramOffset() == other->referenceDiagramOffset());
65 }

```

#### 9.5.3.15 [AbstractCoordinatePlane](#) \* AbstractDiagram::coordinatePlane () const [inherited]

The coordinate plane associated with the diagram.

This determines how coordinates in value space are mapped into pixel space. By default this is a [Cartesian-CoordinatePlane](#).

##### Returns:

The coordinate plane associated with the diagram.

Definition at line 220 of file KDChartAbstractDiagram.cpp.

References [d](#).

Referenced by [KDChart::AbstractDiagram::checkInvariants\(\)](#), [layoutPlanes\(\)](#), [KDChart::Polar-Diagram::paint\(\)](#), [KDChart::AbstractDiagram::paintDataValueTexts\(\)](#), [KDChart::AbstractDiagram::paintMarkers\(\)](#), [KDChart::AbstractPolarDiagram::polarCoordinatePlane\(\)](#), and [setCoordinatePlane\(\)](#).

```

221 {
222     return d->plane;
223 }

```

#### 9.5.3.16 **const QPair< QPointF, QPointF > AbstractDiagram::dataBoundaries () const** [inherited]

Return the bottom left and top right data point, that the diagram will display (unless the grid adjusts these values).

This method returns a cached result of calculations done by `calculateDataBoundaries`. Classes derived from [AbstractDiagram](#) must implement the `calculateDataBoundaries` function, to specify their own way of calculating the data boundaries. If derived classes want to force recalculation of the data boundaries, they can call [setDataBoundariesDirty\(\)](#)

Returned value is in diagram coordinates.

Definition at line 225 of file `KDChartAbstractDiagram.cpp`.

References `KDChart::AbstractDiagram::calculateDataBoundaries()`, and `d`.

Referenced by `KDChart::CartesianCoordinatePlane::getRawDataBoundingRectFromDiagrams()`, `KDChart::PolarCoordinatePlane::layoutDiagrams()`, `KDChart::Plotter::paint()`, `KDChart::LineDiagram::paint()`, and `KDChart::BarDiagram::paint()`.

```

226 {
227     if( d->databoundariesDirty ){
228         d->databoundaries = calculateDataBoundaries ();
229         d->databoundariesDirty = false;
230     }
231     return d->databoundaries;
232 }

```

#### 9.5.3.17 **void AbstractDiagram::dataChanged (const QModelIndex & *topLeft*, const QModelIndex & *bottomRight*)** [virtual, inherited]

[reimplemented]

Definition at line 330 of file `KDChartAbstractDiagram.cpp`.

References `KDChart::AbstractDiagram::setDataBoundariesDirty()`.

```

332 {
333     Q_UNUSED( topLeft );
334     Q_UNUSED( bottomRight );
335     // We are still too dumb to do intelligent updates...
336     setDataBoundariesDirty();
337     scheduleDelayedItemsLayout();
338 }

```

#### 9.5.3.18 **void KDChart::AbstractDiagram::dataHidden ()** [signal, inherited]

This signal is emitted, when the hidden status of at least one data cell was (un)set.

Referenced by `KDChart::AbstractDiagram::setHidden()`.

**9.5.3.19 QList< QBrush > AbstractDiagram::datasetBrushes () const** [inherited]

The set of dataset brushes currently used, for use in legends, etc.

**Note:**

Cell-level override brushes, if set, take precedence over the dataset values, so you might need to check these too, in order to find the brush, that is used for a single cell.

**Returns:**

The current set of dataset brushes.

Definition at line 1018 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::attributesModelRootIndex(), KDChart::AttributesModel::columnCount(), KDChart::DatasetBrushRole, and KDChart::AbstractDiagram::datasetDimension().

Referenced by KDChart::Legend::setBrushesFromDiagram().

```

1019 {
1020     QList<QBrush> ret;
1021     if( model() == 0 )
1022         return ret;
1023
1024     const int columnCount = attributesModel()->columnCount(attributesModelRootIndex());
1025     for( int i = 0; i < columnCount / datasetDimension(); ++i )
1026         ret << qVariantValue<QBrush>( attributesModel()->headerData( i, Qt::Vertical, DatasetBrushRole ));
1027
1028     return ret;
1029 }
```

**9.5.3.20 int AbstractDiagram::datasetDimension () const** [inherited]

The dataset dimension of a diagram determines, how many value dimensions it expects each datapoint to have.

For each dimension it will expect one column of values in the model. If the dimensionality is 1, automatic values will be used for the abscissa.

For example a diagram with the default dimension of 1, will have one column per datapoint (the y values) and will use automatic values for the x axis (1, 2, 3, ... n). If the dimension is 2, the diagram will use the first, (and the third, fifth, etc) columns as X values, and the second, (and the fourth, sixth, etc) column as Y values.

**Returns:**

The dataset dimension of the diagram.

Definition at line 1072 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::brush(), KDChart::AbstractDiagram::compare(), KDChart::AbstractDiagram::datasetBrushes(), KDChart::AbstractDiagram::datasetLabels(), KDChart::AbstractDiagram::datasetMarkers(), KDChart::AbstractDiagram::datasetPens(), KDChart::LineDiagram::getCellValues(), KDChart::CartesianCoordinatePlane::getDataDimensionsList(),

KDChart::TernaryPointDiagram::paint(), KDChart::TernaryLineDiagram::paint(), KDChart::AbstractDiagram::paintDataValueTexts(), KDChart::AbstractDiagram::paintMarkers(), KDChart::AbstractDiagram::pen(), KDChart::AbstractDiagram::setPen(), KDChart::Plotter::setType(), and KDChart::LineDiagram::setType().

```
1073 {
1074     return d->datasetDimension;
1075 }
```

#### 9.5.3.21 QStringList AbstractDiagram::datasetLabels () const [inherited]

The set of dataset labels currently displayed, for use in legends, etc.

##### Returns:

The set of dataset labels currently displayed.

Definition at line 1005 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::attributesModelRootIndex(), KDChart::AttributesModel::columnCount(), KDChart::AbstractDiagram::datasetDimension(), and KDChart::AttributesModel::headerData().

```
1006 {
1007     QStringList ret;
1008     if( model() == 0 )
1009         return ret;
1010
1011     const int columnCount = attributesModel()->columnCount(attributesModelRootIndex());
1012     for( int i = 0; i < columnCount / datasetDimension(); ++i )
1013         ret << attributesModel()->headerData( i, Qt::Horizontal, Qt::DisplayRole ).toString();
1014
1015     return ret;
1016 }
```

#### 9.5.3.22 QList< [MarkerAttributes](#) > AbstractDiagram::datasetMarkers () const [inherited]

The set of dataset markers currently used, for use in legends, etc.

##### Note:

Cell-level override markers, if set, take precedence over the dataset values, so you might need to check these too, in order to find the marker, that is shown for a single cell.

##### Returns:

The current set of dataset brushes.

Definition at line 1044 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::attributesModelRootIndex(), KDChart::AttributesModel::columnCount(), KDChart::AbstractDiagram::datasetDimension(), and KDChart::DataValueLabelAttributesRole.

```

1045 {
1046     QList<MarkerAttributes> ret;
1047     if( model() == 0 )
1048         return ret;
1049
1050     const int columnCount = attributesModel()->columnCount(attributesModelRootIndex());
1051     for( int i = 0; i < columnCount / datasetDimension(); ++i )
1052     {
1053         const DataValueAttributes a =
1054             qVariantValue<DataValueAttributes>( attributesModel()->headerData( i, Qt::Vertical, DataV
1055         ret << a.markerAttributes();
1056     }
1057     return ret;
1058 }

```

#### 9.5.3.23 `QList< QPen > AbstractDiagram::datasetPens () const` [inherited]

The set of dataset pens currently used, for use in legends, etc.

##### Note:

Cell-level override pens, if set, take precedence over the dataset values, so you might need to check these too, in order to find the pens, that is used for a single cell.

##### Returns:

The current set of dataset pens.

Definition at line 1031 of file KDChartAbstractDiagram.cpp.

References `KDChart::AbstractDiagram::attributesModel()`, `KDChart::AbstractDiagram::attributesModelRootIndex()`, `KDChart::AttributesModel::columnCount()`, `KDChart::AbstractDiagram::datasetDimension()`, and `KDChart::DatasetPenRole`.

```

1032 {
1033     QList<QPen> ret;
1034     if( model() == 0 )
1035         return ret;
1036
1037     const int columnCount = attributesModel()->columnCount(attributesModelRootIndex());
1038     for( int i = 0; i < columnCount / datasetDimension(); ++i )
1039         ret << qVariantValue<QPen>( attributesModel()->headerData( i, Qt::Vertical, DatasetPenRole )
1040
1041     return ret;
1042 }

```

#### 9.5.3.24 `DataValueAttributes AbstractDiagram::dataValueAttributes (const QModelIndex & index) const` [inherited]

Retrieve the `DataValueAttributes` for the given index.

This will fall back automatically to what was set at dataset or model level, if there are no datapoint specific settings.

##### Parameters:

*index* The datapoint to retrieve the attributes for.

**Returns:**

The [DataValueAttributes](#) for the given index.

Definition at line 421 of file KDChartAbstractDiagram.cpp.

References [KDChart::AbstractDiagram::attributesModel\(\)](#), [KDChart::AttributesModel::data\(\)](#), and [KDChart::DataValueLabelAttributesRole](#).

```

422 {
423     return qVariantValue<DataValueAttributes>(
424         attributesModel()->data( attributesModel()->mapFromSource(index),
425                                 KDChart::DataValueLabelAttributesRole ));
426 }
```

### 9.5.3.25 [DataValueAttributes](#) [AbstractDiagram::dataValueAttributes](#) (int *column*) const [inherited]

Retrieve the [DataValueAttributes](#) for the given dataset.

This will fall back automatically to what was set at model level, if there are no dataset specific settings.

**Parameters:**

*column* The dataset to retrieve the attributes for.

**Returns:**

The [DataValueAttributes](#) for the given dataset.

Definition at line 414 of file KDChartAbstractDiagram.cpp.

References [KDChart::AbstractDiagram::attributesModel\(\)](#), [KDChart::AbstractDiagram::columnToIndex\(\)](#), [KDChart::AttributesModel::data\(\)](#), and [KDChart::DataValueLabelAttributesRole](#).

```

415 {
416     return qVariantValue<DataValueAttributes>(
417         attributesModel()->data( attributesModel()->mapFromSource(columnToIndex( column )),
418                                 KDChart::DataValueLabelAttributesRole ));
419 }
```

### 9.5.3.26 [DataValueAttributes](#) [AbstractDiagram::dataValueAttributes](#) () const [inherited]

Retrieve the [DataValueAttributes](#) specified globally.

This will fall back automatically to the default settings, if there are no specific settings.

**Returns:**

The global [DataValueAttributes](#).

Definition at line 408 of file KDChartAbstractDiagram.cpp.

References [KDChart::AbstractDiagram::attributesModel\(\)](#), [KDChart::DataValueLabelAttributesRole](#), and [KDChart::AttributesModel::modelData\(\)](#).

Referenced by [KDChart::AbstractDiagram::paintDataValueText\(\)](#), [KDChart::AbstractDiagram::paintMarker\(\)](#), and [KDChart::TernaryLineDiagram::TernaryLineDiagram\(\)](#).

```
409 {  
410     return qVariantValue<DataValueAttributes>(  
411         attributesModel()->modelData( KDChart::DataValueLabelAttributesRole ) );  
412 }
```

#### 9.5.3.27 void AbstractDiagram::doItemsLayout () [virtual, inherited]

[reimplemented]

Definition at line 321 of file KDChartAbstractDiagram.cpp.

References [d](#), and [KDChart::AbstractDiagram::update\(\)](#).

```
322 {  
323     if ( d->plane ) {  
324         d->plane->layoutDiagrams();  
325         update();  
326     }  
327     QAbstractItemView::doItemsLayout();  
328 }
```

#### 9.5.3.28 int AbstractDiagram::horizontalOffset () const [virtual, inherited]

[reimplemented]

Definition at line 950 of file KDChartAbstractDiagram.cpp.

```
951 { return 0; }
```

#### 9.5.3.29 QModelIndex AbstractDiagram::indexAt (const QPoint & *point*) const [virtual, inherited]

[reimplemented]

Definition at line 1098 of file KDChartAbstractDiagram.cpp.

References [d](#).

```
1099 {  
1100     return d->indexAt( point );  
1101 }
```

#### 9.5.3.30 QModelIndexList AbstractDiagram::indexesAt (const QPoint & *point*) const [inherited]

This method is added alongside with `indexAt` from QAIM, since in `kdchart` multiple indexes can be displayed at the same spot.

Definition at line 1103 of file KDChartAbstractDiagram.cpp.

References [d](#).

```
1104 {  
1105     return d->indexesAt( point );  
1106 }
```

### 9.5.3.31 bool AbstractDiagram::isHidden (const QModelIndex & *index*) const [inherited]

Retrieve the hidden status for the given index.

This will fall back automatically to what was set at dataset or diagram level, if there are no datapoint specific settings.

#### Parameters:

*index* The datapoint to retrieve the hidden status for.

#### Returns:

The hidden status for the given index.

Definition at line 380 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AttributesModel::data(), and KDChart::DataHiddenRole.

```
381 {
382     return qVariantValue<bool>(
383         attributesModel()->data(
384             attributesModel()->mapFromSource(index),
385             DataHiddenRole ) );
386 }
```

### 9.5.3.32 bool AbstractDiagram::isHidden (int *column*) const [inherited]

Retrieve the hidden status for the given dataset.

This will fall back automatically to what was set at diagram level, if there are no dataset specific settings.

#### Parameters:

*column* The dataset to retrieve the hidden status for.

#### Returns:

The hidden status for the given dataset.

Definition at line 373 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::columnToIndex(), KDChart::AttributesModel::data(), and KDChart::DataHiddenRole.

```
374 {
375     return qVariantValue<bool>(
376         attributesModel()->data(
377             attributesModel()->mapFromSource(columnToIndex( column )),
378             DataHiddenRole ) );
379 }
```

### 9.5.3.33 bool AbstractDiagram::isHidden () const [inherited]

Retrieve the hidden status specified globally.

This will fall back automatically to the default settings (= not hidden), if there are no specific settings.



**Returns:**

The global hidden status.

Definition at line 367 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DataHiddenRole, and KDChart::AttributesModel::modelData().

Referenced by KDChart::LineDiagram::valueForCellTesting().

```

368 {
369     return qVariantValue<bool>(
370         attributesModel()->modelData( DataHiddenRole ) );
371 }
```

#### 9.5.3.34 bool AbstractDiagram::isIndexHidden (const QModelIndex & *index*) const [virtual, inherited]

[reimplemented]

Definition at line 956 of file KDChartAbstractDiagram.cpp.

```

957 { return true; }
```

#### 9.5.3.35 QStringList AbstractDiagram::itemRowLabels () const [inherited]

The set of item row labels currently displayed, for use in Abscissa axes, etc.

**Returns:**

The set of item row labels currently displayed.

Definition at line 989 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::attributesModelRootIndex(), KDChart::AttributesModel::rowCount(), KDChart::AbstractDiagram::unitPrefix(), and KDChart::AbstractDiagram::unitSuffix().

```

990 {
991     QStringList ret;
992     if( model() ){
993         //qDebug() << "AbstractDiagram::itemRowLabels(): " << attributesModel()->rowCount(attributesModelRootIndex());
994         const int rowCount = attributesModel()->rowCount(attributesModelRootIndex());
995         for( int i = 0; i < rowCount; ++i ){
996             //qDebug() << "item row label: " << attributesModel()->headerData( i, Qt::Vertical, Qt::DisplayRole );
997             ret << unitPrefix( i, Qt::Horizontal, true ) +
998                 attributesModel()->headerData( i, Qt::Vertical, Qt::DisplayRole ).toString() +
999                 unitSuffix( i, Qt::Horizontal, true );
1000         }
1001     }
1002     return ret;
1003 }
```

### 9.5.3.36 void KDChart::AbstractDiagram::layoutChanged ([AbstractDiagram](#) \*) [signal, inherited]

Diagrams are supposed to emit this signal, when the layout of one of their element changes.

Layouts can change, for example, when axes are added or removed, or when the configuration was changed in a way that the axes or the diagram itself are displayed in a different geometry. Changes in the diagrams coordinate system also result in the [layoutChanged\(\)](#) signal being emitted.

Referenced by [KDChart::AbstractDiagram::setDatasetDimension\(\)](#), [KDChart::AbstractPieDiagram::setPieAttributes\(\)](#), [KDChart::AbstractPieDiagram::setThreeDPieAttributes\(\)](#), [KDChart::Plotter::setType\(\)](#), [KDChart::LineDiagram::setType\(\)](#), and [KDChart::BarDiagram::setType\(\)](#).

### 9.5.3.37 void KDChart::AbstractCartesianDiagram::layoutPlanes () [virtual]

Triggers layouting of all coordinate planes on the current chart.

Normally you don't need to call this method. It's handled automatically for you.

Definition at line 115 of file [KDChartAbstractCartesianDiagram.cpp](#).

References [KDChart::AbstractDiagram::coordinatePlane\(\)](#), and [KDChart::AbstractCoordinatePlane::layoutPlanes\(\)](#).

Referenced by [addAxis\(\)](#), and [takeAxis\(\)](#).

```

116 {
117     //qDebug() << "KDChart::AbstractCartesianDiagram::layoutPlanes()";
118     AbstractCoordinatePlane* plane = coordinatePlane();
119     if( plane ){
120         plane->layoutPlanes();
121         //qDebug() << "KDChart::AbstractCartesianDiagram::layoutPlanes() OK";
122     }
123 }
```

### 9.5.3.38 void KDChart::AbstractDiagram::modelsChanged () [signal, inherited]

This signal is emitted, when either the model or the [AttributesModel](#) is replaced.

Referenced by [KDChart::AbstractDiagram::setAttributesModel\(\)](#), and [KDChart::AbstractDiagram::setModel\(\)](#).

### 9.5.3.39 QModelIndex AbstractDiagram::moveCursor (CursorAction *cursorAction*, Qt::KeyboardModifiers *modifiers*) [virtual, inherited]

[reimplemented]

Definition at line 947 of file [KDChartAbstractDiagram.cpp](#).

```

948 { return QModelIndex(); }
```

### 9.5.3.40 virtual const int KDChart::AbstractCartesianDiagram::numberOfAbscissaSegments () const [pure virtual]

Implemented in [KDChart::BarDiagram](#), [KDChart::LineDiagram](#), and [KDChart::Plotter](#).

**9.5.3.41** `virtual const int KDChart::AbstractCartesianDiagram::numberOfOrdinateSegments () const` [pure virtual]

Implemented in [KDChart::BarDiagram](#), [KDChart::LineDiagram](#), and [KDChart::Plotter](#).

**9.5.3.42** `virtual void KDChart::AbstractDiagram::paint (PaintContext * paintContext)` [pure virtual, inherited]

Draw the diagram contents to the rectangle and painter, that are passed in as part of the paint context.

#### Parameters:

*paintContext* All information needed for painting.

Implemented in [KDChart::BarDiagram](#), [KDChart::LineDiagram](#), [KDChart::PieDiagram](#), [KDChart::Plotter](#), [KDChart::PolarDiagram](#), [KDChart::RingDiagram](#), [KDChart::TernaryLineDiagram](#), and [KDChart::TernaryPointDiagram](#).

**9.5.3.43** `void AbstractDiagram::paintDataValueText (QPainter * painter, const QModelIndex & index, const QPointF & pos, double value)` [inherited]

Definition at line 468 of file `KDChartAbstractDiagram.cpp`.

References `d`, `KDChart::DataValueAttributes::dataLabel()`, `KDChart::AbstractDiagram::dataValueAttributes()`, `KDChart::DataValueAttributes::decimalDigits()`, `KDChart::DataValueAttributes::isVisible()`, `KDChartEnums::MeasureOrientationMinimum`, `KDChart::DataValueAttributes::position()`, `KDChart::DataValueAttributes::prefix()`, `KDChart::DataValueAttributes::suffix()`, and `KDChart::DataValueAttributes::textAttributes()`.

Referenced by `KDChart::RingDiagram::paint()`, `KDChart::PolarDiagram::paint()`, and `KDChart::AbstractDiagram::paintDataValueTexts()`.

```

472 {
473     // paint one data series
474     const DataValueAttributes a( dataValueAttributes(index) );
475     if ( !a.isVisible() ) return;
476
477     // handle decimal digits
478     int decimalDigits = a.decimalDigits();
479     int decimalPos = QString::number( value ).indexOf( QLatin1Char( '.' ) );
480     QString roundedValue;
481     if ( a.dataLabel().isNull() ) {
482         if ( decimalPos > 0 && value != 0 )
483             roundedValue = roundValues( value, decimalPos, decimalDigits );
484         else
485             roundedValue = QString::number( value );
486     } else
487         roundedValue = a.dataLabel();
488     // handle prefix and suffix
489     if ( !a.prefix().isNull() )
490         roundedValue.prepend( a.prefix() );
491
492     if ( !a.suffix().isNull() )
493         roundedValue.append( a.suffix() );
494
495     const TextAttributes ta( a.textAttributes() );
496     // FIXME draw the non-text bits, background, etc
497     if ( ta.isVisible() ) {
498

```

```

499     QPointF pt( pos );
500     /* for debugging:
501     PainterSaver painterSaver( painter );
502     painter->setPen( Qt::black );
503     painter->drawLine( pos - QPointF( 1,1), pos + QPointF( 1,1) );
504     painter->drawLine( pos - QPointF(-1,1), pos + QPointF(-1,1) );
505     */
506
507     QTextDocument doc;
508     if( Qt::mightBeRichText( roundedValue ) )
509         doc.setHtml( roundedValue );
510     else
511         doc.setPlainText( roundedValue );
512
513     const RelativePosition relPos( a.position( value >= 0.0 ) );
514     const Qt::Alignment alignBottomLeft = Qt::AlignBottom | Qt::AlignLeft;
515     const QFont calculatedFont( ta.calculatedFont( d->plane, KDChartEnums::MeasureOrientationMinimum, d->plane ) );
516     const QRectF boundRect( d->cachedFontMetrics( calculatedFont, painter->device() )->boundingRect( roundedValue, calculatedFont ) );
517
518     // To place correctly
519     pt.ry() -= boundRect.height();
520
521     // qDebug() << "calculatedFont's point size:" << calculatedFont.pointSizeF();
522     // adjust the text start point position, if alignment is not Bottom/Left
523     if( (relPos.alignment() & alignBottomLeft) != alignBottomLeft ){
524         if( relPos.alignment() & Qt::AlignRight )
525             pt.rx() -= boundRect.width();
526         else if( relPos.alignment() & Qt::AlignHCenter )
527             pt.rx() -= 0.5 * boundRect.width();
528
529         if( relPos.alignment() & Qt::AlignTop )
530             pt.ry() += boundRect.height();
531         else if( relPos.alignment() & Qt::AlignVCenter )
532             pt.ry() += 0.5 * boundRect.height();
533     }
534
535     // FIXME draw the non-text bits, background, etc
536
537     if ( a.showRepetitiveDataLabels() ||
538         pos.x() <= d->lastX ||
539         d->lastRoundedValue != roundedValue ) {
540         d->lastRoundedValue = roundedValue;
541         d->lastX = pos.x();
542         PainterSaver painterSaver( painter );
543
544         doc.setDefaultFont( calculatedFont );
545         QAbstractTextDocumentLayout::PaintContext context;
546         context.palette = palette();
547         context.palette.setColor( QPalette::Text, ta.pen().color() );
548
549         painter->translate( pt );
550         painter->rotate( ta.rotation() );
551
552         QAbstractTextDocumentLayout* layout = doc.documentLayout();
553         layout->draw( painter, context );
554     }
555 }
556 }

```

#### 9.5.3.44 void AbstractDiagram::paintDataValueTexts (QPainter \*painter) [protected, virtual, inherited]

Definition at line 584 of file KDChartAbstractDiagram.cpp.

References [KDChart::AbstractDiagram::checkInvariants\(\)](#), [KDChart::AbstractDiagram::coordinate-](#)

Plane(), KDChart::AbstractDiagram::datasetDimension(), KDChart::AbstractDiagram::paintDataValue-Text(), and KDChart::AbstractCoordinatePlane::translate().

```

585 {
586     if ( !checkInvariants() ) return;
587     const int rowCount = model()->rowCount(rootIndex());
588     const int columnCount = model()->columnCount(rootIndex());
589     for ( int i=datasetDimension()-1; i<columnCount; i += datasetDimension() ) {
590         for ( int j=0; j< rowCount; ++j ) {
591             const QModelIndex index = model()->index( j, i, rootIndex() );
592             double value = model()->data( index ).toDouble();
593             const QPointF pos = coordinatePlane()->translate( QPointF( j, value ) );
594             paintDataValueText( painter, index, pos, value );
595         }
596     }
597 }

```

#### 9.5.3.45 void AbstractDiagram::paintMarker (QPainter \*painter, const QModelIndex &index, const QPointF &pos) [inherited]

Definition at line 600 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::brush(), KDChart::AbstractDiagram::checkInvariants(), d, KDChart::AbstractDiagram::dataValueAttributes(), KDChart::MarkerAttributes::isVisible(), KDChart::DataValueAttributes::isVisible(), KDChart::DataValueAttributes::markerAttributes(), KDChart::MarkerAttributes::markerColor(), KDChart::MarkerAttributes::markerSize(), KDChart::AbstractDiagram::paintMarker(), and KDChart::MarkerAttributes::pen().

```

603 {
604     if ( !checkInvariants() ) return;
605     DataValueAttributes a = dataValueAttributes(index);
606     if ( !a.isVisible() ) return;
607     const MarkerAttributes &ma = a.markerAttributes();
608     if ( !ma.isVisible() ) return;
609
610     PainterSaver painterSaver( painter );
611     QSizeF maSize( ma.markerSize() );
612     QBrush indexBrush( brush( index ) );
613     QPen indexPen( ma.pen() );
614     if ( ma.markerColor().isValid() )
615         indexBrush.setColor( ma.markerColor() );
616
617     paintMarker( painter, ma, indexBrush, indexPen, pos, maSize );
618
619     // workaround: BC cannot be changed, otherwise we would pass the
620     // index down to next-lower paintMarker function. So far, we
621     // basically save a circle of radius maSize at pos in the
622     // reverseMapper. This means that ^^ this version of paintMarker
623     // needs to be called to reverse-map the marker.
624     d->reverseMapper.addCircle( index.row(), index.column(), pos, 2 * maSize );
625 }

```

#### 9.5.3.46 void AbstractDiagram::paintMarker (QPainter \*painter, const MarkerAttributes &markerAttributes, const QBrush &brush, const QPen &, const QPointF &point, const QSizeF &size) [virtual, inherited]

Definition at line 627 of file KDChartAbstractDiagram.cpp.

References `KDChart::MarkerAttributes::Marker1Pixel`, `KDChart::MarkerAttributes::Marker4Pixels`, `KDChart::MarkerAttributes::MarkerCircle`, `KDChart::MarkerAttributes::MarkerCross`, `KDChart::MarkerAttributes::MarkerDiamond`, `KDChart::MarkerAttributes::MarkerFastCross`, `KDChart::MarkerAttributes::MarkerRing`, `KDChart::MarkerAttributes::MarkerSquare`, and `KDChart::MarkerAttributes::markerStyle()`.

Referenced by `KDChart::TernaryPointDiagram::paint()`, `KDChart::TernaryLineDiagram::paint()`, `KDChart::MarkerLayoutItem::paintIntoRect()`, `KDChart::AbstractDiagram::paintMarker()`, and `KDChart::AbstractDiagram::paintMarkers()`.

```

633 {
634     const QPen oldPen( painter->pen() );
635     // Pen is used to paint 4Pixels - 1 Pixel - Ring and FastCross types.
636     // make sure to use the brush color - see above in those cases.
637     const bool isFourPixels = (markerAttributes.markerStyle() == MarkerAttributes::Marker4Pixels);
638     if( isFourPixels || (markerAttributes.markerStyle() == MarkerAttributes::Marker1Pixel) ){
639         // for high-performance point charts with tiny point markers:
640         painter->setPen( QPen( brush.color().light() ) );
641         if( isFourPixels ){
642             const qreal x = pos.x();
643             const qreal y = pos.y();
644             painter->drawLine( QPointF(x-1.0,y-1.0),
645                               QPointF(x+1.0,y-1.0) );
646             painter->drawLine( QPointF(x-1.0,y),
647                               QPointF(x+1.0,y) );
648             painter->drawLine( QPointF(x-1.0,y+1.0),
649                               QPointF(x+1.0,y+1.0) );
650         }
651         painter->drawPoint( pos );
652     }else{
653         PainterSaver painterSaver( painter );
654         // we only a solid line surrounding the markers
655         QPen painterPen( pen );
656         painterPen.setStyle( Qt::SolidLine );
657         painter->setPen( painterPen );
658         painter->setBrush( brush );
659         painter->setRenderHint ( QPainter::Antialiasing );
660         painter->translate( pos );
661         switch ( markerAttributes.markerStyle() ) {
662             case MarkerAttributes::MarkerCircle:
663                 painter->drawEllipse( QRectF( 0 - maSize.height()/2, 0 - maSize.width()/2,
664                                                maSize.height(), maSize.width() ) );
665                 break;
666             case MarkerAttributes::MarkerSquare:
667                 {
668                     QRectF rect( 0 - maSize.width()/2, 0 - maSize.height()/2,
669                                 maSize.width(), maSize.height() );
670                     painter->drawRect( rect );
671                     break;
672                 }
673             case MarkerAttributes::MarkerDiamond:
674                 {
675                     QVector <QPointF > diamondPoints;
676                     QPointF top, left, bottom, right;
677                     top = QPointF( 0, 0 - maSize.height()/2 );
678                     left = QPointF( 0 - maSize.width()/2, 0 );
679                     bottom = QPointF( 0, maSize.height()/2 );
680                     right = QPointF( maSize.width()/2, 0 );
681                     diamondPoints << top << left << bottom << right;
682                     painter->drawPolygon( diamondPoints );
683                     break;
684                 }
685             // both handled on top of the method:
686             case MarkerAttributes::Marker1Pixel:
687             case MarkerAttributes::Marker4Pixels:
688                 break;

```

```

689         case MarkerAttributes::MarkerRing:
690         {
691             painter->setPen( QPen( brush.color() ) );
692             painter->setBrush( Qt::NoBrush );
693             painter->drawEllipse( QRectF( 0 - maSize.height()/2, 0 - maSize.width()/2,
694                                     maSize.height(), maSize.width() ) );
695             break;
696         }
697         case MarkerAttributes::MarkerCross:
698         {
699             QRectF rect( maSize.width()*-0.5, maSize.height()*-0.2,
700                         maSize.width(), maSize.height()*0.4 );
701             painter->drawRect( rect );
702             rect.setTopLeft(QPointF( maSize.width()*-0.2, maSize.height()*-0.5 ));
703             rect.setSize(QSizeF( maSize.width()*0.4, maSize.height() ));
704             painter->drawRect( rect );
705             break;
706         }
707         case MarkerAttributes::MarkerFastCross:
708         {
709             QPointF left, right, top, bottom;
710             left = QPointF( -maSize.width()/2, 0 );
711             right = QPointF( maSize.width()/2, 0 );
712             top = QPointF( 0, -maSize.height()/2 );
713             bottom= QPointF( 0, maSize.height()/2 );
714             painter->setPen( QPen( brush.color() ) );
715             painter->drawLine( left, right );
716             painter->drawLine( top, bottom );
717             break;
718         }
719         default:
720             Q_ASSERT_X ( false, "paintMarkers()",
721                         "Type item does not match a defined Marker Type." );
722     }
723 }
724 painter->setPen( oldPen );
725 }

```

#### 9.5.3.47 void AbstractDiagram::paintMarkers (QPainter \**painter*) [protected, virtual, inherited]

Definition at line 727 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::checkInvariants(), KDChart::AbstractDiagram::coordinatePlane(), KDChart::AbstractDiagram::datasetDimension(), KDChart::AbstractDiagram::paintMarker(), and KDChart::AbstractCoordinatePlane::translate().

```

728 {
729     if ( !checkInvariants() ) return;
730     const int rowCount = model()->rowCount(rootIndex());
731     const int columnCount = model()->columnCount(rootIndex());
732     for ( int i=datasetDimension()-1; i<columnCount; i += datasetDimension() ) {
733         for ( int j=0; j< rowCount; ++j ) {
734             const QModelIndex index = model()->index( j, i, rootIndex() );
735             double value = model()->data( index ).toDouble();
736             const QPointF pos = coordinatePlane()->translate( QPointF( j, value ) );
737             paintMarker( painter, index, pos );
738         }
739     }
740 }

```

**9.5.3.48 QPen AbstractDiagram::pen (const QModelIndex & *index*) const** [inherited]

Retrieve the pen to be used, for painting the datapoint at the given index in the model.

**Parameters:**

*index* The index of the datapoint in the model.

**Returns:**

The pen to use for painting.

Definition at line 788 of file KDChartAbstractDiagram.cpp.

References `KDChart::AbstractDiagram::attributesModel()`, `KDChart::AttributesModel::data()`, `KDChart::AbstractDiagram::datasetDimension()`, `KDChart::DatasetPenRole`, and `KDChart::AbstractDiagram::pen()`.

```

789 {
790     if( datasetDimension() > 1 )
791         return pen( index.column() );
792     return qVariantValue<QPen>(
793         attributesModel()->data(
794             attributesModel()->mapFromSource( index ),
795             DatasetPenRole ) );
796 }
```

**9.5.3.49 QPen AbstractDiagram::pen (int *dataset*) const** [inherited]

Retrieve the pen to be used for the given dataset.

This will fall back automatically to what was set at model level, if there are no dataset specific settings.

**Parameters:**

*dataset* The dataset to retrieve the pen for.

**Returns:**

The pen to use for painting.

Definition at line 777 of file KDChartAbstractDiagram.cpp.

References `KDChart::AbstractDiagram::attributesModel()`, `KDChart::AbstractDiagram::columnToIndex()`, `KDChart::AttributesModel::data()`, `KDChart::AbstractDiagram::datasetDimension()`, and `KDChart::DatasetPenRole`.

```

778 {
779     if( datasetDimension() > 1 )
780         dataset /= datasetDimension();
781
782     return qVariantValue<QPen>(
783         attributesModel()->data(
784             attributesModel()->mapFromSource( columnToIndex( dataset ) ),
785             DatasetPenRole ) );
786 }
```



**9.5.3.50 QPen AbstractDiagram::pen () const** [inherited]

Retrieve the pen to be used for painting datapoints globally.

This will fall back automatically to the default settings, if there are no specific settings.

**Returns:**

The pen to use for painting.

Definition at line 771 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AttributesModel::data(), and KDChart::DatasetPenRole.

Referenced by KDChart::TernaryPointDiagram::paint(), KDChart::TernaryLineDiagram::paint(), and KDChart::AbstractDiagram::pen().

```
772 {
773     return qVariantValue<QPen>(
774         attributesModel()->data( DatasetPenRole ) );
775 }
```

**9.5.3.51 bool AbstractDiagram::percentMode () const** [inherited]

Definition at line 462 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::compare(), and KDChart::CartesianCoordinatePlane::getDataDimensionsList().

```
463 {
464     return d->percent;
465 }
```

**9.5.3.52 void KDChart::AbstractDiagram::propertiesChanged ()** [signal, inherited]

Emitted upon change of a property of the Diagram.

Referenced by KDChart::Plotter::resetLineAttributes(), KDChart::LineDiagram::resetLineAttributes(), KDChart::AbstractDiagram::setAllowOverlappingDataValueTexts(), KDChart::AbstractDiagram::setAntiAliasing(), KDChart::BarDiagram::setBarAttributes(), KDChart::AbstractDiagram::setBrush(), KDChart::AbstractDiagram::setDataValueAttributes(), KDChart::Plotter::setLineAttributes(), KDChart::LineDiagram::setLineAttributes(), KDChart::AbstractDiagram::setPen(), KDChart::AbstractDiagram::setPercentMode(), KDChart::BarDiagram::setThreeDBarAttributes(), KDChart::Plotter::setThreeDLineAttributes(), KDChart::LineDiagram::setThreeDLineAttributes(), KDChart::Plotter::setType(), KDChart::LineDiagram::setType(), KDChart::BarDiagram::setType(), KDChart::Plotter::setValueTrackerAttributes(), and KDChart::LineDiagram::setValueTrackerAttributes().

**9.5.3.53 AbstractCartesianDiagram \* AbstractCartesianDiagram::referenceDiagram () const** [virtual]**Returns:**

this diagram's reference diagram

**See also:**

[setReferenceDiagram](#)

Definition at line 148 of file KDChartAbstractCartesianDiagram.cpp.

References d.

Referenced by compare(), and referenceDiagramIsBarDiagram().

```
149 {
150     return d->referenceDiagram;
151 }
```

#### 9.5.3.54 QPointF AbstractCartesianDiagram::referenceDiagramOffset () const [virtual]

**Returns:**

the relative offset of this diagram's reference diagram

**See also:**

[setReferenceDiagram](#)

Definition at line 153 of file KDChartAbstractCartesianDiagram.cpp.

References d.

Referenced by compare().

```
154 {
155     return d->referenceDiagramOffset;
156 }
```

#### 9.5.3.55 virtual void KDChart::AbstractDiagram::resize (const QSizeF & area) [pure virtual, inherited]

Called by the widget's sizeEvent.

Adjust all internal structures, that are calculated, depending on the size of the widget.

**Parameters:**

*area*

Implemented in [KDChart::BarDiagram](#), [KDChart::LineDiagram](#), [KDChart::PieDiagram](#), [KDChart::Plotter](#), [KDChart::PolarDiagram](#), [KDChart::RingDiagram](#), [KDChart::TernaryLineDiagram](#), and [KDChart::TernaryPointDiagram](#).

Referenced by KDChart::CartesianCoordinatePlane::setGeometry().

#### 9.5.3.56 void AbstractDiagram::scrollTo (const QModelIndex & index, ScrollHint hint = EnsureVisible) [virtual, inherited]

[reimplemented]

Definition at line 942 of file KDChartAbstractDiagram.cpp.

```
943 {}
```

**9.5.3.57 void AbstractDiagram::setAllowOverlappingDataValueTexts (bool *allow*)**  
[inherited]

Set whether data value labels are allowed to overlap.

**Parameters:**

*allow* True means that overlapping labels are allowed.

Definition at line 434 of file KDChartAbstractDiagram.cpp.

References `d`, and `KDChart::AbstractDiagram::propertiesChanged()`.

```
435 {  
436     d->allowOverlappingDataValueTexts = allow;  
437     emit propertiesChanged();  
438 }
```

**9.5.3.58 void AbstractDiagram::setAntiAliasing (bool *enabled*)** [inherited]

Set whether anti-aliasing is to be used while rendering this diagram.

**Parameters:**

*enabled* True means that AA is enabled.

Definition at line 445 of file KDChartAbstractDiagram.cpp.

References `d`, and `KDChart::AbstractDiagram::propertiesChanged()`.

```
446 {  
447     d->antiAliasing = enabled;  
448     emit propertiesChanged();  
449 }
```

**9.5.3.59 void AbstractCartesianDiagram::setAttributesModel ([AttributesModel](#) \* *model*)**  
[virtual]

Associate an [AttributesModel](#) with this diagram.

Note that the diagram does `_not_` take ownership of the [AttributesModel](#). This should thus only be used with [AttributesModels](#) that have been explicitly created by the user, and are owned by her. Setting an [AttributesModel](#) that is internal to another diagram is an error.

Correct:

```
AttributesModel *am = new AttributesModel( model, 0 );  
diagram1->setAttributesModel( am );  
diagram2->setAttributesModel( am );
```

Wrong:

```
diagram1->setAttributesModel( diagram2->attributesModel() );
```

**Parameters:**

*model* The [AttributesModel](#) to use for this diagram.

**See also:**

[AttributesModel](#), [usesExternalAttributesModel](#)

Reimplemented from [KDChart::AbstractDiagram](#).

Definition at line 170 of file `KDChartAbstractCartesianDiagram.cpp`.

References `KDChart::AbstractDiagram::attributesModel()`, `d`, and `KDChart::AbstractDiagram::setAttributesModel()`.

```
171 {
172     AbstractDiagram::setAttributesModel( model );
173     d->compressor.setModel( attributesModel() );
174 }
```

### 9.5.3.60 void AbstractDiagram::setAttributesModelRootIndex (const QModelIndex &) [protected, inherited]

Definition at line 293 of file `KDChartAbstractDiagram.cpp`.

References `d`, and `KDChart::AbstractDiagram::setDataBoundariesDirty()`.

Referenced by `KDChart::AbstractDiagram::setRootIndex()`.

```
294 {
295     d->attributesModelRootIndex=idx;
296     setDataBoundariesDirty();
297     scheduleDelayedItemsLayout();
298 }
```

### 9.5.3.61 void AbstractDiagram::setBrush (const QBrush & brush) [inherited]

Set the brush to be used, for painting all datasets in the model.

**Parameters:**

*brush* The brush to use.

Definition at line 806 of file `KDChartAbstractDiagram.cpp`.

References `KDChart::AbstractDiagram::attributesModel()`, `KDChart::DatasetBrushRole`, `KDChart::AbstractDiagram::propertiesChanged()`, and `KDChart::AttributesModel::setModelData()`.

```
807 {
808     attributesModel()->setModelData(
809         qVariantFromValue( brush ), DatasetBrushRole );
810     emit propertiesChanged();
811 }
```

**9.5.3.62 void AbstractDiagram::setBrush (int *dataset*, const QBrush & *brush*)** [inherited]

Set the brush to be used, for painting the given dataset.

**Parameters:**

*dataset* The dataset's column in the model.

*brush* The brush to use.

Definition at line 813 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DatasetBrushRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setHeaderData().

```
814 {
815     attributesModel()->setHeaderData(
816         column, Qt::Vertical,
817         qVariantFromValue( brush ),
818         DatasetBrushRole );
819     emit propertiesChanged();
820 }
```

**9.5.3.63 void AbstractDiagram::setBrush (const QModelIndex & *index*, const QBrush & *brush*)** [inherited]

Set the brush to be used, for painting the datapoint at the given index.

**Parameters:**

*index* The datapoint's index in the model.

*brush* The brush to use.

Definition at line 798 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DatasetBrushRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setData().

```
799 {
800     attributesModel()->setData(
801         attributesModel()->mapFromSource( index ),
802         qVariantFromValue( brush ), DatasetBrushRole );
803     emit propertiesChanged();
804 }
```

**9.5.3.64 void KDChart::AbstractCartesianDiagram::setCoordinatePlane**  
**([AbstractCoordinatePlane](#) \**plane*)** [virtual]

[reimplemented]

Reimplemented from [KDChart::AbstractDiagram](#).

Definition at line 125 of file KDChartAbstractCartesianDiagram.cpp.

References KDChart::AbstractDiagram::coordinatePlane(), and KDChart::AbstractDiagram::setCoordinatePlane().

```

126 {
127     if( coordinatePlane() ) disconnect( coordinatePlane() );
128     AbstractDiagram::setCoordinatePlane(plane);
129
130     // show the axes, after all have been adjusted
131     // (because they might be dependend on each other)
132     /*
133     if( plane )
134         Q_FOREACH( CartesianAxis* axis, d->axesList )
135             axis->show();
136     else
137         Q_FOREACH( CartesianAxis* axis, d->axesList )
138             axis->hide();
139     */
140 }
```

#### 9.5.3.65 void AbstractDiagram::setDataBoundariesDirty () const [protected, inherited]

Definition at line 234 of file KDChartAbstractDiagram.cpp.

References [d](#).

Referenced by [KDChart::AbstractDiagram::dataChanged\(\)](#), [KDChart::Plotter::resize\(\)](#), [KDChart::LineDiagram::resize\(\)](#), [KDChart::BarDiagram::resize\(\)](#), [KDChart::AbstractDiagram::setAttributesModel\(\)](#), [KDChart::AbstractDiagram::setAttributesModelRootIndex\(\)](#), [KDChart::AbstractDiagram::setDatasetDimension\(\)](#), [KDChart::AbstractDiagram::setModel\(\)](#), [KDChart::BarDiagram::setThreeDBarAttributes\(\)](#), [KDChart::Plotter::setThreeDLineAttributes\(\)](#), [KDChart::LineDiagram::setThreeDLineAttributes\(\)](#), [KDChart::Plotter::setType\(\)](#), [KDChart::LineDiagram::setType\(\)](#), and [KDChart::BarDiagram::setType\(\)](#).

```

235 {
236     d->databoundariesDirty = true;
237 }
```

#### 9.5.3.66 void AbstractDiagram::setDatasetDimension (int *dimension*) [inherited]

Sets the dataset dimension of the diagram.

See also:

[datasetDimension](#).

Parameters:

*dimension*

Definition at line 1077 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::AbstractDiagram::layoutChanged\(\)](#), and [KDChart::AbstractDiagram::setDataBoundariesDirty\(\)](#).

Referenced by [KDChart::Widget::setType\(\)](#), [KDChart::TernaryLineDiagram::TernaryLineDiagram\(\)](#), and [KDChart::TernaryPointDiagram::TernaryPointDiagram\(\)](#).

```

1078 {
1079     if ( d->datasetDimension == dimension ) return;
1080     d->datasetDimension = dimension;
1081     setDataBoundariesDirty();
1082     emit layoutChanged( this );
1083 }
```

### 9.5.3.67 void AbstractDiagram::setDataValueAttributes (const [DataValueAttributes](#) & *a*) [inherited]

Set the [DataValueAttributes](#) for all datapoints in the model.

#### Parameters:

*a* The attributes to set.

Definition at line 428 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::DataValueLabelAttributesRole](#), and [KDChart::AbstractDiagram::propertiesChanged\(\)](#).

```
429 {
430     d->attributesModel->setModelData( qVariantFromValue( a ), DataValueLabelAttributesRole );
431     emit propertiesChanged();
432 }
```

### 9.5.3.68 void AbstractDiagram::setDataValueAttributes (int *dataset*, const [DataValueAttributes](#) & *a*) [inherited]

Set the [DataValueAttributes](#) for the given dataset.

#### Parameters:

*dataset* The dataset to set the attributes for.

*a* The attributes to set.

Definition at line 400 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::DataValueLabelAttributesRole](#), and [KDChart::AbstractDiagram::propertiesChanged\(\)](#).

```
401 {
402     d->attributesModel->setHeaderData(
403         column, Qt::Vertical,
404         qVariantFromValue( a ), DataValueLabelAttributesRole );
405     emit propertiesChanged();
406 }
```

### 9.5.3.69 void AbstractDiagram::setDataValueAttributes (const [QModelIndex](#) & *index*, const [DataValueAttributes](#) & *a*) [inherited]

Set the [DataValueAttributes](#) for the given index.

#### Parameters:

*index* The datapoint to set the attributes for.

*a* The attributes to set.

Definition at line 389 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::DataValueLabelAttributesRole](#), and [KDChart::AbstractDiagram::propertiesChanged\(\)](#).

```

391 {
392     d->attributesModel->setData(
393         d->attributesModel->mapFromSource( index ),
394         qVariantFromValue( a ),
395         DataValueLabelAttributesRole );
396     emit propertiesChanged();
397 }

```

#### 9.5.3.70 void AbstractDiagram::setHidden (bool *hidden*) [inherited]

Hide (or unhide, resp.

) all datapoints in the model.

##### Note:

Hidden data are still taken into account by the coordinate plane, so neither the grid nor your axes' ranges will change, when you hide data. For totally removing data from KD Chart's view you can use another approach: e.g. you could define a proxy model on top of your data model, and register the proxy model calling [setModel\(\)](#) instead of registering your real data model.

##### Parameters:

***hidden*** The hidden status to set.

Definition at line 359 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::AbstractDiagram::dataHidden\(\)](#), and [KDChart::DataHiddenRole](#).

```

360 {
361     d->attributesModel->setModelData(
362         qVariantFromValue( hidden ),
363         DataHiddenRole );
364     emit dataHidden();
365 }

```

#### 9.5.3.71 void AbstractDiagram::setHidden (int *column*, bool *hidden*) [inherited]

Hide (or unhide, resp.

) a dataset.

##### Note:

Hidden data are still taken into account by the coordinate plane, so neither the grid nor your axes' ranges will change, when you hide data. For totally removing data from KD Chart's view you can use another approach: e.g. you could define a proxy model on top of your data model, and register the proxy model calling [setModel\(\)](#) instead of registering your real data model.

##### Parameters:

***column*** The dataset to set the hidden status for.

***hidden*** The hidden status to set.

Definition at line 350 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::AbstractDiagram::dataHidden\(\)](#), and [KDChart::DataHiddenRole](#).



```

351 {
352     d->attributesModel->setHeaderData(
353         column, Qt::Vertical,
354         qVariantFromValue( hidden ),
355         DataHiddenRole );
356     emit dataHidden();
357 }

```

#### 9.5.3.72 void AbstractDiagram::setHidden (const QModelIndex & *index*, bool *hidden*) [inherited]

Hide (or unhide, resp.  
) a data cell.

##### Note:

Hidden data are still taken into account by the coordinate plane, so neither the grid nor your axes' ranges will change, when you hide data. For totally removing data from KD Chart's view you can use another approach: e.g. you could define a proxy model on top of your data model, and register the proxy model calling [setModel\(\)](#) instead of registering your real data model.

##### Parameters:

***index*** The datapoint to set the hidden status for.

***hidden*** The hidden status to set.

Definition at line 341 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::AbstractDiagram::dataHidden\(\)](#), and [KDChart::DataHiddenRole](#).

```

342 {
343     d->attributesModel->setData(
344         d->attributesModel->mapFromSource( index ),
345         qVariantFromValue( hidden ),
346         DataHiddenRole );
347     emit dataHidden();
348 }

```

#### 9.5.3.73 void AbstractCartesianDiagram::setModel (QAbstractItemModel \* *model*) [virtual]

Associate a model with the diagram.

Reimplemented from [KDChart::AbstractDiagram](#).

Definition at line 164 of file KDChartAbstractCartesianDiagram.cpp.

References [KDChart::AbstractDiagram::attributesModel\(\)](#), [d](#), and [KDChart::AbstractDiagram::setModel\(\)](#).

```

165 {
166     AbstractDiagram::setModel( model );
167     d->compressor.setModel( attributesModel() );
168 }

```

**9.5.3.74 void AbstractDiagram::setPen (const QPen & *pen*)** [inherited]

Set the pen to be used, for painting all datasets in the model.

**Parameters:**

*pen* The pen to use.

Definition at line 753 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DatasetPenRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setModelData().

```

754 {
755     attributesModel()->setModelData(
756         qVariantFromValue( pen ), DatasetPenRole );
757     emit propertiesChanged();
758 }
```

**9.5.3.75 void AbstractDiagram::setPen (int *dataset*, const QPen & *pen*)** [inherited]

Set the pen to be used, for painting the given dataset.

**Parameters:**

*dataset* The dataset's row in the model.

*pen* The pen to use.

Definition at line 760 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::datasetDimension(), KDChart::DatasetPenRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setHeaderData().

```

761 {
762     if( datasetDimension() > 1 )
763         column *= datasetDimension();
764     attributesModel()->setHeaderData(
765         column, Qt::Vertical,
766         qVariantFromValue( pen ),
767         DatasetPenRole );
768     emit propertiesChanged();
769 }
```

**9.5.3.76 void AbstractDiagram::setPen (const QModelIndex & *index*, const QPen & *pen*)** [inherited]

Set the pen to be used, for painting the datapoint at the given index.

**Parameters:**

*index* The datapoint's index in the model.

*pen* The pen to use.

Definition at line 743 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::datasetDimension(), KDChart::DatasetPenRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setData().

```

744 {
745     if( datasetDimension() > 1 )
746         return setPen( index.column(), pen );
747     attributesModel()->setData(
748         attributesModel()->mapFromSource( index ),
749         qVariantFromValue( pen ), DatasetPenRole );
750     emit propertiesChanged();
751 }
```

#### 9.5.3.77 void AbstractDiagram::setPercentMode (bool *percent*) [inherited]

Definition at line 456 of file KDChartAbstractDiagram.cpp.

References d, and KDChart::AbstractDiagram::propertiesChanged().

Referenced by KDChart::LineDiagram::setType(), and KDChart::BarDiagram::setType().

```

457 {
458     d->percent = percent;
459     emit propertiesChanged();
460 }
```

#### 9.5.3.78 void AbstractCartesianDiagram::setReferenceDiagram ([AbstractCartesianDiagram \\*](#) *diagram*, const QPointF & *offset* = QPointF()) [virtual]

Makes this diagram use another diagram *diagram* as reference diagram with relative offset *offset*.

To share cartesian axes between different diagrams there might be cases when you need that. Normally you don't.

**See also:**

examples/SharedAbscissa

Definition at line 142 of file KDChartAbstractCartesianDiagram.cpp.

References d.

```

143 {
144     d->referenceDiagram = diagram;
145     d->referenceDiagramOffset = offset;
146 }
```

#### 9.5.3.79 void AbstractCartesianDiagram::setRootIndex (const QModelIndex & *index*) [virtual]

Set the root index in the model, where the diagram starts referencing data for display.

[reimplemented]

Reimplemented from [KDChart::AbstractDiagram](#).

Definition at line 158 of file KDChartAbstractCartesianDiagram.cpp.

References [d](#), and [KDChart::AbstractDiagram::setRootIndex\(\)](#).

```
159 {
160     d->compressor.setRootIndex( index );
161     AbstractDiagram::setRootIndex( index );
162 }
```

#### 9.5.3.80 void AbstractDiagram::setSelection (const QRect & *rect*, QItemSelection-Model::SelectionFlags *command*) [virtual, inherited]

[reimplemented]

Definition at line 959 of file KDChartAbstractDiagram.cpp.

References [d](#).

```
960 {
961     const QModelIndexList indexes = d->indexesIn( rect );
962     QItemSelection selection;
963     KDAB_FOREACH( const QModelIndex& index, indexes )
964     {
965         selection.append( QItemSelectionRange( index ) );
966     }
967     selectionModel()->select( selection, command );
968 }
```

#### 9.5.3.81 void AbstractDiagram::setUnitPrefix (const QString & *prefix*, Qt::Orientation *orientation*) [inherited]

Sets the unit prefix for all values.

##### Parameters:

*prefix* the prefix to be set

*orientation* the orientation of the axis to set

Definition at line 864 of file KDChartAbstractDiagram.cpp.

References [d](#).

```
865 {
866     d->unitPrefix[ orientation ] = prefix;
867 }
```

#### 9.5.3.82 void AbstractDiagram::setUnitPrefix (const QString & *prefix*, int *column*, Qt::Orientation *orientation*) [inherited]

Sets the unit prefix for one value.

##### Parameters:

*prefix* the prefix to be set

*column* the value using that prefix

*orientation* the orientation of the axis to set

Definition at line 854 of file KDChartAbstractDiagram.cpp.

References d.

```
855 {  
856     d->unitPrefixMap[ column ][ orientation ]= prefix;  
857 }
```

#### 9.5.3.83 void AbstractDiagram::setUnitSuffix (const QString & *suffix*, Qt::Orientation *orientation*) [inherited]

Sets the unit suffix for all values.

##### Parameters:

*suffix* the suffix to be set

*orientation* the orientation of the axis to set

Definition at line 885 of file KDChartAbstractDiagram.cpp.

References d.

```
886 {  
887     d->unitSuffix[ orientation ] = suffix;  
888 }
```

#### 9.5.3.84 void AbstractDiagram::setUnitSuffix (const QString & *suffix*, int *column*, Qt::Orientation *orientation*) [inherited]

Sets the unit suffix for one value.

##### Parameters:

*suffix* the suffix to be set

*column* the value using that suffix

*orientation* the orientation of the axis to set

Definition at line 875 of file KDChartAbstractDiagram.cpp.

References d.

```
876 {  
877     d->unitSuffixMap[ column ][ orientation ]= suffix;  
878 }
```

### 9.5.3.85 void AbstractCartesianDiagram::takeAxis ([CartesianAxis](#) \* *axis*) [virtual]

Removes the axis from the diagram, without deleting it.

The diagram no longer owns the axis, so it is the caller's responsibility to delete the axis.

See also:

[addAxis](#)

Definition at line 100 of file `KDChartAbstractCartesianDiagram.cpp`.

References `d`, `KDChart::AbstractAxis::deleteObserver()`, `layoutPlanes()`, and `KDChart::AbstractLayoutItem::setParentWidget()`.

Referenced by `KDChart::Widget::setType()`, and `KDChart::CartesianAxis::~~CartesianAxis()`.

```

101 {
102     const int idx = d->axesList.indexOf( axis );
103     if( idx != -1 )
104         d->axesList.takeAt( idx );
105     axis->deleteObserver( this );
106     axis->setParentWidget( 0 );
107     layoutPlanes();
108 }
```

### 9.5.3.86 virtual double KDChart::AbstractCartesianDiagram::threeDItemDepth (int *column*) const [protected, pure virtual]

Returns:

the 3D item depth of the data set *column*

Implemented in [KDChart::BarDiagram](#), [KDChart::LineDiagram](#), and [KDChart::Plotter](#).

### 9.5.3.87 virtual double KDChart::AbstractCartesianDiagram::threeDItemDepth (const QModelIndex & *index*) const [protected, pure virtual]

Returns:

the 3D item depth of the model index *index*

Implemented in [KDChart::BarDiagram](#), [KDChart::LineDiagram](#), and [KDChart::Plotter](#).

### 9.5.3.88 QString AbstractDiagram::unitPrefix (Qt::Orientation *orientation*) const [inherited]

Returns the global unit prefix.

Parameters:

*orientation* the orientation of the axis

Returns:

the unit prefix

Definition at line 908 of file KDChartAbstractDiagram.cpp.

References d.

```
909 {  
910     return d->unitPrefix[ orientation ];  
911 }
```

#### 9.5.3.89 QString AbstractDiagram::unitPrefix (int *column*, Qt::Orientation *orientation*, bool *fallback* = false) const [inherited]

Returns the unit prefix for a special value.

##### Parameters:

*column* the value which's prefix is requested

*orientation* the orientation of the axis

*fallback* if true, the global prefix is return when no specific one is set for that value

##### Returns:

the unit prefix

Definition at line 897 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::itemRowLabels(), and KDChart::CartesianAxis::paintCtx().

```
898 {  
899     if( !fallback || d->unitPrefixMap[ column ].contains( orientation ) )  
900         return d->unitPrefixMap[ column ][ orientation ];  
901     return d->unitPrefix[ orientation ];  
902 }
```

#### 9.5.3.90 QString AbstractDiagram::unitSuffix (Qt::Orientation *orientation*) const [inherited]

Returns the global unit suffix.

##### Parameters:

*orientation* the orientation of the axis

##### Returns:

the unit suffix

Definition at line 931 of file KDChartAbstractDiagram.cpp.

References d.

```
932 {  
933     return d->unitSuffix[ orientation ];  
934 }
```

### 9.5.3.91 QString AbstractDiagram::unitSuffix (int *column*, Qt::Orientation *orientation*, bool *fallback* = false) const [inherited]

Returns the unit suffix for a special value.

#### Parameters:

*column* the value which's suffix is requested

*orientation* the orientation of the axis

*fallback* if true, the global suffix is return when no specific one is set for that value

#### Returns:

the unit suffix

Definition at line 920 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::itemRowLabels(), and KDChart::CartesianAxis::paintCtx().

```

921 {
922     if( !fallback || d->unitSuffixMap[ column ].contains( orientation ) )
923         return d->unitSuffixMap[ column ][ orientation ];
924     return d->unitSuffix[ orientation ];
925 }
```

### 9.5.3.92 void AbstractDiagram::update () const [inherited]

Definition at line 1091 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::doItemsLayout().

```

1092 {
1093     //qDebug("KDChart::AbstractDiagram::update() called");
1094     if( d->plane )
1095         d->plane->update();
1096 }
```

### 9.5.3.93 void KDChart::AbstractDiagram::useDefaultColors () [inherited]

Set the palette to be used, for painting datasets to the default palette.

#### See also:

[KDChart::Palette](#). FIXME: fold into one usePalette ([KDChart::Palette&](#)) method

Definition at line 974 of file KDChartAbstractDiagram.cpp.

References d, and KDChart::AttributesModel::PaletteTypeDefault.

```

975 {
976     d->attributesModel->setPaletteType( AttributesModel::PaletteTypeDefault );
977 }
```



**9.5.3.94 void KDChart::AbstractDiagram::useRainbowColors ()** [inherited]

Set the palette to be used, for painting datasets to the rainbow palette.

**See also:**

[KDChart::Palette](#).

Definition at line 984 of file KDChartAbstractDiagram.cpp.

References [d](#), and [KDChart::AttributesModel::PaletteTypeRainbow](#).

```
985 {  
986     d->attributesModel->setPaletteType( AttributesModel::PaletteTypeRainbow );  
987 }
```

**9.5.3.95 bool AbstractDiagram::usesExternalAttributesModel () const** [virtual, inherited]

Returns whether the diagram is using its own built-in attributes model or an attributes model that was set via [setAttributesModel](#).

**See also:**

[setAttributesModel](#)

Definition at line 274 of file KDChartAbstractDiagram.cpp.

References [d](#).

```
275 {  
276     return d->usesExternalAttributesModel();  
277 }
```

**9.5.3.96 void KDChart::AbstractDiagram::useSubduedColors ()** [inherited]

Set the palette to be used, for painting datasets to the subdued palette.

**See also:**

[KDChart::Palette](#).

Definition at line 979 of file KDChartAbstractDiagram.cpp.

References [d](#), and [KDChart::AttributesModel::PaletteTypeSubdued](#).

```
980 {  
981     d->attributesModel->setPaletteType( AttributesModel::PaletteTypeSubdued );  
982 }
```

### 9.5.3.97 **double AbstractDiagram::valueForCell (int *row*, int *column*) const** [protected, inherited]

Helper method, retrieving the data value (DisplayRole) for a given row and column.

#### Parameters:

***row*** The row to query.

***column*** The column to query.

#### Returns:

The value of the display role at the given row and column as a double.

### Deprecated

Definition at line 1085 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModelRootIndex(), and d.

```
1086 {
1087     return d->attributesModel->data(
1088         d->attributesModel->index( row, column, attributesModelRootIndex() ) ).toDouble();
1089 }
```

### 9.5.3.98 **int AbstractDiagram::verticalOffset () const** [virtual, inherited]

[reimplemented]

Definition at line 953 of file KDChartAbstractDiagram.cpp.

```
954 { return 0; }
```

### 9.5.3.99 **QRect AbstractDiagram::visualRect (const QModelIndex & *index*) const** [virtual, inherited]

[reimplemented]

Definition at line 937 of file KDChartAbstractDiagram.cpp.

```
938 {
939     return QRect();
940 }
```

### 9.5.3.100 **QRegion AbstractDiagram::visualRegionForSelection (const QItemSelection & *selection*) const** [virtual, inherited]

[reimplemented]

Definition at line 970 of file KDChartAbstractDiagram.cpp.

```
971 { return QRegion(); }
```

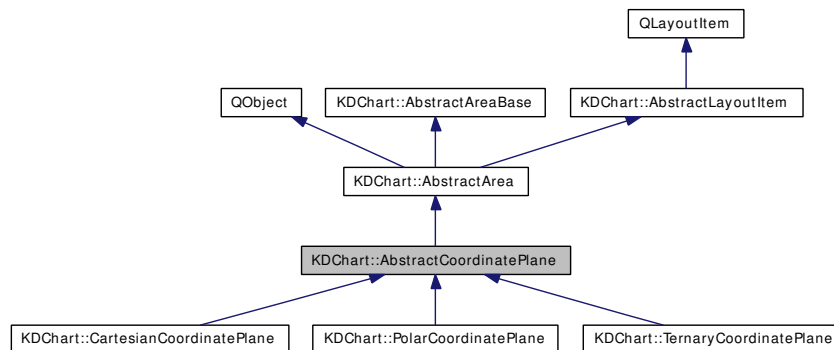
The documentation for this class was generated from the following files:

- [KDChartAbstractCartesianDiagram.h](#)
- [KDChartAbstractCartesianDiagram.cpp](#)

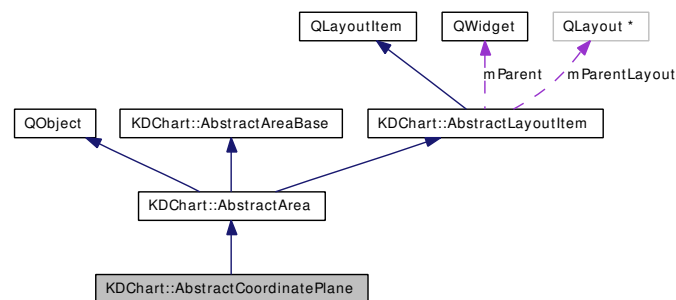
## 9.6 KDChart::AbstractCoordinatePlane Class Reference

```
#include <KDChartAbstractCoordinatePlane.h>
```

Inheritance diagram for KDChart::AbstractCoordinatePlane:



Collaboration diagram for KDChart::AbstractCoordinatePlane:



### 9.6.1 Detailed Description

Base class common for all coordinate planes, [CartesianCoordinatePlane](#), [PolarCoordinatePlane](#), [TernaryCoordinatePlane](#).

Definition at line 47 of file `KDChartAbstractCoordinatePlane.h`.

### Public Types

- enum [AxesCalcMode](#) {  
[Linear](#),  
[Logarithmic](#) }

### Public Slots

- void [layoutPlanes](#) ()  
*Calling [layoutPlanes\(\)](#) on the plane triggers the global `KDChart::Chart::slotLayoutPlanes()`.*

- void [relayout](#) ()  
*Calling [relayout\(\)](#) on the plane triggers the global `KDChart::Chart::slotRelayout()`.*
- void [setGridNeedsRecalculate](#) ()  
*Used by the chart to clear the cached grid data.*
- void [update](#) ()  
*Calling [update\(\)](#) on the plane triggers the global `KDChart::Chart::update()`.*

## Signals

- void [destroyedCoordinatePlane](#) ([AbstractCoordinatePlane](#) \*)  
*Emitted when this coordinate plane is destroyed.*
- void [needLayoutPlanes](#) ()  
*Emitted when plane needs to trigger the Chart's layouting of the coord.*
- void [needRelayout](#) ()  
*Emitted when plane needs to trigger the Chart's layouting.*
- void [needUpdate](#) ()  
*Emitted when plane needs to update its drawings.*
- void [positionChanged](#) ([AbstractArea](#) \*)
- void [propertiesChanged](#) ()  
*Emitted upon change of a property of the Coordinate Plane or any of its components.*

## Public Member Functions

- virtual void [addDiagram](#) ([AbstractDiagram](#) \*diagram)  
*Adds a diagram to this coordinate plane.*
- void [alignToReferencePoint](#) (const [RelativePosition](#) &position)
- [BackgroundAttributes](#) [backgroundAttributes](#) () const
- virtual int [bottomOverlap](#) (bool doNotRecalculate=false) const  
*This is called at layout time by `KDChart::AutoSpacerLayoutItem::sizeHint()`.*
- bool [compare](#) (const [AbstractAreaBase](#) \*other) const  
*Returns true if both areas have the same settings.*
- [AbstractDiagram](#) \* [diagram](#) ()  
**Returns:**  
*The first diagram associated with this coordinate plane.*
- [ConstAbstractDiagramList](#) [diagrams](#) () const  
**Returns:**  
*The list of diagrams associated with this coordinate plane.*

- [AbstractDiagramList diagrams \(\)](#)

**Returns:**

*The list of diagrams associated with this coordinate plane.*

- virtual Qt::Orientations [expandingDirections \(\)](#) const  
*pure virtual in [QLayoutItem](#)*

- [FrameAttributes frameAttributes \(\)](#) const

- virtual QRect [geometry \(\)](#) const  
*pure virtual in [QLayoutItem](#)*

- void [getFrameLeadings](#) (int &left, int &top, int &right, int &bottom) const

- [GridAttributes globalGridAttributes \(\)](#) const

**Returns:**

*The grid attributes used by this coordinate plane.*

- [DataDimensionsList gridDimensionsList \(\)](#)

*Returns the dimensions used for drawing the grid lines.*

- virtual bool [isEmpty \(\)](#) const  
*pure virtual in [QLayoutItem](#)*

- bool [isRubberBandZoomingEnabled \(\)](#) const

**Returns:**

*Whether zooming with a rubber band using the mouse is enabled.*

- const bool [isVisiblePoint](#) (const QPointF &point) const

*Tests, if a point is visible on the coordinate plane.*

- virtual void [layoutDiagrams \(\)=0](#)

*Distribute the available space among the diagrams and axes.*

- virtual int [leftOverlap](#) (bool doNotRecalculate=false) const

*This is called at layout time by [KDChart::AutoSpacerLayoutItem::sizeHint\(\)](#).*

- virtual QSize [maximumSize \(\)](#) const  
*pure virtual in [QLayoutItem](#)*

- virtual QSize [minimumSize \(\)](#) const  
*pure virtual in [QLayoutItem](#)*

- virtual QSize [minimumSizeHint \(\)](#) const  
*[reimplemented]*

- void [mouseDoubleClickEvent](#) (QMouseEvent \*event)
- void [mouseMoveEvent](#) (QMouseEvent \*event)
- void [mousePressEvent](#) (QMouseEvent \*event)
- void [mouseReleaseEvent](#) (QMouseEvent \*event)

- virtual void [paint](#) (QPainter \*)=0
- virtual void [paintAll](#) (QPainter &painter)
 

*Call paintAll, if you want the background and the frame to be drawn before the normal [paint\(\)](#) is invoked automatically.*
- virtual void [paintBackground](#) (QPainter &painter, const QRect &rectangle)
- virtual void [paintCtx](#) (PaintContext \*context)
 

*Default impl: Paint the complete item using its layouted position and size.*
- virtual void [paintFrame](#) (QPainter &painter, const QRect &rectangle)
- virtual void [paintIntoRect](#) (QPainter &painter, const QRect &rect)
 

*Draws the background and frame, then calls [paint\(\)](#).*
- const [Chart](#) \* [parent](#) () const
- [Chart](#) \* [parent](#) ()
- QLayout \* [parentLayout](#) ()
- [AbstractCoordinatePlane](#) \* [referenceCoordinatePlane](#) () const
 

*There are two ways, in which planes can be caused to interact, in where they are put layouting wise: The first is the reference plane.*
- void [removeFromParentLayout](#) ()
- virtual void [replaceDiagram](#) ([AbstractDiagram](#) \*diagram, [AbstractDiagram](#) \*oldDiagram=0)
 

*Replaces the old diagram, or appends the diagram, if there is none yet.*
- virtual int [rightOverlap](#) (bool doNotRecalculate=false) const
 

*This is called at layout time by [KDChart::AutoSpacerLayoutItem::sizeHint\(\)](#).*
- void [setBackgroundAttributes](#) (const [BackgroundAttributes](#) &a)
- void [setFrameAttributes](#) (const [FrameAttributes](#) &a)
- virtual void [setGeometry](#) (const QRect &r)
 

*pure virtual in [QLayoutItem](#)*
- void [setGlobalGridAttributes](#) (const [GridAttributes](#) &)
- void [setParent](#) ([Chart](#) \*parent)
 

*Called internally by [KDChart::Chart](#).*
- void [setParentLayout](#) (QLayout \*lay)
- virtual void [setParentWidget](#) (QWidget \*widget)
 

*Inform the item about its widget: This enables the item, to trigger that widget's update, whenever the size of the item's contents has changed.*
- void [setReferenceCoordinatePlane](#) ([AbstractCoordinatePlane](#) \*plane)
 

*Set another coordinate plane to be used as the reference plane for this one.*
- void [setRubberBandZoomingEnabled](#) (bool enable)
 

*Enables or disables zooming with a rubber band using the mouse.*
- virtual void [setZoomCenter](#) (const QPointF &center)

*Set the point (in value coordinates) to be used as the center point in zoom operations.*

- virtual void [setZoomFactorX](#) (double factor)  
*Sets the zoom factor in horizontal direction, that is applied to all coordinate transformations.*
- virtual void [setZoomFactorY](#) (double factor)  
*Sets the zoom factor in vertical direction, that is applied to all coordinate transformations.*
- virtual [AbstractCoordinatePlane](#) \* [sharedAxisMasterPlane](#) (QPainter \*p=0)
- virtual QSize [sizeHint](#) () const  
*pure virtual in [QLayoutItem](#)*
- virtual void [sizeHintChanged](#) () const  
*Report changed size hint: ask the parent widget to recalculate the layout.*
- virtual QSizePolicy [sizePolicy](#) () const  
*[reimplemented]*
- virtual void [takeDiagram](#) ([AbstractDiagram](#) \*diagram)  
*Removes the diagram from the plane, without deleting it.*
- virtual int [topOverlap](#) (bool doNotRecalculate=false) const  
*This is called at layout time by [KDChart::AutoSpacerLayoutItem::sizeHint\(\)](#).*
- virtual const QPointF [translate](#) (const QPointF &diagramPoint) const=0  
*Translate the given point in value space coordinates to a position in pixel space.*
- virtual QPointF [zoomCenter](#) () const  
**Returns:**  
*The center point (in value coordinates) of the coordinate plane, that is used for zoom operations.*
- virtual double [zoomFactorX](#) () const  
**Returns:**  
*The zoom factor in horizontal direction, that is applied to all coordinate transformations.*
- virtual double [zoomFactorY](#) () const  
**Returns:**  
*The zoom factor in vertical direction, that is applied to all coordinate transformations.*
- virtual ~[AbstractCoordinatePlane](#) ()

## Static Public Member Functions

- static void [paintBackgroundAttributes](#) (QPainter &painter, const QRect &rectangle, const [KDChart::BackgroundAttributes](#) &attributes)
- static void [paintFrameAttributes](#) (QPainter &painter, const QRect &rectangle, const [KDChart::FrameAttributes](#) &attributes)



## Protected Member Functions

- [AbstractCoordinatePlane](#) ([Chart](#) \*parent=0)
- virtual [QRect](#) [areaGeometry](#) () const
- virtual [DataDimensionsList](#) [getDataDimensionsList](#) () const=0
- [QRect](#) [innerRect](#) () const
- virtual void [positionHasChanged](#) ()

## Protected Attributes

- [QWidget](#) \* [mParent](#)
- [QLayout](#) \* [mParentLayout](#)

## 9.6.2 Member Enumeration Documentation

### 9.6.2.1 enum [KDChart::AbstractCoordinatePlane::AxesCalcMode](#)

Enumerator:

*Linear*

*Logarithmic*

Definition at line 57 of file [KDChartAbstractCoordinatePlane.h](#).

```
57 { Linear, Logarithmic };
```

## 9.6.3 Constructor & Destructor Documentation

### 9.6.3.1 [AbstractCoordinatePlane::AbstractCoordinatePlane](#) ([Chart](#) \*parent = 0) [[explicit](#), [protected](#)]

Definition at line 55 of file [KDChartAbstractCoordinatePlane.cpp](#).

References [d](#), and [parent\(\)](#).

```
56     : AbstractArea ( new Private() )
57 {
58     d->parent = parent;
59     d->init();
60 }
```

### 9.6.3.2 [AbstractCoordinatePlane::~~AbstractCoordinatePlane](#) () [[virtual](#)]

Definition at line 62 of file [KDChartAbstractCoordinatePlane.cpp](#).

References [destroyedCoordinatePlane\(\)](#).

```
63 {
64     emit destroyedCoordinatePlane( this );
65 }
```

## 9.6.4 Member Function Documentation

### 9.6.4.1 void AbstractCoordinatePlane::addDiagram ([AbstractDiagram](#) \* *diagram*) [virtual]

Adds a diagram to this coordinate plane.

#### Parameters:

*diagram* The diagram to add.

#### See also:

[replaceDiagram](#), [takeDiagram](#)

Reimplemented in [KDChart::CartesianCoordinatePlane](#), [KDChart::PolarCoordinatePlane](#), and [KDChart::TernaryCoordinatePlane](#).

Definition at line 72 of file `KDChartAbstractCoordinatePlane.cpp`.

References `d`, `diagram()`, `layoutDiagrams()`, `layoutPlanes()`, `KDChart::AbstractDiagram::setCoordinatePlane()`, and `update()`.

Referenced by `KDChart::TernaryCoordinatePlane::addDiagram()`, `KDChart::PolarCoordinatePlane::addDiagram()`, `KDChart::CartesianCoordinatePlane::addDiagram()`, and `replaceDiagram()`.

```

73 {
74     // diagrams are invisible and paint through their paint() method
75     diagram->hide();
76
77     d->diagrams.append( diagram );
78     diagram->setParent( d->parent );
79     diagram->setCoordinatePlane( this );
80     layoutDiagrams();
81     layoutPlanes(); // there might be new axes, etc
82     update();
83 }
```

### 9.6.4.2 void AbstractAreaBase::alignToReferencePoint (const [RelativePosition](#) & *position*) [inherited]

Definition at line 90 of file `KDChartAbstractAreaBase.cpp`.

```

91 {
92     Q_UNUSED( position );
93     // PENDING(kalle) FIXME
94     qWarning( "Sorry, not implemented: void AbstractAreaBase::alignToReferencePoint( const RelativePosi
95 }
```

### 9.6.4.3 QRect AbstractArea::areaGeometry () const [protected, virtual, inherited]

Implements [KDChart::AbstractAreaBase](#).

Definition at line 150 of file `KDChartAbstractArea.cpp`.

Referenced by `KDChart::CartesianCoordinatePlane::drawingArea()`, `KDChart::TernaryCoordinatePlane::layoutDiagrams()`, `KDChart::PolarCoordinatePlane::layoutDiagrams()`, `KDChart::TernaryCoordinatePlane::paint()`, `KDChart::CartesianAxis::paint()`, `KDChart::AbstractArea::paintAll()`, and `KDChart::CartesianAxis::paintCtx()`.

```

151 {
152     return geometry();
153 }

```

#### 9.6.4.4 BackgroundAttributes AbstractAreaBase::backgroundAttributes () const [inherited]

Definition at line 120 of file KDChartAbstractAreaBase.cpp.

References d.

Referenced by KDChart::AbstractAreaBase::compare(), and updateCommonBrush().

```

121 {
122     return d->backgroundAttributes;
123 }

```

#### 9.6.4.5 int AbstractArea::bottomOverlap (bool *doNotRecalculate* = false) const [virtual, inherited]

This is called at layout time by KDChart::AutoSpacerLayoutItem::sizeHint().

The method triggers AbstractArea::sizeHint() to find out the amount of overlap at the bottom edge of the area.

##### Note:

The default implementation is not using any caching, it might make sense to implement a more sophisticated solution for derived classes that have complex work to do in sizeHint(). All we have here is a primitive flag to be set by the caller if it is sure that no sizeHint() needs to be called.

Definition at line 101 of file KDChartAbstractArea.cpp.

References d.

Referenced by KDChart::CartesianAxis::maximumSize().

```

102 {
103     // Re-calculate the sizes,
104     // so we also get the amountOf..Overlap members set newly:
105     if( ! doNotRecalculate )
106         sizeHint();
107     return d->amountOfBottomOverlap;
108 }

```

#### 9.6.4.6 bool AbstractAreaBase::compare (const AbstractAreaBase \* *other*) const [inherited]

Returns true if both areas have the same settings.

Definition at line 75 of file KDChartAbstractAreaBase.cpp.

References KDChart::AbstractAreaBase::backgroundAttributes(), and KDChart::AbstractAreaBase::frameAttributes().

```

76 {
77     if( other == this ) return true;
78     if( ! other ){

```

```

79         //qDebug() << "CartesianAxis::compare() cannot compare to Null pointer";
80         return false;
81     }
82     /*
83     qDebug() << "AbstractAreaBase:" << (frameAttributes() == other->frameAttributes())
84     << (backgroundAttributes() == other->backgroundAttributes()) << "\n";
85     */
86     return (frameAttributes() == other->frameAttributes()) &&
87            (backgroundAttributes() == other->backgroundAttributes());
88 }

```

#### 9.6.4.7 void KDChart::AbstractCoordinatePlane::destroyedCoordinatePlane ([AbstractCoordinatePlane \\*](#)) [signal]

Emitted when this coordinate plane is destroyed.

Referenced by `~AbstractCoordinatePlane()`.

#### 9.6.4.8 [AbstractDiagram \\*](#) AbstractCoordinatePlane::diagram ()

##### Returns:

The first diagram associated with this coordinate plane.

Definition at line 117 of file `KDChartAbstractCoordinatePlane.cpp`.

References d.

Referenced by `KDChart::TernaryCoordinatePlane::addDiagram()`, `KDChart::PolarCoordinatePlane::addDiagram()`, `KDChart::CartesianCoordinatePlane::addDiagram()`, `addDiagram()`, `KDChart::Widget::diagram()`, `KDChart::CartesianCoordinatePlane::getRawDataBoundingRectFromDiagrams()`, `KDChart::TernaryCoordinatePlane::layoutDiagrams()`, `KDChart::PolarCoordinatePlane::layoutDiagrams()`, `replaceDiagram()`, `KDChart::CartesianCoordinatePlane::setGeometry()`, `KDChart::PolarCoordinatePlane::setStartPosition()`, `KDChart::CartesianCoordinatePlane::sharedAxisMasterPlane()`, and `takeDiagram()`.

```

118 {
119     if ( d->diagrams.isEmpty() )
120     {
121         return 0;
122     } else {
123         return d->diagrams.first();
124     }
125 }

```

#### 9.6.4.9 [ConstAbstractDiagramList](#) AbstractCoordinatePlane::diagrams () const

##### Returns:

The list of diagrams associated with this coordinate plane.

Definition at line 132 of file `KDChartAbstractCoordinatePlane.cpp`.

References d.

```

133 {

```

```

134     ConstAbstractDiagramList list;
135 #ifndef QT_NO_STL
136     qCopy( d->diagrams.begin(), d->diagrams.end(), std::back_inserter( list ) );
137 #else
138     Q_FOREACH( AbstractDiagram * a, d->diagrams )
139         list.push_back( a );
140 #endif
141     return list;
142 }

```

#### 9.6.4.10 [AbstractDiagramList](#) AbstractCoordinatePlane::diagrams ()

##### Returns:

The list of diagrams associated with this coordinate plane.

Definition at line 127 of file KDChartAbstractCoordinatePlane.cpp.

References [d](#).

Referenced by [KDChart::CartesianCoordinatePlane::getDataDimensionsList\(\)](#), [KDChart::CartesianCoordinatePlane::getRawDataBoundingRectFromDiagrams\(\)](#), [KDChart::TernaryCoordinatePlane::layoutDiagrams\(\)](#), [KDChart::PolarCoordinatePlane::layoutDiagrams\(\)](#), [KDChart::CartesianCoordinatePlane::layoutDiagrams\(\)](#), [KDChart::Chart::mouseDoubleClickEvent\(\)](#), [KDChart::Chart::mouseMoveEvent\(\)](#), [KDChart::Chart::mousePressEvent\(\)](#), [KDChart::Chart::mouseReleaseEvent\(\)](#), [KDChart::TernaryCoordinatePlane::paint\(\)](#), [KDChart::PolarCoordinatePlane::paint\(\)](#), [KDChart::CartesianCoordinatePlane::paint\(\)](#), and [KDChart::CartesianCoordinatePlane::setGeometry\(\)](#).

```

128 {
129     return d->diagrams;
130 }

```

#### 9.6.4.11 [Qt::Orientations](#) KDChart::AbstractCoordinatePlane::expandingDirections () const [virtual]

pure virtual in [QLayoutItem](#)

Definition at line 212 of file KDChartAbstractCoordinatePlane.cpp.

```

213 {
214     return Qt::Vertical | Qt::Horizontal;
215 }

```

#### 9.6.4.12 [FrameAttributes](#) AbstractAreaBase::frameAttributes () const [inherited]

Definition at line 106 of file KDChartAbstractAreaBase.cpp.

References [d](#).

Referenced by [KDChart::Legend::clone\(\)](#), [KDChart::AbstractAreaBase::compare\(\)](#), and [updateCommonBrush\(\)](#).

```

107 {
108     return d->frameAttributes;
109 }

```

#### 9.6.4.13 **QRect** **KDChart::AbstractCoordinatePlane::geometry () const** [virtual]

pure virtual in [QLayoutItem](#)

Definition at line 246 of file `KDChartAbstractCoordinatePlane.cpp`.

References [d](#).

Referenced by `KDChart::Chart::mouseDoubleClickEvent()`, `KDChart::Chart::mouseMoveEvent()`, `mouseMoveEvent()`, `KDChart::Chart::mousePressEvent()`, `KDChart::Chart::mouseReleaseEvent()`, `mouseReleaseEvent()`, and `KDChart::PolarCoordinatePlane::paint()`.

```
247 {
248     return d->geometry;
249 }
```

#### 9.6.4.14 **virtual** [DataDimensionsList](#) **KDChart::AbstractCoordinatePlane::getDataDimensionsList () const** [protected, pure virtual]

Implemented in [KDChart::CartesianCoordinatePlane](#), [KDChart::PolarCoordinatePlane](#), and [KDChart::TernaryCoordinatePlane](#).

#### 9.6.4.15 **void** **AbstractAreaBase::getFrameLeadings (int & left, int & top, int & right, int & bottom) const** [inherited]

Definition at line 212 of file `KDChartAbstractAreaBase.cpp`.

References [d](#).

Referenced by `KDChart::AbstractAreaBase::innerRect()`, and `KDChart::AbstractAreaWidget::paintAll()`.

```
213 {
214     if( d && d->frameAttributes.isVisible() ){
215         const int padding = qMax( d->frameAttributes.padding(), 0 );
216         left    = padding;
217         top     = padding;
218         right   = padding;
219         bottom  = padding;
220     }else{
221         left    = 0;
222         top     = 0;
223         right   = 0;
224         bottom  = 0;
225     }
226 }
```

#### 9.6.4.16 [GridAttributes](#) **KDChart::AbstractCoordinatePlane::globalGridAttributes () const**

##### Returns:

The grid attributes used by this coordinate plane.

##### See also:

[setGlobalGridAttributes](#)  
[CartesianCoordinatePlane::gridAttributes](#)

Definition at line 161 of file KDChartAbstractCoordinatePlane.cpp.

References d.

Referenced by KDChart::PolarCoordinatePlane::gridAttributes(), and KDChart::CartesianCoordinatePlane::gridAttributes().

```
162 {
163     return d->gridAttributes;
164 }
```

#### 9.6.4.17 [KDChart::DataDimensionsList](#) KDChart::AbstractCoordinatePlane::gridDimensionsList ()

Returns the dimensions used for drawing the grid lines.

Returned data is the result of (cached) grid calculations, so - if you need that information for your own tasks - make sure to call again this function after every data modification that has changed the data range, since grid calculation is based upon the data range, thus the grid start/end might have changed if the data was changed.

##### Note:

Returned list will contain different numbers of [DataDimension](#), depending on the kind of coordinate plane used. For [CartesianCoordinatePlane](#) two [DataDimension](#) are returned: the first representing grid lines in X direction (matching the Abscissa axes) and the second indicating vertical grid lines (or Ordinate axes, resp.).

##### Returns:

The dimensions used for drawing the grid lines.

##### See also:

[DataDimension](#)

Definition at line 166 of file KDChartAbstractCoordinatePlane.cpp.

References d.

Referenced by KDChart::CartesianCoordinatePlane::layoutDiagrams(), and KDChart::CartesianAxis::maximumSize().

```
167 {
168     //KDChart::DataDimensionsList l( d->grid->updateData( this ) );
169     //qDebug() << "AbstractCoordinatePlane::gridDimensionsList() Y-range:" << l.last().end - l.last().
170     //qDebug() << "AbstractCoordinatePlane::gridDimensionsList() X-range:" << l.first().end - l.first().
171     return d->grid->updateData( this );
172 }
```

#### 9.6.4.18 [QRect](#) AbstractAreaBase::innerRect () const [protected, inherited]

Definition at line 228 of file KDChartAbstractAreaBase.cpp.

References KDChart::AbstractAreaBase::areaGeometry(), and KDChart::AbstractAreaBase::getFrameLeadings().

Referenced by KDChart::TextArea::paintAll(), and KDChart::AbstractArea::paintAll().

```
229 {
230     int left;
231     int top;
232     int right;
233     int bottom;
234     setFrameLeadings( left, top, right, bottom );
235     return
236         QRect( QPoint(0,0), areaGeometry().size() )
237             .adjusted( left, top, -right, -bottom );
238 }
```

#### 9.6.4.19 bool KDChart::AbstractCoordinatePlane::isEmpty () const [virtual]

pure virtual in [QLayoutItem](#)

Definition at line 205 of file KDChartAbstractCoordinatePlane.cpp.

Referenced by KDChart::CartesianCoordinatePlane::layoutDiagrams().

```
206 {
207     return false; // never empty!
208     // coordinate planes with no associated diagrams
209     // are showing a default grid of (1..10, 1..10) stepWidth 1
210 }
```

#### 9.6.4.20 bool KDChart::AbstractCoordinatePlane::isRubberBandZoomingEnabled () const

##### Returns:

Whether zooming with a rubber band using the mouse is enabled.

Definition at line 280 of file KDChartAbstractCoordinatePlane.cpp.

References d.

```
281 {
282     return d->enableRubberBandZooming;
283 }
```

#### 9.6.4.21 const bool KDChart::AbstractCoordinatePlane::isVisiblePoint (const QPointF & point) const

Tests, if a point is visible on the coordinate plane.

##### Note:

Before calling this function the point must have been translated into coordinate plane space.

Definition at line 403 of file KDChartAbstractCoordinatePlane.cpp.

References d.

```
404 {
405     return d->isVisiblePoint( this, point );
406 }
```



#### 9.6.4.22 virtual void KDChart::AbstractCoordinatePlane::layoutDiagrams () [pure virtual]

Distribute the available space among the diagrams and axes.

Implemented in [KDChart::CartesianCoordinatePlane](#), [KDChart::PolarCoordinatePlane](#), and [KDChart::TernaryCoordinatePlane](#).

Referenced by [addDiagram\(\)](#), [replaceDiagram\(\)](#), and [takeDiagram\(\)](#).

#### 9.6.4.23 void KDChart::AbstractCoordinatePlane::layoutPlanes () [slot]

Calling [layoutPlanes\(\)](#) on the plane triggers the global [KDChart::Chart::slotLayoutPlanes\(\)](#).

Definition at line 263 of file [KDChartAbstractCoordinatePlane.cpp](#).

References [needLayoutPlanes\(\)](#).

Referenced by [addDiagram\(\)](#), [KDChart::CartesianAxis::layoutPlanes\(\)](#), [KDChart::AbstractCartesianDiagram::layoutPlanes\(\)](#), and [replaceDiagram\(\)](#).

```
264 {
265     //qDebug("KDChart::AbstractCoordinatePlane::relayout() called");
266     emit needLayoutPlanes();
267 }
```

#### 9.6.4.24 int AbstractArea::leftOverlap (bool *doNotRecalculate* = false) const [virtual, inherited]

This is called at layout time by [KDChart::AutoSpacerLayoutItem::sizeHint\(\)](#).

The method triggers [AbstractArea::sizeHint\(\)](#) to find out the amount of overlap at the left edge of the area.

##### Note:

The default implementation is not using any caching, it might make sense to implement a more sophisticated solution for derived classes that have complex work to do in [sizeHint\(\)](#). All we have here is a primitive flag to be set by the caller if it is sure that no [sizeHint\(\)](#) needs to be called.

Definition at line 77 of file [KDChartAbstractArea.cpp](#).

References [d](#).

Referenced by [KDChart::CartesianAxis::maximumSize\(\)](#).

```
78 {
79     // Re-calculate the sizes,
80     // so we also get the amountOf..Overlap members set newly:
81     if( ! doNotRecalculate )
82         sizeHint();
83     return d->amountOfLeftOverlap;
84 }
```

#### 9.6.4.25 QSize KDChart::AbstractCoordinatePlane::maximumSize () const [virtual]

pure virtual in [QLayoutItem](#)

Definition at line 217 of file KDChartAbstractCoordinatePlane.cpp.

Referenced by `sizeHint()`.

```

218 {
219     // No maximum size set. Especially not parent()->size(), we are not layouting
220     // to the parent widget's size when using Chart::paint()!
221     return QSize(QLAYOUTSIZE_MAX, QLAYOUTSIZE_MAX);
222 }
```

#### 9.6.4.26 QSize KDChart::AbstractCoordinatePlane::minimumSize () const [virtual]

pure virtual in [QLayoutItem](#)

Definition at line 224 of file KDChartAbstractCoordinatePlane.cpp.

```

225 {
226     return QSize(60, 60); // this default can be overwritten by derived classes
227 }
```

#### 9.6.4.27 QSize KDChart::AbstractCoordinatePlane::minimumSizeHint () const [virtual]

[reimplemented]

Reimplemented in [KDChart::TernaryCoordinatePlane](#).

Definition at line 144 of file KDChartAbstractCoordinatePlane.cpp.

```

145 {
146     return QSize( 200, 200 );
147 }
```

#### 9.6.4.28 void KDChart::AbstractCoordinatePlane::mouseDoubleClickEvent (QMouseEvent \* event)

Definition at line 325 of file KDChartAbstractCoordinatePlane.cpp.

References `d`, and `mousePressEvent()`.

Referenced by `KDChart::Chart::mouseDoubleClickEvent()`.

```

326 {
327     if( event->button() == Qt::RightButton )
328     {
329         // othwise the second click gets lost
330         // which is pretty annoying when zooming out fast
331         mousePressEvent( event );
332     }
333     KDAB_FOREACH( AbstractDiagram * a, d->diagrams )
334     {
335         a->mouseDoubleClickEvent( event );
336     }
337 }
```

**9.6.4.29 void KDChart::AbstractCoordinatePlane::mouseMoveEvent (QMouseEvent \* event)**

Definition at line 387 of file KDChartAbstractCoordinatePlane.cpp.

References `d`, and `geometry()`.

Referenced by `KDChart::Chart::mouseMoveEvent()`.

```

388 {
389     if( d->rubberBand != 0 )
390     {
391         const QRect normalized = QRect( d->rubberBandOrigin, event->pos() ).normalized();
392         d->rubberBand->setGeometry( normalized & geometry() );
393
394         event->accept();
395     }
396
397     KDAB_FOREACH( AbstractDiagram * a, d->diagrams )
398     {
399         a->mouseMoveEvent( event );
400     }
401 }
```

**9.6.4.30 void KDChart::AbstractCoordinatePlane::mousePressEvent (QMouseEvent \* event)**

Definition at line 285 of file KDChartAbstractCoordinatePlane.cpp.

References `d`, `parent()`, `setZoomCenter()`, `setZoomFactorX()`, and `setZoomFactorY()`.

Referenced by `mouseDoubleClickEvent()`, and `KDChart::Chart::mousePressEvent()`.

```

286 {
287     if( event->button() == Qt::LeftButton )
288     {
289         if( d->enableRubberBandZooming && d->rubberBand == 0 )
290             d->rubberBand = new QRubberBand( QRubberBand::Rectangle, qobject_cast< QWidget* >( parent() ) );
291
292         if( d->rubberBand != 0 )
293         {
294             d->rubberBandOrigin = event->pos();
295             d->rubberBand->setGeometry( QRect( event->pos(), QSize() ) );
296             d->rubberBand->show();
297
298             event->accept();
299         }
300     }
301     else if( event->button() == Qt::RightButton )
302     {
303         if( d->enableRubberBandZooming && !d->rubberBandZoomConfigHistory.isEmpty() )
304         {
305             // restore the last config from the stack
306             ZoomParameters config = d->rubberBandZoomConfigHistory.pop();
307             setZoomFactorX( config.xFactor );
308             setZoomFactorY( config.yFactor );
309             setZoomCenter( config.center() );
310
311             QWidget* const p = qobject_cast< QWidget* >( parent() );
312             if( p != 0 )
313                 p->update();
314
315             event->accept();
316         }
317     }
318 }
```

```

319     KDAB_FOREACH( AbstractDiagram * a, d->diagrams )
320     {
321         a->mousePressEvent( event );
322     }
323 }

```

#### 9.6.4.31 void KDChart::AbstractCoordinatePlane::mouseReleaseEvent (QMouseEvent \* event)

Definition at line 339 of file KDChartAbstractCoordinatePlane.cpp.

References `d`, `geometry()`, `setZoomCenter()`, `setZoomFactorX()`, `setZoomFactorY()`, `zoomCenter()`, `zoomFactorX()`, and `zoomFactorY()`.

Referenced by `KDChart::Chart::mouseReleaseEvent()`.

```

340 {
341     if( d->rubberBand != 0 )
342     {
343         // save the old config on the stack
344         d->rubberBandZoomConfigHistory.push( ZoomParameters( zoomFactorX(), zoomFactorY(), zoomCenter() ) );
345
346         // this is the height/width of the rubber band in pixel space
347         const double rubberWidth = static_cast< double >( d->rubberBand->width() );
348         const double rubberHeight = static_cast< double >( d->rubberBand->height() );
349
350         // this is the center of the rubber band in pixel space
351         const double rubberCenterX = static_cast< double >( d->rubberBand->geometry().center().x() - geometry().center().x() );
352         const double rubberCenterY = static_cast< double >( d->rubberBand->geometry().center().y() - geometry().center().y() );
353
354         // this is the height/width of the plane in pixel space
355         const double myWidth = static_cast< double >( geometry().width() );
356         const double myHeight = static_cast< double >( geometry().height() );
357
358         // this describes the new center of zooming, relative to the plane pixel space
359         const double newCenterX = rubberCenterX / myWidth / zoomFactorX() + zoomCenter().x() - 0.5 / zoomFactorX();
360         const double newCenterY = rubberCenterY / myHeight / zoomFactorY() + zoomCenter().y() - 0.5 / zoomFactorY();
361
362         // this will be the new zoom factor
363         const double newZoomFactorX = zoomFactorX() * myWidth / rubberWidth;
364         const double newZoomFactorY = zoomFactorY() * myHeight / rubberHeight;
365
366         // and this the new center
367         const QPointF newZoomCenter( newCenterX, newCenterY );
368
369         setZoomFactorX( newZoomFactorX );
370         setZoomFactorY( newZoomFactorY );
371         setZoomCenter( newZoomCenter );
372
373
374         d->rubberBand->parentWidget()->update();
375         delete d->rubberBand;
376         d->rubberBand = 0;
377
378         event->accept();
379     }
380
381     KDAB_FOREACH( AbstractDiagram * a, d->diagrams )
382     {
383         a->mouseReleaseEvent( event );
384     }
385 }

```

**9.6.4.32 void KDChart::AbstractCoordinatePlane::needLayoutPlanes ()** [signal]

Emitted when plane needs to trigger the Chart's layouting of the coord.

planes.

Referenced by layoutPlanes().

**9.6.4.33 void KDChart::AbstractCoordinatePlane::needRelayout ()** [signal]

Emitted when plane needs to trigger the Chart's layouting.

Referenced by relayout().

**9.6.4.34 void KDChart::AbstractCoordinatePlane::needUpdate ()** [signal]

Emitted when plane needs to update its drawings.

Referenced by update().

**9.6.4.35 virtual void KDChart::AbstractLayoutItem::paint (QPainter \*)** [pure virtual, inherited]

Implemented in [KDChart::CartesianAxis](#), [KDChart::CartesianCoordinatePlane](#), [KDChart::TextLayoutItem](#), [KDChart::MarkerLayoutItem](#), [KDChart::LineLayoutItem](#), [KDChart::LineWithMarkerLayoutItem](#), [KDChart::HorizontalLineLayoutItem](#), [KDChart::VerticalLineLayoutItem](#), [KDChart::AutoSpacerLayoutItem](#), [KDChart::PolarCoordinatePlane](#), [KDChart::TernaryAxis](#), and [KDChart::TernaryCoordinatePlane](#).

Referenced by [KDChart::Legend::paint\(\)](#), [KDChart::AbstractLayoutItem::paintAll\(\)](#), [KDChart::AbstractArea::paintAll\(\)](#), and [KDChart::AbstractLayoutItem::paintCtx\(\)](#).

**9.6.4.36 void AbstractArea::paintAll (QPainter & painter)** [virtual, inherited]

Call paintAll, if you want the background and the frame to be drawn before the normal [paint\(\)](#) is invoked automatically.

Reimplemented from [KDChart::AbstractLayoutItem](#).

Reimplemented in [KDChart::TernaryAxis](#).

Definition at line 123 of file [KDChartAbstractArea.cpp](#).

References [KDChart::AbstractArea::areaGeometry\(\)](#), [d](#), [KDChart::AbstractAreaBase::innerRect\(\)](#), [KDChart::AbstractLayoutItem::paint\(\)](#), [KDChart::AbstractAreaBase::paintBackground\(\)](#), and [KDChart::AbstractAreaBase::paintFrame\(\)](#).

Referenced by [KDChart::AbstractArea::paintIntoRect\(\)](#).

```

124 {
125     // Paint the background and frame
126     const QRect overlappingArea( geometry().adjusted(
127         -d->amountOfLeftOverlap,
128         -d->amountOfTopOverlap,
129         d->amountOfRightOverlap,
130         d->amountOfBottomOverlap ) );
131     paintBackground( painter, overlappingArea );
132     paintFrame(      painter, overlappingArea );
133 }
```

```

134 // temporarily adjust the widget size, to be sure all content gets calculated
135 // to fit into the inner rectangle
136 const QRect oldGeometry( areaGeometry() );
137 QRect inner( innerRect() );
138 inner.moveTo(
139     oldGeometry.left() + inner.left(),
140     oldGeometry.top() + inner.top() );
141 const bool needAdjustGeometry = oldGeometry != inner;
142 if( needAdjustGeometry )
143     setGeometry( inner );
144 paint( &painter );
145 if( needAdjustGeometry )
146     setGeometry( oldGeometry );
147 //qDebug() << "AbstractAreaWidget::paintAll() done.";
148 }

```

#### 9.6.4.37 void AbstractAreaBase::paintBackground (QPainter & painter, const QRect & rectangle) [virtual, inherited]

Definition at line 196 of file KDChartAbstractAreaBase.cpp.

References `d`, and `KDChart::AbstractAreaBase::paintBackgroundAttributes()`.

Referenced by `KDChart::TextArea::paintAll()`, `KDChart::AbstractAreaWidget::paintAll()`, and `KDChart::AbstractArea::paintAll()`.

```

197 {
198     Q_ASSERT_X ( d != 0, "AbstractAreaBase::paintBackground()",
199                 "Private class was not initialized!" );
200     paintBackgroundAttributes( painter, rect, d->backgroundAttributes );
201 }

```

#### 9.6.4.38 void AbstractAreaBase::paintBackgroundAttributes (QPainter & painter, const QRect & rectangle, const KDChart::BackgroundAttributes & attributes) [static, inherited]

Definition at line 127 of file KDChartAbstractAreaBase.cpp.

References `KDChart::BackgroundAttributes::BackgroundPixmapModeCentered`, `KDChart::BackgroundAttributes::BackgroundPixmapModeNone`, `KDChart::BackgroundAttributes::BackgroundPixmapModeScaled`, `KDChart::BackgroundAttributes::BackgroundPixmapModeStretched`, `KDChart::BackgroundAttributes::brush()`, `KDChart::BackgroundAttributes::isVisible()`, `KDChart::BackgroundAttributes::pixmap()`, and `KDChart::BackgroundAttributes::pixmapMode()`.

Referenced by `KDChart::AbstractAreaBase::paintBackground()`.

```

129 {
130     if( !attributes.isVisible() ) return;
131
132     /* first draw the brush (may contain a pixmap)*/
133     if( Qt::NoBrush != attributes.brush().style() ) {
134         KDChart::PainterSaver painterSaver( &painter );
135         painter.setPen( Qt::NoPen );
136         const QPointF newTopLeft( painter.deviceMatrix().map( rect.topLeft() ) );
137         painter.setBrushOrigin( newTopLeft );
138         painter.setBrush( attributes.brush() );
139         painter.drawRect( rect.adjusted( 0, 0, -1, -1 ) );
140     }
141     /* next draw the backPixmap over the brush */

```

```

142     if( !attributes.pixmap().isNull() &&
143         attributes.pixmapMode() != BackgroundAttributes::BackgroundPixmapModeNone ) {
144         QPointF ol = rect.topLeft();
145         if( BackgroundAttributes::BackgroundPixmapModeCentered == attributes.pixmapMode() )
146         {
147             ol.setX( rect.center().x() - attributes.pixmap().width() / 2 );
148             ol.setY( rect.center().y() - attributes.pixmap().height() / 2 );
149             painter.drawPixmap( ol, attributes.pixmap() );
150         } else {
151             QMatrix m;
152             double zW = (double)rect.width() / (double)attributes.pixmap().width();
153             double zH = (double)rect.height() / (double)attributes.pixmap().height();
154             switch( attributes.pixmapMode() ) {
155                 case BackgroundAttributes::BackgroundPixmapModeScaled:
156                 {
157                     double z;
158                     z = qMin( zW, zH );
159                     m.scale( z, z );
160                 }
161                 break;
162                 case BackgroundAttributes::BackgroundPixmapModeStretched:
163                     m.scale( zW, zH );
164                     break;
165                 default:
166                     ; // Cannot happen, previously checked
167             }
168             QPixmap pm = attributes.pixmap().transformed( m );
169             ol.setX( rect.center().x() - pm.width() / 2 );
170             ol.setY( rect.center().y() - pm.height() / 2 );
171             painter.drawPixmap( ol, pm );
172         }
173     }
174 }

```

#### 9.6.4.39 void KDChart::AbstractLayoutItem::paintCtx ([PaintContext](#) \* *context*) [virtual, inherited]

Default impl: Paint the complete item using its layouted position and size.

Reimplemented in [KDChart::CartesianAxis](#), and [KDChart::TernaryAxis](#).

Definition at line 77 of file `KDChartLayoutItems.cpp`.

References `KDChart::AbstractLayoutItem::paint()`, and `KDChart::PaintContext::painter()`.

```

78 {
79     if( context )
80         paint( context->painter() );
81 }

```

#### 9.6.4.40 void AbstractAreaBase::paintFrame (QPainter & *painter*, const QRect & *rectangle*) [virtual, inherited]

Definition at line 204 of file `KDChartAbstractAreaBase.cpp`.

References `d`, and `KDChart::AbstractAreaBase::paintFrameAttributes()`.

Referenced by `KDChart::TextArea::paintAll()`, `KDChart::AbstractAreaWidget::paintAll()`, and `KDChart::AbstractArea::paintAll()`.

```

205 {

```

```

206     Q_ASSERT_X ( d != 0, "AbstractAreaBase::paintFrame()",
207                 "Private class was not initialized!" );
208     paintFrameAttributes( painter, rect, d->frameAttributes );
209 }

```

#### 9.6.4.41 void AbstractAreaBase::paintFrameAttributes (QPainter & *painter*, const QRect & *rectangle*, const [KDChart::FrameAttributes](#) & *attributes*) [static, inherited]

Definition at line 177 of file `KDChartAbstractAreaBase.cpp`.

References `KDChart::FrameAttributes::isVisible()`, and `KDChart::FrameAttributes::pen()`.

Referenced by `KDChart::AbstractAreaBase::paintFrame()`.

```

179 {
180
181     if( !attributes.isVisible() ) return;
182
183     // Note: We set the brush to NoBrush explicitly here.
184     //       Otherwise we might get a filled rectangle, so any
185     //       previously drawn background would be overwritten by that area.
186
187     const QPen    oldPen(    painter.pen() );
188     const QBrush  oldBrush(  painter.brush() );
189     painter.setPen(    attributes.pen() );
190     painter.setBrush(  Qt::NoBrush );
191     painter.drawRect( rect.adjusted( 0, 0, -1, -1 ) );
192     painter.setBrush(  oldBrush );
193     painter.setPen(    oldPen );
194 }

```

#### 9.6.4.42 void AbstractArea::paintIntoRect (QPainter & *painter*, const QRect & *rect*) [virtual, inherited]

Draws the background and frame, then calls [paint\(\)](#).

In most cases there is no need to overwrite this method in a derived class, but you would overwrite [AbstractLayoutItem::paint\(\)](#) instead.

Definition at line 111 of file `KDChartAbstractArea.cpp`.

References `KDChart::AbstractArea::paintAll()`.

```

112 {
113     const QRect oldGeometry( geometry() );
114     if( oldGeometry != rect )
115         setGeometry( rect );
116     painter.translate( rect.left(), rect.top() );
117     paintAll( painter );
118     painter.translate( -rect.left(), -rect.top() );
119     if( oldGeometry != rect )
120         setGeometry( oldGeometry );
121 }

```

#### 9.6.4.43 const [KDChart::Chart](#) \* KDChart::AbstractCoordinatePlane::parent () const

Definition at line 194 of file `KDChartAbstractCoordinatePlane.cpp`.

References `d`.



```

195 {
196     return d->parent;
197 }

```

#### 9.6.4.44 [KDChart::Chart](#) \* KDChart::AbstractCoordinatePlane::parent ()

Definition at line 199 of file KDChartAbstractCoordinatePlane.cpp.

References [d](#).

Referenced by [AbstractCoordinatePlane\(\)](#), [KDChart::CartesianAxis::maximumSize\(\)](#), [mousePressEvent\(\)](#), and [setParent\(\)](#).

```

200 {
201     return d->parent;
202 }

```

#### 9.6.4.45 [QLayout\\*](#) KDChart::AbstractLayoutItem::parentLayout () [inherited]

Definition at line 76 of file KDChartLayoutItems.h.

```

77     {
78         return mParentLayout;
79     }

```

#### 9.6.4.46 void KDChart::AbstractArea::positionChanged ([AbstractArea](#) \*) [signal, inherited]

Referenced by [KDChart::AbstractArea::positionHasChanged\(\)](#).

#### 9.6.4.47 void AbstractArea::positionHasChanged () [protected, virtual, inherited]

Reimplemented from [KDChart::AbstractAreaBase](#).

Definition at line 155 of file KDChartAbstractArea.cpp.

References [KDChart::AbstractArea::positionChanged\(\)](#).

```

156 {
157     emit positionChanged( this );
158 }

```

#### 9.6.4.48 void KDChart::AbstractCoordinatePlane::propertiesChanged () [signal]

Emitted upon change of a property of the Coordinate Plane or any of its components.

Referenced by [KDChart::CartesianCoordinatePlane::addDiagram\(\)](#), [KDChart::CartesianCoordinatePlane::adjustHorizontalRangeToData\(\)](#), [KDChart::CartesianCoordinatePlane::adjustRangesToData\(\)](#), [KDChart::CartesianCoordinatePlane::adjustVerticalRangeToData\(\)](#), [KDChart::CartesianCoordinatePlane::setAutoAdjustGridToZoom\(\)](#), [KDChart::CartesianCoordinatePlane::setAutoAdjustHorizontalRangeToData\(\)](#), [KDChart::CartesianCoordinatePlane::setAutoAdjustVerticalRangeToData\(\)](#), [KDChart::CartesianCoordinatePlane::setAxesCalcModes\(\)](#), [KDChart::CartesianCoordinatePlane::setAxesCalcModeX\(\)](#), and [KDChart::CartesianCoordinatePlane::setAxesCalcModeY\(\)](#).

KDChart::PolarCoordinatePlane::setGridAttributes(), KDChart::CartesianCoordinatePlane::setGridAttributes(), KDChart::CartesianCoordinatePlane::setHorizontalRange(), KDChart::CartesianCoordinatePlane::setHorizontalRangeReversed(), KDChart::CartesianCoordinatePlane::setIsometricScaling(), KDChart::CartesianCoordinatePlane::setVerticalRange(), KDChart::CartesianCoordinatePlane::setVerticalRangeReversed(), KDChart::CartesianCoordinatePlane::setZoomCenter(), KDChart::CartesianCoordinatePlane::setZoomFactorX(), and KDChart::CartesianCoordinatePlane::setZoomFactorY().

#### 9.6.4.49 **AbstractCoordinatePlane** \* KDChart::AbstractCoordinatePlane::referenceCoordinatePlane () const

There are two ways, in which planes can be caused to interact, in where they are put layouting wise: The first is the reference plane.

If such a reference plane is set, on a plane, it will use the same cell in the layout as that one. In addition to this, planes can share an axis. In that case they will be layed out in relation to each other as suggested by the position of the axis. If, for example Plane1 and Plane2 share an axis at position Left, that will result in the layout: Axis Plane1 Plane 2, vertically. If Plane1 also happens to be Plane2's reference plane, both planes are drawn over each other. The reference plane concept allows two planes to share the same space even if neither has any axis, and in case there are shared axis, it is used to decided, whether the planes should be painted on top of each other or layed out vertically or horizontally next to each other.

#### Returns:

The reference coordinate plane associated with this one.

Definition at line 184 of file KDChartAbstractCoordinatePlane.cpp.

References d.

```
185 {
186     return d->referenceCoordinatePlane;
187 }
```

#### 9.6.4.50 **void** KDChart::AbstractCoordinatePlane::relayout () [slot]

Calling [relayout\(\)](#) on the plane triggers the global KDChart::Chart::slotRelayout().

Definition at line 257 of file KDChartAbstractCoordinatePlane.cpp.

References needRelayout().

```
258 {
259     //QDebug("KDChart::AbstractCoordinatePlane::relayout() called");
260     emit needRelayout();
261 }
```

#### 9.6.4.51 **void** KDChart::AbstractLayoutItem::removeFromParentLayout () [inherited]

Definition at line 80 of file KDChartLayoutItems.h.

Referenced by KDChart::Chart::takeCoordinatePlane().

```
81     {
82         if ( mParentLayout ) {
```

```

83         if( widget() )
84             mParentLayout->removeWidget( widget() );
85         else
86             mParentLayout->removeItem( this );
87     }
88 }

```

#### 9.6.4.52 void AbstractCoordinatePlane::replaceDiagram ([AbstractDiagram](#) \* *diagram*, [AbstractDiagram](#) \* *oldDiagram* = 0) [virtual]

Replaces the old diagram, or appends the diagram, if there is none yet.

##### Parameters:

***diagram*** The diagram to be used instead of the old diagram. This parameter must not be zero, or the method will do nothing.

***oldDiagram*** The diagram to be removed by the new diagram. This diagram will be deleted automatically. If the parameter is omitted, the very first diagram will be replaced. In case, there was no diagram yet, the new diagram will just be added.

##### Note:

If you want to re-use the old diagram, call `takeDiagram` and `addDiagram`, instead of using `replaceDiagram`.

##### See also:

[addDiagram](#), [takeDiagram](#)

Definition at line 86 of file `KDChartAbstractCoordinatePlane.cpp`.

References `addDiagram()`, `d`, `diagram()`, `layoutDiagrams()`, `layoutPlanes()`, `takeDiagram()`, and `update()`.

Referenced by `KDChart::Widget::setType()`.

```

87 {
88     if( diagram && oldDiagram_ != diagram ){
89         AbstractDiagram* oldDiagram = oldDiagram_;
90         if( d->diagrams.count() ){
91             if( ! oldDiagram )
92                 oldDiagram = d->diagrams.first();
93             takeDiagram( oldDiagram );
94         }
95         delete oldDiagram;
96         addDiagram( diagram );
97         layoutDiagrams();
98         layoutPlanes(); // there might be new axes, etc
99         update();
100     }
101 }

```

#### 9.6.4.53 int AbstractArea::rightOverlap (bool *doNotRecalculate* = false) const [virtual, inherited]

This is called at layout time by [KDChart::AutoSpacerLayoutItem::sizeHint\(\)](#).

The method triggers `AbstractArea::sizeHint()` to find out the amount of overlap at the right edge of the area.

**Note:**

The default implementation is not using any caching, it might make sense to implement a more sophisticated solution for derived classes that have complex work to do in `sizeHint()`. All we have here is a primitive flag to be set by the caller if it is sure that no `sizeHint()` needs to be called.

Definition at line 85 of file `KDChartAbstractArea.cpp`.

References `d`.

Referenced by `KDChart::CartesianAxis::maximumSize()`.

```

86 {
87     // Re-calculate the sizes,
88     // so we also get the amountOf..Overlap members set newly:
89     if( ! doNotRecalculate )
90         sizeHint();
91     return d->amountOfRightOverlap;
92 }
```

#### 9.6.4.54 void AbstractAreaBase::setBackgroundAttributes (const [BackgroundAttributes](#) & a) [inherited]

Definition at line 111 of file `KDChartAbstractAreaBase.cpp`.

References `d`, and `KDChart::AbstractAreaBase::positionHasChanged()`.

```

112 {
113     if( d->backgroundAttributes == a )
114         return;
115
116     d->backgroundAttributes = a;
117     positionHasChanged();
118 }
```

#### 9.6.4.55 void AbstractAreaBase::setFrameAttributes (const [FrameAttributes](#) & a) [inherited]

Definition at line 97 of file `KDChartAbstractAreaBase.cpp`.

References `d`, and `KDChart::AbstractAreaBase::positionHasChanged()`.

Referenced by `KDChart::Legend::clone()`.

```

98 {
99     if( d->frameAttributes == a )
100         return;
101
102     d->frameAttributes = a;
103     positionHasChanged();
104 }
```

#### 9.6.4.56 void KDChart::AbstractCoordinatePlane::setGeometry (const [QRect](#) & r) [virtual]

pure virtual in [QLayoutItem](#)

**Note:**

Do not call this function directly, unless you know exactly what you are doing. Geometry management is done by KD Chart's internal layouting measures.

Reimplemented in [KDChart::CartesianCoordinatePlane](#).

Definition at line 236 of file KDChartAbstractCoordinatePlane.cpp.

References [d](#).

Referenced by [KDChart::CartesianCoordinatePlane::setGeometry\(\)](#).

```
237 {
238 //     qDebug() << "KDChart::AbstractCoordinatePlane::setGeometry(" << r << ") called";
239     if( d->geometry != r ){
240         d->geometry = r;
241         // Note: We do *not* call update() here
242         //         because it would invoke KDChart::update() recursively.
243     }
244 }
```

**9.6.4.57 void KDChart::AbstractCoordinatePlane::setGlobalGridAttributes (const GridAttributes &)**

Set the grid attributes to be used by this coordinate plane.

To disable grid painting, for example, your code should like this:

```
GridAttributes ga = plane->globalGridAttributes();
ga.setGlobalGridVisible( false );
plane->setGlobalGridAttributes( ga );
```

**See also:**

[globalGridAttributes](#)  
[CartesianCoordinatePlane::setGridAttributes](#)

Definition at line 155 of file KDChartAbstractCoordinatePlane.cpp.

References [d](#), and [update\(\)](#).

```
156 {
157     d->gridAttributes = a;
158     update();
159 }
```

**9.6.4.58 void KDChart::AbstractCoordinatePlane::setGridNeedsRecalculate () [slot]**

Used by the chart to clear the cached grid data.

Definition at line 174 of file KDChartAbstractCoordinatePlane.cpp.

References [d](#).

Referenced by [KDChart::Chart::resizeEvent\(\)](#).

```
175 {
176     d->grid->setNeedRecalculate();
177 }
```

**9.6.4.59 void KDChart::AbstractCoordinatePlane::setParent ([Chart](#) \* *parent*)**

Called internally by [KDChart::Chart](#).

Definition at line 189 of file KDChartAbstractCoordinatePlane.cpp.

References [d](#), and [parent\(\)](#).

Referenced by [KDChart::Chart::addCoordinatePlane\(\)](#), and [KDChart::Chart::takeCoordinatePlane\(\)](#).

```
190 {
191     d->parent = parent;
192 }
```

**9.6.4.60 void KDChart::AbstractLayoutItem::setParentLayout ([QLayout](#) \* *lay*)** [inherited]

Definition at line 72 of file KDChartLayoutItems.h.

```
73     {
74         mParentLayout = lay;
75     }
```

**9.6.4.61 void KDChart::AbstractLayoutItem::setParentWidget ([QWidget](#) \* *widget*)** [virtual, inherited]

Inform the item about its widget: This enables the item, to trigger that widget's update, whenever the size of the item's contents has changed.

Thus, you need to call `setParentWidget` on every item, that has a non-fixed size.

Definition at line 64 of file KDChartLayoutItems.cpp.

References [KDChart::AbstractLayoutItem::mParent](#).

Referenced by [KDChart::HeaderFooter::setParent\(\)](#), and [KDChart::AbstractCartesianDiagram::takeAxis\(\)](#).

```
65 {
66     mParent = widget;
67 }
```

**9.6.4.62 void KDChart::AbstractCoordinatePlane::setReferenceCoordinatePlane ([AbstractCoordinatePlane](#) \* *plane*)**

Set another coordinate plane to be used as the reference plane for this one.

**Parameters:**

*plane* The coordinate plane to be used the reference plane for this one.

**See also:**

[referenceCoordinatePlane](#)

Definition at line 179 of file KDChartAbstractCoordinatePlane.cpp.

References [d](#).

```
180 {  
181     d->referenceCoordinatePlane = plane;  
182 }
```

#### 9.6.4.63 void KDChart::AbstractCoordinatePlane::setRubberBandZoomingEnabled (bool *enable*)

Enables or disables zooming with a rubber band using the mouse.

Definition at line 269 of file KDChartAbstractCoordinatePlane.cpp.

References [d](#).

```
270 {  
271     d->enableRubberBandZooming = enable;  
272  
273     if( !enable && d->rubberBand != 0 )  
274     {  
275         delete d->rubberBand;  
276         d->rubberBand = 0;  
277     }  
278 }
```

#### 9.6.4.64 virtual void KDChart::AbstractCoordinatePlane::setZoomCenter (const QPointF &*center*) [virtual]

Set the point (in value coordinates) to be used as the center point in zoom operations.

##### Parameters:

***center*** The point to use.

Reimplemented in [KDChart::CartesianCoordinatePlane](#), and [KDChart::PolarCoordinatePlane](#).

Definition at line 182 of file KDChartAbstractCoordinatePlane.h.

Referenced by [mousePressEvent\(\)](#), and [mouseReleaseEvent\(\)](#).

```
182 { Q_UNUSED( center ); }
```

#### 9.6.4.65 virtual void KDChart::AbstractCoordinatePlane::setZoomFactorX (double *factor*) [virtual]

Sets the zoom factor in horizontal direction, that is applied to all coordinate transformations.

##### Parameters:

***factor*** The new zoom factor

Reimplemented in [KDChart::CartesianCoordinatePlane](#), and [KDChart::PolarCoordinatePlane](#).

Definition at line 162 of file KDChartAbstractCoordinatePlane.h.

Referenced by [mousePressEvent\(\)](#), and [mouseReleaseEvent\(\)](#).

```
162 { Q_UNUSED( factor ); }
```

#### 9.6.4.66 **virtual void KDChart::AbstractCoordinatePlane::setZoomFactorY (double *factor*)** [virtual]

Sets the zoom factor in vertical direction, that is applied to all coordinate transformations.

##### Parameters:

*factor* The new zoom factor

Reimplemented in [KDChart::CartesianCoordinatePlane](#), and [KDChart::PolarCoordinatePlane](#).

Definition at line 169 of file [KDChartAbstractCoordinatePlane.h](#).

Referenced by [mousePressEvent\(\)](#), and [mouseReleaseEvent\(\)](#).

```
169 { Q_UNUSED( factor ); }
```

#### 9.6.4.67 **AbstractCoordinatePlane \* KDChart::AbstractCoordinatePlane::sharedAxisMaster-Plane (QPainter \* *p* = 0)** [virtual]

Reimplemented in [KDChart::CartesianCoordinatePlane](#).

Definition at line 408 of file [KDChartAbstractCoordinatePlane.cpp](#).

Referenced by [KDChart::Plotter::paint\(\)](#), [KDChart::LineDiagram::paint\(\)](#), and [KDChart::BarDiagram::paint\(\)](#).

```
409 {
410     Q_UNUSED( p );
411     return this;
412 }
```

#### 9.6.4.68 **QSize KDChart::AbstractCoordinatePlane::sizeHint () const** [virtual]

pure virtual in [QLayoutItem](#)

Definition at line 229 of file [KDChartAbstractCoordinatePlane.cpp](#).

References [maximumSize\(\)](#).

```
230 {
231     // we return our maximum (which is the full size of the Chart)
232     // even if we know the plane will be smaller
233     return maximumSize();
234 }
```

#### 9.6.4.69 **void KDChart::AbstractLayoutItem::sizeHintChanged () const** [virtual, inherited]

Report changed size hint: ask the parent widget to recalculate the layout.

Definition at line 86 of file [KDChartLayoutItems.cpp](#).

References [KDChart::AbstractLayoutItem::mParent](#).

Referenced by [KDChart::TextLayoutItem::sizeHint\(\)](#).



```

87 {
88     // This is exactly like what QWidget::updateGeometry does.
89     // qDebug("KDChart::AbstractLayoutItem::sizeHintChanged() called");
90     if( mParent ) {
91         if ( mParent->layout() )
92             mParent->layout()->invalidate();
93         else
94             QApplication::postEvent( mParent, new QEvent( QEvent::LayoutRequest ) );
95     }
96 }

```

#### 9.6.4.70 QSizePolicy KDChart::AbstractCoordinatePlane::sizePolicy () const [virtual]

[reimplemented]

Reimplemented in [KDChart::TernaryCoordinatePlane](#).

Definition at line 150 of file KDChartAbstractCoordinatePlane.cpp.

```

151 {
152     return QSizePolicy( QSizePolicy::MinimumExpanding, QSizePolicy::MinimumExpanding );
153 }

```

#### 9.6.4.71 void AbstractCoordinatePlane::takeDiagram ([AbstractDiagram](#) \* *diagram*) [virtual]

Removes the diagram from the plane, without deleting it.

The plane no longer owns the diagram, so it is the caller's responsibility to delete the diagram.

See also:

[addDiagram](#), [replaceDiagram](#)

Definition at line 104 of file KDChartAbstractCoordinatePlane.cpp.

References [d](#), [diagram\(\)](#), [layoutDiagrams\(\)](#), [KDChart::AbstractDiagram::setCoordinatePlane\(\)](#), and [update\(\)](#).

Referenced by [replaceDiagram\(\)](#).

```

105 {
106     const int idx = d->diagrams.indexOf( diagram );
107     if( idx != -1 ){
108         d->diagrams.removeAt( idx );
109         diagram->setParent( 0 );
110         diagram->setCoordinatePlane( 0 );
111         layoutDiagrams();
112         update();
113     }
114 }

```

#### 9.6.4.72 int AbstractArea::topOverlap (bool *doNotRecalculate* = false) const [virtual, inherited]

This is called at layout time by [KDChart::AutoSpacerLayoutItem::sizeHint\(\)](#).

The method triggers [AbstractArea::sizeHint\(\)](#) to find out the amount of overlap at the top edge of the area.

**Note:**

The default implementation is not using any caching, it might make sense to implement a more sophisticated solution for derived classes that have complex work to do in `sizeHint()`. All we have here is a primitive flag to be set by the caller if it is sure that no `sizeHint()` needs to be called.

Definition at line 93 of file `KDChartAbstractArea.cpp`.

References d.

Referenced by `KDChart::CartesianAxis::maximumSize()`.

```

94 {
95     // Re-calculate the sizes,
96     // so we also get the amountOf..Overlap members set newly:
97     if( ! doNotRecalculate )
98         sizeHint();
99     return d->amountOfTopOverlap;
100 }
```

#### 9.6.4.73 **virtual const QPointF KDChart::AbstractCoordinatePlane::translate (const QPointF & diagramPoint) const** [pure virtual]

Translate the given point in value space coordinates to a position in pixel space.

**Parameters:**

*diagramPoint* The point in value coordinates.

**Returns:**

The translated point.

Implemented in [KDChart::CartesianCoordinatePlane](#), [KDChart::PolarCoordinatePlane](#), and [KDChart::TernaryCoordinatePlane](#).

Referenced by `KDChart::PolarDiagram::paint()`, `KDChart::AbstractDiagram::paintDataValueTexts()`, and `KDChart::AbstractDiagram::paintMarkers()`.

#### 9.6.4.74 **void KDChart::AbstractCoordinatePlane::update ()** [slot]

Calling [update\(\)](#) on the plane triggers the global `KDChart::Chart::update()`.

Definition at line 251 of file `KDChartAbstractCoordinatePlane.cpp`.

References `needUpdate()`.

Referenced by `addDiagram()`, `KDChart::CartesianCoordinatePlane::layoutDiagrams()`, `replaceDiagram()`, `KDChart::PolarCoordinatePlane::resetGridAttributes()`, `KDChart::CartesianCoordinatePlane::resetGridAttributes()`, `setGlobalGridAttributes()`, `KDChart::PolarCoordinatePlane::setGridAttributes()`, `KDChart::CartesianCoordinatePlane::setGridAttributes()`, and `takeDiagram()`.

```

252 {
253     //qDebug("KDChart::AbstractCoordinatePlane::update() called");
254     emit needUpdate();
255 }
```

**9.6.4.75 virtual QPointF KDChart::AbstractCoordinatePlane::zoomCenter () const** [virtual]**Returns:**

The center point (in value coordinates) of the coordinate plane, that is used for zoom operations.

Reimplemented in [KDChart::CartesianCoordinatePlane](#), and [KDChart::PolarCoordinatePlane](#).

Definition at line 175 of file KDChartAbstractCoordinatePlane.h.

Referenced by `mouseReleaseEvent()`.

```
175 { return QPointF(0.0, 0.0); }
```

**9.6.4.76 virtual double KDChart::AbstractCoordinatePlane::zoomFactorX () const** [virtual]**Returns:**

The zoom factor in horizontal direction, that is applied to all coordinate transformations.

Reimplemented in [KDChart::CartesianCoordinatePlane](#), and [KDChart::PolarCoordinatePlane](#).

Definition at line 149 of file KDChartAbstractCoordinatePlane.h.

Referenced by `mouseReleaseEvent()`.

```
149 { return 1.0; }
```

**9.6.4.77 virtual double KDChart::AbstractCoordinatePlane::zoomFactorY () const** [virtual]**Returns:**

The zoom factor in vertical direction, that is applied to all coordinate transformations.

Reimplemented in [KDChart::CartesianCoordinatePlane](#), and [KDChart::PolarCoordinatePlane](#).

Definition at line 155 of file KDChartAbstractCoordinatePlane.h.

Referenced by `mouseReleaseEvent()`.

```
155 { return 1.0; }
```

## 9.6.5 Member Data Documentation

**9.6.5.1 QWidget\* KDChart::AbstractLayoutItem::mParent** [protected, inherited]

Definition at line 90 of file KDChartLayoutItems.h.

Referenced by `KDChart::AbstractLayoutItem::setParentWidget()`, `KDChart::TextLayoutItem::setText()`, `KDChart::TextLayoutItem::setTextAttributes()`, and `KDChart::AbstractLayoutItem::sizeHintChanged()`.

### 9.6.5.2 `QLayout*` [KDChart::AbstractLayoutItem::mParentLayout](#) [protected, inherited]

Definition at line 91 of file `KDChartLayoutItems.h`.

Referenced by `KDChart::AutoSpacerLayoutItem::paint()`.

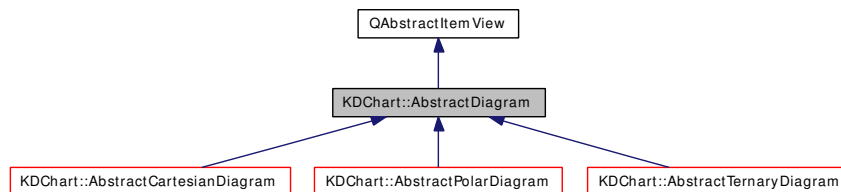
The documentation for this class was generated from the following files:

- [KDChartAbstractCoordinatePlane.h](#)
- [KDChartAbstractCoordinatePlane.cpp](#)

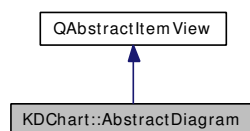
## 9.7 KDChart::AbstractDiagram Class Reference

```
#include <KDChartAbstractDiagram.h>
```

Inheritance diagram for KDChart::AbstractDiagram:



Collaboration diagram for KDChart::AbstractDiagram:



### 9.7.1 Detailed Description

[AbstractDiagram](#) defines the interface for diagram classes.

[AbstractDiagram](#) is the base class for diagram classes ("chart types").

It defines the interface, that needs to be implemented for the diagram, to function within the [KDChart](#) framework. It extends Interview's [QAbstractItemView](#).

Definition at line 53 of file `KDChartAbstractDiagram.h`.

### Signals

- void [dataHidden](#) ()  
*This signal is emitted, when the hidden status of at least one data cell was (un)set.*
- void [layoutChanged](#) ([AbstractDiagram](#) \*)  
*Diagrams are supposed to emit this signal, when the layout of one of their element changes.*
- void [modelsChanged](#) ()  
*This signal is emitted, when either the model or the [AttributesModel](#) is replaced.*
- void [propertiesChanged](#) ()  
*Emitted upon change of a property of the Diagram.*

### Public Member Functions

- bool [allowOverlappingDataValueTexts](#) () const

**Returns:**

*Whether data value labels are allowed to overlap.*

- bool [antiAliasing](#) () const

**Returns:**

*Whether anti-aliasing is to be used for rendering this diagram.*

- virtual [AttributesModel](#) \* [attributesModel](#) () const

*Returns the [AttributesModel](#), that is used by this diagram.*

- QBrush [brush](#) (const QModelIndex &index) const

*Retrieve the brush to be used, for painting the datapoint at the given index in the model.*

- QBrush [brush](#) (int dataset) const

*Retrieve the brush to be used for the given dataset.*

- QBrush [brush](#) () const

*Retrieve the brush to be used for painting datapoints globally.*

- bool [compare](#) (const [AbstractDiagram](#) \*other) const

*Returns true if both diagrams have the same settings.*

- [AbstractCoordinatePlane](#) \* [coordinatePlane](#) () const

*The coordinate plane associated with the diagram.*

- const QPair< QPointF, QPointF > [dataBoundaries](#) () const

*Return the bottom left and top right data point, that the diagram will display (unless the grid adjusts these values).*

- virtual void [dataChanged](#) (const QModelIndex &topLeft, const QModelIndex &bottomRight)

*[reimplemented]*

- QList< QBrush > [datasetBrushes](#) () const

*The set of dataset brushes currently used, for use in legends, etc.*

- int [datasetDimension](#) () const

*The dataset dimension of a diagram determines, how many value dimensions it expects each datapoint to have.*

- QStringList [datasetLabels](#) () const

*The set of dataset labels currently displayed, for use in legends, etc.*

- QList< [MarkerAttributes](#) > [datasetMarkers](#) () const

*The set of dataset markers currently used, for use in legends, etc.*

- QList< QPen > [datasetPens](#) () const

*The set of dataset pens currently used, for use in legends, etc.*

- [DataValueAttributes](#) [dataValueAttributes](#) (const QModelIndex &index) const

*Retrieve the [DataValueAttributes](#) for the given index.*

- [DataValueAttributes dataValueAttributes](#) (int column) const  
*Retrieve the [DataValueAttributes](#) for the given dataset.*
- [DataValueAttributes dataValueAttributes](#) () const  
*Retrieve the [DataValueAttributes](#) specified globally.*
- virtual void [doItemsLayout](#) ()  
*[reimplemented]*
- virtual int [horizontalOffset](#) () const  
*[reimplemented]*
- virtual QModelIndex [indexAt](#) (const QPoint &point) const  
*[reimplemented]*
- QModelIndexList [indexesAt](#) (const QPoint &point) const  
*This method is added alongside with [indexAt](#) from QAIM, since in kdchart multiple indexes can be displayed at the same spot.*
- bool [isHidden](#) (const QModelIndex &index) const  
*Retrieve the hidden status for the given index.*
- bool [isHidden](#) (int column) const  
*Retrieve the hidden status for the given dataset.*
- bool [isHidden](#) () const  
*Retrieve the hidden status specified globally.*
- virtual bool [isIndexHidden](#) (const QModelIndex &index) const  
*[reimplemented]*
- QStringList [itemRowLabels](#) () const  
*The set of item row labels currently displayed, for use in Abscissa axes, etc.*
- virtual QModelIndex [moveCursor](#) (CursorAction cursorAction, Qt::KeyboardModifiers modifiers)  
*[reimplemented]*
- virtual void [paint](#) (PaintContext \*paintContext)=0  
*Draw the diagram contents to the rectangle and painter, that are passed in as part of the paint context.*
- void [paintDataValueText](#) (QPainter \*painter, const QModelIndex &index, const QPointF &pos, double value)
- void [paintMarker](#) (QPainter \*painter, const QModelIndex &index, const QPointF &pos)
- virtual void [paintMarker](#) (QPainter \*painter, const [MarkerAttributes](#) &markerAttributes, const QBrush &brush, const QPen &, const QPointF &point, const QSizeF &size)
- QPen [pen](#) (const QModelIndex &index) const  
*Retrieve the pen to be used, for painting the datapoint at the given index in the model.*
- QPen [pen](#) (int dataset) const

*Retrieve the pen to be used for the given dataset.*

- `QPen pen () const`

*Retrieve the pen to be used for painting datapoints globally.*

- `bool percentMode () const`

- `virtual void resize (const QSizeF &area)=0`

*Called by the widget's sizeEvent.*

- `virtual void scrollTo (const QModelIndex &index, ScrollHint hint=EnsureVisible)`

*[reimplemented]*

- `void setAllowOverlappingDataValueTexts (bool allow)`

*Set whether data value labels are allowed to overlap.*

- `void setAntiAliasing (bool enabled)`

*Set whether anti-aliasing is to be used while rendering this diagram.*

- `virtual void setAttributesModel (AttributesModel *model)`

*Associate an [AttributesModel](#) with this diagram.*

- `void setBrush (const QBrush &brush)`

*Set the brush to be used, for painting all datasets in the model.*

- `void setBrush (int dataset, const QBrush &brush)`

*Set the brush to be used, for painting the given dataset.*

- `void setBrush (const QModelIndex &index, const QBrush &brush)`

*Set the brush to be used, for painting the datapoint at the given index.*

- `virtual void setCoordinatePlane (AbstractCoordinatePlane *plane)`

*Set the coordinate plane associated with the diagram.*

- `void setDatasetDimension (int dimension)`

*Sets the dataset dimension of the diagram.*

- `void setDataValueAttributes (const DataValueAttributes &a)`

*Set the [DataValueAttributes](#) for all datapoints in the model.*

- `void setDataValueAttributes (int dataset, const DataValueAttributes &a)`

*Set the [DataValueAttributes](#) for the given dataset.*

- `void setDataValueAttributes (const QModelIndex &index, const DataValueAttributes &a)`

*Set the [DataValueAttributes](#) for the given index.*

- `void setHidden (bool hidden)`

*Hide (or unhide, resp.)*

- `void setHidden (int column, bool hidden)`

*Hide (or unhide, resp.)*



- void [setHidden](#) (const QModelIndex &index, bool hidden)  
*Hide (or unhide, resp).*
- virtual void [setModel](#) (QAbstractItemModel \*model)  
*Associate a model with the diagram.*
- void [setPen](#) (const QPen &pen)  
*Set the pen to be used, for painting all datasets in the model.*
- void [setPen](#) (int dataset, const QPen &pen)  
*Set the pen to be used, for painting the given dataset.*
- void [setPen](#) (const QModelIndex &index, const QPen &pen)  
*Set the pen to be used, for painting the datapoint at the given index.*
- void [setPercentMode](#) (bool percent)
- virtual void [setRootIndex](#) (const QModelIndex &idx)  
*Set the root index in the model, where the diagram starts referencing data for display.*
- virtual void [setSelection](#) (const QRect &rect, QItemSelectionModel::SelectionFlags command)  
*[reimplemented]*
- void [setUnitPrefix](#) (const QString &prefix, Qt::Orientation orientation)  
*Sets the unit prefix for all values.*
- void [setUnitPrefix](#) (const QString &prefix, int column, Qt::Orientation orientation)  
*Sets the unit prefix for one value.*
- void [setUnitSuffix](#) (const QString &suffix, Qt::Orientation orientation)  
*Sets the unit suffix for all values.*
- void [setUnitSuffix](#) (const QString &suffix, int column, Qt::Orientation orientation)  
*Sets the unit suffix for one value.*
- QString [unitPrefix](#) (Qt::Orientation orientation) const  
*Returns the global unit prefix.*
- QString [unitPrefix](#) (int column, Qt::Orientation orientation, bool fallback=false) const  
*Returns the unit prefix for a special value.*
- QString [unitSuffix](#) (Qt::Orientation orientation) const  
*Returns the global unit suffix.*
- QString [unitSuffix](#) (int column, Qt::Orientation orientation, bool fallback=false) const  
*Returns the unit suffix for a special value.*
- void [update](#) () const
- void [useDefaultColors](#) ()  
*Set the palette to be used, for painting datasets to the default palette.*

- void [useRainbowColors](#) ()  
*Set the palette to be used, for painting datasets to the rainbow palette.*
- virtual bool [usesExternalAttributesModel](#) () const  
*Returns whether the diagram is using its own built-in attributes model or an attributes model that was set via [setAttributesModel](#).*
- void [useSubduedColors](#) ()  
*Set the palette to be used, for painting datasets to the subdued palette.*
- virtual int [verticalOffset](#) () const  
*[reimplemented]*
- virtual QRect [visualRect](#) (const QModelIndex &index) const  
*[reimplemented]*
- virtual QRegion [visualRegionForSelection](#) (const QItemSelection &selection) const  
*[reimplemented]*
- virtual [~AbstractDiagram](#) ()

## Protected Member Functions

- [AbstractDiagram](#) (QWidget \*parent=0, [AbstractCoordinatePlane](#) \*plane=0)
- [AbstractDiagram](#) (Private \*p, QWidget \*parent, [AbstractCoordinatePlane](#) \*plane)
- QModelIndex [attributesModelRootIndex](#) () const
- virtual const QPair< QPointF, QPointF > [calculateDataBoundaries](#) () const=0
- virtual bool [checkInvariants](#) (bool justReturnTheStatus=false) const
- QModelIndex [columnToIndex](#) (int column) const
- virtual void [paintDataValueTexts](#) (QPainter \*painter)
- virtual void [paintMarkers](#) (QPainter \*painter)
- void [setAttributesModelRootIndex](#) (const QModelIndex &)
- void [setDataBoundariesDirty](#) () const
- double [valueForCell](#) (int row, int column) const  
*Helper method, retrieving the data value (DisplayRole) for a given row and column.*

## 9.7.2 Constructor & Destructor Documentation

**9.7.2.1** [KDChart::AbstractDiagram::AbstractDiagram](#) (Private \*p, [QWidget](#) \*parent, [AbstractCoordinatePlane](#) \*plane) [explicit, protected]

**9.7.2.2** [AbstractDiagram::AbstractDiagram](#) ([QWidget](#) \*parent = 0, [AbstractCoordinatePlane](#) \*plane = 0) [explicit, protected]

Definition at line 111 of file [KDChartAbstractDiagram.cpp](#).

```
112      : QAbstractItemView ( parent ), _d( new Private() )
113 {
114     _d->init( plane );
115     init();
116 }
```

### 9.7.2.3 AbstractDiagram::~~AbstractDiagram () [virtual]

Definition at line 118 of file KDChartAbstractDiagram.cpp.

```
119 {
120     delete _d;
121 }
```

## 9.7.3 Member Function Documentation

### 9.7.3.1 bool AbstractDiagram::allowOverlappingDataValueTexts () const

#### Returns:

Whether data value labels are allowed to overlap.

Definition at line 440 of file KDChartAbstractDiagram.cpp.

References [d](#).

Referenced by [compare\(\)](#).

```
441 {
442     return d->allowOverlappingDataValueTexts;
443 }
```

### 9.7.3.2 bool AbstractDiagram::antiAliasing () const

#### Returns:

Whether anti-aliasing is to be used for rendering this diagram.

Definition at line 451 of file KDChartAbstractDiagram.cpp.

References [d](#).

Referenced by [compare\(\)](#).

```
452 {
453     return d->antiAliasing;
454 }
```

### 9.7.3.3 [AttributesModel](#) \* AbstractDiagram::attributesModel () const [virtual]

Returns the [AttributesModel](#), that is used by this diagram.

By default each diagram owns its own [AttributesModel](#), which should never be deleted. Only if a user-supplied [AttributesModel](#) has been set does the pointer returned here not belong to the diagram.

**Returns:**

The [AttributesModel](#) associated with the diagram.

**See also:**

[setAttributesModel](#)

Definition at line 280 of file `KDChartAbstractDiagram.cpp`.

References d.

Referenced by `brush()`, `compare()`, `datasetBrushes()`, `datasetLabels()`, `datasetMarkers()`, `datasetPens()`, `dataValueAttributes()`, `isHidden()`, `itemRowLabels()`, `KDChart::RingDiagram::paint()`, `KDChart::PolarDiagram::paint()`, `pen()`, `KDChart::AbstractCartesianDiagram::setAttributesModel()`, `KDChart::BarDiagram::setBarAttributes()`, `setBrush()`, `KDChart::AbstractCartesianDiagram::setModel()`, `setPen()`, and `KDChart::TernaryLineDiagram::TernaryLineDiagram()`.

```
281 {
282     return d->attributesModel;
283 }
```

**9.7.3.4 QModelIndex AbstractDiagram::attributesModelRootIndex () const** `[protected]`

returns a `QModelIndex` pointing into the [AttributesModel](#) that corresponds to the root index of the diagram.

Definition at line 302 of file `KDChartAbstractDiagram.cpp`.

References d.

Referenced by `datasetBrushes()`, `datasetLabels()`, `datasetMarkers()`, `datasetPens()`, `itemRowLabels()`, `KDChart::Plotter::numberOfAbscissaSegments()`, `KDChart::LineDiagram::numberOfAbscissaSegments()`, `KDChart::BarDiagram::numberOfAbscissaSegments()`, `KDChart::Plotter::numberOfOrdinateSegments()`, `KDChart::LineDiagram::numberOfOrdinateSegments()`, `KDChart::BarDiagram::numberOfOrdinateSegments()`, `valueForCell()`, and `KDChart::LineDiagram::valueForCellTesting()`.

```
303 {
304     if ( !d->attributesModelRootIndex.isValid() )
305         d->attributesModelRootIndex = d->attributesModel->mapFromSource( rootIndex() );
306     return d->attributesModelRootIndex;
307 }
```

**9.7.3.5 QBrush AbstractDiagram::brush (const QModelIndex & index) const**

Retrieve the brush to be used, for painting the datapoint at the given index in the model.

**Parameters:**

*index* The index of the datapoint in the model.

**Returns:**

The brush to use for painting.

Definition at line 838 of file KDChartAbstractDiagram.cpp.

References `attributesModel()`, `brush()`, `KDChart::AttributesModel::data()`, `KDChart::DatasetBrushRole`, and `datasetDimension()`.

```

839 {
840     if( datasetDimension() > 1 )
841         return brush( index.column() );
842     return qVariantValue<QBrush>(
843         attributesModel()->data(
844             attributesModel()->mapFromSource( index ),
845             DatasetBrushRole ) );
846 }
```

### 9.7.3.6 QBrush AbstractDiagram::brush (int *dataset*) const

Retrieve the brush to be used for the given dataset.

This will fall back automatically to what was set at model level, if there are no dataset specific settings.

#### Parameters:

*dataset* The dataset to retrieve the brush for.

#### Returns:

The brush to use for painting.

Definition at line 828 of file KDChartAbstractDiagram.cpp.

References `attributesModel()`, `columnToIndex()`, `KDChart::AttributesModel::data()`, `KDChart::DatasetBrushRole`, and `datasetDimension()`.

```

829 {
830     if( datasetDimension() > 1 )
831         dataset /= datasetDimension();
832     return qVariantValue<QBrush>(
833         attributesModel()->data(
834             attributesModel()->mapFromSource( columnToIndex( dataset ) ),
835             DatasetBrushRole ) );
836 }
```

### 9.7.3.7 QBrush AbstractDiagram::brush () const

Retrieve the brush to be used for painting datapoints globally.

This will fall back automatically to the default settings, if there are no specific settings.

#### Returns:

The brush to use for painting.

Definition at line 822 of file KDChartAbstractDiagram.cpp.

References `attributesModel()`, `KDChart::AttributesModel::data()`, and `KDChart::DatasetBrushRole`.

Referenced by `brush()`, `KDChart::TernaryPointDiagram::paint()`, `KDChart::TernaryLineDiagram::paint()`, `KDChart::RingDiagram::paint()`, `KDChart::PolarDiagram::paint()`, and `paintMarker()`.

```

823 {
824     return qVariantValue<QBrush>(
825         attributesModel()->data( DatasetBrushRole ) );
826 }

```

#### 9.7.3.8 virtual const QPair<QPointF, QPointF> KDChart::AbstractDiagram::calculateDataBoundaries() const [protected, pure virtual]

Implemented in [KDChart::BarDiagram](#), [KDChart::LineDiagram](#), [KDChart::PieDiagram](#), [KDChart::Plotter](#), [KDChart::PolarDiagram](#), [KDChart::RingDiagram](#), [KDChart::AbstractTernaryDiagram](#), [KDChart::TernaryLineDiagram](#), and [KDChart::TernaryPointDiagram](#).

Referenced by `dataBoundaries()`.

#### 9.7.3.9 bool AbstractDiagram::checkInvariants (bool *justReturnTheStatus* = false) const [protected, virtual]

Definition at line 1060 of file `KDChartAbstractDiagram.cpp`.

References `coordinatePlane()`.

Referenced by `KDChart::RingDiagram::calculateDataBoundaries()`, `KDChart::PolarDiagram::calculateDataBoundaries()`, `KDChart::Plotter::calculateDataBoundaries()`, `KDChart::PieDiagram::calculateDataBoundaries()`, `KDChart::LineDiagram::calculateDataBoundaries()`, `KDChart::BarDiagram::calculateDataBoundaries()`, `KDChart::RingDiagram::paint()`, `KDChart::PolarDiagram::paint()`, `KDChart::Plotter::paint()`, `KDChart::PieDiagram::paint()`, `KDChart::LineDiagram::paint()`, `KDChart::BarDiagram::paint()`, `paintDataValueTexts()`, `paintMarker()`, and `paintMarkers()`.

```

1061 {
1062     if( ! justReturnTheStatus ){
1063         Q_ASSERT_X ( model(), "AbstractDiagram::checkInvariants()",
1064             "There is no usable model set, for the diagram." );
1065
1066         Q_ASSERT_X ( coordinatePlane(), "AbstractDiagram::checkInvariants()",
1067             "There is no usable coordinate plane set, for the diagram." );
1068     }
1069     return model() && coordinatePlane();
1070 }

```

#### 9.7.3.10 QModelIndex AbstractDiagram::columnToIndex (int *column*) const [protected]

Definition at line 309 of file `KDChartAbstractDiagram.cpp`.

Referenced by `KDChart::BarDiagram::barAttributes()`, `brush()`, `dataValueAttributes()`, `isHidden()`, `KDChart::Plotter::lineAttributes()`, `KDChart::LineDiagram::lineAttributes()`, `pen()`, `KDChart::AbstractPieDiagram::pieAttributes()`, `KDChart::BarDiagram::threeDBarAttributes()`, `KDChart::Plotter::threeDLineAttributes()`, `KDChart::LineDiagram::threeDLineAttributes()`, and `KDChart::AbstractPieDiagram::threeDPieAttributes()`.

```

310 {    // FIXME (Mirko): shouldn't this be headerData? instead of the index for the first row?
311     if( model() )
312         return QModelIndex( model()->index( 0, column, rootIndex() ) );
313     return QModelIndex();
314 }

```

**9.7.3.11 bool AbstractDiagram::compare (const AbstractDiagram \* other) const**

Returns true if both diagrams have the same settings.

Definition at line 129 of file KDChartAbstractDiagram.cpp.

References `allowOverlappingDataValueTexts()`, `antiAliasing()`, `attributesModel()`, `KDChart::AttributesModel::compare()`, `datasetDimension()`, and `percentMode()`.

```

130 {
131     if( other == this ) return true;
132     if( ! other ){
133         //qDebug() << "AbstractDiagram::compare() cannot compare to Null pointer";
134         return false;
135     }
136     /*
137     qDebug() << "\n                AbstractDiagram::compare() QAbstractScrollArea:";
138         // compare QAbstractScrollArea properties
139     qDebug() <<
140         ((horizontalScrollBarPolicy() == other->horizontalScrollBarPolicy()) &&
141         (verticalScrollBarPolicy()   == other->verticalScrollBarPolicy()));
142     qDebug() << "AbstractDiagram::compare() QFrame:";
143         // compare QFrame properties
144     qDebug() <<
145         ((frameShadow() == other->frameShadow()) &&
146         (frameShape()   == other->frameShape()) &&
147         (frameWidth()   == other->frameWidth()) &&
148         (lineWidth()    == other->lineWidth()) &&
149         (midLineWidth() == other->midLineWidth()));
150     qDebug() << "AbstractDiagram::compare() QAbstractItemView:";
151         // compare QAbstractItemView properties
152     qDebug() <<
153         ((alternatingRowColors() == other->alternatingRowColors()) &&
154         (hasAutoScroll()         == other->hasAutoScroll()) &&
155 #if QT_VERSION > 0x040199
156         (dragDropMode()          == other->dragDropMode()) &&
157         (dragDropOverwriteMode() == other->dragDropOverwriteMode()) &&
158         (horizontalScrollMode()  == other->horizontalScrollMode()) &&
159         (verticalScrollMode()    == other->verticalScrollMode()) &&
160 #endif
161         (dragEnabled()           == other->dragEnabled()) &&
162         (editTriggers()          == other->editTriggers()) &&
163         (iconSize()              == other->iconSize()) &&
164         (selectionBehavior()     == other->selectionBehavior()) &&
165         (selectionMode()         == other->selectionMode()) &&
166         (showDropIndicator()     == other->showDropIndicator()) &&
167         (tabKeyNavigation()      == other->tabKeyNavigation()) &&
168         (textElideMode()         == other->textElideMode()));
169     qDebug() << "AbstractDiagram::compare() AttributesModel: ";
170         // compare all of the properties stored in the attributes model
171     qDebug() << attributesModel()->compare( other->attributesModel() );
172     qDebug() << "AbstractDiagram::compare() own:";
173         // compare own properties
174     qDebug() <<
175         ((rootIndex().column() == other->rootIndex().column()) &&
176         (rootIndex().row()     == other->rootIndex().row()) &&
177         (allowOverlappingDataValueTexts() == other->allowOverlappingDataValueTexts()) &&
178         (antiAliasing()          == other->antiAliasing()) &&
179         (percentMode()           == other->percentMode()) &&
180         (datasetDimension()      == other->datasetDimension()));
181     */
182     return // compare QAbstractScrollArea properties
183         (horizontalScrollBarPolicy() == other->horizontalScrollBarPolicy()) &&
184         (verticalScrollBarPolicy()   == other->verticalScrollBarPolicy()) &&
185         // compare QFrame properties
186         (frameShadow() == other->frameShadow()) &&
187         (frameShape()  == other->frameShape()) &&

```

```

188 // frameWidth is a read-only property defined by the style, it should not be in here:
189         // (frameWidth() == other->frameWidth()) &&
190         (lineWidth() == other->lineWidth()) &&
191         (midLineWidth() == other->midLineWidth()) &&
192         // compare QAbstractItemView properties
193         (alternatingRowColors() == other->alternatingRowColors()) &&
194         (hasAutoScroll() == other->hasAutoScroll()) &&
195 #if QT_VERSION > 0x040199
196         (dragDropMode() == other->dragDropMode()) &&
197         (dragDropOverwriteMode() == other->dragDropOverwriteMode()) &&
198         (horizontalScrollMode() == other->horizontalScrollMode()) &&
199         (verticalScrollMode() == other->verticalScrollMode()) &&
200 #endif
201         (dragEnabled() == other->dragEnabled()) &&
202         (editTriggers() == other->editTriggers()) &&
203         (iconSize() == other->iconSize()) &&
204         (selectionBehavior() == other->selectionBehavior()) &&
205         (selectionMode() == other->selectionMode()) &&
206         (showDropIndicator() == other->showDropIndicator()) &&
207         (tabKeyNavigation() == other->tabKeyNavigation()) &&
208         (textElideMode() == other->textElideMode()) &&
209         // compare all of the properties stored in the attributes model
210         attributesModel()->compare( other->attributesModel() ) &&
211         // compare own properties
212         (rootIndex().column() == other->rootIndex().column()) &&
213         (rootIndex().row() == other->rootIndex().row()) &&
214         (allowOverlappingDataValueTexts() == other->allowOverlappingDataValueTexts()) &&
215         (antiAliasing() == other->antiAliasing()) &&
216         (percentMode() == other->percentMode()) &&
217         (datasetDimension() == other->datasetDimension());
218 }

```

### 9.7.3.12 [AbstractCoordinatePlane](#) \* [AbstractDiagram::coordinatePlane\(\)](#) const

The coordinate plane associated with the diagram.

This determines how coordinates in value space are mapped into pixel space. By default this is a [Cartesian-CoordinatePlane](#).

#### Returns:

The coordinate plane associated with the diagram.

Definition at line 220 of file `KDChartAbstractDiagram.cpp`.

References d.

Referenced by `checkInvariants()`, `KDChart::AbstractCartesianDiagram::layoutPlanes()`, `KDChart::PolarDiagram::paint()`, `paintDataValueTexts()`, `paintMarkers()`, `KDChart::AbstractPolarDiagram::polarCoordinatePlane()`, and `KDChart::AbstractCartesianDiagram::setCoordinatePlane()`.

```

221 {
222     return d->plane;
223 }

```

### 9.7.3.13 `const QPair< QPointF, QPointF > AbstractDiagram::dataBoundaries()` const

Return the bottom left and top right data point, that the diagram will display (unless the grid adjusts these values).



This method returns a cached result of calculations done by `calculateDataBoundaries`. Classes derived from [AbstractDiagram](#) must implement the `calculateDataBoundaries` function, to specify their own way of calculating the data boundaries. If derived classes want to force recalculation of the data boundaries, they can call [setDataBoundariesDirty\(\)](#)

Returned value is in diagram coordinates.

Definition at line 225 of file `KDChartAbstractDiagram.cpp`.

References `calculateDataBoundaries()`, and `d`.

Referenced by `KDChart::CartesianCoordinatePlane::getRawDataBoundingRectFromDiagrams()`, `KDChart::PolarCoordinatePlane::layoutDiagrams()`, `KDChart::Plotter::paint()`, `KDChart::LineDiagram::paint()`, and `KDChart::BarDiagram::paint()`.

```

226 {
227     if( d->databoundariesDirty ){
228         d->databoundaries = calculateDataBoundaries ();
229         d->databoundariesDirty = false;
230     }
231     return d->databoundaries;
232 }
```

#### 9.7.3.14 void AbstractDiagram::dataChanged (const QModelIndex & *topLeft*, const QModelIndex & *bottomRight*) [virtual]

[reimplemented]

Definition at line 330 of file `KDChartAbstractDiagram.cpp`.

References `setDataBoundariesDirty()`.

```

332 {
333     Q_UNUSED( topLeft );
334     Q_UNUSED( bottomRight );
335     // We are still too dumb to do intelligent updates...
336     setDataBoundariesDirty();
337     scheduleDelayedItemsLayout();
338 }
```

#### 9.7.3.15 void KDChart::AbstractDiagram::dataHidden () [signal]

This signal is emitted, when the hidden status of at least one data cell was (un)set.

Referenced by `setHidden()`.

#### 9.7.3.16 QList< QBrush > AbstractDiagram::datasetBrushes () const

The set of dataset brushes currently used, for use in legends, etc.

##### Note:

Cell-level override brushes, if set, take precedence over the dataset values, so you might need to check these too, in order to find the brush, that is used for a single cell.

##### Returns:

The current set of dataset brushes.

Definition at line 1018 of file KDChartAbstractDiagram.cpp.

References `attributesModel()`, `attributesModelRootIndex()`, `KDChart::AttributesModel::columnCount()`, `KDChart::DatasetBrushRole`, and `datasetDimension()`.

Referenced by `KDChart::Legend::setBrushesFromDiagram()`.

```

1019 {
1020     QList<QBrush> ret;
1021     if( model() == 0 )
1022         return ret;
1023
1024     const int columnCount = attributesModel()->columnCount(attributesModelRootIndex());
1025     for( int i = 0; i < columnCount / datasetDimension(); ++i )
1026         ret << QVariantValue<QBrush>( attributesModel()->headerData( i, Qt::Vertical, DatasetBrushRole ),
1027                                     QVariant::Brush );
1028     return ret;
1029 }
```

### 9.7.3.17 `int AbstractDiagram::datasetDimension () const`

The dataset dimension of a diagram determines, how many value dimensions it expects each datapoint to have.

For each dimension it will expect one column of values in the model. If the dimensionality is 1, automatic values will be used for the abscissa.

For example a diagram with the default dimension of 1, will have one column per datapoint (the y values) and will use automatic values for the x axis (1, 2, 3, ... n). If the dimension is 2, the diagram will use the first, (and the third, fifth, etc) columns as X values, and the second, (and the fourth, sixth, etc) column as Y values.

#### Returns:

The dataset dimension of the diagram.

Definition at line 1072 of file KDChartAbstractDiagram.cpp.

References `d`.

Referenced by `brush()`, `compare()`, `datasetBrushes()`, `datasetLabels()`, `datasetMarkers()`, `datasetPens()`, `KDChart::LineDiagram::getCellValues()`, `KDChart::CartesianCoordinatePlane::getDataDimensionsList()`, `KDChart::TernaryPointDiagram::paint()`, `KDChart::TernaryLineDiagram::paint()`, `paintDataValueTexts()`, `paintMarkers()`, `pen()`, `setPen()`, `KDChart::Plotter::setType()`, and `KDChart::LineDiagram::setType()`.

```

1073 {
1074     return d->datasetDimension;
1075 }
```

### 9.7.3.18 `QStringList AbstractDiagram::datasetLabels () const`

The set of dataset labels currently displayed, for use in legends, etc.

#### Returns:

The set of dataset labels currently displayed.

Definition at line 1005 of file KDChartAbstractDiagram.cpp.

References `attributesModel()`, `attributesModelRootIndex()`, `KDChart::AttributesModel::columnCount()`, `datasetDimension()`, and `KDChart::AttributesModel::headerData()`.

```

1006 {
1007     QStringList ret;
1008     if( model() == 0 )
1009         return ret;
1010
1011     const int columnCount = attributesModel()->columnCount(attributesModelRootIndex());
1012     for( int i = 0; i < columnCount / datasetDimension(); ++i )
1013         ret << attributesModel()->headerData( i, Qt::Horizontal, Qt::DisplayRole ).toString();
1014
1015     return ret;
1016 }
```

### 9.7.3.19 QList< [MarkerAttributes](#) > AbstractDiagram::datasetMarkers () const

The set of dataset markers currently used, for use in legends, etc.

#### Note:

Cell-level override markers, if set, take precedence over the dataset values, so you might need to check these too, in order to find the marker, that is shown for a single cell.

#### Returns:

The current set of dataset brushes.

Definition at line 1044 of file KDChartAbstractDiagram.cpp.

References `attributesModel()`, `attributesModelRootIndex()`, `KDChart::AttributesModel::columnCount()`, `datasetDimension()`, and `KDChart::DataValueLabelAttributesRole`.

```

1045 {
1046     QList<MarkerAttributes> ret;
1047     if( model() == 0 )
1048         return ret;
1049
1050     const int columnCount = attributesModel()->columnCount(attributesModelRootIndex());
1051     for( int i = 0; i < columnCount / datasetDimension(); ++i )
1052     {
1053         const DataValueAttributes a =
1054             qVariantValue<DataValueAttributes>( attributesModel()->headerData( i, Qt::Vertical, DataValueLabelAttributesRole ) );
1055         ret << a.markerAttributes();
1056     }
1057     return ret;
1058 }
```

### 9.7.3.20 QList< QPen > AbstractDiagram::datasetPens () const

The set of dataset pens currently used, for use in legends, etc.

#### Note:

Cell-level override pens, if set, take precedence over the dataset values, so you might need to check these too, in order to find the pens, that is used for a single cell.

**Returns:**

The current set of dataset pens.

Definition at line 1031 of file KDChartAbstractDiagram.cpp.

References `attributesModel()`, `attributesModelRootIndex()`, `KDChart::AttributesModel::columnCount()`, `datasetDimension()`, and `KDChart::DatasetPenRole`.

```

1032 {
1033     QList<QPen> ret;
1034     if( model() == 0 )
1035         return ret;
1036
1037     const int columnCount = attributesModel()->columnCount(attributesModelRootIndex());
1038     for( int i = 0; i < columnCount / datasetDimension(); ++i )
1039         ret << qVariantValue<QPen>( attributesModel()->headerData( i, Qt::Vertical, DatasetPenRole ) );
1040
1041     return ret;
1042 }
```

#### 9.7.3.21 **DataValueAttributes** AbstractDiagram::dataValueAttributes (const QModelIndex & *index*) const

Retrieve the **DataValueAttributes** for the given index.

This will fall back automatically to what was set at dataset or model level, if there are no datapoint specific settings.

**Parameters:**

*index* The datapoint to retrieve the attributes for.

**Returns:**

The **DataValueAttributes** for the given index.

Definition at line 421 of file KDChartAbstractDiagram.cpp.

References `attributesModel()`, `KDChart::AttributesModel::data()`, and `KDChart::DataValueLabelAttributesRole`.

```

422 {
423     return qVariantValue<DataValueAttributes>(
424         attributesModel()->data( attributesModel()->mapFromSource(index),
425                                 KDChart::DataValueLabelAttributesRole ) );
426 }
```

#### 9.7.3.22 **DataValueAttributes** AbstractDiagram::dataValueAttributes (int *column*) const

Retrieve the **DataValueAttributes** for the given dataset.

This will fall back automatically to what was set at model level, if there are no dataset specific settings.

**Parameters:**

*column* The dataset to retrieve the attributes for.

**Returns:**

The [DataValueAttributes](#) for the given dataset.

Definition at line 414 of file KDChartAbstractDiagram.cpp.

References [attributesModel\(\)](#), [columnToIndex\(\)](#), [KDChart::AttributesModel::data\(\)](#), and [KDChart::DataValueLabelAttributesRole](#).

```
415 {
416     return qVariantValue<DataValueAttributes>(
417         attributesModel()->data( attributesModel()->mapFromSource(columnToIndex( column )),
418         KDChart::DataValueLabelAttributesRole ) );
419 }
```

**9.7.3.23 [DataValueAttributes](#) AbstractDiagram::dataValueAttributes () const**

Retrieve the [DataValueAttributes](#) specified globally.

This will fall back automatically to the default settings, if there are no specific settings.

**Returns:**

The global [DataValueAttributes](#).

Definition at line 408 of file KDChartAbstractDiagram.cpp.

References [attributesModel\(\)](#), [KDChart::DataValueLabelAttributesRole](#), and [KDChart::AttributesModel::modelData\(\)](#).

Referenced by [paintDataValueText\(\)](#), [paintMarker\(\)](#), and [KDChart::TernaryLineDiagram::TernaryLineDiagram\(\)](#).

```
409 {
410     return qVariantValue<DataValueAttributes>(
411         attributesModel()->modelData( KDChart::DataValueLabelAttributesRole ) );
412 }
```

**9.7.3.24 void AbstractDiagram::doItemsLayout () [virtual]**

[reimplemented]

Definition at line 321 of file KDChartAbstractDiagram.cpp.

References [d](#), and [update\(\)](#).

```
322 {
323     if ( d->plane ) {
324         d->plane->layoutDiagrams();
325         update();
326     }
327     QAbstractItemView::doItemsLayout();
328 }
```

**9.7.3.25 int AbstractDiagram::horizontalOffset () const** [virtual]

[reimplemented]

Definition at line 950 of file KDChartAbstractDiagram.cpp.

```
951 { return 0; }
```

**9.7.3.26 QModelIndex AbstractDiagram::indexAt (const QPoint & *point*) const** [virtual]

[reimplemented]

Definition at line 1098 of file KDChartAbstractDiagram.cpp.

References d.

```
1099 {
1100     return d->indexAt( point );
1101 }
```

**9.7.3.27 QModelIndexList AbstractDiagram::indexesAt (const QPoint & *point*) const**

This method is added alongside with indexAt from QAIM, since in kdchart multiple indexes can be displayed at the same spot.

Definition at line 1103 of file KDChartAbstractDiagram.cpp.

References d.

```
1104 {
1105     return d->indexesAt( point );
1106 }
```

**9.7.3.28 bool AbstractDiagram::isHidden (const QModelIndex & *index*) const**

Retrieve the hidden status for the given index.

This will fall back automatically to what was set at dataset or diagram level, if there are no datapoint specific settings.

**Parameters:**

***index*** The datapoint to retrieve the hidden status for.

**Returns:**

The hidden status for the given index.

Definition at line 380 of file KDChartAbstractDiagram.cpp.

References attributesModel(), KDChart::AttributesModel::data(), and KDChart::DataHiddenRole.

```
381 {
382     return qVariantValue<bool>(
383         attributesModel()->data(
384             attributesModel()->mapFromSource(index),
385             DataHiddenRole ) );
386 }
```

**9.7.3.29 bool AbstractDiagram::isHidden (int *column*) const**

Retrieve the hidden status for the given dataset.

This will fall back automatically to what was set at diagram level, if there are no dataset specific settings.

**Parameters:**

*column* The dataset to retrieve the hidden status for.

**Returns:**

The hidden status for the given dataset.

Definition at line 373 of file KDChartAbstractDiagram.cpp.

References attributesModel(), columnToIndex(), KDChart::AttributesModel::data(), and KDChart::DataHiddenRole.

```

374 {
375     return qVariantValue<bool>(
376         attributesModel()->data(
377             attributesModel()->mapFromSource(columnToIndex( column )),
378             DataHiddenRole ) );
379 }
```

**9.7.3.30 bool AbstractDiagram::isHidden () const**

Retrieve the hidden status specified globally.

This will fall back automatically to the default settings ( = not hidden), if there are no specific settings.

**Returns:**

The global hidden status.

Definition at line 367 of file KDChartAbstractDiagram.cpp.

References attributesModel(), KDChart::DataHiddenRole, and KDChart::AttributesModel::modelData().

Referenced by KDChart::LineDiagram::valueForCellTesting().

```

368 {
369     return qVariantValue<bool>(
370         attributesModel()->modelData( DataHiddenRole ) );
371 }
```

**9.7.3.31 bool AbstractDiagram::isIndexHidden (const QModelIndex & *index*) const** [virtual]

[reimplemented]

Definition at line 956 of file KDChartAbstractDiagram.cpp.

```

957 { return true; }
```

### 9.7.3.32 QStringList AbstractDiagram::itemRowLabels () const

The set of item row labels currently displayed, for use in Abscissa axes, etc.

#### Returns:

The set of item row labels currently displayed.

Definition at line 989 of file KDChartAbstractDiagram.cpp.

References attributesModel(), attributesModelRootIndex(), KDChart::AttributesModel::rowCount(), unitPrefix(), and unitSuffix().

```

990 {
991     QStringList ret;
992     if( model() ){
993         //qDebug() << "AbstractDiagram::itemRowLabels(): " << attributesModel()->rowCount(attributesModelRootIndex());
994         const int rowCount = attributesModel()->rowCount(attributesModelRootIndex());
995         for( int i = 0; i < rowCount; ++i ){
996             //qDebug() << "item row label: " << attributesModel()->headerData( i, Qt::Vertical, Qt::DisplayRole ).toString();
997             ret << unitPrefix( i, Qt::Horizontal, true ) +
998                 attributesModel()->headerData( i, Qt::Vertical, Qt::DisplayRole ).toString() +
999                 unitSuffix( i, Qt::Horizontal, true );
1000         }
1001     }
1002     return ret;
1003 }
```

### 9.7.3.33 void KDChart::AbstractDiagram::layoutChanged (AbstractDiagram \*) [signal]

Diagrams are supposed to emit this signal, when the layout of one of their element changes.

Layouts can change, for example, when axes are added or removed, or when the configuration was changed in a way that the axes or the diagram itself are displayed in a different geometry. Changes in the diagrams coordinate system also result in the [layoutChanged\(\)](#) signal being emitted.

Referenced by setDatasetDimension(), KDChart::AbstractPieDiagram::setPieAttributes(), KDChart::AbstractPieDiagram::setThreeDPieAttributes(), KDChart::Plotter::setType(), KDChart::LineDiagram::setType(), and KDChart::BarDiagram::setType().

### 9.7.3.34 void KDChart::AbstractDiagram::modelsChanged () [signal]

This signal is emitted, when either the model or the [AttributesModel](#) is replaced.

Referenced by setAttributesModel(), and setModel().

### 9.7.3.35 QModelIndex AbstractDiagram::moveCursor (CursorAction cursorAction, Qt::KeyboardModifiers modifiers) [virtual]

[reimplemented]

Definition at line 947 of file KDChartAbstractDiagram.cpp.

```

948 { return QModelIndex(); }
```



### 9.7.3.36 virtual void KDChart::AbstractDiagram::paint ([PaintContext](#) \* *paintContext*) [pure virtual]

Draw the diagram contents to the rectangle and painter, that are passed in as part of the paint context.

#### Parameters:

*paintContext* All information needed for painting.

Implemented in [KDChart::BarDiagram](#), [KDChart::LineDiagram](#), [KDChart::PieDiagram](#), [KDChart::Plotter](#), [KDChart::PolarDiagram](#), [KDChart::RingDiagram](#), [KDChart::TernaryLineDiagram](#), and [KDChart::TernaryPointDiagram](#).

### 9.7.3.37 void AbstractDiagram::paintDataValueText (QPainter \* *painter*, const QModelIndex & *index*, const QPointF & *pos*, double *value*)

Definition at line 468 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::DataValueAttributes::dataLabel\(\)](#), [dataValueAttributes\(\)](#), [KDChart::DataValueAttributes::decimalDigits\(\)](#), [KDChart::DataValueAttributes::isVisible\(\)](#), [KDChartEnums::MeasureOrientationMinimum](#), [KDChart::DataValueAttributes::position\(\)](#), [KDChart::DataValueAttributes::prefix\(\)](#), [KDChart::DataValueAttributes::suffix\(\)](#), and [KDChart::DataValueAttributes::textAttributes\(\)](#).

Referenced by [KDChart::RingDiagram::paint\(\)](#), [KDChart::PolarDiagram::paint\(\)](#), and [paintDataValueTexts\(\)](#).

```

472 {
473     // paint one data series
474     const DataValueAttributes a( dataValueAttributes(index) );
475     if ( !a.isVisible() ) return;
476
477     // handle decimal digits
478     int decimalDigits = a.decimalDigits();
479     int decimalPos = QString::number( value ).indexOf( QLatin1Char( '.' ) );
480     QString roundedValue;
481     if ( a.dataLabel().isNull() ) {
482         if ( decimalPos > 0 && value != 0 )
483             roundedValue = roundValues( value, decimalPos, decimalDigits );
484         else
485             roundedValue = QString::number( value );
486     } else
487         roundedValue = a.dataLabel();
488     // handle prefix and suffix
489     if ( !a.prefix().isNull() )
490         roundedValue.prepend( a.prefix() );
491
492     if ( !a.suffix().isNull() )
493         roundedValue.append( a.suffix() );
494
495     const TextAttributes ta( a.textAttributes() );
496     // FIXME draw the non-text bits, background, etc
497     if ( ta.isVisible() ) {
498         QPointF pt( pos );
499         /* for debugging:
500         PainterSaver painterSaver( painter );
501         painter->setPen( Qt::black );
502         painter->drawLine( pos - QPointF( 1,1), pos + QPointF( 1,1) );
503         painter->drawLine( pos - QPointF(-1,1), pos + QPointF(-1,1) );
504         */
505     }
506 }
```

```

507     QTextDocument doc;
508     if( Qt::mightBeRichText( roundedValue ) )
509         doc.setHtml( roundedValue );
510     else
511         doc.setPlainText( roundedValue );
512
513     const RelativePosition relPos( a.position( value >= 0.0 ) );
514     const Qt::Alignment alignBottomLeft = Qt::AlignBottom | Qt::AlignLeft;
515     const QFont calculatedFont( ta.calculatedFont( d->plane, KDChartEnums::MeasureOrientationMinimum );
516     const QRectF boundRect( d->cachedFontMetrics( calculatedFont, painter->device() )->boundingRect( roundedValue );
517
518     // To place correctly
519     pt.ry() -= boundRect.height();
520
521     //qDebug() << "calculatedFont's point size:" << calculatedFont.pointSizeF();
522     // adjust the text start point position, if alignment is not Bottom/Left
523     if( (relPos.alignment() & alignBottomLeft) != alignBottomLeft ){
524         if( relPos.alignment() & Qt::AlignRight )
525             pt.rx() -= boundRect.width();
526         else if( relPos.alignment() & Qt::AlignHCenter )
527             pt.rx() -= 0.5 * boundRect.width();
528
529         if( relPos.alignment() & Qt::AlignTop )
530             pt.ry() += boundRect.height();
531         else if( relPos.alignment() & Qt::AlignVCenter )
532             pt.ry() += 0.5 * boundRect.height();
533     }
534
535     // FIXME draw the non-text bits, background, etc
536
537     if ( a.showRepetitiveDataLabels() ||
538         pos.x() <= d->lastX ||
539         d->lastRoundedValue != roundedValue ) {
540         d->lastRoundedValue = roundedValue;
541         d->lastX = pos.x();
542         PainterSaver painterSaver( painter );
543
544         doc.setDefaultFont( calculatedFont );
545         QAbstractTextDocumentLayout::PaintContext context;
546         context.palette = palette();
547         context.palette.setColor(QPalette::Text, ta.pen().color() );
548
549         painter->translate( pt );
550         painter->rotate( ta.rotation() );
551
552         QAbstractTextDocumentLayout* layout = doc.documentLayout();
553         layout->draw( painter, context );
554     }
555 }
556 }

```

### 9.7.3.38 void AbstractDiagram::paintDataValueTexts (QPainter \* *painter*) [protected, virtual]

Definition at line 584 of file KDChartAbstractDiagram.cpp.

References [checkInvariants\(\)](#), [coordinatePlane\(\)](#), [datasetDimension\(\)](#), [paintDataValueText\(\)](#), and [KDChart::AbstractCoordinatePlane::translate\(\)](#).

```

585 {
586     if ( !checkInvariants() ) return;
587     const int rowCount = model()->rowCount( rootIndex() );
588     const int columnCount = model()->columnCount( rootIndex() );
589     for ( int i=datasetDimension()-1; i<columnCount; i += datasetDimension() ) {

```

```

590         for ( int j=0; j< rowCount; ++j ) {
591             const QModelIndex index = model()->index( j, i, rootIndex() );
592             double value = model()->data( index ).toDouble();
593             const QPointF pos = coordinatePlane()->translate( QPointF( j, value ) );
594             paintDataValueText( painter, index, pos, value );
595         }
596     }
597 }

```

### 9.7.3.39 void AbstractDiagram::paintMarker (QPainter \* *painter*, const QModelIndex & *index*, const QPointF & *pos*)

Definition at line 600 of file KDChartAbstractDiagram.cpp.

References [brush\(\)](#), [checkInvariants\(\)](#), [d](#), [dataValueAttributes\(\)](#), [KDChart::MarkerAttributes::isVisible\(\)](#), [KDChart::DataValueAttributes::isVisible\(\)](#), [KDChart::DataValueAttributes::markerAttributes\(\)](#), [KDChart::MarkerAttributes::markerColor\(\)](#), [KDChart::MarkerAttributes::markerSize\(\)](#), [paintMarker\(\)](#), and [KDChart::MarkerAttributes::pen\(\)](#).

```

603 {
604     if ( !checkInvariants() ) return;
605     DataValueAttributes a = dataValueAttributes(index);
606     if ( !a.isVisible() ) return;
607     const MarkerAttributes &ma = a.markerAttributes();
608     if ( !ma.isVisible() ) return;
609
610     PainterSaver painterSaver( painter );
611     QSizeF maSize( ma.markerSize() );
612     QBrush indexBrush( brush( index ) );
613     QPen indexPen( ma.pen() );
614     if ( ma.markerColor().isValid() )
615         indexBrush.setColor( ma.markerColor() );
616
617     paintMarker( painter, ma, indexBrush, indexPen, pos, maSize );
618
619     // workaround: BC cannot be changed, otherwise we would pass the
620     // index down to next-lower paintMarker function. So far, we
621     // basically save a circle of radius maSize at pos in the
622     // reverseMapper. This means that ^^^ this version of paintMarker
623     // needs to be called to reverse-map the marker.
624     d->reverseMapper.addCircle( index.row(), index.column(), pos, 2 * maSize );
625 }

```

### 9.7.3.40 void AbstractDiagram::paintMarker (QPainter \* *painter*, const [MarkerAttributes](#) & *markerAttributes*, const QBrush & *brush*, const QPen & *pen*, const QPointF & *point*, const QSizeF & *size*) [virtual]

Definition at line 627 of file KDChartAbstractDiagram.cpp.

References [KDChart::MarkerAttributes::Marker1Pixel](#), [KDChart::MarkerAttributes::Marker4Pixels](#), [KDChart::MarkerAttributes::MarkerCircle](#), [KDChart::MarkerAttributes::MarkerCross](#), [KDChart::MarkerAttributes::MarkerDiamond](#), [KDChart::MarkerAttributes::MarkerFastCross](#), [KDChart::MarkerAttributes::MarkerRing](#), [KDChart::MarkerAttributes::MarkerSquare](#), and [KDChart::MarkerAttributes::markerStyle\(\)](#).

Referenced by [KDChart::TernaryPointDiagram::paint\(\)](#), [KDChart::TernaryLineDiagram::paint\(\)](#), [KDChart::MarkerLayoutItem::paintIntoRect\(\)](#), [paintMarker\(\)](#), and [paintMarkers\(\)](#).

```

633 {

```

```

634     const QPen oldPen( painter->pen() );
635     // Pen is used to paint 4Pixels - 1 Pixel - Ring and FastCross types.
636     // make sure to use the brush color - see above in those cases.
637     const bool isFourPixels = (markerAttributes.markerStyle() == MarkerAttributes::Marker4Pixels);
638     if( isFourPixels || (markerAttributes.markerStyle() == MarkerAttributes::Marker1Pixel) ){
639         // for high-performance point charts with tiny point markers:
640         painter->setPen( QPen( brush.color().light() ) );
641         if( isFourPixels ){
642             const qreal x = pos.x();
643             const qreal y = pos.y();
644             painter->drawLine( QPointF(x-1.0,y-1.0),
645                               QPointF(x+1.0,y-1.0) );
646             painter->drawLine( QPointF(x-1.0,y),
647                               QPointF(x+1.0,y) );
648             painter->drawLine( QPointF(x-1.0,y+1.0),
649                               QPointF(x+1.0,y+1.0) );
650         }
651         painter->drawPoint( pos );
652     }else{
653         PainterSaver painterSaver( painter );
654         // we only a solid line surrounding the markers
655         QPen painterPen( pen );
656         painterPen.setStyle( Qt::SolidLine );
657         painter->setPen( painterPen );
658         painter->setBrush( brush );
659         painter->setRenderHint ( QPainter::Antialiasing );
660         painter->translate( pos );
661         switch ( markerAttributes.markerStyle() ) {
662             case MarkerAttributes::MarkerCircle:
663                 painter->drawEllipse( QRectF( 0 - maSize.height()/2, 0 - maSize.width()/2,
664                                                maSize.height(), maSize.width() ) );
665                 break;
666             case MarkerAttributes::MarkerSquare:
667                 {
668                     QRectF rect( 0 - maSize.width()/2, 0 - maSize.height()/2,
669                                 maSize.width(), maSize.height() );
670                     painter->drawRect( rect );
671                     break;
672                 }
673             case MarkerAttributes::MarkerDiamond:
674                 {
675                     QVector<QPointF> diamondPoints;
676                     QPointF top, left, bottom, right;
677                     top = QPointF( 0, 0 - maSize.height()/2 );
678                     left = QPointF( 0 - maSize.width()/2, 0 );
679                     bottom = QPointF( 0, maSize.height()/2 );
680                     right = QPointF( maSize.width()/2, 0 );
681                     diamondPoints << top << left << bottom << right;
682                     painter->drawPolygon( diamondPoints );
683                     break;
684                 }
685             // both handled on top of the method:
686             case MarkerAttributes::Marker1Pixel:
687             case MarkerAttributes::Marker4Pixels:
688                 break;
689             case MarkerAttributes::MarkerRing:
690                 {
691                     painter->setPen( QPen( brush.color() ) );
692                     painter->setBrush( Qt::NoBrush );
693                     painter->drawEllipse( QRectF( 0 - maSize.height()/2, 0 - maSize.width()/2,
694                                                    maSize.height(), maSize.width() ) );
695                     break;
696                 }
697             case MarkerAttributes::MarkerCross:
698                 {
699                     QRectF rect( maSize.width()*-0.5, maSize.height()*-0.2,
700                                 maSize.width(), maSize.height()*0.4 );

```

```

701         painter->drawRect( rect );
702         rect.setTopLeft(QPointF( maSize.width()*-0.2, maSize.height()*-0.5 ));
703         rect.setSize(QSizeF( maSize.width()*0.4, maSize.height() ));
704         painter->drawRect( rect );
705         break;
706     }
707     case MarkerAttributes::MarkerFastCross:
708     {
709         QPointF left, right, top, bottom;
710         left = QPointF( -maSize.width()/2, 0 );
711         right = QPointF( maSize.width()/2, 0 );
712         top = QPointF( 0, -maSize.height()/2 );
713         bottom= QPointF( 0, maSize.height()/2 );
714         painter->setPen( QPen( brush.color() ) );
715         painter->drawLine( left, right );
716         painter->drawLine( top, bottom );
717         break;
718     }
719     default:
720         Q_ASSERT_X ( false, "paintMarkers()",
721                     "Type item does not match a defined Marker Type." );
722 }
723 }
724 painter->setPen( oldPen );
725 }

```

#### 9.7.3.41 void AbstractDiagram::paintMarkers (QPainter \* *painter*) [protected, virtual]

Definition at line 727 of file KDChartAbstractDiagram.cpp.

References [checkInvariants\(\)](#), [coordinatePlane\(\)](#), [datasetDimension\(\)](#), [paintMarker\(\)](#), and [KDChart::AbstractCoordinatePlane::translate\(\)](#).

```

728 {
729     if ( !checkInvariants() ) return;
730     const int rowCount = model()->rowCount( rootIndex() );
731     const int columnCount = model()->columnCount( rootIndex() );
732     for ( int i=datasetDimension()-1; i<columnCount; i += datasetDimension() ) {
733         for ( int j=0; j<rowCount; ++j ) {
734             const QModelIndex index = model()->index( j, i, rootIndex() );
735             double value = model()->data( index ).toDouble();
736             const QPointF pos = coordinatePlane()->translate( QPointF( j, value ) );
737             paintMarker( painter, index, pos );
738         }
739     }
740 }

```

#### 9.7.3.42 QPen AbstractDiagram::pen (const QModelIndex & *index*) const

Retrieve the pen to be used, for painting the datapoint at the given index in the model.

##### Parameters:

*index* The index of the datapoint in the model.

##### Returns:

The pen to use for painting.

Definition at line 788 of file KDChartAbstractDiagram.cpp.

References `attributesModel()`, `KDChart::AttributesModel::data()`, `datasetDimension()`, `KDChart::DatasetPenRole`, and `pen()`.

```

789 {
790     if( datasetDimension() > 1 )
791         return pen( index.column() );
792     return QVariantValue<QPen>(
793         attributesModel()->data(
794             attributesModel()->mapFromSource( index ),
795             DatasetPenRole ) );
796 }
```

#### 9.7.3.43 QPen AbstractDiagram::pen (int *dataset*) const

Retrieve the pen to be used for the given dataset.

This will fall back automatically to what was set at model level, if there are no dataset specific settings.

##### Parameters:

*dataset* The dataset to retrieve the pen for.

##### Returns:

The pen to use for painting.

Definition at line 777 of file KDChartAbstractDiagram.cpp.

References `attributesModel()`, `columnToIndex()`, `KDChart::AttributesModel::data()`, `datasetDimension()`, and `KDChart::DatasetPenRole`.

```

778 {
779     if( datasetDimension() > 1 )
780         dataset /= datasetDimension();
781
782     return QVariantValue<QPen>(
783         attributesModel()->data(
784             attributesModel()->mapFromSource( columnToIndex( dataset ) ),
785             DatasetPenRole ) );
786 }
```

#### 9.7.3.44 QPen AbstractDiagram::pen () const

Retrieve the pen to be used for painting datapoints globally.

This will fall back automatically to the default settings, if there are no specific settings.

##### Returns:

The pen to use for painting.

Definition at line 771 of file KDChartAbstractDiagram.cpp.

References `attributesModel()`, `KDChart::AttributesModel::data()`, and `KDChart::DatasetPenRole`.

Referenced by `KDChart::TernaryPointDiagram::paint()`, `KDChart::TernaryLineDiagram::paint()`, and `pen()`.

```

772 {
773     return qVariantValue<QPen>(
774         attributesModel()->data( DatasetPenRole ) );
775 }

```

#### 9.7.3.45 bool AbstractDiagram::percentMode () const

Definition at line 462 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by compare(), and KDChart::CartesianCoordinatePlane::getDataDimensionsList().

```

463 {
464     return d->percent;
465 }

```

#### 9.7.3.46 void KDChart::AbstractDiagram::propertiesChanged () [signal]

Emitted upon change of a property of the Diagram.

Referenced by KDChart::Plotter::resetLineAttributes(), KDChart::LineDiagram::resetLineAttributes(), setAllowOverlappingDataValueTexts(), setAntiAliasing(), KDChart::BarDiagram::setBarAttributes(), setBrush(), setDataValueAttributes(), KDChart::Plotter::setLineAttributes(), KDChart::LineDiagram::setLineAttributes(), setPen(), setPercentMode(), KDChart::BarDiagram::setThreeDBarAttributes(), KDChart::Plotter::setThreeDLineAttributes(), KDChart::LineDiagram::setThreeDLineAttributes(), KDChart::Plotter::setType(), KDChart::LineDiagram::setType(), KDChart::BarDiagram::setType(), KDChart::Plotter::setValueTrackerAttributes(), and KDChart::LineDiagram::setValueTrackerAttributes().

#### 9.7.3.47 virtual void KDChart::AbstractDiagram::resize (const QSizeF & area) [pure virtual]

Called by the widget's sizeEvent.

Adjust all internal structures, that are calculated, depending on the size of the widget.

**Parameters:**

*area*

Implemented in [KDChart::BarDiagram](#), [KDChart::LineDiagram](#), [KDChart::PieDiagram](#), [KDChart::Plotter](#), [KDChart::PolarDiagram](#), [KDChart::RingDiagram](#), [KDChart::TernaryLineDiagram](#), and [KDChart::TernaryPointDiagram](#).

Referenced by KDChart::CartesianCoordinatePlane::setGeometry().

#### 9.7.3.48 void AbstractDiagram::scrollTo (const QModelIndex & index, ScrollHint hint = EnsureVisible) [virtual]

[reimplemented]

Definition at line 942 of file KDChartAbstractDiagram.cpp.

```

943 {}

```

### 9.7.3.49 void AbstractDiagram::setAllowOverlappingDataValueTexts (bool *allow*)

Set whether data value labels are allowed to overlap.

#### Parameters:

*allow* True means that overlapping labels are allowed.

Definition at line 434 of file KDChartAbstractDiagram.cpp.

References [d](#), and [propertiesChanged\(\)](#).

```
435 {  
436     d->allowOverlappingDataValueTexts = allow;  
437     emit propertiesChanged();  
438 }
```

### 9.7.3.50 void AbstractDiagram::setAntiAliasing (bool *enabled*)

Set whether anti-aliasing is to be used while rendering this diagram.

#### Parameters:

*enabled* True means that AA is enabled.

Definition at line 445 of file KDChartAbstractDiagram.cpp.

References [d](#), and [propertiesChanged\(\)](#).

```
446 {  
447     d->antiAliasing = enabled;  
448     emit propertiesChanged();  
449 }
```

### 9.7.3.51 void AbstractDiagram::setAttributesModel ([AttributesModel](#) \* *model*) [virtual]

Associate an [AttributesModel](#) with this diagram.

Note that the diagram does *\_not\_* take ownership of the [AttributesModel](#). This should thus only be used with [AttributesModels](#) that have been explicitly created by the user, and are owned by her. Setting an [AttributesModel](#) that is internal to another diagram is an error.

Correct:

```
AttributesModel *am = new AttributesModel( model, 0 );  
diagram1->setAttributesModel( am );  
diagram2->setAttributesModel( am );
```

Wrong:

```
diagram1->setAttributesModel( diagram2->attributesModel() );
```

#### Parameters:

*model* The [AttributesModel](#) to use for this diagram.



See also:

[AttributesModel](#), [usesExternalAttributesModel](#)

Reimplemented in [KDChart::AbstractCartesianDiagram](#).

Definition at line 255 of file `KDChartAbstractDiagram.cpp`.

References `d`, `modelsChanged()`, and `setDataBoundariesDirty()`.

Referenced by `KDChart::AbstractCartesianDiagram::setAttributesModel()`.

```

256 {
257     if( amodel->sourceModel() != model() ) {
258         qWarning("KDChart::AbstractDiagram::setAttributesModel() failed: "
259                "Trying to set an attributesmodel which works on a different "
260                "model than the diagram.");
261         return;
262     }
263     if( qobject_cast<PrivateAttributesModel*>(amodel) ) {
264         qWarning("KDChart::AbstractDiagram::setAttributesModel() failed: "
265                "Trying to set an attributesmodel that is private to another diagram.");
266         return;
267     }
268     d->setAttributesModel(amodel);
269     scheduleDelayedItemsLayout();
270     setDataBoundariesDirty();
271     emit modelsChanged();
272 }
```

#### 9.7.3.52 void AbstractDiagram::setAttributesModelRootIndex (const QModelIndex &) [protected]

Definition at line 293 of file `KDChartAbstractDiagram.cpp`.

References `d`, and `setDataBoundariesDirty()`.

Referenced by `setRootIndex()`.

```

294 {
295     d->attributesModelRootIndex=idx;
296     setDataBoundariesDirty();
297     scheduleDelayedItemsLayout();
298 }
```

#### 9.7.3.53 void AbstractDiagram::setBrush (const QBrush & brush)

Set the brush to be used, for painting all datasets in the model.

**Parameters:**

*brush* The brush to use.

Definition at line 806 of file `KDChartAbstractDiagram.cpp`.

References `attributesModel()`, `KDChart::DatasetBrushRole`, `propertiesChanged()`, and `KDChart::AttributesModel::setModelData()`.

```

807 {
808     attributesModel()->setModelData(
809         qVariantFromValue( brush ), DatasetBrushRole );
810     emit propertiesChanged();
811 }

```

#### 9.7.3.54 void AbstractDiagram::setBrush (int *dataset*, const QBrush & *brush*)

Set the brush to be used, for painting the given dataset.

##### Parameters:

***dataset*** The dataset's column in the model.

***brush*** The brush to use.

Definition at line 813 of file KDChartAbstractDiagram.cpp.

References `attributesModel()`, `KDChart::DatasetBrushRole`, `propertiesChanged()`, and `KDChart::AttributesModel::setHeaderData()`.

```

814 {
815     attributesModel()->setHeaderData(
816         column, Qt::Vertical,
817         qVariantFromValue( brush ),
818         DatasetBrushRole );
819     emit propertiesChanged();
820 }

```

#### 9.7.3.55 void AbstractDiagram::setBrush (const QModelIndex & *index*, const QBrush & *brush*)

Set the brush to be used, for painting the datapoint at the given index.

##### Parameters:

***index*** The datapoint's index in the model.

***brush*** The brush to use.

Definition at line 798 of file KDChartAbstractDiagram.cpp.

References `attributesModel()`, `KDChart::DatasetBrushRole`, `propertiesChanged()`, and `KDChart::AttributesModel::setData()`.

```

799 {
800     attributesModel()->setData(
801         attributesModel()->mapFromSource( index ),
802         qVariantFromValue( brush ), DatasetBrushRole );
803     emit propertiesChanged();
804 }

```

#### 9.7.3.56 void AbstractDiagram::setCoordinatePlane ([AbstractCoordinatePlane](#) \* *plane*) [virtual]

Set the coordinate plane associated with the diagram.

This determines how coordinates in value space are mapped into pixel space. The chart takes ownership.

**Returns:**

The coordinate plane associated with the diagram.

Reimplemented in [KDChart::AbstractCartesianDiagram](#).

Definition at line 316 of file KDChartAbstractDiagram.cpp.

References [d](#).

Referenced by [KDChart::AbstractCoordinatePlane::addDiagram\(\)](#), [KDChart::AbstractCartesianDiagram::setCoordinatePlane\(\)](#), and [KDChart::AbstractCoordinatePlane::takeDiagram\(\)](#).

```
317 {
318     d->plane = parent;
319 }
```

**9.7.3.57 void AbstractDiagram::setDataBoundariesDirty () const** [protected]

Definition at line 234 of file KDChartAbstractDiagram.cpp.

References [d](#).

Referenced by [dataChanged\(\)](#), [KDChart::Plotter::resize\(\)](#), [KDChart::LineDiagram::resize\(\)](#), [KDChart::BarDiagram::resize\(\)](#), [setAttributesModel\(\)](#), [setAttributesModelRootIndex\(\)](#), [setDatasetDimension\(\)](#), [setModel\(\)](#), [KDChart::BarDiagram::setThreeDBarAttributes\(\)](#), [KDChart::Plotter::setThreeDLineAttributes\(\)](#), [KDChart::LineDiagram::setThreeDLineAttributes\(\)](#), [KDChart::Plotter::setType\(\)](#), [KDChart::LineDiagram::setType\(\)](#), and [KDChart::BarDiagram::setType\(\)](#).

```
235 {
236     d->databoundariesDirty = true;
237 }
```

**9.7.3.58 void AbstractDiagram::setDatasetDimension (int *dimension*)**

Sets the dataset dimension of the diagram.

**See also:**

[datasetDimension](#).

**Parameters:**

*dimension*

Definition at line 1077 of file KDChartAbstractDiagram.cpp.

References [d](#), [layoutChanged\(\)](#), and [setDataBoundariesDirty\(\)](#).

Referenced by [KDChart::Widget::setType\(\)](#), [KDChart::TernaryLineDiagram::TernaryLineDiagram\(\)](#), and [KDChart::TernaryPointDiagram::TernaryPointDiagram\(\)](#).

```
1078 {
1079     if ( d->datasetDimension == dimension ) return;
1080     d->datasetDimension = dimension;
1081     setDataBoundariesDirty();
1082     emit layoutChanged( this );
1083 }
```

**9.7.3.59 void AbstractDiagram::setDataValueAttributes (const [DataValueAttributes](#) & *a*)**

Set the [DataValueAttributes](#) for all datapoints in the model.

**Parameters:**

*a* The attributes to set.

Definition at line 428 of file KDChartAbstractDiagram.cpp.

References `d`, `KDChart::DataValueLabelAttributesRole`, and `propertiesChanged()`.

```
429 {
430     d->attributesModel->setModelData( qVariantFromValue( a ), DataValueLabelAttributesRole );
431     emit propertiesChanged();
432 }
```

**9.7.3.60 void AbstractDiagram::setDataValueAttributes (int *dataset*, const [DataValueAttributes](#) & *a*)**

Set the [DataValueAttributes](#) for the given dataset.

**Parameters:**

*dataset* The dataset to set the attributes for.

*a* The attributes to set.

Definition at line 400 of file KDChartAbstractDiagram.cpp.

References `d`, `KDChart::DataValueLabelAttributesRole`, and `propertiesChanged()`.

```
401 {
402     d->attributesModel->setHeaderData(
403         column, Qt::Vertical,
404         qVariantFromValue( a ), DataValueLabelAttributesRole );
405     emit propertiesChanged();
406 }
```

**9.7.3.61 void AbstractDiagram::setDataValueAttributes (const QModelIndex & *index*, const [DataValueAttributes](#) & *a*)**

Set the [DataValueAttributes](#) for the given index.

**Parameters:**

*index* The datapoint to set the attributes for.

*a* The attributes to set.

Definition at line 389 of file KDChartAbstractDiagram.cpp.

References `d`, `KDChart::DataValueLabelAttributesRole`, and `propertiesChanged()`.

```
391 {
392     d->attributesModel->setData(
393         d->attributesModel->mapFromSource( index ),
394         qVariantFromValue( a ),
395         DataValueLabelAttributesRole );
396     emit propertiesChanged();
397 }
```

### 9.7.3.62 void AbstractDiagram::setHidden (bool *hidden*)

Hide (or unhide, resp.  
) all datapoints in the model.

**Note:**

Hidden data are still taken into account by the coordinate plane, so neither the grid nor your axes' ranges will change, when you hide data. For totally removing data from KD Chart's view you can use another approach: e.g. you could define a proxy model on top of your data model, and register the proxy model calling [setModel\(\)](#) instead of registering your real data model.

**Parameters:**

*hidden* The hidden status to set.

Definition at line 359 of file KDChartAbstractDiagram.cpp.

References [d](#), [dataHidden\(\)](#), and [KDChart::DataHiddenRole](#).

```
360 {  
361     d->attributesModel->setModelData(  
362         qVariantFromValue( hidden ),  
363         DataHiddenRole );  
364     emit dataHidden();  
365 }
```

### 9.7.3.63 void AbstractDiagram::setHidden (int *column*, bool *hidden*)

Hide (or unhide, resp.  
) a dataset.

**Note:**

Hidden data are still taken into account by the coordinate plane, so neither the grid nor your axes' ranges will change, when you hide data. For totally removing data from KD Chart's view you can use another approach: e.g. you could define a proxy model on top of your data model, and register the proxy model calling [setModel\(\)](#) instead of registering your real data model.

**Parameters:**

*column* The dataset to set the hidden status for.

*hidden* The hidden status to set.

Definition at line 350 of file KDChartAbstractDiagram.cpp.

References [d](#), [dataHidden\(\)](#), and [KDChart::DataHiddenRole](#).

```
351 {  
352     d->attributesModel->setHeaderData(  
353         column, Qt::Vertical,  
354         qVariantFromValue( hidden ),  
355         DataHiddenRole );  
356     emit dataHidden();  
357 }
```

### 9.7.3.64 void AbstractDiagram::setHidden (const QModelIndex & *index*, bool *hidden*)

Hide (or unhide, resp.

) a data cell.

#### Note:

Hidden data are still taken into account by the coordinate plane, so neither the grid nor your axes' ranges will change, when you hide data. For totally removing data from KD Chart's view you can use another approach: e.g. you could define a proxy model on top of your data model, and register the proxy model calling [setModel\(\)](#) instead of registering your real data model.

#### Parameters:

***index*** The datapoint to set the hidden status for.

***hidden*** The hidden status to set.

Definition at line 341 of file KDChartAbstractDiagram.cpp.

References [d](#), [dataHidden\(\)](#), and [KDChart::DataHiddenRole](#).

```

342 {
343     d->attributesModel->setData(
344         d->attributesModel->mapFromSource( index ),
345         QVariantFromValue( hidden ),
346         DataHiddenRole );
347     emit dataHidden();
348 }
```

### 9.7.3.65 void AbstractDiagram::setModel (QAbstractItemModel \* *model*) [virtual]

Associate a model with the diagram.

Reimplemented in [KDChart::AbstractCartesianDiagram](#).

Definition at line 239 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::AttributesModel::initFrom\(\)](#), [modelsChanged\(\)](#), and [setDataBoundariesDirty\(\)](#).

Referenced by [KDChart::AbstractCartesianDiagram::setModel\(\)](#), and [KDChart::Widget::setType\(\)](#).

```

240 {
241     QAbstractItemView::setModel( newModel );
242     AttributesModel* amodel = new PrivateAttributesModel( newModel, this );
243     amodel->initFrom( d->attributesModel );
244     d->setAttributesModel( amodel );
245     scheduleDelayedItemsLayout();
246     setDataBoundariesDirty();
247     emit modelsChanged();
248 }
```

### 9.7.3.66 void AbstractDiagram::setPen (const QPen & *pen*)

Set the pen to be used, for painting all datasets in the model.

#### Parameters:

***pen*** The pen to use.

Definition at line 753 of file KDChartAbstractDiagram.cpp.

References `attributesModel()`, `KDChart::DatasetPenRole`, `propertiesChanged()`, and `KDChart::AttributesModel::setModelData()`.

```
754 {
755     attributesModel()->setModelData(
756         qVariantFromValue( pen ), DatasetPenRole );
757     emit propertiesChanged();
758 }
```

### 9.7.3.67 void AbstractDiagram::setPen (int *dataset*, const QPen & *pen*)

Set the pen to be used, for painting the given dataset.

#### Parameters:

***dataset*** The dataset's row in the model.

***pen*** The pen to use.

Definition at line 760 of file KDChartAbstractDiagram.cpp.

References `attributesModel()`, `datasetDimension()`, `KDChart::DatasetPenRole`, `propertiesChanged()`, and `KDChart::AttributesModel::setHeaderData()`.

```
761 {
762     if( datasetDimension() > 1 )
763         column *= datasetDimension();
764     attributesModel()->setHeaderData(
765         column, Qt::Vertical,
766         qVariantFromValue( pen ),
767         DatasetPenRole );
768     emit propertiesChanged();
769 }
```

### 9.7.3.68 void AbstractDiagram::setPen (const QModelIndex & *index*, const QPen & *pen*)

Set the pen to be used, for painting the datapoint at the given index.

#### Parameters:

***index*** The datapoint's index in the model.

***pen*** The pen to use.

Definition at line 743 of file KDChartAbstractDiagram.cpp.

References `attributesModel()`, `datasetDimension()`, `KDChart::DatasetPenRole`, `propertiesChanged()`, and `KDChart::AttributesModel::setData()`.

```
744 {
745     if( datasetDimension() > 1 )
746         return setPen( index.column(), pen );
747     attributesModel()->setData(
748         attributesModel()->mapFromSource( index ),
749         qVariantFromValue( pen ), DatasetPenRole );
750     emit propertiesChanged();
751 }
```

**9.7.3.69 void AbstractDiagram::setPercentMode (bool *percent*)**

Definition at line 456 of file KDChartAbstractDiagram.cpp.

References `d`, and `propertiesChanged()`.

Referenced by `KDChart::LineDiagram::setType()`, and `KDChart::BarDiagram::setType()`.

```

457 {
458     d->percent = percent;
459     emit propertiesChanged();
460 }
```

**9.7.3.70 void AbstractDiagram::setRootIndex (const QModelIndex & *idx*) [virtual]**

Set the root index in the model, where the diagram starts referencing data for display.

[reimplemented]

Reimplemented in [KDChart::AbstractCartesianDiagram](#).

Definition at line 286 of file KDChartAbstractDiagram.cpp.

References `d`, and `setAttributesModelRootIndex()`.

Referenced by `KDChart::AbstractCartesianDiagram::setRootIndex()`.

```

287 {
288     QAbstractItemView::setRootIndex(idx);
289     setAttributesModelRootIndex( d->attributesModel->mapFromSource(idx) );
290 }
```

**9.7.3.71 void AbstractDiagram::setSelection (const QRect & *rect*, QItemSelection-Model::SelectionFlags *command*) [virtual]**

[reimplemented]

Definition at line 959 of file KDChartAbstractDiagram.cpp.

References `d`.

```

960 {
961     const QModelIndexList indexes = d->indexesIn( rect );
962     QItemSelection selection;
963     KDAB_FOREACH( const QModelIndex& index, indexes )
964     {
965         selection.append( QItemSelectionRange( index ) );
966     }
967     selectionModel()->select( selection, command );
968 }
```

**9.7.3.72 void AbstractDiagram::setUnitPrefix (const QString & *prefix*, Qt::Orientation *orientation*)**

Sets the unit prefix for all values.



**Parameters:**

*prefix* the prefix to be set

*orientation* the orientation of the axis to set

Definition at line 864 of file KDChartAbstractDiagram.cpp.

References d.

```
865 {  
866     d->unitPrefix[ orientation ] = prefix;  
867 }
```

**9.7.3.73 void AbstractDiagram::setUnitPrefix (const QString & *prefix*, int *column*, Qt::Orientation *orientation*)**

Sets the unit prefix for one value.

**Parameters:**

*prefix* the prefix to be set

*column* the value using that prefix

*orientation* the orientation of the axis to set

Definition at line 854 of file KDChartAbstractDiagram.cpp.

References d.

```
855 {  
856     d->unitPrefixMap[ column ][ orientation ]= prefix;  
857 }
```

**9.7.3.74 void AbstractDiagram::setUnitSuffix (const QString & *suffix*, Qt::Orientation *orientation*)**

Sets the unit suffix for all values.

**Parameters:**

*suffix* the suffix to be set

*orientation* the orientation of the axis to set

Definition at line 885 of file KDChartAbstractDiagram.cpp.

References d.

```
886 {  
887     d->unitSuffix[ orientation ] = suffix;  
888 }
```

### 9.7.3.75 void AbstractDiagram::setUnitSuffix (const QString & *suffix*, int *column*, Qt::Orientation *orientation*)

Sets the unit suffix for one value.

#### Parameters:

*suffix* the suffix to be set  
*column* the value using that suffix  
*orientation* the orientation of the axis to set

Definition at line 875 of file KDChartAbstractDiagram.cpp.

References d.

```
876 {  
877     d->unitSuffixMap[ column ][ orientation ]= suffix;  
878 }
```

### 9.7.3.76 QString AbstractDiagram::unitPrefix (Qt::Orientation *orientation*) const

Returns the global unit prefix.

#### Parameters:

*orientation* the orientation of the axis

#### Returns:

the unit prefix

Definition at line 908 of file KDChartAbstractDiagram.cpp.

References d.

```
909 {  
910     return d->unitPrefix[ orientation ];  
911 }
```

### 9.7.3.77 QString AbstractDiagram::unitPrefix (int *column*, Qt::Orientation *orientation*, bool *fallback* = false) const

Returns the unit prefix for a special value.

#### Parameters:

*column* the value which's prefix is requested  
*orientation* the orientation of the axis  
*fallback* if true, the global prefix is return when no specific one is set for that value

#### Returns:

the unit prefix

Definition at line 897 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by itemRowLabels(), and KDChart::CartesianAxis::paintCtx().

```
898 {  
899     if( !fallback || d->unitPrefixMap[ column ].contains( orientation ) )  
900         return d->unitPrefixMap[ column ][ orientation ];  
901     return d->unitPrefix[ orientation ];  
902 }
```

### 9.7.3.78 QString AbstractDiagram::unitSuffix (Qt::Orientation *orientation*) const

Returns the global unit suffix.

#### Parameters:

***orientation*** the orientation of the axis

#### Returns:

the unit suffix

Definition at line 931 of file KDChartAbstractDiagram.cpp.

References d.

```
932 {  
933     return d->unitSuffix[ orientation ];  
934 }
```

### 9.7.3.79 QString AbstractDiagram::unitSuffix (int *column*, Qt::Orientation *orientation*, bool *fallback* = false) const

Returns the unit suffix for a special value.

#### Parameters:

***column*** the value which's suffix is requested

***orientation*** the orientation of the axis

***fallback*** if true, the global suffix is return when no specific one is set for that value

#### Returns:

the unit suffix

Definition at line 920 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by itemRowLabels(), and KDChart::CartesianAxis::paintCtx().

```
921 {  
922     if( !fallback || d->unitSuffixMap[ column ].contains( orientation ) )  
923         return d->unitSuffixMap[ column ][ orientation ];  
924     return d->unitSuffix[ orientation ];  
925 }
```

### 9.7.3.80 void AbstractDiagram::update () const

Definition at line 1091 of file KDChartAbstractDiagram.cpp.

References [d](#).

Referenced by [doItemsLayout\(\)](#).

```
1092 {  
1093     //qDebug("KDChart::AbstractDiagram::update() called");  
1094     if ( d->plane )  
1095         d->plane->update();  
1096 }
```

### 9.7.3.81 void KDChart::AbstractDiagram::useDefaultColors ()

Set the palette to be used, for painting datasets to the default palette.

See also:

[KDChart::Palette](#). FIXME: fold into one [usePalette \(KDChart::Palette&\)](#) method

Definition at line 974 of file KDChartAbstractDiagram.cpp.

References [d](#), and [KDChart::AttributesModel::PaletteTypeDefault](#).

```
975 {  
976     d->attributesModel->setPaletteType( AttributesModel::PaletteTypeDefault );  
977 }
```

### 9.7.3.82 void KDChart::AbstractDiagram::useRainbowColors ()

Set the palette to be used, for painting datasets to the rainbow palette.

See also:

[KDChart::Palette](#).

Definition at line 984 of file KDChartAbstractDiagram.cpp.

References [d](#), and [KDChart::AttributesModel::PaletteTypeRainbow](#).

```
985 {  
986     d->attributesModel->setPaletteType( AttributesModel::PaletteTypeRainbow );  
987 }
```

### 9.7.3.83 bool AbstractDiagram::usesExternalAttributesModel () const [virtual]

Returns whether the diagram is using its own built-in attributes model or an attributes model that was set via [setAttributesModel](#).

See also:

[setAttributesModel](#)

Definition at line 274 of file KDChartAbstractDiagram.cpp.

References `d`.

```
275 {  
276     return d->usesExternalAttributesModel();  
277 }
```

#### 9.7.3.84 void KDChart::AbstractDiagram::useSubduedColors ()

Set the palette to be used, for painting datasets to the subdued palette.

**See also:**

[KDChart::Palette](#).

Definition at line 979 of file KDChartAbstractDiagram.cpp.

References `d`, and `KDChart::AttributesModel::PaletteTypeSubdued`.

```
980 {  
981     d->attributesModel->setPaletteType( AttributesModel::PaletteTypeSubdued );  
982 }
```

#### 9.7.3.85 double AbstractDiagram::valueForCell (int row, int column) const [protected]

Helper method, retrieving the data value (DisplayRole) for a given row and column.

**Parameters:**

*row* The row to query.

*column* The column to query.

**Returns:**

The value of the display role at the given row and column as a double.

**Deprecated**

Definition at line 1085 of file KDChartAbstractDiagram.cpp.

References `attributesModelRootIndex()`, and `d`.

```
1086 {  
1087     return d->attributesModel->data(  
1088         d->attributesModel->index( row, column, attributesModelRootIndex() ) ).toDouble();  
1089 }
```

#### 9.7.3.86 int AbstractDiagram::verticalOffset () const [virtual]

[reimplemented]

Definition at line 953 of file KDChartAbstractDiagram.cpp.

```
954 { return 0; }
```

**9.7.3.87 QRect AbstractDiagram::visualRect (const QModelIndex & *index*) const** [virtual]

[reimplemented]

Definition at line 937 of file KDChartAbstractDiagram.cpp.

```
938 {  
939     return QRect();  
940 }
```

**9.7.3.88 QRegion AbstractDiagram::visualRegionForSelection (const QItemSelection & *selection*) const** [virtual]

[reimplemented]

Definition at line 970 of file KDChartAbstractDiagram.cpp.

```
971 { return QRegion(); }
```

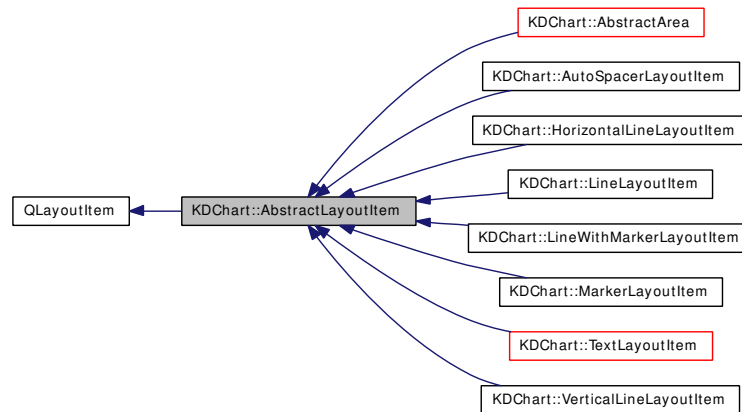
The documentation for this class was generated from the following files:

- [KDChartAbstractDiagram.h](#)
- [KDChartAbstractDiagram.cpp](#)

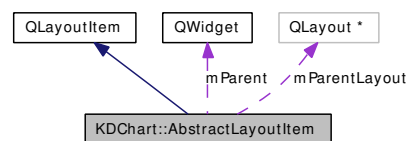
## 9.8 KDChart::AbstractLayoutItem Class Reference

```
#include <KDChartLayoutItems.h>
```

Inheritance diagram for KDChart::AbstractLayoutItem:



Collaboration diagram for KDChart::AbstractLayoutItem:



### 9.8.1 Detailed Description

Base class for all layout items of KD Chart.

Definition at line 50 of file KDChartLayoutItems.h.

### Public Member Functions

- [AbstractLayoutItem](#) (Qt::Alignment itemAlignment=0)
- virtual void [paint](#) (QPainter \*)=0
- virtual void [paintAll](#) (QPainter &painter)

*Default impl: just call paint.*

- virtual void [paintCtx](#) ([PaintContext](#) \*context)

*Default impl: Paint the complete item using its layouted position and size.*

- QLayout \* [parentLayout](#) ()
- void [removeFromParentLayout](#) ()
- void [setParentLayout](#) (QLayout \*lay)
- virtual void [setParentWidget](#) (QWidget \*widget)

*Inform the item about its widget: This enables the item, to trigger that widget's update, whenever the size of the item's contents has changed.*

- virtual void [sizeHintChanged](#) () const

*Report changed size hint: ask the parent widget to recalculate the layout.*

## Protected Attributes

- [QWidget](#) \* [mParent](#)
- [QLayout](#) \* [mParentLayout](#)

## 9.8.2 Constructor & Destructor Documentation

### 9.8.2.1 [KDChart::AbstractLayoutItem::AbstractLayoutItem](#) ([Qt::Alignment](#) *itemAlignment* = 0)

Definition at line 53 of file [KDChartLayoutItems.h](#).

```

53                                     :
54     QLayoutItem( itemAlignment ),
55     mParent( 0 ),
56     mParentLayout( 0 ) {}

```

## 9.8.3 Member Function Documentation

### 9.8.3.1 virtual void [KDChart::AbstractLayoutItem::paint](#) ([QPainter](#) \*) [pure virtual]

Implemented in [KDChart::CartesianAxis](#), [KDChart::CartesianCoordinatePlane](#), [KDChart::TextLayoutItem](#), [KDChart::MarkerLayoutItem](#), [KDChart::LineLayoutItem](#), [KDChart::LineWithMarkerLayoutItem](#), [KDChart::HorizontalLineLayoutItem](#), [KDChart::VerticalLineLayoutItem](#), [KDChart::AutoSpacerLayoutItem](#), [KDChart::PolarCoordinatePlane](#), [KDChart::TernaryAxis](#), and [KDChart::TernaryCoordinatePlane](#).

Referenced by [KDChart::Legend::paint\(\)](#), [paintAll\(\)](#), [KDChart::AbstractArea::paintAll\(\)](#), and [paintCtx\(\)](#).

### 9.8.3.2 void [KDChart::AbstractLayoutItem::paintAll](#) ([QPainter](#) & *painter*) [virtual]

Default impl: just call paint.

Derived classes like [KDChart::AbstractArea](#) are providing additional action here.

Reimplemented in [KDChart::AbstractArea](#), [KDChart::TextArea](#), and [KDChart::TernaryAxis](#).

Definition at line 69 of file [KDChartLayoutItems.cpp](#).

References [paint\(\)](#).

```

70 {
71     paint( &painter );
72 }

```

### 9.8.3.3 void [KDChart::AbstractLayoutItem::paintCtx](#) ([PaintContext](#) \* *context*) [virtual]

Default impl: Paint the complete item using its layouted position and size.



Reimplemented in [KDChart::CartesianAxis](#), and [KDChart::TernaryAxis](#).

Definition at line 77 of file `KDChartLayoutItems.cpp`.

References `paint()`, and `KDChart::PaintContext::painter()`.

```
78 {
79     if( context )
80         paint( context->painter() );
81 }
```

#### 9.8.3.4 QLayout\* KDChart::AbstractLayoutItem::parentLayout ()

Definition at line 76 of file `KDChartLayoutItems.h`.

```
77     {
78         return mParentLayout;
79     }
```

#### 9.8.3.5 void KDChart::AbstractLayoutItem::removeFromParentLayout ()

Definition at line 80 of file `KDChartLayoutItems.h`.

Referenced by `KDChart::Chart::takeCoordinatePlane()`.

```
81     {
82         if( mParentLayout ){
83             if( widget() )
84                 mParentLayout->removeWidget( widget() );
85             else
86                 mParentLayout->removeItem( this );
87         }
88     }
```

#### 9.8.3.6 void KDChart::AbstractLayoutItem::setParentLayout (QLayout \* lay)

Definition at line 72 of file `KDChartLayoutItems.h`.

```
73     {
74         mParentLayout = lay;
75     }
```

#### 9.8.3.7 void KDChart::AbstractLayoutItem::setParentWidget (QWidget \* widget) [virtual]

Inform the item about its widget: This enables the item, to trigger that widget's update, whenever the size of the item's contents has changed.

Thus, you need to call `setParentWidget` on every item, that has a non-fixed size.

Definition at line 64 of file `KDChartLayoutItems.cpp`.

References `mParent`.

Referenced by `KDChart::HeaderFooter::setParent()`, and `KDChart::AbstractCartesianDiagram::takeAxis()`.

```
65 {  
66     mParent = widget;  
67 }
```

#### 9.8.3.8 void KDChart::AbstractLayoutItem::sizeHintChanged () const [virtual]

Report changed size hint: ask the parent widget to recalculate the layout.

Definition at line 86 of file KDChartLayoutItems.cpp.

References mParent.

Referenced by KDChart::TextLayoutItem::sizeHint().

```
87 {  
88     // This is exactly like what QWidget::updateGeometry does.  
89     qDebug("KDChart::AbstractLayoutItem::sizeHintChanged() called");  
90     if( mParent ) {  
91         if ( mParent->layout() )  
92             mParent->layout()->invalidate();  
93         else  
94             QApplication::postEvent( mParent, new QEvent( QEvent::LayoutRequest ) );  
95     }  
96 }
```

### 9.8.4 Member Data Documentation

#### 9.8.4.1 QWidget\* KDChart::AbstractLayoutItem::mParent [protected]

Definition at line 90 of file KDChartLayoutItems.h.

Referenced by setParentWidget(), KDChart::TextLayoutItem::setText(), KDChart::TextLayoutItem::setTextAttributes(), and sizeHintChanged().

#### 9.8.4.2 QLayout\* KDChart::AbstractLayoutItem::mParentLayout [protected]

Definition at line 91 of file KDChartLayoutItems.h.

Referenced by KDChart::AutoSpacerLayoutItem::paint().

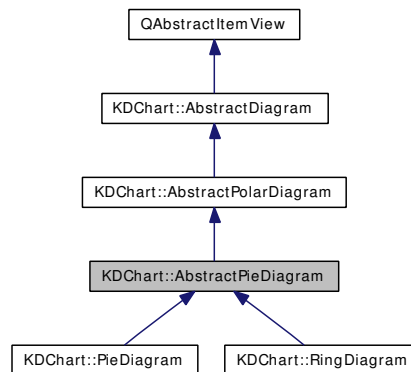
The documentation for this class was generated from the following files:

- [KDChartLayoutItems.h](#)
- [KDChartLayoutItems.cpp](#)

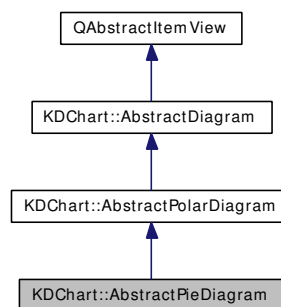
## 9.9 KDChart::AbstractPieDiagram Class Reference

```
#include <KDChartAbstractPieDiagram.h>
```

Inheritance diagram for KDChart::AbstractPieDiagram:



Collaboration diagram for KDChart::AbstractPieDiagram:



### 9.9.1 Detailed Description

Base class for any diagram type.

Definition at line 42 of file KDChartAbstractPieDiagram.h.

#### Signals

- void `dataHidden` ()  
*This signal is emitted, when the hidden status of at least one data cell was (un)set.*
- void `layoutChanged` (`AbstractDiagram *`)  
*Diagrams are supposed to emit this signal, when the layout of one of their element changes.*
- void `modelsChanged` ()  
*This signal is emitted, when either the model or the *AttributesModel* is replaced.*
- void `propertiesChanged` ()

*Emitted upon change of a property of the Diagram.*

## Public Member Functions

- [AbstractPieDiagram](#) ([QWidget](#) \*parent=0, [PolarCoordinatePlane](#) \*plane=0)
- bool [allowOverlappingDataValueTexts](#) () const  
*Returns:*  
*Whether data value labels are allowed to overlap.*
- bool [antiAliasing](#) () const  
*Returns:*  
*Whether anti-aliasing is to be used for rendering this diagram.*
- virtual [AttributesModel](#) \* [attributesModel](#) () const  
*Returns the [AttributesModel](#), that is used by this diagram.*
- [QBrush](#) [brush](#) (const [QModelIndex](#) &index) const  
*Retrieve the brush to be used, for painting the datapoint at the given index in the model.*
- [QBrush](#) [brush](#) (int dataset) const  
*Retrieve the brush to be used for the given dataset.*
- [QBrush](#) [brush](#) () const  
*Retrieve the brush to be used for painting datapoints globally.*
- int [columnCount](#) () const
- bool [compare](#) (const [AbstractDiagram](#) \*other) const  
*Returns true if both diagrams have the same settings.*
- [AbstractCoordinatePlane](#) \* [coordinatePlane](#) () const  
*The coordinate plane associated with the diagram.*
- const [QPair](#)< [QPointF](#), [QPointF](#) > [dataBoundaries](#) () const  
*Return the bottom left and top right data point, that the diagram will display (unless the grid adjusts these values).*
- virtual void [dataChanged](#) (const [QModelIndex](#) &topLeft, const [QModelIndex](#) &bottomRight)  
*[reimplemented]*
- [QList](#)< [QBrush](#) > [datasetBrushes](#) () const  
*The set of dataset brushes currently used, for use in legends, etc.*
- int [datasetDimension](#) () const  
*The dataset dimension of a diagram determines, how many value dimensions it expects each datapoint to have.*
- [QStringList](#) [datasetLabels](#) () const  
*The set of dataset labels currently displayed, for use in legends, etc.*

- QList< [MarkerAttributes](#) > [datasetMarkers](#) () const  
*The set of dataset markers currently used, for use in legends, etc.*
- QList< QPen > [datasetPens](#) () const  
*The set of dataset pens currently used, for use in legends, etc.*
- [DataValueAttributes](#) [dataValueAttributes](#) (const QModelIndex &index) const  
*Retrieve the [DataValueAttributes](#) for the given index.*
- [DataValueAttributes](#) [dataValueAttributes](#) (int column) const  
*Retrieve the [DataValueAttributes](#) for the given dataset.*
- [DataValueAttributes](#) [dataValueAttributes](#) () const  
*Retrieve the [DataValueAttributes](#) specified globally.*
- virtual void [doItemsLayout](#) ()  
*[reimplemented]*
- qreal [granularity](#) () const  
**Returns:**  
*the granularity.*
- virtual int [horizontalOffset](#) () const  
*[reimplemented]*
- virtual QModelIndex [indexAt](#) (const QPoint &point) const  
*[reimplemented]*
- QModelIndexList [indexesAt](#) (const QPoint &point) const  
*This method is added alongside with [indexAt](#) from QAIM, since in kdchart multiple indexes can be displayed at the same spot.*
- bool [isHidden](#) (const QModelIndex &index) const  
*Retrieve the hidden status for the given index.*
- bool [isHidden](#) (int column) const  
*Retrieve the hidden status for the given dataset.*
- bool [isHidden](#) () const  
*Retrieve the hidden status specified globally.*
- virtual bool [isIndexHidden](#) (const QModelIndex &index) const  
*[reimplemented]*
- QStringList [itemRowLabels](#) () const  
*The set of item row labels currently displayed, for use in Abscissa axes, etc.*
- virtual QModelIndex [moveCursor](#) (CursorAction cursorAction, Qt::KeyboardModifiers modifiers)  
*[reimplemented]*

- virtual double [numberOfGridRings](#) () const=0
- virtual double [numberOfValuesPerDataset](#) () const=0
- virtual void [paint](#) ([PaintContext](#) \*paintContext)=0  
*Draw the diagram contents to the rectangle and painter, that are passed in as part of the paint context.*
- void [paintDataValueText](#) (QPainter \*painter, const QModelIndex &index, const QPointF &pos, double value)
- void [paintMarker](#) (QPainter \*painter, const QModelIndex &index, const QPointF &pos)
- virtual void [paintMarker](#) (QPainter \*painter, const [MarkerAttributes](#) &markerAttributes, const QBrush &brush, const QPen &, const QPointF &point, const QSizeF &size)
- QPen [pen](#) (const QModelIndex &index) const  
*Retrieve the pen to be used, for painting the datapoint at the given index in the model.*
- QPen [pen](#) (int dataset) const  
*Retrieve the pen to be used for the given dataset.*
- QPen [pen](#) () const  
*Retrieve the pen to be used for painting datapoints globally.*
- bool [percentMode](#) () const
- [PieAttributes](#) [pieAttributes](#) (const QModelIndex &index) const
- [PieAttributes](#) [pieAttributes](#) (int column) const
- [PieAttributes](#) [pieAttributes](#) () const
- const [PolarCoordinatePlane](#) \* [polarCoordinatePlane](#) () const
- virtual void [resize](#) (const QSizeF &area)=0  
*Called by the widget's sizeEvent.*
- virtual void [scrollTo](#) (const QModelIndex &index, ScrollHint hint=EnsureVisible)  
*[reimplemented]*
- void [setAllowOverlappingDataValueTexts](#) (bool allow)  
*Set whether data value labels are allowed to overlap.*
- void [setAntiAliasing](#) (bool enabled)  
*Set whether anti-aliasing is to be used while rendering this diagram.*
- virtual void [setAttributesModel](#) ([AttributesModel](#) \*model)  
*Associate an [AttributesModel](#) with this diagram.*
- void [setBrush](#) (const QBrush &brush)  
*Set the brush to be used, for painting all datasets in the model.*
- void [setBrush](#) (int dataset, const QBrush &brush)  
*Set the brush to be used, for painting the given dataset.*
- void [setBrush](#) (const QModelIndex &index, const QBrush &brush)  
*Set the brush to be used, for painting the datapoint at the given index.*
- virtual void [setCoordinatePlane](#) ([AbstractCoordinatePlane](#) \*plane)  
*Set the coordinate plane associated with the diagram.*

- void [setDatasetDimension](#) (int dimension)  
*Sets the dataset dimension of the diagram.*
- void [setDataValueAttributes](#) (const [DataValueAttributes](#) &a)  
*Set the [DataValueAttributes](#) for all datapoints in the model.*
- void [setDataValueAttributes](#) (int dataset, const [DataValueAttributes](#) &a)  
*Set the [DataValueAttributes](#) for the given dataset.*
- void [setDataValueAttributes](#) (const QModelIndex &index, const [DataValueAttributes](#) &a)  
*Set the [DataValueAttributes](#) for the given index.*
- void [setGranularity](#) (qreal value)  
*Set the granularity: the smaller the granularity the more your diagram segments will show facettes instead of rounded segments.*
- void [setHidden](#) (bool hidden)  
*Hide (or unhide, resp.)*
- void [setHidden](#) (int column, bool hidden)  
*Hide (or unhide, resp.)*
- void [setHidden](#) (const QModelIndex &index, bool hidden)  
*Hide (or unhide, resp.)*
- virtual void [setModel](#) (QAbstractItemModel \*model)  
*Associate a model with the diagram.*
- void [setPen](#) (const QPen &pen)  
*Set the pen to be used, for painting all datasets in the model.*
- void [setPen](#) (int dataset, const QPen &pen)  
*Set the pen to be used, for painting the given dataset.*
- void [setPen](#) (const QModelIndex &index, const QPen &pen)  
*Set the pen to be used, for painting the datapoint at the given index.*
- void [setPercentMode](#) (bool percent)
- void [setPieAttributes](#) (int column, const [PieAttributes](#) &a)
- void [setPieAttributes](#) (const [PieAttributes](#) &a)
- virtual void [setRootIndex](#) (const QModelIndex &idx)  
*Set the root index in the model, where the diagram starts referencing data for display.*
- virtual void [setSelection](#) (const QRect &rect, QItemSelectionModel::SelectionFlags command)  
*[reimplemented]*
- void [setStartPosition](#) (int degrees)
- void [setThreeDPieAttributes](#) (const QModelIndex &index, const [ThreeDPieAttributes](#) &a)
- void [setThreeDPieAttributes](#) (int column, const [ThreeDPieAttributes](#) &a)

- void [setThreeDPieAttributes](#) (const [ThreeDPieAttributes](#) &a)
- void [setUnitPrefix](#) (const QString &prefix, Qt::Orientation orientation)  
*Sets the unit prefix for all values.*
- void [setUnitPrefix](#) (const QString &prefix, int column, Qt::Orientation orientation)  
*Sets the unit prefix for one value.*
- void [setUnitSuffix](#) (const QString &suffix, Qt::Orientation orientation)  
*Sets the unit suffix for all values.*
- void [setUnitSuffix](#) (const QString &suffix, int column, Qt::Orientation orientation)  
*Sets the unit suffix for one value.*
- int [startPosition](#) () const
- [ThreeDPieAttributes](#) [threeDPieAttributes](#) (const QModelIndex &index) const
- [ThreeDPieAttributes](#) [threeDPieAttributes](#) (int column) const
- [ThreeDPieAttributes](#) [threeDPieAttributes](#) () const
- QString [unitPrefix](#) (Qt::Orientation orientation) const  
*Returns the global unit prefix.*
- QString [unitPrefix](#) (int column, Qt::Orientation orientation, bool fallback=false) const  
*Returns the unit prefix for a special value.*
- QString [unitSuffix](#) (Qt::Orientation orientation) const  
*Returns the global unit suffix.*
- QString [unitSuffix](#) (int column, Qt::Orientation orientation, bool fallback=false) const  
*Returns the unit suffix for a special value.*
- void [update](#) () const
- void [useDefaultColors](#) ()  
*Set the palette to be used, for painting datasets to the default palette.*
- void [useRainbowColors](#) ()  
*Set the palette to be used, for painting datasets to the rainbow palette.*
- virtual bool [usesExternalAttributesModel](#) () const  
*Returns whether the diagram is using its own built-in attributes model or an attributes model that was set via [setAttributesModel](#).*
- void [useSubduedColors](#) ()  
*Set the palette to be used, for painting datasets to the subdued palette.*
- virtual double [valueTotals](#) () const=0
- virtual int [verticalOffset](#) () const  
*[reimplemented]*
- virtual QRect [visualRect](#) (const QModelIndex &index) const  
*[reimplemented]*



- virtual QRegion [visualRegionForSelection](#) (const QItemSelection &selection) const  
[reimplemented]
- virtual [~AbstractPieDiagram](#) ()

## Protected Member Functions

- QModelIndex [attributesModelRootIndex](#) () const
- virtual const QPair< QPointF, QPointF > [calculateDataBoundaries](#) () const=0
- virtual bool [checkInvariants](#) (bool justReturnTheStatus=false) const
- QModelIndex [columnToIndex](#) (int column) const
- virtual void [paintDataValueTexts](#) (QPainter \*painter)
- virtual void [paintMarkers](#) (QPainter \*painter)
- void [setAttributesModelRootIndex](#) (const QModelIndex &)
- void [setDataBoundariesDirty](#) () const
- double [valueForCell](#) (int row, int column) const

*Helper method, retrieving the data value (DisplayRole) for a given row and column.*

## 9.9.2 Constructor & Destructor Documentation

### 9.9.2.1 AbstractPieDiagram::AbstractPieDiagram (QWidget \*parent = 0, PolarCoordinatePlane \*plane = 0) [explicit]

Definition at line 46 of file KDChartAbstractPieDiagram.cpp.

```

46                                     :
47     AbstractPolarDiagram( new Private(), parent, plane )
48 {
49     init();
50 }
```

### 9.9.2.2 AbstractPieDiagram::~~AbstractPieDiagram () [virtual]

Definition at line 52 of file KDChartAbstractPieDiagram.cpp.

```

53 {
54 }
```

## 9.9.3 Member Function Documentation

### 9.9.3.1 bool AbstractDiagram::allowOverlappingDataValueTexts () const [inherited]

#### Returns:

Whether data value labels are allowed to overlap.

Definition at line 440 of file KDChartAbstractDiagram.cpp.

References [d](#).

Referenced by [KDChart::AbstractDiagram::compare\(\)](#).

```

441 {
442     return d->allowOverlappingDataValueTexts;
443 }

```

### 9.9.3.2 bool AbstractDiagram::antiAliasing () const [inherited]

#### Returns:

Whether anti-aliasing is to be used for rendering this diagram.

Definition at line 451 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::compare().

```

452 {
453     return d->antiAliasing;
454 }

```

### 9.9.3.3 [AttributesModel](#) \* AbstractDiagram::attributesModel () const [virtual, inherited]

Returns the [AttributesModel](#), that is used by this diagram.

By default each diagram owns its own [AttributesModel](#), which should never be deleted. Only if a user-supplied [AttributesModel](#) has been set does the pointer returned here not belong to the diagram.

#### Returns:

The [AttributesModel](#) associated with the diagram.

#### See also:

[setAttributesModel](#)

Definition at line 280 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::brush(), KDChart::AbstractDiagram::compare(), KDChart::AbstractDiagram::datasetBrushes(), KDChart::AbstractDiagram::datasetLabels(), KDChart::AbstractDiagram::datasetMarkers(), KDChart::AbstractDiagram::datasetPens(), KDChart::AbstractDiagram::dataValueAttributes(), KDChart::AbstractDiagram::isHidden(), KDChart::AbstractDiagram::itemRowLabels(), KDChart::RingDiagram::paint(), KDChart::PolarDiagram::paint(), KDChart::AbstractDiagram::pen(), KDChart::AbstractCartesianDiagram::setAttributesModel(), KDChart::BarDiagram::setBarAttributes(), KDChart::AbstractDiagram::setBrush(), KDChart::AbstractCartesianDiagram::setModel(), KDChart::AbstractDiagram::setPen(), and KDChart::TernaryLineDiagram::TernaryLineDiagram().

```

281 {
282     return d->attributesModel;
283 }

```

#### 9.9.3.4 QModelIndex AbstractDiagram::attributesModelRootIndex () const [protected, inherited]

returns a QModelIndex pointing into the [AttributesModel](#) that corresponds to the root index of the diagram.

Definition at line 302 of file KDChartAbstractDiagram.cpp.

References [d](#).

Referenced by [KDChart::AbstractDiagram::datasetBrushes\(\)](#), [KDChart::AbstractDiagram::datasetLabels\(\)](#), [KDChart::AbstractDiagram::datasetMarkers\(\)](#), [KDChart::AbstractDiagram::datasetPens\(\)](#), [KDChart::AbstractDiagram::itemRowLabels\(\)](#), [KDChart::Plotter::numberOfAbscissaSegments\(\)](#), [KDChart::LineDiagram::numberOfAbscissaSegments\(\)](#), [KDChart::BarDiagram::numberOfAbscissaSegments\(\)](#), [KDChart::Plotter::numberOfOrdinateSegments\(\)](#), [KDChart::LineDiagram::numberOfOrdinateSegments\(\)](#), [KDChart::BarDiagram::numberOfOrdinateSegments\(\)](#), [KDChart::AbstractDiagram::valueForCell\(\)](#), and [KDChart::LineDiagram::valueForCellTesting\(\)](#).

```
303 {
304     if ( !d->attributesModelRootIndex.isValid() )
305         d->attributesModelRootIndex = d->attributesModel->mapFromSource( rootIndex() );
306     return d->attributesModelRootIndex;
307 }
```

#### 9.9.3.5 QBrush AbstractDiagram::brush (const QModelIndex & index) const [inherited]

Retrieve the brush to be used, for painting the datapoint at the given index in the model.

##### Parameters:

*index* The index of the datapoint in the model.

##### Returns:

The brush to use for painting.

Definition at line 838 of file KDChartAbstractDiagram.cpp.

References [KDChart::AbstractDiagram::attributesModel\(\)](#), [KDChart::AbstractDiagram::brush\(\)](#), [KDChart::AttributesModel::data\(\)](#), [KDChart::DatasetBrushRole](#), and [KDChart::AbstractDiagram::datasetDimension\(\)](#).

```
839 {
840     if( datasetDimension() > 1 )
841         return brush( index.column() );
842     return QVariantValue<QBrush>(
843         attributesModel()->data(
844             attributesModel()->mapFromSource( index ),
845             DatasetBrushRole ) );
846 }
```

#### 9.9.3.6 QBrush AbstractDiagram::brush (int dataset) const [inherited]

Retrieve the brush to be used for the given dataset.

This will fall back automatically to what was set at model level, if there are no dataset specific settings.

**Parameters:**

*dataset* The dataset to retrieve the brush for.

**Returns:**

The brush to use for painting.

Definition at line 828 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::columnToIndex(), KDChart::AttributesModel::data(), KDChart::DatasetBrushRole, and KDChart::AbstractDiagram::datasetDimension().

```

829 {
830     if( datasetDimension() > 1 )
831         dataset /= datasetDimension();
832     return qVariantValue<QBrush>(
833         attributesModel()->data(
834             attributesModel()->mapFromSource( columnToIndex( dataset ) ),
835             DatasetBrushRole ) );
836 }
```

**9.9.3.7 QBrush AbstractDiagram::brush () const** [inherited]

Retrieve the brush to be used for painting datapoints globally.

This will fall back automatically to the default settings, if there are no specific settings.

**Returns:**

The brush to use for painting.

Definition at line 822 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AttributesModel::data(), and KDChart::DatasetBrushRole.

Referenced by KDChart::AbstractDiagram::brush(), KDChart::TernaryPointDiagram::paint(), KDChart::TernaryLineDiagram::paint(), KDChart::RingDiagram::paint(), KDChart::PolarDiagram::paint(), and KDChart::AbstractDiagram::paintMarker().

```

823 {
824     return qVariantValue<QBrush>(
825         attributesModel()->data( DatasetBrushRole ) );
826 }
```

**9.9.3.8 virtual const QPair<QPointF, QPointF> KDChart::AbstractDiagram::calculateDataBoundaries () const** [protected, pure virtual, inherited]

Implemented in KDChart::BarDiagram, KDChart::LineDiagram, KDChart::PieDiagram, KDChart::Plotter, KDChart::PolarDiagram, KDChart::RingDiagram, KDChart::AbstractTernaryDiagram, KDChart::TernaryLineDiagram, and KDChart::TernaryPointDiagram.

Referenced by KDChart::AbstractDiagram::dataBoundaries().

### 9.9.3.9 bool AbstractDiagram::checkInvariants (bool *justReturnTheStatus* = false) const

[protected, virtual, inherited]

Definition at line 1060 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::coordinatePlane().

Referenced by KDChart::RingDiagram::calculateDataBoundaries(), KDChart::PolarDiagram::calculateDataBoundaries(), KDChart::Plotter::calculateDataBoundaries(), KDChart::PieDiagram::calculateDataBoundaries(), KDChart::LineDiagram::calculateDataBoundaries(), KDChart::BarDiagram::calculateDataBoundaries(), KDChart::RingDiagram::paint(), KDChart::PolarDiagram::paint(), KDChart::Plotter::paint(), KDChart::PieDiagram::paint(), KDChart::LineDiagram::paint(), KDChart::BarDiagram::paint(), KDChart::AbstractDiagram::paintDataValueTexts(), KDChart::AbstractDiagram::paintMarker(), and KDChart::AbstractDiagram::paintMarkers().

```

1061 {
1062     if( ! justReturnTheStatus ){
1063         Q_ASSERT_X ( model(), "AbstractDiagram::checkInvariants()",
1064                     "There is no usable model set, for the diagram." );
1065
1066         Q_ASSERT_X ( coordinatePlane(), "AbstractDiagram::checkInvariants()",
1067                     "There is no usable coordinate plane set, for the diagram." );
1068     }
1069     return model() && coordinatePlane();
1070 }
```

### 9.9.3.10 int AbstractPolarDiagram::columnCount () const

[inherited]

Definition at line 60 of file KDChartAbstractPolarDiagram.cpp.

References KDChart::AbstractPolarDiagram::numberOfValuesPerDataset().

Referenced by KDChart::PieDiagram::calculateDataBoundaries(), KDChart::PieDiagram::paint(), and KDChart::PieDiagram::valueTotals().

```

61 {
62     return static_cast<int>( numberOfValuesPerDataset() );
63 }
```

### 9.9.3.11 QModelIndex AbstractDiagram::columnToIndex (int *column*) const

[protected, inherited]

Definition at line 309 of file KDChartAbstractDiagram.cpp.

Referenced by KDChart::BarDiagram::barAttributes(), KDChart::AbstractDiagram::brush(), KDChart::AbstractDiagram::dataValueAttributes(), KDChart::AbstractDiagram::isHidden(), KDChart::Plotter::lineAttributes(), KDChart::LineDiagram::lineAttributes(), KDChart::AbstractDiagram::pen(), pieAttributes(), KDChart::BarDiagram::threeDBarAttributes(), KDChart::Plotter::threeDLineAttributes(), KDChart::LineDiagram::threeDLineAttributes(), and threeDPieAttributes().

```

310 {    // FIXME (Mirko): shouldn't this be headerData? instead of the index for the first row?
311     if( model() )
312         return QModelIndex( model()->index( 0, column, rootIndex() ) );
313     return QModelIndex();
314 }
```

### 9.9.3.12 bool AbstractDiagram::compare (const AbstractDiagram \* other) const [inherited]

Returns true if both diagrams have the same settings.

Definition at line 129 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::allowOverlappingDataValueTexts(), KDChart::AbstractDiagram::antiAliasing(), KDChart::AbstractDiagram::attributesModel(), KDChart::AttributesModel::compare(), KDChart::AbstractDiagram::datasetDimension(), and KDChart::AbstractDiagram::percentMode().

```

130 {
131     if( other == this ) return true;
132     if( ! other ){
133         //qDebug() << "AbstractDiagram::compare() cannot compare to Null pointer";
134         return false;
135     }
136     /*
137     qDebug() << "\n                AbstractDiagram::compare() QAbstractScrollArea:";
138         // compare QAbstractScrollArea properties
139     qDebug() <<
140         ((horizontalScrollBarPolicy() == other->horizontalScrollBarPolicy()) &&
141         (verticalScrollBarPolicy() == other->verticalScrollBarPolicy()));
142     qDebug() << "AbstractDiagram::compare() QFrame:";
143         // compare QFrame properties
144     qDebug() <<
145         ((frameShadow() == other->frameShadow()) &&
146         (frameShape() == other->frameShape()) &&
147         (frameWidth() == other->frameWidth()) &&
148         (lineWidth() == other->lineWidth()) &&
149         (midLineWidth() == other->midLineWidth()));
150     qDebug() << "AbstractDiagram::compare() QAbstractItemView:";
151         // compare QAbstractItemView properties
152     qDebug() <<
153         ((alternatingRowColors() == other->alternatingRowColors()) &&
154         (hasAutoScroll() == other->hasAutoScroll()) &&
155 #if QT_VERSION > 0x040199
156         (dragDropMode() == other->dragDropMode()) &&
157         (dragDropOverwriteMode() == other->dragDropOverwriteMode()) &&
158         (horizontalScrollMode() == other->horizontalScrollMode()) &&
159         (verticalScrollMode() == other->verticalScrollMode()) &&
160 #endif
161         (dragEnabled() == other->dragEnabled()) &&
162         (editTriggers() == other->editTriggers()) &&
163         (iconSize() == other->iconSize()) &&
164         (selectionBehavior() == other->selectionBehavior()) &&
165         (selectionMode() == other->selectionMode()) &&
166         (showDropIndicator() == other->showDropIndicator()) &&
167         (tabKeyNavigation() == other->tabKeyNavigation()) &&
168         (textElideMode() == other->textElideMode()));
169     qDebug() << "AbstractDiagram::compare() AttributesModel: ";
170         // compare all of the properties stored in the attributes model
171     qDebug() << attributesModel()->compare( other->attributesModel() );
172     qDebug() << "AbstractDiagram::compare() own:";
173         // compare own properties
174     qDebug() <<
175         ((rootIndex().column() == other->rootIndex().column()) &&
176         (rootIndex().row() == other->rootIndex().row()) &&
177         (allowOverlappingDataValueTexts() == other->allowOverlappingDataValueTexts()) &&
178         (antiAliasing() == other->antiAliasing()) &&
179         (percentMode() == other->percentMode()) &&
180         (datasetDimension() == other->datasetDimension()));
181     */
182     return // compare QAbstractScrollArea properties
183         (horizontalScrollBarPolicy() == other->horizontalScrollBarPolicy()) &&
184         (verticalScrollBarPolicy() == other->verticalScrollBarPolicy()) &&

```

```

185         // compare QFrame properties
186         (frameShadow() == other->frameShadow()) &&
187         (frameShape() == other->frameShape()) &&
188 // frameWidth is a read-only property defined by the style, it should not be in here:
189         // (frameWidth() == other->frameWidth()) &&
190         (lineWidth() == other->lineWidth()) &&
191         (midLineWidth() == other->midLineWidth()) &&
192         // compare QAbstractItemView properties
193         (alternatingRowColors() == other->alternatingRowColors()) &&
194         (hasAutoScroll() == other->hasAutoScroll()) &&
195 #if QT_VERSION > 0x040199
196         (dragDropMode() == other->dragDropMode()) &&
197         (dragDropOverwriteMode() == other->dragDropOverwriteMode()) &&
198         (horizontalScrollMode() == other->horizontalScrollMode()) &&
199         (verticalScrollMode() == other->verticalScrollMode()) &&
200 #endif
201         (dragEnabled() == other->dragEnabled()) &&
202         (editTriggers() == other->editTriggers()) &&
203         (iconSize() == other->iconSize()) &&
204         (selectionBehavior() == other->selectionBehavior()) &&
205         (selectionMode() == other->selectionMode()) &&
206         (showDropIndicator() == other->showDropIndicator()) &&
207         (tabKeyNavigation() == other->tabKeyNavigation()) &&
208         (textElideMode() == other->textElideMode()) &&
209         // compare all of the properties stored in the attributes model
210         attributesModel()->compare( other->attributesModel() ) &&
211         // compare own properties
212         (rootIndex().column() == other->rootIndex().column()) &&
213         (rootIndex().row() == other->rootIndex().row()) &&
214         (allowOverlappingDataValueTexts() == other->allowOverlappingDataValueTexts()) &&
215         (antiAliasing() == other->antiAliasing()) &&
216         (percentMode() == other->percentMode()) &&
217         (datasetDimension() == other->datasetDimension());
218 }

```

#### 9.9.3.13 [AbstractCoordinatePlane](#) \* [AbstractDiagram::coordinatePlane\(\)](#) const [inherited]

The coordinate plane associated with the diagram.

This determines how coordinates in value space are mapped into pixel space. By default this is a [Cartesian-CoordinatePlane](#).

#### Returns:

The coordinate plane associated with the diagram.

Definition at line 220 of file `KDChartAbstractDiagram.cpp`.

References `d`.

Referenced by `KDChart::AbstractDiagram::checkInvariants()`, `KDChart::AbstractCartesianDiagram::layoutPlanes()`, `KDChart::PolarDiagram::paint()`, `KDChart::AbstractDiagram::paintDataValueTexts()`, `KDChart::AbstractDiagram::paintMarkers()`, `KDChart::AbstractPolarDiagram::polarCoordinatePlane()`, and `KDChart::AbstractCartesianDiagram::setCoordinatePlane()`.

```

221 {
222     return d->plane;
223 }

```

### 9.9.3.14 `const QPair< QPointF, QPointF > AbstractDiagram::dataBoundaries () const` [inherited]

Return the bottom left and top right data point, that the diagram will display (unless the grid adjusts these values).

This method returns a cached result of calculations done by `calculateDataBoundaries`. Classes derived from [AbstractDiagram](#) must implement the `calculateDataBoundaries` function, to specify their own way of calculating the data boundaries. If derived classes want to force recalculation of the data boundaries, they can call [setDataBoundariesDirty\(\)](#)

Returned value is in diagram coordinates.

Definition at line 225 of file `KDChartAbstractDiagram.cpp`.

References `KDChart::AbstractDiagram::calculateDataBoundaries()`, and `d`.

Referenced by `KDChart::CartesianCoordinatePlane::getRawDataBoundingRectFromDiagrams()`, `KDChart::PolarCoordinatePlane::layoutDiagrams()`, `KDChart::Plotter::paint()`, `KDChart::LineDiagram::paint()`, and `KDChart::BarDiagram::paint()`.

```
226 {
227     if( d->databoundariesDirty ){
228         d->databoundaries = calculateDataBoundaries ();
229         d->databoundariesDirty = false;
230     }
231     return d->databoundaries;
232 }
```

### 9.9.3.15 `void AbstractDiagram::dataChanged (const QModelIndex & topLeft, const QModelIndex & bottomRight)` [virtual, inherited]

[reimplemented]

Definition at line 330 of file `KDChartAbstractDiagram.cpp`.

References `KDChart::AbstractDiagram::setDataBoundariesDirty()`.

```
332 {
333     Q_UNUSED( topLeft );
334     Q_UNUSED( bottomRight );
335     // We are still too dumb to do intelligent updates...
336     setDataBoundariesDirty();
337     scheduleDelayedItemsLayout();
338 }
```

### 9.9.3.16 `void KDChart::AbstractDiagram::dataHidden ()` [signal, inherited]

This signal is emitted, when the hidden status of at least one data cell was (un)set.

Referenced by `KDChart::AbstractDiagram::setHidden()`.

### 9.9.3.17 `QList< QBrush > AbstractDiagram::datasetBrushes () const` [inherited]

The set of dataset brushes currently used, for use in legends, etc.



**Note:**

Cell-level override brushes, if set, take precedence over the dataset values, so you might need to check these too, in order to find the brush, that is used for a single cell.

**Returns:**

The current set of dataset brushes.

Definition at line 1018 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::attributesModelRootIndex(), KDChart::AttributesModel::columnCount(), KDChart::DatasetBrushRole, and KDChart::AbstractDiagram::datasetDimension().

Referenced by KDChart::Legend::setBrushesFromDiagram().

```

1019 {
1020     QList<QBrush> ret;
1021     if( model() == 0 )
1022         return ret;
1023
1024     const int columnCount = attributesModel()->columnCount(attributesModelRootIndex());
1025     for( int i = 0; i < columnCount / datasetDimension(); ++i )
1026         ret << QVariantValue<QBrush>( attributesModel()->headerData( i, Qt::Vertical, DatasetBrushRole ),
1027
1028     return ret;
1029 }
```

**9.9.3.18 int AbstractDiagram::datasetDimension () const** [inherited]

The dataset dimension of a diagram determines, how many value dimensions it expects each datapoint to have.

For each dimension it will expect one column of values in the model. If the dimensionality is 1, automatic values will be used for the abscissa.

For example a diagram with the default dimension of 1, will have one column per datapoint (the y values) and will use automatic values for the x axis (1, 2, 3, ... n). If the dimension is 2, the diagram will use the first, (and the third, fifth, etc) columns as X values, and the second, (and the fourth, sixth, etc) column as Y values.

**Returns:**

The dataset dimension of the diagram.

Definition at line 1072 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::brush(), KDChart::AbstractDiagram::compare(), KDChart::AbstractDiagram::datasetBrushes(), KDChart::AbstractDiagram::datasetLabels(), KDChart::AbstractDiagram::datasetMarkers(), KDChart::AbstractDiagram::datasetPens(), KDChart::LineDiagram::getCellValues(), KDChart::CartesianCoordinatePlane::getDataDimensionsList(), KDChart::TernaryPointDiagram::paint(), KDChart::TernaryLineDiagram::paint(), KDChart::AbstractDiagram::paintDataValueTexts(), KDChart::AbstractDiagram::paintMarkers(), KDChart::AbstractDiagram::pen(), KDChart::AbstractDiagram::setPen(), KDChart::Plotter::setType(), and KDChart::LineDiagram::setType().

```

1073 {
1074     return d->datasetDimension();
1075 }

```

### 9.9.3.19 QStringList AbstractDiagram::datasetLabels () const [inherited]

The set of dataset labels currently displayed, for use in legends, etc.

#### Returns:

The set of dataset labels currently displayed.

Definition at line 1005 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::attributesModelRootIndex(), KDChart::AttributesModel::columnCount(), KDChart::AbstractDiagram::datasetDimension(), and KDChart::AttributesModel::headerData().

```

1006 {
1007     QStringList ret;
1008     if( model() == 0 )
1009         return ret;
1010
1011     const int columnCount = attributesModel()->columnCount(attributesModelRootIndex());
1012     for( int i = 0; i < columnCount / datasetDimension(); ++i )
1013         ret << attributesModel()->headerData( i, Qt::Horizontal, Qt::DisplayRole ).toString();
1014
1015     return ret;
1016 }

```

### 9.9.3.20 QList< [MarkerAttributes](#) > AbstractDiagram::datasetMarkers () const [inherited]

The set of dataset markers currently used, for use in legends, etc.

#### Note:

Cell-level override markers, if set, take precedence over the dataset values, so you might need to check these too, in order to find the marker, that is shown for a single cell.

#### Returns:

The current set of dataset brushes.

Definition at line 1044 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::attributesModelRootIndex(), KDChart::AttributesModel::columnCount(), KDChart::AbstractDiagram::datasetDimension(), and KDChart::DataValueLabelAttributesRole.

```

1045 {
1046     QList<MarkerAttributes> ret;
1047     if( model() == 0 )
1048         return ret;
1049
1050     const int columnCount = attributesModel()->columnCount(attributesModelRootIndex());
1051     for( int i = 0; i < columnCount / datasetDimension(); ++i )

```

```

1052     {
1053         const DataValueAttributes a =
1054             qVariantValue<DataValueAttributes>( attributesModel()->headerData( i, Qt::Vertical, DataV
1055         ret << a.markerAttributes();
1056     }
1057     return ret;
1058 }

```

### 9.9.3.21 QList< QPen > AbstractDiagram::datasetPens () const [inherited]

The set of dataset pens currently used, for use in legends, etc.

#### Note:

Cell-level override pens, if set, take precedence over the dataset values, so you might need to check these too, in order to find the pens, that is used for a single cell.

#### Returns:

The current set of dataset pens.

Definition at line 1031 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::attributesModelRootIndex(), KDChart::AttributesModel::columnCount(), KDChart::AbstractDiagram::datasetDimension(), and KDChart::DatasetPenRole.

```

1032 {
1033     QList<QPen> ret;
1034     if( model() == 0 )
1035         return ret;
1036
1037     const int columnCount = attributesModel()->columnCount(attributesModelRootIndex());
1038     for( int i = 0; i < columnCount / datasetDimension(); ++i )
1039         ret << qVariantValue<QPen>( attributesModel()->headerData( i, Qt::Vertical, DatasetPenRole )
1040
1041     return ret;
1042 }

```

### 9.9.3.22 DataValueAttributes AbstractDiagram::dataValueAttributes (const QModelIndex & index) const [inherited]

Retrieve the [DataValueAttributes](#) for the given index.

This will fall back automatically to what was set at dataset or model level, if there are no datapoint specific settings.

#### Parameters:

*index* The datapoint to retrieve the attributes for.

#### Returns:

The [DataValueAttributes](#) for the given index.

Definition at line 421 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AttributesModel::data(), and KDChart::DataValueLabelAttributesRole.

```

422 {
423     return qVariantValue<DataValueAttributes>(
424         attributesModel()->data( attributesModel()->mapFromSource(index),
425                                 KDChart::DataValueLabelAttributesRole ));
426 }

```

### 9.9.3.23 [DataValueAttributes](#) AbstractDiagram::dataValueAttributes (int *column*) const [inherited]

Retrieve the [DataValueAttributes](#) for the given dataset.

This will fall back automatically to what was set at model level, if there are no dataset specific settings.

#### Parameters:

*column* The dataset to retrieve the attributes for.

#### Returns:

The [DataValueAttributes](#) for the given dataset.

Definition at line 414 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::columnToIndex(), KDChart::AttributesModel::data(), and KDChart::DataValueLabelAttributesRole.

```

415 {
416     return qVariantValue<DataValueAttributes>(
417         attributesModel()->data( attributesModel()->mapFromSource(columnToIndex( column )),
418                                 KDChart::DataValueLabelAttributesRole ));
419 }

```

### 9.9.3.24 [DataValueAttributes](#) AbstractDiagram::dataValueAttributes () const [inherited]

Retrieve the [DataValueAttributes](#) specified globally.

This will fall back automatically to the default settings, if there are no specific settings.

#### Returns:

The global [DataValueAttributes](#).

Definition at line 408 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DataValueLabelAttributesRole, and KDChart::AttributesModel::modelData().

Referenced by KDChart::AbstractDiagram::paintDataValueText(), KDChart::AbstractDiagram::paintMarker(), and KDChart::TernaryLineDiagram::TernaryLineDiagram().

```

409 {
410     return qVariantValue<DataValueAttributes>(
411         attributesModel()->modelData( KDChart::DataValueLabelAttributesRole ));
412 }

```

**9.9.3.25 void AbstractDiagram::doItemsLayout ()** [virtual, inherited]

[reimplemented]

Definition at line 321 of file KDChartAbstractDiagram.cpp.

References d, and KDChart::AbstractDiagram::update().

```
322 {  
323     if ( d->plane ) {  
324         d->plane->layoutDiagrams();  
325         update();  
326     }  
327     QAbstractItemView::doItemsLayout();  
328 }
```

**9.9.3.26 qreal AbstractPieDiagram::granularity () const**

**Returns:**

the granularity.

Definition at line 69 of file KDChartAbstractPieDiagram.cpp.

References d.

Referenced by KDChart::PieDiagram::paint().

```
70 {  
71     return (d->granularity < 0.05 || d->granularity > 36.0)  
72         ? 1.0  
73         : d->granularity;  
74 }
```

**9.9.3.27 int AbstractDiagram::horizontalOffset () const** [virtual, inherited]

[reimplemented]

Definition at line 950 of file KDChartAbstractDiagram.cpp.

```
951 { return 0; }
```

**9.9.3.28 QModelIndex AbstractDiagram::indexAt (const QPoint & *point*) const** [virtual, inherited]

[reimplemented]

Definition at line 1098 of file KDChartAbstractDiagram.cpp.

References d.

```
1099 {  
1100     return d->indexAt( point );  
1101 }
```

### 9.9.3.29 QModelIndexList AbstractDiagram::indexesAt (const QPoint & *point*) const [inherited]

This method is added alongside with `indexAt` from QAIM, since in `kdchart` multiple indexes can be displayed at the same spot.

Definition at line 1103 of file `KDChartAbstractDiagram.cpp`.

References `d`.

```
1104 {
1105     return d->indexesAt( point );
1106 }
```

### 9.9.3.30 bool AbstractDiagram::isHidden (const QModelIndex & *index*) const [inherited]

Retrieve the hidden status for the given index.

This will fall back automatically to what was set at dataset or diagram level, if there are no datapoint specific settings.

#### Parameters:

*index* The datapoint to retrieve the hidden status for.

#### Returns:

The hidden status for the given index.

Definition at line 380 of file `KDChartAbstractDiagram.cpp`.

References `KDChart::AbstractDiagram::attributesModel()`, `KDChart::AttributesModel::data()`, and `KDChart::DataHiddenRole`.

```
381 {
382     return qVariantValue<bool>(
383         attributesModel()->data(
384             attributesModel()->mapFromSource(index),
385             DataHiddenRole ) );
386 }
```

### 9.9.3.31 bool AbstractDiagram::isHidden (int *column*) const [inherited]

Retrieve the hidden status for the given dataset.

This will fall back automatically to what was set at diagram level, if there are no dataset specific settings.

#### Parameters:

*column* The dataset to retrieve the hidden status for.

#### Returns:

The hidden status for the given dataset.

Definition at line 373 of file `KDChartAbstractDiagram.cpp`.

References `KDChart::AbstractDiagram::attributesModel()`, `KDChart::AbstractDiagram::columnToIndex()`, `KDChart::AttributesModel::data()`, and `KDChart::DataHiddenRole`.

```

374 {
375     return qVariantValue<bool>(
376         attributesModel()->data(
377             attributesModel()->mapFromSource(columnToIndex( column )),
378             DataHiddenRole ) );
379 }

```

### 9.9.3.32 bool AbstractDiagram::isHidden () const [inherited]

Retrieve the hidden status specied globally.

This will fall back automatically to the default settings ( = not hidden), if there are no specific settings.

#### Returns:

The global hidden status.

Definition at line 367 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DataHiddenRole, and KDChart::AttributesModel::modelData().

Referenced by KDChart::LineDiagram::valueForCellTesting().

```

368 {
369     return qVariantValue<bool>(
370         attributesModel()->modelData( DataHiddenRole ) );
371 }

```

### 9.9.3.33 bool AbstractDiagram::isIndexHidden (const QModelIndex & index) const [virtual, inherited]

[reimplemented]

Definition at line 956 of file KDChartAbstractDiagram.cpp.

```

957 { return true; }

```

### 9.9.3.34 QStringList AbstractDiagram::itemRowLabels () const [inherited]

The set of item row labels currently displayed, for use in Abscissa axes, etc.

#### Returns:

The set of item row labels currently displayed.

Definition at line 989 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::attributesModel-RootIndex(), KDChart::AttributesModel::rowCount(), KDChart::AbstractDiagram::unitPrefix(), and KDChart::AbstractDiagram::unitSuffix().

```

990 {
991     QStringList ret;

```

```

992     if( model() ){
993         //qDebug() << "AbstractDiagram::itemRowLabels(): " << attributesModel()->rowCount(attributesModelRootIndex());
994         const int rowCount = attributesModel()->rowCount(attributesModelRootIndex());
995         for( int i = 0; i < rowCount; ++i ){
996             //qDebug() << "item row label: " << attributesModel()->headerData( i, Qt::Vertical, Qt::DisplayRole );
997             ret << unitPrefix( i, Qt::Horizontal, true ) +
998                 attributesModel()->headerData( i, Qt::Vertical, Qt::DisplayRole ).toString() +
999                 unitSuffix( i, Qt::Horizontal, true );
1000         }
1001     }
1002     return ret;
1003 }

```

**9.9.3.35 void KDChart::AbstractDiagram::layoutChanged ([AbstractDiagram](#) \*)** [signal, inherited]

Diagrams are supposed to emit this signal, when the layout of one of their element changes.

Layouts can change, for example, when axes are added or removed, or when the configuration was changed in a way that the axes or the diagram itself are displayed in a different geometry. Changes in the diagrams coordinate system also result in the [layoutChanged\(\)](#) signal being emitted.

Referenced by [KDChart::AbstractDiagram::setDatasetDimension\(\)](#), [setPieAttributes\(\)](#), [setThreeDPieAttributes\(\)](#), [KDChart::Plotter::setType\(\)](#), [KDChart::LineDiagram::setType\(\)](#), and [KDChart::BarDiagram::setType\(\)](#).

**9.9.3.36 void KDChart::AbstractDiagram::modelsChanged ()** [signal, inherited]

This signal is emitted, when either the model or the [AttributesModel](#) is replaced.

Referenced by [KDChart::AbstractDiagram::setAttributesModel\(\)](#), and [KDChart::AbstractDiagram::setModel\(\)](#).

**9.9.3.37 QModelIndex AbstractDiagram::moveCursor (CursorAction *cursorAction*, Qt::KeyboardModifiers *modifiers*)** [virtual, inherited]

[reimplemented]

Definition at line 947 of file [KDChartAbstractDiagram.cpp](#).

```

948 { return QModelIndex(); }

```

**9.9.3.38 virtual double KDChart::AbstractPolarDiagram::numberOfGridRings () const** [pure virtual, inherited]

Implemented in [KDChart::PieDiagram](#), [KDChart::PolarDiagram](#), and [KDChart::RingDiagram](#).

**9.9.3.39 virtual double KDChart::AbstractPolarDiagram::numberOfValuesPerDataset () const** [pure virtual, inherited]

Implemented in [KDChart::PieDiagram](#), [KDChart::PolarDiagram](#), and [KDChart::RingDiagram](#).

Referenced by [KDChart::AbstractPolarDiagram::columnCount\(\)](#).



### 9.9.3.40 virtual void KDChart::AbstractDiagram::paint (PaintContext \* paintContext) [pure virtual, inherited]

Draw the diagram contents to the rectangle and painter, that are passed in as part of the paint context.

#### Parameters:

*paintContext* All information needed for painting.

Implemented in [KDChart::BarDiagram](#), [KDChart::LineDiagram](#), [KDChart::PieDiagram](#), [KDChart::Plotter](#), [KDChart::PolarDiagram](#), [KDChart::RingDiagram](#), [KDChart::TernaryLineDiagram](#), and [KDChart::TernaryPointDiagram](#).

### 9.9.3.41 void AbstractDiagram::paintDataValueText (QPainter \* painter, const QModelIndex & index, const QPointF & pos, double value) [inherited]

Definition at line 468 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::DataValueAttributes::dataLabel\(\)](#), [KDChart::AbstractDiagram::dataValueAttributes\(\)](#), [KDChart::DataValueAttributes::decimalDigits\(\)](#), [KDChart::DataValueAttributes::isVisible\(\)](#), [KDChartEnums::MeasureOrientationMinimum](#), [KDChart::DataValueAttributes::position\(\)](#), [KDChart::DataValueAttributes::prefix\(\)](#), [KDChart::DataValueAttributes::suffix\(\)](#), and [KDChart::DataValueAttributes::textAttributes\(\)](#).

Referenced by [KDChart::RingDiagram::paint\(\)](#), [KDChart::PolarDiagram::paint\(\)](#), and [KDChart::AbstractDiagram::paintDataValueTexts\(\)](#).

```

472 {
473     // paint one data series
474     const DataValueAttributes a( dataValueAttributes(index) );
475     if ( !a.isVisible() ) return;
476
477     // handle decimal digits
478     int decimalDigits = a.decimalDigits();
479     int decimalPos = QString::number( value ).indexOf( QLatin1Char( '.' ) );
480     QString roundedValue;
481     if ( a.dataLabel().isNull() ) {
482         if ( decimalPos > 0 && value != 0 )
483             roundedValue = roundValues( value, decimalPos, decimalDigits );
484         else
485             roundedValue = QString::number( value );
486     } else
487         roundedValue = a.dataLabel();
488     // handle prefix and suffix
489     if ( !a.prefix().isNull() )
490         roundedValue.prepend( a.prefix() );
491
492     if ( !a.suffix().isNull() )
493         roundedValue.append( a.suffix() );
494
495     const TextAttributes ta( a.textAttributes() );
496     // FIXME draw the non-text bits, background, etc
497     if ( ta.isVisible() ) {
498
499         QPointF pt( pos );
500         /* for debugging:
501         PainterSaver painterSaver( painter );
502         painter->setPen( Qt::black );
503         painter->drawLine( pos - QPointF( 1,1), pos + QPointF( 1,1) );
504         painter->drawLine( pos - QPointF(-1,1), pos + QPointF(-1,1) );
505         */

```

```

506
507     QTextDocument doc;
508     if( Qt::mightBeRichText( roundedValue ) )
509         doc.setHtml( roundedValue );
510     else
511         doc.setPlainText( roundedValue );
512
513     const RelativePosition relPos( a.position( value >= 0.0 ) );
514     const Qt::Alignment alignBottomLeft = Qt::AlignBottom | Qt::AlignLeft;
515     const QFont calculatedFont( ta.calculatedFont( d->plane, KDChartEnums::MeasureOrientationMinimum ) );
516     const QRectF boundRect( d->cachedFontMetrics( calculatedFont, painter->device() )->boundingRect( roundedValue ) );
517
518     // To place correctly
519     pt.ry() -= boundRect.height();
520
521     //qDebug() << "calculatedFont's point size:" << calculatedFont.pointSizeF();
522     // adjust the text start point position, if alignment is not Bottom/Left
523     if( (relPos.alignment() & alignBottomLeft) != alignBottomLeft ){
524         if( relPos.alignment() & Qt::AlignRight )
525             pt.rx() -= boundRect.width();
526         else if( relPos.alignment() & Qt::AlignHCenter )
527             pt.rx() -= 0.5 * boundRect.width();
528
529         if( relPos.alignment() & Qt::AlignTop )
530             pt.ry() += boundRect.height();
531         else if( relPos.alignment() & Qt::AlignVCenter )
532             pt.ry() += 0.5 * boundRect.height();
533     }
534
535     // FIXME draw the non-text bits, background, etc
536
537     if ( a.showRepetitiveDataLabels() ||
538         pos.x() <= d->lastX ||
539         d->lastRoundedValue != roundedValue ) {
540         d->lastRoundedValue = roundedValue;
541         d->lastX = pos.x();
542         PainterSaver painterSaver( painter );
543
544         doc.setDefaultFont( calculatedFont );
545         QAbstractTextDocumentLayout::PaintContext context;
546         context.palette = palette();
547         context.palette.setColor( QPalette::Text, ta.pen().color() );
548
549         painter->translate( pt );
550         painter->rotate( ta.rotation() );
551
552         QAbstractTextDocumentLayout* layout = doc.documentLayout();
553         layout->draw( painter, context );
554     }
555 }
556 }

```

#### 9.9.3.42 void AbstractDiagram::paintDataValueTexts (QPainter \* *painter*) [protected, virtual, inherited]

Definition at line 584 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::checkInvariants(), KDChart::AbstractDiagram::coordinatePlane(), KDChart::AbstractDiagram::datasetDimension(), KDChart::AbstractDiagram::paintDataValueText(), and KDChart::AbstractCoordinatePlane::translate().

```

585 {
586     if ( !checkInvariants() ) return;
587     const int rowCount = model()->rowCount( rootIndex() );

```

```

588     const int columnCount = model()->columnCount(rootIndex());
589     for ( int i=datasetDimension()-1; i<columnCount; i += datasetDimension() ) {
590         for ( int j=0; j< rowCount; ++j ) {
591             const QModelIndex index = model()->index( j, i, rootIndex() );
592             double value = model()->data( index ).toDouble();
593             const QPointF pos = coordinatePlane()->translate( QPointF( j, value ) );
594             paintDataValueText( painter, index, pos, value );
595         }
596     }
597 }

```

### 9.9.3.43 void AbstractDiagram::paintMarker (QPainter \* *painter*, const QModelIndex & *index*, const QPointF & *pos*) [inherited]

Definition at line 600 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::brush(), KDChart::AbstractDiagram::checkInvariants(), d, KDChart::AbstractDiagram::dataValueAttributes(), KDChart::MarkerAttributes::isVisible(), KDChart::DataValueAttributes::isVisible(), KDChart::DataValueAttributes::markerAttributes(), KDChart::MarkerAttributes::markerColor(), KDChart::MarkerAttributes::markerSize(), KDChart::AbstractDiagram::paintMarker(), and KDChart::MarkerAttributes::pen().

```

603 {
604     if ( !checkInvariants() ) return;
605     DataValueAttributes a = dataValueAttributes(index);
606     if ( !a.isVisible() ) return;
607     const MarkerAttributes &ma = a.markerAttributes();
608     if ( !ma.isVisible() ) return;
609
610     PainterSaver painterSaver( painter );
611     QSizeF maSize( ma.markerSize() );
612     QBrush indexBrush( brush( index ) );
613     QPen indexPen( ma.pen() );
614     if ( ma.markerColor().isValid() )
615         indexBrush.setColor( ma.markerColor() );
616
617     paintMarker( painter, ma, indexBrush, indexPen, pos, maSize );
618
619     // workaround: BC cannot be changed, otherwise we would pass the
620     // index down to next-lower paintMarker function. So far, we
621     // basically save a circle of radius maSize at pos in the
622     // reverseMapper. This means that ^^ this version of paintMarker
623     // needs to be called to reverse-map the marker.
624     d->reverseMapper.addCircle( index.row(), index.column(), pos, 2 * maSize );
625 }

```

### 9.9.3.44 void AbstractDiagram::paintMarker (QPainter \* *painter*, const MarkerAttributes & *markerAttributes*, const QBrush & *brush*, const QPen & *pen*, const QPointF & *point*, const QSizeF & *size*) [virtual, inherited]

Definition at line 627 of file KDChartAbstractDiagram.cpp.

References KDChart::MarkerAttributes::Marker1Pixel, KDChart::MarkerAttributes::Marker4Pixels, KDChart::MarkerAttributes::MarkerCircle, KDChart::MarkerAttributes::MarkerCross, KDChart::MarkerAttributes::MarkerDiamond, KDChart::MarkerAttributes::MarkerFastCross, KDChart::MarkerAttributes::MarkerRing, KDChart::MarkerAttributes::MarkerSquare, and KDChart::MarkerAttributes::markerStyle().

Referenced by `KDChart::TernaryPointDiagram::paint()`, `KDChart::TernaryLineDiagram::paint()`, `KDChart::MarkerLayoutItem::paintIntoRect()`, `KDChart::AbstractDiagram::paintMarker()`, and `KDChart::AbstractDiagram::paintMarkers()`.

```

633 {
634     const QPen oldPen( painter->pen() );
635     // Pen is used to paint 4Pixels - 1 Pixel - Ring and FastCross types.
636     // make sure to use the brush color - see above in those cases.
637     const bool isFourPixels = (markerAttributes.markerStyle() == MarkerAttributes::Marker4Pixels);
638     if( isFourPixels || (markerAttributes.markerStyle() == MarkerAttributes::Marker1Pixel) ){
639         // for high-performance point charts with tiny point markers:
640         painter->setPen( QPen( brush.color().light() ) );
641         if( isFourPixels ){
642             const qreal x = pos.x();
643             const qreal y = pos.y();
644             painter->drawLine( QPointF(x-1.0,y-1.0),
645                             QPointF(x+1.0,y-1.0) );
646             painter->drawLine( QPointF(x-1.0,y),
647                             QPointF(x+1.0,y) );
648             painter->drawLine( QPointF(x-1.0,y+1.0),
649                             QPointF(x+1.0,y+1.0) );
650         }
651         painter->drawPoint( pos );
652     }else{
653         PainterSaver painterSaver( painter );
654         // we only a solid line surrounding the markers
655         QPen painterPen( pen );
656         painterPen.setStyle( Qt::SolidLine );
657         painter->setPen( painterPen );
658         painter->setBrush( brush );
659         painter->setRenderHint ( QPainter::Antialiasing );
660         painter->translate( pos );
661         switch ( markerAttributes.markerStyle() ) {
662             case MarkerAttributes::MarkerCircle:
663                 painter->drawEllipse( QRectF( 0 - maSize.height()/2, 0 - maSize.width()/2,
664                                             maSize.height(), maSize.width() ) );
665                 break;
666             case MarkerAttributes::MarkerSquare:
667                 {
668                     QRectF rect( 0 - maSize.width()/2, 0 - maSize.height()/2,
669                               maSize.width(), maSize.height() );
670                     painter->drawRect( rect );
671                     break;
672                 }
673             case MarkerAttributes::MarkerDiamond:
674                 {
675                     QVector <QPointF > diamondPoints;
676                     QPointF top, left, bottom, right;
677                     top = QPointF( 0, 0 - maSize.height()/2 );
678                     left = QPointF( 0 - maSize.width()/2, 0 );
679                     bottom = QPointF( 0, maSize.height()/2 );
680                     right = QPointF( maSize.width()/2, 0 );
681                     diamondPoints << top << left << bottom << right;
682                     painter->drawPolygon( diamondPoints );
683                     break;
684                 }
685             // both handled on top of the method:
686             case MarkerAttributes::Marker1Pixel:
687             case MarkerAttributes::Marker4Pixels:
688                 break;
689             case MarkerAttributes::MarkerRing:
690                 {
691                     painter->setPen( QPen( brush.color() ) );
692                     painter->setBrush( Qt::NoBrush );
693                     painter->drawEllipse( QRectF( 0 - maSize.height()/2, 0 - maSize.width()/2,
694                                             maSize.height(), maSize.width() ) );
695                     break;

```

```

696     }
697     case MarkerAttributes::MarkerCross:
698     {
699         QRectF rect( maSize.width()*-0.5, maSize.height()*-0.2,
700                     maSize.width(), maSize.height()*0.4 );
701         painter->drawRect( rect );
702         rect.setTopLeft(QPointF( maSize.width()*-0.2, maSize.height()*-0.5 ));
703         rect.setSize(QSizeF( maSize.width()*0.4, maSize.height() ));
704         painter->drawRect( rect );
705         break;
706     }
707     case MarkerAttributes::MarkerFastCross:
708     {
709         QPointF left, right, top, bottom;
710         left = QPointF( -maSize.width()/2, 0 );
711         right = QPointF( maSize.width()/2, 0 );
712         top = QPointF( 0, -maSize.height()/2 );
713         bottom= QPointF( 0, maSize.height()/2 );
714         painter->setPen( QPen( brush.color() ) );
715         painter->drawLine( left, right );
716         painter->drawLine( top, bottom );
717         break;
718     }
719     default:
720         Q_ASSERT_X ( false, "paintMarkers()",
721                     "Type item does not match a defined Marker Type." );
722 }
723 }
724 painter->setPen( oldPen );
725 }

```

#### 9.9.3.45 void AbstractDiagram::paintMarkers (QPainter \* *painter*) [protected, virtual, inherited]

Definition at line 727 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::checkInvariants(), KDChart::AbstractDiagram::coordinatePlane(), KDChart::AbstractDiagram::datasetDimension(), KDChart::AbstractDiagram::paintMarker(), and KDChart::AbstractCoordinatePlane::translate().

```

728 {
729     if ( !checkInvariants() ) return;
730     const int rowCount = model()->rowCount(rootIndex());
731     const int columnCount = model()->columnCount(rootIndex());
732     for ( int i=datasetDimension()-1; i<columnCount; i += datasetDimension() ) {
733         for ( int j=0; j< rowCount; ++j ) {
734             const QModelIndex index = model()->index( j, i, rootIndex() );
735             double value = model()->data( index ).toDouble();
736             const QPointF pos = coordinatePlane()->translate( QPointF( j, value ) );
737             paintMarker( painter, index, pos );
738         }
739     }
740 }

```

#### 9.9.3.46 QPen AbstractDiagram::pen (const QModelIndex & *index*) const [inherited]

Retrieve the pen to be used, for painting the datapoint at the given index in the model.

##### Parameters:

*index* The index of the datapoint in the model.

**Returns:**

The pen to use for painting.

Definition at line 788 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AttributesModel::data(), KDChart::AbstractDiagram::datasetDimension(), KDChart::DatasetPenRole, and KDChart::AbstractDiagram::pen().

```

789 {
790     if( datasetDimension() > 1 )
791         return pen( index.column() );
792     return qVariantValue<QPen>(
793         attributesModel()->data(
794             attributesModel()->mapFromSource( index ),
795             DatasetPenRole ) );
796 }
```

**9.9.3.47 QPen AbstractDiagram::pen (int *dataset*) const** [inherited]

Retrieve the pen to be used for the given dataset.

This will fall back automatically to what was set at model level, if there are no dataset specific settings.

**Parameters:**

*dataset* The dataset to retrieve the pen for.

**Returns:**

The pen to use for painting.

Definition at line 777 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::columnToIndex(), KDChart::AttributesModel::data(), KDChart::AbstractDiagram::datasetDimension(), and KDChart::DatasetPenRole.

```

778 {
779     if( datasetDimension() > 1 )
780         dataset /= datasetDimension();
781     return qVariantValue<QPen>(
782         attributesModel()->data(
783             attributesModel()->mapFromSource( columnToIndex( dataset ) ),
784             DatasetPenRole ) );
785 }
786 }
```

**9.9.3.48 QPen AbstractDiagram::pen () const** [inherited]

Retrieve the pen to be used for painting datapoints globally.

This will fall back automatically to the default settings, if there are no specific settings.

**Returns:**

The pen to use for painting.

Definition at line 771 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AttributesModel::data(), and KDChart::DatasetPenRole.

Referenced by KDChart::TernaryPointDiagram::paint(), KDChart::TernaryLineDiagram::paint(), and KDChart::AbstractDiagram::pen().

```
772 {
773     return qVariantValue<QPen>(
774         attributesModel()->data( DatasetPenRole ) );
775 }
```

#### 9.9.3.49 bool AbstractDiagram::percentMode () const [inherited]

Definition at line 462 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::compare(), and KDChart::CartesianCoordinatePlane::getDataDimensionsList().

```
463 {
464     return d->percent;
465 }
```

#### 9.9.3.50 [PieAttributes](#) AbstractPieDiagram::pieAttributes (const QModelIndex & *index*) const

Definition at line 122 of file KDChartAbstractPieDiagram.cpp.

References d, and KDChart::PieAttributesRole.

```
123 {
124     return qVariantValue<PieAttributes>(
125         d->attributesModel->data(
126             d->attributesModel->mapFromSource( index ),
127             PieAttributesRole ) );
128 }
```

#### 9.9.3.51 [PieAttributes](#) AbstractPieDiagram::pieAttributes (int *column*) const

Definition at line 114 of file KDChartAbstractPieDiagram.cpp.

References KDChart::AbstractDiagram::columnToIndex(), d, and KDChart::PieAttributesRole.

```
115 {
116     return qVariantValue<PieAttributes>(
117         d->attributesModel->data(
118             d->attributesModel->mapFromSource( columnToIndex( column ) ).column(),
119             PieAttributesRole ) );
120 }
```

### 9.9.3.52 [PieAttributes](#) AbstractPieDiagram::pieAttributes () const

Definition at line 105 of file KDChartAbstractPieDiagram.cpp.

References [d](#), and [KDChart::PieAttributesRole](#).

Referenced by [KDChart::PieDiagram::calculateDataBoundaries\(\)](#), and [KDChart::PieDiagram::paint\(\)](#).

```
106 {
107     return qVariantValue<PieAttributes>(
108         d->attributesModel->data( PieAttributesRole ) );
109 }
```

### 9.9.3.53 const [PolarCoordinatePlane](#) \* AbstractPolarDiagram::polarCoordinatePlane () const [inherited]

Definition at line 55 of file KDChartAbstractPolarDiagram.cpp.

References [KDChart::AbstractDiagram::coordinatePlane\(\)](#).

Referenced by [KDChart::PieDiagram::paint\(\)](#).

```
56 {
57     return dynamic_cast<const PolarCoordinatePlane*>( coordinatePlane() );
58 }
```

### 9.9.3.54 void KDChart::AbstractDiagram::propertiesChanged () [signal, inherited]

Emitted upon change of a property of the Diagram.

Referenced by [KDChart::Plotter::resetLineAttributes\(\)](#), [KDChart::LineDiagram::resetLineAttributes\(\)](#), [KDChart::AbstractDiagram::setAllowOverlappingDataValueTexts\(\)](#), [KDChart::AbstractDiagram::setAntiAliasing\(\)](#), [KDChart::BarDiagram::setBarAttributes\(\)](#), [KDChart::AbstractDiagram::setBrush\(\)](#), [KDChart::AbstractDiagram::setDataValueAttributes\(\)](#), [KDChart::Plotter::setLineAttributes\(\)](#), [KDChart::LineDiagram::setLineAttributes\(\)](#), [KDChart::AbstractDiagram::setPen\(\)](#), [KDChart::AbstractDiagram::setPercentMode\(\)](#), [KDChart::BarDiagram::setThreeDBarAttributes\(\)](#), [KDChart::Plotter::setThreeDLineAttributes\(\)](#), [KDChart::LineDiagram::setThreeDLineAttributes\(\)](#), [KDChart::Plotter::setType\(\)](#), [KDChart::LineDiagram::setType\(\)](#), [KDChart::BarDiagram::setType\(\)](#), [KDChart::Plotter::setValueTrackerAttributes\(\)](#), and [KDChart::LineDiagram::setValueTrackerAttributes\(\)](#).

### 9.9.3.55 virtual void KDChart::AbstractDiagram::resize (const QSizeF & area) [pure virtual, inherited]

Called by the widget's sizeEvent.

Adjust all internal structures, that are calculated, depending on the size of the widget.

**Parameters:**

*area*

Implemented in [KDChart::BarDiagram](#), [KDChart::LineDiagram](#), [KDChart::PieDiagram](#), [KDChart::Plotter](#), [KDChart::PolarDiagram](#), [KDChart::RingDiagram](#), [KDChart::TernaryLineDiagram](#), and [KDChart::TernaryPointDiagram](#).

Referenced by [KDChart::CartesianCoordinatePlane::setGeometry\(\)](#).



**9.9.3.56 void AbstractDiagram::scrollTo (const QModelIndex & *index*, ScrollHint *hint* = EnsureVisible)** [virtual, inherited]

[reimplemented]

Definition at line 942 of file KDChartAbstractDiagram.cpp.

```
943 {}
```

**9.9.3.57 void AbstractDiagram::setAllowOverlappingDataValueTexts (bool *allow*)** [inherited]

Set whether data value labels are allowed to overlap.

**Parameters:**

*allow* True means that overlapping labels are allowed.

Definition at line 434 of file KDChartAbstractDiagram.cpp.

References `d`, and `KDChart::AbstractDiagram::propertiesChanged()`.

```
435 {  
436     d->allowOverlappingDataValueTexts = allow;  
437     emit propertiesChanged();  
438 }
```

**9.9.3.58 void AbstractDiagram::setAntiAliasing (bool *enabled*)** [inherited]

Set whether anti-aliasing is to be used while rendering this diagram.

**Parameters:**

*enabled* True means that AA is enabled.

Definition at line 445 of file KDChartAbstractDiagram.cpp.

References `d`, and `KDChart::AbstractDiagram::propertiesChanged()`.

```
446 {  
447     d->antiAliasing = enabled;  
448     emit propertiesChanged();  
449 }
```

**9.9.3.59 void AbstractDiagram::setAttributesModel ([AttributesModel](#) \* *model*)** [virtual, inherited]

Associate an [AttributesModel](#) with this diagram.

Note that the diagram does `_not_` take ownership of the [AttributesModel](#). This should thus only be used with [AttributesModels](#) that have been explicitly created by the user, and are owned by her. Setting an [AttributesModel](#) that is internal to another diagram is an error.

Correct:

```
AttributesModel *am = new AttributesModel( model, 0 );
diagram1->setAttributesModel( am );
diagram2->setAttributesModel( am );
```

Wrong:

```
diagram1->setAttributesModel( diagram2->attributesModel() );
```

#### Parameters:

*model* The [AttributesModel](#) to use for this diagram.

See also:

[AttributesModel](#), [usesExternalAttributesModel](#)

Reimplemented in [KDChart::AbstractCartesianDiagram](#).

Definition at line 255 of file `KDChartAbstractDiagram.cpp`.

References `d`, [KDChart::AbstractDiagram::modelsChanged\(\)](#), and [KDChart::AbstractDiagram::setDataBoundariesDirty\(\)](#).

Referenced by [KDChart::AbstractCartesianDiagram::setAttributesModel\(\)](#).

```
256 {
257     if( amodel->sourceModel() != model() ) {
258         qWarning("KDChart::AbstractDiagram::setAttributesModel() failed: "
259                 "Trying to set an attributesmodel which works on a different "
260                 "model than the diagram.");
261         return;
262     }
263     if( qobject_cast<PrivateAttributesModel*>(amodel) ) {
264         qWarning("KDChart::AbstractDiagram::setAttributesModel() failed: "
265                 "Trying to set an attributesmodel that is private to another diagram.");
266         return;
267     }
268     d->setAttributesModel(amodel);
269     scheduleDelayedItemsLayout();
270     setDataBoundariesDirty();
271     emit modelsChanged();
272 }
```

#### 9.9.3.60 void AbstractDiagram::setAttributesModelRootIndex (const QModelIndex &) [protected, inherited]

Definition at line 293 of file `KDChartAbstractDiagram.cpp`.

References `d`, and [KDChart::AbstractDiagram::setDataBoundariesDirty\(\)](#).

Referenced by [KDChart::AbstractDiagram::setRootIndex\(\)](#).

```
294 {
295     d->attributesModelRootIndex=idx;
296     setDataBoundariesDirty();
297     scheduleDelayedItemsLayout();
298 }
```

**9.9.3.61 void AbstractDiagram::setBrush (const QBrush & *brush*)** [inherited]

Set the brush to be used, for painting all datasets in the model.

**Parameters:**

***brush*** The brush to use.

Definition at line 806 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DatasetBrushRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setModelData().

```
807 {
808     attributesModel()->setModelData(
809         qVariantFromValue( brush ), DatasetBrushRole );
810     emit propertiesChanged();
811 }
```

**9.9.3.62 void AbstractDiagram::setBrush (int *dataset*, const QBrush & *brush*)** [inherited]

Set the brush to be used, for painting the given dataset.

**Parameters:**

***dataset*** The dataset's column in the model.

***brush*** The brush to use.

Definition at line 813 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DatasetBrushRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setHeaderData().

```
814 {
815     attributesModel()->setHeaderData(
816         column, Qt::Vertical,
817         qVariantFromValue( brush ),
818         DatasetBrushRole );
819     emit propertiesChanged();
820 }
```

**9.9.3.63 void AbstractDiagram::setBrush (const QModelIndex & *index*, const QBrush & *brush*)** [inherited]

Set the brush to be used, for painting the datapoint at the given index.

**Parameters:**

***index*** The datapoint's index in the model.

***brush*** The brush to use.

Definition at line 798 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DatasetBrushRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setData().

```

799 {
800     attributesModel()->setData(
801         attributesModel()->mapFromSource( index ),
802         qVariantFromValue( brush ), DatasetBrushRole );
803     emit propertiesChanged();
804 }

```

#### 9.9.3.64 void AbstractDiagram::setCoordinatePlane ([AbstractCoordinatePlane](#) \* *plane*) [virtual, inherited]

Set the coordinate plane associated with the diagram.

This determines how coordinates in value space are mapped into pixel space. The chart takes ownership.

##### Returns:

The coordinate plane associated with the diagram.

Reimplemented in [KDChart::AbstractCartesianDiagram](#).

Definition at line 316 of file `KDChartAbstractDiagram.cpp`.

References d.

Referenced by `KDChart::AbstractCoordinatePlane::addDiagram()`, `KDChart::AbstractCartesianDiagram::setCoordinatePlane()`, and `KDChart::AbstractCoordinatePlane::takeDiagram()`.

```

317 {
318     d->plane = parent;
319 }

```

#### 9.9.3.65 void AbstractDiagram::setDataBoundariesDirty () const [protected, inherited]

Definition at line 234 of file `KDChartAbstractDiagram.cpp`.

References d.

Referenced by `KDChart::AbstractDiagram::dataChanged()`, `KDChart::Plotter::resize()`, `KDChart::LineDiagram::resize()`, `KDChart::BarDiagram::resize()`, `KDChart::AbstractDiagram::setAttributesModel()`, `KDChart::AbstractDiagram::setAttributesModelRootIndex()`, `KDChart::AbstractDiagram::setDatasetDimension()`, `KDChart::AbstractDiagram::setModel()`, `KDChart::BarDiagram::setThreeDBarAttributes()`, `KDChart::Plotter::setThreeDLineAttributes()`, `KDChart::LineDiagram::setThreeDLineAttributes()`, `KDChart::Plotter::setType()`, `KDChart::LineDiagram::setType()`, and `KDChart::BarDiagram::setType()`.

```

235 {
236     d->databoundariesDirty = true;
237 }

```

#### 9.9.3.66 void AbstractDiagram::setDatasetDimension (int *dimension*) [inherited]

Sets the dataset dimension of the diagram.

##### See also:

[datasetDimension](#).

**Parameters:***dimension*

Definition at line 1077 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::AbstractDiagram::layoutChanged\(\)](#), and [KDChart::AbstractDiagram::setDataBoundariesDirty\(\)](#).

Referenced by [KDChart::Widget::setType\(\)](#), [KDChart::TernaryLineDiagram::TernaryLineDiagram\(\)](#), and [KDChart::TernaryPointDiagram::TernaryPointDiagram\(\)](#).

```
1078 {
1079     if ( d->datasetDimension == dimension ) return;
1080     d->datasetDimension = dimension;
1081     setDataBoundariesDirty();
1082     emit layoutChanged( this );
1083 }
```

### 9.9.3.67 void AbstractDiagram::setDataValueAttributes (const [DataValueAttributes](#) & *a*) [inherited]

Set the [DataValueAttributes](#) for all datapoints in the model.

**Parameters:***a* The attributes to set.

Definition at line 428 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::DataValueLabelAttributesRole](#), and [KDChart::AbstractDiagram::propertiesChanged\(\)](#).

```
429 {
430     d->attributesModel->setModelData( qVariantFromValue( a ), DataValueLabelAttributesRole );
431     emit propertiesChanged();
432 }
```

### 9.9.3.68 void AbstractDiagram::setDataValueAttributes (int *dataset*, const [DataValueAttributes](#) & *a*) [inherited]

Set the [DataValueAttributes](#) for the given dataset.

**Parameters:***dataset* The dataset to set the attributes for.*a* The attributes to set.

Definition at line 400 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::DataValueLabelAttributesRole](#), and [KDChart::AbstractDiagram::propertiesChanged\(\)](#).

```
401 {
402     d->attributesModel->setHeaderData(
403         column, Qt::Vertical,
404         qVariantFromValue( a ), DataValueLabelAttributesRole );
405     emit propertiesChanged();
406 }
```

### 9.9.3.69 void AbstractDiagram::setDataValueAttributes (const QModelIndex & *index*, const [DataValueAttributes](#) & *a*) [inherited]

Set the [DataValueAttributes](#) for the given index.

#### Parameters:

*index* The datapoint to set the attributes for.

*a* The attributes to set.

Definition at line 389 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::DataValueLabelAttributesRole](#), and [KDChart::AbstractDiagram::propertiesChanged\(\)](#).

```

391 {
392     d->attributesModel->setData(
393         d->attributesModel->mapFromSource( index ),
394         qVariantFromValue( a ),
395         DataValueLabelAttributesRole );
396     emit propertiesChanged();
397 }
```

### 9.9.3.70 void AbstractPieDiagram::setGranularity (qreal *value*)

Set the granularity: the smaller the granularity the more your diagram segments will show facettes instead of rounded segments.

#### Parameters:

*value* the granularity value between 0.05 (one twentieth of a degree) and 36.0 (one tenth of a full circle), other values will be interpreted as 1.0.

Definition at line 64 of file KDChartAbstractPieDiagram.cpp.

References [d](#).

```

65 {
66     d->granularity = value;
67 }
```

### 9.9.3.71 void AbstractDiagram::setHidden (bool *hidden*) [inherited]

Hide (or unhide, resp.

) all datapoints in the model.

#### Note:

Hidden data are still taken into account by the coordinate plane, so neither the grid nor your axes' ranges will change, when you hide data. For totally removing data from KD Chart's view you can use another approach: e.g. you could define a proxy model on top of your data model, and register the proxy model calling [setModel\(\)](#) instead of registering your real data model.

**Parameters:**

*hidden* The hidden status to set.

Definition at line 359 of file KDChartAbstractDiagram.cpp.

References `d`, `KDChart::AbstractDiagram::dataHidden()`, and `KDChart::DataHiddenRole`.

```
360 {  
361     d->attributesModel->setModelData(  
362         qVariantFromValue( hidden ),  
363         DataHiddenRole );  
364     emit dataHidden();  
365 }
```

**9.9.3.72 void AbstractDiagram::setHidden (int *column*, bool *hidden*)** [inherited]

Hide (or unhide, resp.

) a dataset.

**Note:**

Hidden data are still taken into account by the coordinate plane, so neither the grid nor your axes' ranges will change, when you hide data. For totally removing data from KD Chart's view you can use another approach: e.g. you could define a proxy model on top of your data model, and register the proxy model calling [setModel\(\)](#) instead of registering your real data model.

**Parameters:**

*column* The dataset to set the hidden status for.

*hidden* The hidden status to set.

Definition at line 350 of file KDChartAbstractDiagram.cpp.

References `d`, `KDChart::AbstractDiagram::dataHidden()`, and `KDChart::DataHiddenRole`.

```
351 {  
352     d->attributesModel->setHeaderData(  
353         column, Qt::Vertical,  
354         qVariantFromValue( hidden ),  
355         DataHiddenRole );  
356     emit dataHidden();  
357 }
```

**9.9.3.73 void AbstractDiagram::setHidden (const QModelIndex & *index*, bool *hidden*)**  
[inherited]

Hide (or unhide, resp.

) a data cell.

**Note:**

Hidden data are still taken into account by the coordinate plane, so neither the grid nor your axes' ranges will change, when you hide data. For totally removing data from KD Chart's view you can use another approach: e.g. you could define a proxy model on top of your data model, and register the proxy model calling [setModel\(\)](#) instead of registering your real data model.

**Parameters:**

***index*** The datapoint to set the hidden status for.

***hidden*** The hidden status to set.

Definition at line 341 of file KDChartAbstractDiagram.cpp.

References `d`, `KDChart::AbstractDiagram::dataHidden()`, and `KDChart::DataHiddenRole`.

```

342 {
343     d->attributesModel->setData(
344         d->attributesModel->mapFromSource( index ),
345         qVariantFromValue( hidden ),
346         DataHiddenRole );
347     emit dataHidden();
348 }
```

#### 9.9.3.74 void AbstractDiagram::setModel (QAbstractItemModel \* *model*) [virtual, inherited]

Associate a model with the diagram.

Reimplemented in [KDChart::AbstractCartesianDiagram](#).

Definition at line 239 of file KDChartAbstractDiagram.cpp.

References `d`, `KDChart::AttributesModel::initFrom()`, `KDChart::AbstractDiagram::modelsChanged()`, and `KDChart::AbstractDiagram::setDataBoundariesDirty()`.

Referenced by `KDChart::AbstractCartesianDiagram::setModel()`, and `KDChart::Widget::setType()`.

```

240 {
241     QAbstractItemView::setModel( newModel );
242     AttributesModel* amodel = new PrivateAttributesModel( newModel, this );
243     amodel->initFrom( d->attributesModel );
244     d->setAttributesModel( amodel );
245     scheduleDelayedItemsLayout();
246     setDataBoundariesDirty();
247     emit modelsChanged();
248 }
```

#### 9.9.3.75 void AbstractDiagram::setPen (const QPen & *pen*) [inherited]

Set the pen to be used, for painting all datasets in the model.

**Parameters:**

***pen*** The pen to use.

Definition at line 753 of file KDChartAbstractDiagram.cpp.

References `KDChart::AbstractDiagram::attributesModel()`, `KDChart::DatasetPenRole`, `KDChart::AbstractDiagram::propertiesChanged()`, and `KDChart::AttributesModel::setModelData()`.

```

754 {
755     attributesModel()->setModelData(
756         qVariantFromValue( pen ), DatasetPenRole );
757     emit propertiesChanged();
758 }
```



**9.9.3.76 void AbstractDiagram::setPen (int *dataset*, const QPen & *pen*)** [inherited]

Set the pen to be used, for painting the given dataset.

**Parameters:**

*dataset* The dataset's row in the model.

*pen* The pen to use.

Definition at line 760 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::datasetDimension(), KDChart::DatasetPenRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setHeaderData().

```

761 {
762     if( datasetDimension() > 1 )
763         column *= datasetDimension();
764     attributesModel()->setHeaderData(
765         column, Qt::Vertical,
766         qVariantFromValue( pen ),
767         DatasetPenRole );
768     emit propertiesChanged();
769 }
```

**9.9.3.77 void AbstractDiagram::setPen (const QModelIndex & *index*, const QPen & *pen*)** [inherited]

Set the pen to be used, for painting the datapoint at the given index.

**Parameters:**

*index* The datapoint's index in the model.

*pen* The pen to use.

Definition at line 743 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::datasetDimension(), KDChart::DatasetPenRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setData().

```

744 {
745     if( datasetDimension() > 1 )
746         return setPen( index.column(), pen );
747     attributesModel()->setData(
748         attributesModel()->mapFromSource( index ),
749         qVariantFromValue( pen ), DatasetPenRole );
750     emit propertiesChanged();
751 }
```

**9.9.3.78 void AbstractDiagram::setPercentMode (bool *percent*)** [inherited]

Definition at line 456 of file KDChartAbstractDiagram.cpp.

References d, and KDChart::AbstractDiagram::propertiesChanged().

Referenced by KDChart::LineDiagram::setType(), and KDChart::BarDiagram::setType().

```

457 {
458     d->percent = percent;
459     emit propertiesChanged();
460 }

```

### 9.9.3.79 void AbstractPieDiagram::setPieAttributes (int *column*, const [PieAttributes](#) & *a*)

Definition at line 95 of file `KDChartAbstractPieDiagram.cpp`.

References `d`, `KDChart::AbstractDiagram::layoutChanged()`, and `KDChart::PieAttributesRole`.

```

96 {
97     d->attributesModel->setHeaderData(
98         column, Qt::Vertical, qVariantFromValue( attrs ), PieAttributesRole );
99     emit layoutChanged( this );
100 }

```

### 9.9.3.80 void AbstractPieDiagram::setPieAttributes (const [PieAttributes](#) & *a*)

Definition at line 89 of file `KDChartAbstractPieDiagram.cpp`.

References `d`, `KDChart::AbstractDiagram::layoutChanged()`, and `KDChart::PieAttributesRole`.

```

90 {
91     d->attributesModel->setModelData( qVariantFromValue( attrs ), PieAttributesRole );
92     emit layoutChanged( this );
93 }

```

### 9.9.3.81 void AbstractDiagram::setRootIndex (const [QModelIndex](#) & *idx*) [virtual, inherited]

Set the root index in the model, where the diagram starts referencing data for display.

[reimplemented]

Reimplemented in [KDChart::AbstractCartesianDiagram](#).

Definition at line 286 of file `KDChartAbstractDiagram.cpp`.

References `d`, and `KDChart::AbstractDiagram::setAttributesModelRootIndex()`.

Referenced by `KDChart::AbstractCartesianDiagram::setRootIndex()`.

```

287 {
288     QAbstractItemView::setRootIndex(idx);
289     setAttributesModelRootIndex( d->attributesModel->mapFromSource(idx) );
290 }

```

### 9.9.3.82 void AbstractDiagram::setSelection (const [QRect](#) & *rect*, [QItemSelectionModel::SelectionFlags](#) *command*) [virtual, inherited]

[reimplemented]

Definition at line 959 of file `KDChartAbstractDiagram.cpp`.

References `d`.

```

960 {
961     const QModelIndexList indexes = d->indexesIn( rect );
962     QItemSelection selection;
963     KDAB_FOREACH( const QModelIndex& index, indexes )
964     {
965         selection.append( QItemSelectionRange( index ) );
966     }
967     selectionModel()->select( selection, command );
968 }

```

### 9.9.3.83 void AbstractPieDiagram::setStartPosition (int *degrees*)

#### Deprecated

Use [PolarCoordinatePlane::setStartPosition\( qreal degrees \)](#) instead.

Definition at line 77 of file KDChartAbstractPieDiagram.cpp.

```

78 {
79     Q_UNUSED( degrees );
80     qWarning() << "Deprecated AbstractPieDiagram::setStartPosition() called, setting ignored.";
81 }

```

### 9.9.3.84 void AbstractPieDiagram::setThreeDPieAttributes (const QModelIndex & *index*, const [ThreeDPieAttributes](#) & *a*)

Definition at line 144 of file KDChartAbstractPieDiagram.cpp.

References [KDChart::AbstractDiagram::layoutChanged\(\)](#), and [KDChart::ThreeDPieAttributesRole](#).

```

145 {
146     model()->setData( index, QVariantFromValue( tda ), ThreeDPieAttributesRole );
147     emit layoutChanged( this );
148 }

```

### 9.9.3.85 void AbstractPieDiagram::setThreeDPieAttributes (int *column*, const [ThreeDPieAttributes](#) & *a*)

Definition at line 137 of file KDChartAbstractPieDiagram.cpp.

References [d](#), [KDChart::AbstractDiagram::layoutChanged\(\)](#), and [KDChart::ThreeDPieAttributesRole](#).

```

138 {
139     d->attributesModel->setHeaderData(
140         column, Qt::Vertical, QVariantFromValue( tda ), ThreeDPieAttributesRole );
141     emit layoutChanged( this );
142 }

```

### 9.9.3.86 void AbstractPieDiagram::setThreeDPieAttributes (const [ThreeDPieAttributes](#) & *a*)

Definition at line 131 of file KDChartAbstractPieDiagram.cpp.

References [d](#), [KDChart::AbstractDiagram::layoutChanged\(\)](#), and [KDChart::ThreeDPieAttributesRole](#).

```

132 {
133     d->attributesModel->setModelData( qVariantFromValue( tda ), ThreeDPieAttributesRole );
134     emit layoutChanged( this );
135 }

```

#### 9.9.3.87 void AbstractDiagram::setUnitPrefix (const QString & *prefix*, Qt::Orientation *orientation*) [inherited]

Sets the unit prefix for all values.

##### Parameters:

*prefix* the prefix to be set

*orientation* the orientation of the axis to set

Definition at line 864 of file KDChartAbstractDiagram.cpp.

References d.

```

865 {
866     d->unitPrefix[ orientation ] = prefix;
867 }

```

#### 9.9.3.88 void AbstractDiagram::setUnitPrefix (const QString & *prefix*, int *column*, Qt::Orientation *orientation*) [inherited]

Sets the unit prefix for one value.

##### Parameters:

*prefix* the prefix to be set

*column* the value using that prefix

*orientation* the orientation of the axis to set

Definition at line 854 of file KDChartAbstractDiagram.cpp.

References d.

```

855 {
856     d->unitPrefixMap[ column ][ orientation ]= prefix;
857 }

```

#### 9.9.3.89 void AbstractDiagram::setUnitSuffix (const QString & *suffix*, Qt::Orientation *orientation*) [inherited]

Sets the unit suffix for all values.

##### Parameters:

*suffix* the suffix to be set

*orientation* the orientation of the axis to set

Definition at line 885 of file KDChartAbstractDiagram.cpp.

References [d](#).

```
886 {
887     d->unitSuffix[ orientation ] = suffix;
888 }
```

### 9.9.3.90 void AbstractDiagram::setUnitSuffix (const QString &suffix, int column, Qt::Orientation orientation) [inherited]

Sets the unit suffix for one value.

#### Parameters:

*suffix* the suffix to be set

*column* the value using that suffix

*orientation* the orientation of the axis to set

Definition at line 875 of file KDChartAbstractDiagram.cpp.

References [d](#).

```
876 {
877     d->unitSuffixMap[ column ][ orientation ]= suffix;
878 }
```

### 9.9.3.91 int AbstractPieDiagram::startPosition () const

#### Deprecated

Use [qreal PolarCoordinatePlane::startPosition](#) instead.

Definition at line 83 of file KDChartAbstractPieDiagram.cpp.

```
84 {
85     qWarning() << "Deprecated AbstractPieDiagram::startPosition() called.";
86     return 0;
87 }
```

### 9.9.3.92 [ThreeDPieAttributes](#) AbstractPieDiagram::threeDPieAttributes (const QModelIndex &index) const

Definition at line 170 of file KDChartAbstractPieDiagram.cpp.

References [d](#), and [KDChart::ThreeDPieAttributesRole](#).

```
171 {
172     return qVariantValue<ThreeDPieAttributes>(
173         d->attributesModel->data(
174             d->attributesModel->mapFromSource( index ),
175             ThreeDPieAttributesRole ) );
176 }
```

### 9.9.3.93 [ThreeDPieAttributes](#) AbstractPieDiagram::threeDPieAttributes (int *column*) const

Definition at line 162 of file KDChartAbstractPieDiagram.cpp.

References KDChart::AbstractDiagram::columnToIndex(), d, and KDChart::ThreeDPieAttributesRole.

```
163 {
164     return qVariantValue<ThreeDPieAttributes>(
165         d->attributesModel->data(
166             d->attributesModel->mapFromSource( columnToIndex( column ) ).column(),
167             ThreeDPieAttributesRole ) );
168 }
```

### 9.9.3.94 [ThreeDPieAttributes](#) AbstractPieDiagram::threeDPieAttributes () const

Definition at line 153 of file KDChartAbstractPieDiagram.cpp.

References d, and KDChart::ThreeDPieAttributesRole.

Referenced by KDChart::PieDiagram::paint().

```
154 {
155     return qVariantValue<ThreeDPieAttributes>(
156         d->attributesModel->data( ThreeDPieAttributesRole ) );
157 }
```

### 9.9.3.95 [QString](#) AbstractDiagram::unitPrefix (Qt::Orientation *orientation*) const [inherited]

Returns the global unit prefix.

#### Parameters:

*orientation* the orientation of the axis

#### Returns:

the unit prefix

Definition at line 908 of file KDChartAbstractDiagram.cpp.

References d.

```
909 {
910     return d->unitPrefix[ orientation ];
911 }
```

### 9.9.3.96 [QString](#) AbstractDiagram::unitPrefix (int *column*, Qt::Orientation *orientation*, bool *fallback* = false) const [inherited]

Returns the unit prefix for a special value.

#### Parameters:

*column* the value which's prefix is requested

*orientation* the orientation of the axis

*fallback* if true, the global prefix is return when no specific one is set for that value

#### Returns:

the unit prefix

Definition at line 897 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::itemRowLabels(), and KDChart::CartesianAxis::paintCtx().

```
898 {
899     if( !fallback || d->unitPrefixMap[ column ].contains( orientation ) )
900         return d->unitPrefixMap[ column ][ orientation ];
901     return d->unitPrefix[ orientation ];
902 }
```

#### 9.9.3.97 QString AbstractDiagram::unitSuffix (Qt::Orientation *orientation*) const [inherited]

Returns the global unit suffix.

#### Parameters:

*orientation* the orientation of the axis

#### Returns:

the unit suffix

Definition at line 931 of file KDChartAbstractDiagram.cpp.

References d.

```
932 {
933     return d->unitSuffix[ orientation ];
934 }
```

#### 9.9.3.98 QString AbstractDiagram::unitSuffix (int *column*, Qt::Orientation *orientation*, bool *fallback* = false) const [inherited]

Returns the unit suffix for a special value.

#### Parameters:

*column* the value which's suffix is requested

*orientation* the orientation of the axis

*fallback* if true, the global suffix is return when no specific one is set for that value

#### Returns:

the unit suffix

Definition at line 920 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::itemRowLabels(), and KDChart::CartesianAxis::paintCtx().

```

921 {
922     if( !fallback || d->unitSuffixMap[ column ].contains( orientation ) )
923         return d->unitSuffixMap[ column ][ orientation ];
924     return d->unitSuffix[ orientation ];
925 }
```

### 9.9.3.99 void AbstractDiagram::update () const [inherited]

Definition at line 1091 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::doItemsLayout().

```

1092 {
1093     //qDebug("KDChart::AbstractDiagram::update() called");
1094     if( d->plane )
1095         d->plane->update();
1096 }
```

### 9.9.3.100 void KDChart::AbstractDiagram::useDefaultColors () [inherited]

Set the palette to be used, for painting datasets to the default palette.

See also:

[KDChart::Palette](#). FIXME: fold into one usePalette ([KDChart::Palette&](#)) method

Definition at line 974 of file KDChartAbstractDiagram.cpp.

References d, and KDChart::AttributesModel::PaletteTypeDefault.

```

975 {
976     d->attributesModel->setPaletteType( AttributesModel::PaletteTypeDefault );
977 }
```

### 9.9.3.101 void KDChart::AbstractDiagram::useRainbowColors () [inherited]

Set the palette to be used, for painting datasets to the rainbow palette.

See also:

[KDChart::Palette](#).

Definition at line 984 of file KDChartAbstractDiagram.cpp.

References d, and KDChart::AttributesModel::PaletteTypeRainbow.

```

985 {
986     d->attributesModel->setPaletteType( AttributesModel::PaletteTypeRainbow );
987 }
```



**9.9.3.102** `bool AbstractDiagram::usesExternalAttributesModel () const` [virtual, inherited]

Returns whether the diagram is using its own built-in attributes model or an attributes model that was set via `setAttributesModel`.

**See also:**

[setAttributesModel](#)

Definition at line 274 of file `KDChartAbstractDiagram.cpp`.

References `d`.

```
275 {  
276     return d->usesExternalAttributesModel();  
277 }
```

**9.9.3.103** `void KDChart::AbstractDiagram::useSubduedColors ()` [inherited]

Set the palette to be used, for painting datasets to the subdued palette.

**See also:**

[KDChart::Palette](#).

Definition at line 979 of file `KDChartAbstractDiagram.cpp`.

References `d`, and `KDChart::AttributesModel::PaletteTypeSubdued`.

```
980 {  
981     d->attributesModel->setPaletteType( AttributesModel::PaletteTypeSubdued );  
982 }
```

**9.9.3.104** `double AbstractDiagram::valueForCell (int row, int column) const` [protected, inherited]

Helper method, retrieving the data value (`DisplayRole`) for a given row and column.

**Parameters:**

*row* The row to query.

*column* The column to query.

**Returns:**

The value of the display role at the given row and column as a double.

**Deprecated**

Definition at line 1085 of file `KDChartAbstractDiagram.cpp`.

References `KDChart::AbstractDiagram::attributesModelRootIndex()`, and `d`.

```

1086 {
1087     return d->attributesModel->data(
1088         d->attributesModel->index( row, column, attributesModelRootIndex() ) ).toDouble();
1089 }

```

**9.9.3.105** `virtual double KDChart::AbstractPolarDiagram::valueTotals () const` [pure virtual, inherited]

Implemented in [KDChart::PieDiagram](#), [KDChart::PolarDiagram](#), and [KDChart::RingDiagram](#).

Referenced by [KDChart::PolarCoordinatePlane::layoutDiagrams\(\)](#).

**9.9.3.106** `int AbstractDiagram::verticalOffset () const` [virtual, inherited]

[reimplemented]

Definition at line 953 of file [KDChartAbstractDiagram.cpp](#).

```

954 { return 0; }

```

**9.9.3.107** `QRect AbstractDiagram::visualRect (const QModelIndex & index) const` [virtual, inherited]

[reimplemented]

Definition at line 937 of file [KDChartAbstractDiagram.cpp](#).

```

938 {
939     return QRect();
940 }

```

**9.9.3.108** `QRegion AbstractDiagram::visualRegionForSelection (const QItemSelection & selection) const` [virtual, inherited]

[reimplemented]

Definition at line 970 of file [KDChartAbstractDiagram.cpp](#).

```

971 { return QRegion(); }

```

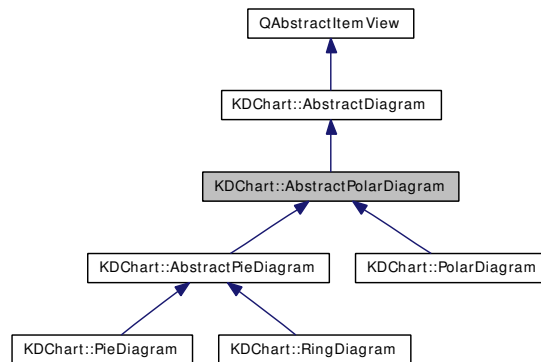
The documentation for this class was generated from the following files:

- [KDChartAbstractPieDiagram.h](#)
- [KDChartAbstractPieDiagram.cpp](#)

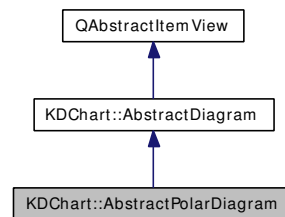
## 9.10 KDChart::AbstractPolarDiagram Class Reference

```
#include <KDChartAbstractPolarDiagram.h>
```

Inheritance diagram for KDChart::AbstractPolarDiagram:



Collaboration diagram for KDChart::AbstractPolarDiagram:



### 9.10.1 Detailed Description

Base class for diagrams based on a polar coordinate system.

Definition at line 39 of file KDChartAbstractPolarDiagram.h.

#### Signals

- void [dataHidden](#) ()  
*This signal is emitted, when the hidden status of at least one data cell was (un)set.*
- void [layoutChanged](#) ([AbstractDiagram](#) \*)  
*Diagrams are supposed to emit this signal, when the layout of one of their element changes.*
- void [modelsChanged](#) ()  
*This signal is emitted, when either the model or the [AttributesModel](#) is replaced.*
- void [propertiesChanged](#) ()  
*Emitted upon change of a property of the Diagram.*

## Public Member Functions

- [AbstractPolarDiagram](#) ([QWidget](#) \*parent=0, [PolarCoordinatePlane](#) \*plane=0)
- bool [allowOverlappingDataValueTexts](#) () const  
*Returns:*  
*Whether data value labels are allowed to overlap.*
- bool [antiAliasing](#) () const  
*Returns:*  
*Whether anti-aliasing is to be used for rendering this diagram.*
- virtual [AttributesModel](#) \* [attributesModel](#) () const  
*Returns the [AttributesModel](#), that is used by this diagram.*
- [QBrush](#) [brush](#) (const [QModelIndex](#) &index) const  
*Retrieve the brush to be used, for painting the datapoint at the given index in the model.*
- [QBrush](#) [brush](#) (int dataset) const  
*Retrieve the brush to be used for the given dataset.*
- [QBrush](#) [brush](#) () const  
*Retrieve the brush to be used for painting datapoints globally.*
- int [columnCount](#) () const
- bool [compare](#) (const [AbstractDiagram](#) \*other) const  
*Returns true if both diagrams have the same settings.*
- [AbstractCoordinatePlane](#) \* [coordinatePlane](#) () const  
*The coordinate plane associated with the diagram.*
- const [QPair](#)< [QPointF](#), [QPointF](#) > [dataBoundaries](#) () const  
*Return the bottom left and top right data point, that the diagram will display (unless the grid adjusts these values).*
- virtual void [dataChanged](#) (const [QModelIndex](#) &topLeft, const [QModelIndex](#) &bottomRight)  
*[reimplemented]*
- [QList](#)< [QBrush](#) > [datasetBrushes](#) () const  
*The set of dataset brushes currently used, for use in legends, etc.*
- int [datasetDimension](#) () const  
*The dataset dimension of a diagram determines, how many value dimensions it expects each datapoint to have.*
- [QStringList](#) [datasetLabels](#) () const  
*The set of dataset labels currently displayed, for use in legends, etc.*
- [QList](#)< [MarkerAttributes](#) > [datasetMarkers](#) () const  
*The set of dataset markers currently used, for use in legends, etc.*

- QList< QPen > [datasetPens](#) () const  
*The set of dataset pens currently used, for use in legends, etc.*
- [DataValueAttributes dataValueAttributes](#) (const QModelIndex &index) const  
*Retrieve the [DataValueAttributes](#) for the given index.*
- [DataValueAttributes dataValueAttributes](#) (int column) const  
*Retrieve the [DataValueAttributes](#) for the given dataset.*
- [DataValueAttributes dataValueAttributes](#) () const  
*Retrieve the [DataValueAttributes](#) specified globally.*
- virtual void [doItemsLayout](#) ()  
*[reimplemented]*
- virtual int [horizontalOffset](#) () const  
*[reimplemented]*
- virtual QModelIndex [indexAt](#) (const QPoint &point) const  
*[reimplemented]*
- QModelIndexList [indexesAt](#) (const QPoint &point) const  
*This method is added alongside with indexAt from QAIM, since in kdchart multiple indexes can be displayed at the same spot.*
- bool [isHidden](#) (const QModelIndex &index) const  
*Retrieve the hidden status for the given index.*
- bool [isHidden](#) (int column) const  
*Retrieve the hidden status for the given dataset.*
- bool [isHidden](#) () const  
*Retrieve the hidden status specified globally.*
- virtual bool [isIndexHidden](#) (const QModelIndex &index) const  
*[reimplemented]*
- QStringList [itemRowLabels](#) () const  
*The set of item row labels currently displayed, for use in Abscissa axes, etc.*
- virtual QModelIndex [moveCursor](#) (CursorAction cursorAction, Qt::KeyboardModifiers modifiers)  
*[reimplemented]*
- virtual double [numberOfGridRings](#) () const=0
- virtual double [numberOfValuesPerDataset](#) () const=0
- virtual void [paint](#) (PaintContext \*paintContext)=0  
*Draw the diagram contents to the rectangle and painter, that are passed in as part of the paint context.*
- void [paintDataValueText](#) (QPainter \*painter, const QModelIndex &index, const QPointF &pos, double value)

- void [paintMarker](#) (QPainter \*painter, const QModelIndex &index, const QPointF &pos)
- virtual void [paintMarker](#) (QPainter \*painter, const [MarkerAttributes](#) &markerAttributes, const QBrush &brush, const QPen &, const QPointF &point, const QSizeF &size)
- QPen [pen](#) (const QModelIndex &index) const  
*Retrieve the pen to be used, for painting the datapoint at the given index in the model.*
- QPen [pen](#) (int dataset) const  
*Retrieve the pen to be used for the given dataset.*
- QPen [pen](#) () const  
*Retrieve the pen to be used for painting datapoints globally.*
- bool [percentMode](#) () const
- const [PolarCoordinatePlane](#) \* [polarCoordinatePlane](#) () const
- virtual void [resize](#) (const QSizeF &area)=0  
*Called by the widget's sizeEvent.*
- virtual void [scrollTo](#) (const QModelIndex &index, ScrollHint hint=Ensure Visible)  
*[reimplemented]*
- void [setAllowOverlappingDataValueTexts](#) (bool allow)  
*Set whether data value labels are allowed to overlap.*
- void [setAntiAliasing](#) (bool enabled)  
*Set whether anti-aliasing is to be used while rendering this diagram.*
- virtual void [setAttributesModel](#) ([AttributesModel](#) \*model)  
*Associate an [AttributesModel](#) with this diagram.*
- void [setBrush](#) (const QBrush &brush)  
*Set the brush to be used, for painting all datasets in the model.*
- void [setBrush](#) (int dataset, const QBrush &brush)  
*Set the brush to be used, for painting the given dataset.*
- void [setBrush](#) (const QModelIndex &index, const QBrush &brush)  
*Set the brush to be used, for painting the datapoint at the given index.*
- virtual void [setCoordinatePlane](#) ([AbstractCoordinatePlane](#) \*plane)  
*Set the coordinate plane associated with the diagram.*
- void [setDatasetDimension](#) (int dimension)  
*Sets the dataset dimension of the diagram.*
- void [setDataValueAttributes](#) (const [DataValueAttributes](#) &a)  
*Set the [DataValueAttributes](#) for all datapoints in the model.*
- void [setDataValueAttributes](#) (int dataset, const [DataValueAttributes](#) &a)  
*Set the [DataValueAttributes](#) for the given dataset.*

- void [setDataValueAttributes](#) (const QModelIndex &index, const [DataValueAttributes](#) &a)  
*Set the [DataValueAttributes](#) for the given index.*
- void [setHidden](#) (bool hidden)  
*Hide (or unhide, resp.*
- void [setHidden](#) (int column, bool hidden)  
*Hide (or unhide, resp.*
- void [setHidden](#) (const QModelIndex &index, bool hidden)  
*Hide (or unhide, resp.*
- virtual void [setModel](#) (QAbstractItemModel \*model)  
*Associate a model with the diagram.*
- void [setPen](#) (const QPen &pen)  
*Set the pen to be used, for painting all datasets in the model.*
- void [setPen](#) (int dataset, const QPen &pen)  
*Set the pen to be used, for painting the given dataset.*
- void [setPen](#) (const QModelIndex &index, const QPen &pen)  
*Set the pen to be used, for painting the datapoint at the given index.*
- void [setPercentMode](#) (bool percent)
- virtual void [setRootIndex](#) (const QModelIndex &idx)  
*Set the root index in the model, where the diagram starts referencing data for display.*
- virtual void [setSelection](#) (const QRect &rect, QItemSelectionModel::SelectionFlags command)  
*[reimplemented]*
- void [setUnitPrefix](#) (const QString &prefix, Qt::Orientation orientation)  
*Sets the unit prefix for all values.*
- void [setUnitPrefix](#) (const QString &prefix, int column, Qt::Orientation orientation)  
*Sets the unit prefix for one value.*
- void [setUnitSuffix](#) (const QString &suffix, Qt::Orientation orientation)  
*Sets the unit suffix for all values.*
- void [setUnitSuffix](#) (const QString &suffix, int column, Qt::Orientation orientation)  
*Sets the unit suffix for one value.*
- QString [unitPrefix](#) (Qt::Orientation orientation) const  
*Returns the global unit prefix.*
- QString [unitPrefix](#) (int column, Qt::Orientation orientation, bool fallback=false) const  
*Returns the unit prefix for a special value.*
- QString [unitSuffix](#) (Qt::Orientation orientation) const

*Returns the global unit suffix.*

- QString [unitSuffix](#) (int column, Qt::Orientation orientation, bool fallback=false) const

*Returns the unit suffix for a special value.*

- void [update](#) () const
- void [useDefaultColors](#) ()

*Set the palette to be used, for painting datasets to the default palette.*

- void [useRainbowColors](#) ()

*Set the palette to be used, for painting datasets to the rainbow palette.*

- virtual bool [usesExternalAttributesModel](#) () const

*Returns whether the diagram is using its own built-in attributes model or an attributes model that was set via [setAttributesModel](#).*

- void [useSubduedColors](#) ()

*Set the palette to be used, for painting datasets to the subdued palette.*

- virtual double [valueTotals](#) () const=0
- virtual int [verticalOffset](#) () const

*[reimplemented]*

- virtual QRect [visualRect](#) (const QModelIndex &index) const

*[reimplemented]*

- virtual QRegion [visualRegionForSelection](#) (const QItemSelection &selection) const

*[reimplemented]*

- virtual [~AbstractPolarDiagram](#) ()

## Protected Member Functions

- QModelIndex [attributesModelRootIndex](#) () const
- virtual const QPair< QPointF, QPointF > [calculateDataBoundaries](#) () const=0
- virtual bool [checkInvariants](#) (bool justReturnTheStatus=false) const
- QModelIndex [columnToIndex](#) (int column) const
- virtual void [paintDataValueTexts](#) (QPainter \*painter)
- virtual void [paintMarkers](#) (QPainter \*painter)
- void [setAttributesModelRootIndex](#) (const QModelIndex &)
- void [setDataBoundariesDirty](#) () const
- double [valueForCell](#) (int row, int column) const

*Helper method, retrieving the data value (DisplayRole) for a given row and column.*



## 9.10.2 Constructor & Destructor Documentation

### 9.10.2.1 AbstractPolarDiagram::AbstractPolarDiagram (QWidget \* *parent* = 0, PolarCoordinatePlane \* *plane* = 0) [explicit]

Definition at line 48 of file KDCartAbstractPolarDiagram.cpp.

```
50      : AbstractDiagram ( new Private(), parent, plane )
51  {
52  }
```

### 9.10.2.2 virtual KDCart::AbstractPolarDiagram::~~AbstractPolarDiagram () [virtual]

Definition at line 48 of file KDCartAbstractPolarDiagram.h.

```
48 {}
```

## 9.10.3 Member Function Documentation

### 9.10.3.1 bool AbstractDiagram::allowOverlappingDataValueTexts () const [inherited]

#### Returns:

Whether data value labels are allowed to overlap.

Definition at line 440 of file KDCartAbstractDiagram.cpp.

References d.

Referenced by KDCart::AbstractDiagram::compare().

```
441 {
442     return d->allowOverlappingDataValueTexts;
443 }
```

### 9.10.3.2 bool AbstractDiagram::antiAliasing () const [inherited]

#### Returns:

Whether anti-aliasing is to be used for rendering this diagram.

Definition at line 451 of file KDCartAbstractDiagram.cpp.

References d.

Referenced by KDCart::AbstractDiagram::compare().

```
452 {
453     return d->antiAliasing;
454 }
```

### 9.10.3.3 **AttributesModel** \* **AbstractDiagram::attributesModel () const** [virtual, inherited]

Returns the [AttributesModel](#), that is used by this diagram.

By default each diagram owns its own [AttributesModel](#), which should never be deleted. Only if a user-supplied [AttributesModel](#) has been set does the pointer returned here not belong to the diagram.

#### Returns:

The [AttributesModel](#) associated with the diagram.

#### See also:

[setAttributeModel](#)

Definition at line 280 of file `KDChartAbstractDiagram.cpp`.

References d.

Referenced by `KDChart::AbstractDiagram::brush()`, `KDChart::AbstractDiagram::compare()`, `KDChart::AbstractDiagram::datasetBrushes()`, `KDChart::AbstractDiagram::datasetLabels()`, `KDChart::AbstractDiagram::datasetMarkers()`, `KDChart::AbstractDiagram::datasetPens()`, `KDChart::AbstractDiagram::dataValueAttributes()`, `KDChart::AbstractDiagram::isHidden()`, `KDChart::AbstractDiagram::itemRowLabels()`, `KDChart::RingDiagram::paint()`, `KDChart::PolarDiagram::paint()`, `KDChart::AbstractDiagram::pen()`, `KDChart::AbstractCartesianDiagram::setAttributeModel()`, `KDChart::BarDiagram::setBarAttributes()`, `KDChart::AbstractDiagram::setBrush()`, `KDChart::AbstractCartesianDiagram::setModel()`, `KDChart::AbstractDiagram::setPen()`, and `KDChart::TernaryLineDiagram::TernaryLineDiagram()`.

```
281 {
282     return d->attributesModel;
283 }
```

### 9.10.3.4 **QModelIndex** **AbstractDiagram::attributesModelRootIndex () const** [protected, inherited]

returns a `QModelIndex` pointing into the [AttributesModel](#) that corresponds to the root index of the diagram.

Definition at line 302 of file `KDChartAbstractDiagram.cpp`.

References d.

Referenced by `KDChart::AbstractDiagram::datasetBrushes()`, `KDChart::AbstractDiagram::datasetLabels()`, `KDChart::AbstractDiagram::datasetMarkers()`, `KDChart::AbstractDiagram::datasetPens()`, `KDChart::AbstractDiagram::itemRowLabels()`, `KDChart::Plotter::numberOfAbscissaSegments()`, `KDChart::LineDiagram::numberOfAbscissaSegments()`, `KDChart::BarDiagram::numberOfAbscissaSegments()`, `KDChart::Plotter::numberOfOrdinateSegments()`, `KDChart::LineDiagram::numberOfOrdinateSegments()`, `KDChart::BarDiagram::numberOfOrdinateSegments()`, `KDChart::AbstractDiagram::valueForCell()`, and `KDChart::LineDiagram::valueForCellTesting()`.

```
303 {
304     if ( !d->attributesModelRootIndex.isValid() )
305         d->attributesModelRootIndex = d->attributesModel->mapFromSource( rootIndex() );
306     return d->attributesModelRootIndex;
307 }
```

**9.10.3.5 QBrush AbstractDiagram::brush (const QModelIndex & *index*) const** [inherited]

Retrieve the brush to be used, for painting the datapoint at the given index in the model.

**Parameters:**

*index* The index of the datapoint in the model.

**Returns:**

The brush to use for painting.

Definition at line 838 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::brush(), KDChart::AttributesModel::data(), KDChart::DatasetBrushRole, and KDChart::AbstractDiagram::datasetDimension().

```
839 {
840     if( datasetDimension() > 1 )
841         return brush( index.column() );
842     return qVariantValue<QBrush>(
843         attributesModel()->data(
844             attributesModel()->mapFromSource( index ),
845             DatasetBrushRole ) );
846 }
```

**9.10.3.6 QBrush AbstractDiagram::brush (int *dataset*) const** [inherited]

Retrieve the brush to be used for the given dataset.

This will fall back automatically to what was set at model level, if there are no dataset specific settings.

**Parameters:**

*dataset* The dataset to retrieve the brush for.

**Returns:**

The brush to use for painting.

Definition at line 828 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::columnToIndex(), KDChart::AttributesModel::data(), KDChart::DatasetBrushRole, and KDChart::AbstractDiagram::datasetDimension().

```
829 {
830     if( datasetDimension() > 1 )
831         dataset /= datasetDimension();
832     return qVariantValue<QBrush>(
833         attributesModel()->data(
834             attributesModel()->mapFromSource( columnToIndex( dataset ) ),
835             DatasetBrushRole ) );
836 }
```

### 9.10.3.7 QBrush AbstractDiagram::brush () const [inherited]

Retrieve the brush to be used for painting datapoints globally.

This will fall back automatically to the default settings, if there are no specific settings.

#### Returns:

The brush to use for painting.

Definition at line 822 of file KDChartAbstractDiagram.cpp.

References [KDChart::AbstractDiagram::attributesModel\(\)](#), [KDChart::AttributesModel::data\(\)](#), and [KDChart::DatasetBrushRole](#).

Referenced by [KDChart::AbstractDiagram::brush\(\)](#), [KDChart::TernaryPointDiagram::paint\(\)](#), [KDChart::TernaryLineDiagram::paint\(\)](#), [KDChart::RingDiagram::paint\(\)](#), [KDChart::PolarDiagram::paint\(\)](#), and [KDChart::AbstractDiagram::paintMarker\(\)](#).

```
823 {
824     return qVariantValue<QBrush>(
825         attributesModel()->data( DatasetBrushRole ) );
826 }
```

### 9.10.3.8 virtual const QPair<QPointF, QPointF> KDChart::AbstractDiagram::calculateDataBoundaries () const [protected, pure virtual, inherited]

Implemented in [KDChart::BarDiagram](#), [KDChart::LineDiagram](#), [KDChart::PieDiagram](#), [KDChart::Plotter](#), [KDChart::PolarDiagram](#), [KDChart::RingDiagram](#), [KDChart::AbstractTernaryDiagram](#), [KDChart::TernaryLineDiagram](#), and [KDChart::TernaryPointDiagram](#).

Referenced by [KDChart::AbstractDiagram::dataBoundaries\(\)](#).

### 9.10.3.9 bool AbstractDiagram::checkInvariants (bool *justReturnTheStatus* = false) const [protected, virtual, inherited]

Definition at line 1060 of file KDChartAbstractDiagram.cpp.

References [KDChart::AbstractDiagram::coordinatePlane\(\)](#).

Referenced by [KDChart::RingDiagram::calculateDataBoundaries\(\)](#), [KDChart::PolarDiagram::calculateDataBoundaries\(\)](#), [KDChart::Plotter::calculateDataBoundaries\(\)](#), [KDChart::PieDiagram::calculateDataBoundaries\(\)](#), [KDChart::LineDiagram::calculateDataBoundaries\(\)](#), [KDChart::BarDiagram::calculateDataBoundaries\(\)](#), [KDChart::RingDiagram::paint\(\)](#), [KDChart::PolarDiagram::paint\(\)](#), [KDChart::Plotter::paint\(\)](#), [KDChart::PieDiagram::paint\(\)](#), [KDChart::LineDiagram::paint\(\)](#), [KDChart::BarDiagram::paint\(\)](#), [KDChart::AbstractDiagram::paintDataValueTexts\(\)](#), [KDChart::AbstractDiagram::paintMarker\(\)](#), and [KDChart::AbstractDiagram::paintMarkers\(\)](#).

```
1061 {
1062     if( ! justReturnTheStatus ){
1063         Q_ASSERT_X ( model(), "AbstractDiagram::checkInvariants()",
1064             "There is no usable model set, for the diagram." );
1065
1066         Q_ASSERT_X ( coordinatePlane(), "AbstractDiagram::checkInvariants()",
1067             "There is no usable coordinate plane set, for the diagram." );
1068     }
1069     return model() && coordinatePlane();
1070 }
```

**9.10.3.10 int AbstractPolarDiagram::columnCount () const**

Definition at line 60 of file KDChartAbstractPolarDiagram.cpp.

References `numberOfValuesPerDataset()`.

Referenced by `KDChart::PieDiagram::calculateDataBoundaries()`, `KDChart::PieDiagram::paint()`, and `KDChart::PieDiagram::valueTotals()`.

```
61 {
62     return static_cast<int>( numberOfValuesPerDataset() );
63 }
```

**9.10.3.11 QModelIndex AbstractDiagram::columnToIndex (int *column*) const** [protected, inherited]

Definition at line 309 of file KDChartAbstractDiagram.cpp.

Referenced by `KDChart::BarDiagram::barAttributes()`, `KDChart::AbstractDiagram::brush()`, `KDChart::AbstractDiagram::dataValueAttributes()`, `KDChart::AbstractDiagram::isHidden()`, `KDChart::Plotter::lineAttributes()`, `KDChart::LineDiagram::lineAttributes()`, `KDChart::AbstractDiagram::pen()`, `KDChart::AbstractPieDiagram::pieAttributes()`, `KDChart::BarDiagram::threeDBarAttributes()`, `KDChart::Plotter::threeDLineAttributes()`, `KDChart::LineDiagram::threeDLineAttributes()`, and `KDChart::AbstractPieDiagram::threeDPieAttributes()`.

```
310 {    // FIXME (Mirko): shouldn't this be headerData? instead of the index for the first row?
311     if( model() )
312         return QModelIndex( model()->index( 0, column, rootIndex() ) );
313     return QModelIndex();
314 }
```

**9.10.3.12 bool AbstractDiagram::compare (const [AbstractDiagram](#) \* *other*) const** [inherited]

Returns true if both diagrams have the same settings.

Definition at line 129 of file KDChartAbstractDiagram.cpp.

References `KDChart::AbstractDiagram::allowOverlappingDataValueTexts()`, `KDChart::AbstractDiagram::antiAliasing()`, `KDChart::AbstractDiagram::attributesModel()`, `KDChart::AttributesModel::compare()`, `KDChart::AbstractDiagram::datasetDimension()`, and `KDChart::AbstractDiagram::percentMode()`.

```
130 {
131     if( other == this ) return true;
132     if( ! other ){
133         //qDebug() << "AbstractDiagram::compare() cannot compare to Null pointer";
134         return false;
135     }
136     /*
137     qDebug() << "\n          AbstractDiagram::compare() QAbstractScrollArea:";
138         // compare QAbstractScrollArea properties
139     qDebug() <<
140         ((horizontalScrollBarPolicy() == other->horizontalScrollBarPolicy()) &&
141         (verticalScrollBarPolicy()    == other->verticalScrollBarPolicy()));
142     qDebug() << "AbstractDiagram::compare() QFrame:";
143         // compare QFrame properties
```

```

144     qDebug() <<
145         ((frameShadow() == other->frameShadow()) &&
146         (frameShape() == other->frameShape()) &&
147         (frameWidth() == other->frameWidth()) &&
148         (lineWidth() == other->lineWidth()) &&
149         (midLineWidth() == other->midLineWidth()));
150     qDebug() << "AbstractDiagram::compare() QAbstractItemView:";
151     // compare QAbstractItemView properties
152     qDebug() <<
153         ((alternatingRowColors() == other->alternatingRowColors()) &&
154         (hasAutoScroll() == other->hasAutoScroll()) &&
155 #if QT_VERSION > 0x040199
156         (dragDropMode() == other->dragDropMode()) &&
157         (dragDropOverwriteMode() == other->dragDropOverwriteMode()) &&
158         (horizontalScrollMode() == other->horizontalScrollMode()) &&
159         (verticalScrollMode() == other->verticalScrollMode()) &&
160 #endif
161         (dragEnabled() == other->dragEnabled()) &&
162         (editTriggers() == other->editTriggers()) &&
163         (iconSize() == other->iconSize()) &&
164         (selectionBehavior() == other->selectionBehavior()) &&
165         (selectionMode() == other->selectionMode()) &&
166         (showDropIndicator() == other->showDropIndicator()) &&
167         (tabKeyNavigation() == other->tabKeyNavigation()) &&
168         (textElideMode() == other->textElideMode()));
169     qDebug() << "AbstractDiagram::compare() AttributesModel: ";
170     // compare all of the properties stored in the attributes model
171     qDebug() << attributesModel()->compare( other->attributesModel() );
172     qDebug() << "AbstractDiagram::compare() own:";
173     // compare own properties
174     qDebug() <<
175         ((rootIndex().column() == other->rootIndex().column()) &&
176         (rootIndex().row() == other->rootIndex().row()) &&
177         (allowOverlappingDataValueTexts() == other->allowOverlappingDataValueTexts()) &&
178         (antiAliasing() == other->antiAliasing()) &&
179         (percentMode() == other->percentMode()) &&
180         (datasetDimension() == other->datasetDimension()));
181     */
182     return // compare QAbstractScrollArea properties
183         (horizontalScrollBarPolicy() == other->horizontalScrollBarPolicy()) &&
184         (verticalScrollBarPolicy() == other->verticalScrollBarPolicy()) &&
185         // compare QFrame properties
186         (frameShadow() == other->frameShadow()) &&
187         (frameShape() == other->frameShape()) &&
188     // frameWidth is a read-only property defined by the style, it should not be in here:
189     // (frameWidth() == other->frameWidth()) &&
190     (lineWidth() == other->lineWidth()) &&
191     (midLineWidth() == other->midLineWidth()) &&
192     // compare QAbstractItemView properties
193     (alternatingRowColors() == other->alternatingRowColors()) &&
194     (hasAutoScroll() == other->hasAutoScroll()) &&
195 #if QT_VERSION > 0x040199
196     (dragDropMode() == other->dragDropMode()) &&
197     (dragDropOverwriteMode() == other->dragDropOverwriteMode()) &&
198     (horizontalScrollMode() == other->horizontalScrollMode()) &&
199     (verticalScrollMode() == other->verticalScrollMode()) &&
200 #endif
201     (dragEnabled() == other->dragEnabled()) &&
202     (editTriggers() == other->editTriggers()) &&
203     (iconSize() == other->iconSize()) &&
204     (selectionBehavior() == other->selectionBehavior()) &&
205     (selectionMode() == other->selectionMode()) &&
206     (showDropIndicator() == other->showDropIndicator()) &&
207     (tabKeyNavigation() == other->tabKeyNavigation()) &&
208     (textElideMode() == other->textElideMode()) &&
209     // compare all of the properties stored in the attributes model
210     attributesModel()->compare( other->attributesModel() ) &&

```

```

211         // compare own properties
212         (rootIndex().column() == other->rootIndex().column()) &&
213         (rootIndex().row() == other->rootIndex().row()) &&
214         (allowOverlappingDataValueTexts() == other->allowOverlappingDataValueTexts()) &&
215         (antiAliasing() == other->antiAliasing()) &&
216         (percentMode() == other->percentMode()) &&
217         (datasetDimension() == other->datasetDimension());
218     }

```

### 9.10.3.13 AbstractCoordinatePlane \* AbstractDiagram::coordinatePlane () const

[inherited]

The coordinate plane associated with the diagram.

This determines how coordinates in value space are mapped into pixel space. By default this is a [Cartesian-CoordinatePlane](#).

#### Returns:

The coordinate plane associated with the diagram.

Definition at line 220 of file KDChartAbstractDiagram.cpp.

References [d](#).

Referenced by [KDChart::AbstractDiagram::checkInvariants\(\)](#), [KDChart::AbstractCartesianDiagram::layoutPlanes\(\)](#), [KDChart::PolarDiagram::paint\(\)](#), [KDChart::AbstractDiagram::paintDataValueTexts\(\)](#), [KDChart::AbstractDiagram::paintMarkers\(\)](#), [polarCoordinatePlane\(\)](#), and [KDChart::AbstractCartesianDiagram::setCoordinatePlane\(\)](#).

```

221 {
222     return d->plane;
223 }

```

### 9.10.3.14 const QPair< QPointF, QPointF > AbstractDiagram::dataBoundaries () const

[inherited]

Return the bottom left and top right data point, that the diagram will display (unless the grid adjusts these values).

This method returns a cached result of calculations done by [calculateDataBoundaries](#). Classes derived from [AbstractDiagram](#) must implement the [calculateDataBoundaries](#) function, to specify their own way of calculating the data boundaries. If derived classes want to force recalculation of the data boundaries, they can call [setDataBoundariesDirty\(\)](#)

Returned value is in diagram coordinates.

Definition at line 225 of file KDChartAbstractDiagram.cpp.

References [KDChart::AbstractDiagram::calculateDataBoundaries\(\)](#), and [d](#).

Referenced by [KDChart::CartesianCoordinatePlane::getRawDataBoundingRectFromDiagrams\(\)](#), [KDChart::PolarCoordinatePlane::layoutDiagrams\(\)](#), [KDChart::Plotter::paint\(\)](#), [KDChart::LineDiagram::paint\(\)](#), and [KDChart::BarDiagram::paint\(\)](#).

```

226 {
227     if ( d->databoundariesDirty ) {
228         d->databoundaries = calculateDataBoundaries ();

```

```

229         d->databoundariesDirty = false;
230     }
231     return d->databoundaries;
232 }

```

### 9.10.3.15 void AbstractDiagram::dataChanged (const QModelIndex & *topLeft*, const QModelIndex & *bottomRight*) [virtual, inherited]

[reimplemented]

Definition at line 330 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::setDataBoundariesDirty().

```

332 {
333     Q_UNUSED( topLeft );
334     Q_UNUSED( bottomRight );
335     // We are still too dumb to do intelligent updates...
336     setDataBoundariesDirty();
337     scheduleDelayedItemsLayout();
338 }

```

### 9.10.3.16 void KDChart::AbstractDiagram::dataHidden () [signal, inherited]

This signal is emitted, when the hidden status of at least one data cell was (un)set.

Referenced by KDChart::AbstractDiagram::setHidden().

### 9.10.3.17 QList< QBrush > AbstractDiagram::datasetBrushes () const [inherited]

The set of dataset brushes currently used, for use in legends, etc.

#### Note:

Cell-level override brushes, if set, take precedence over the dataset values, so you might need to check these too, in order to find the brush, that is used for a single cell.

#### Returns:

The current set of dataset brushes.

Definition at line 1018 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::attributesModelRootIndex(), KDChart::AttributesModel::columnCount(), KDChart::DatasetBrushRole, and KDChart::AbstractDiagram::datasetDimension().

Referenced by KDChart::Legend::setBrushesFromDiagram().

```

1019 {
1020     QList<QBrush> ret;
1021     if( model() == 0 )
1022         return ret;
1023
1024     const int columnCount = attributesModel()->columnCount(attributesModelRootIndex());
1025     for( int i = 0; i < columnCount / datasetDimension(); ++i )

```



```

1026         ret << qVariantValue<QBrush>( attributesModel()->headerData( i, Qt::Vertical, DatasetBrushRol
1027
1028     return ret;
1029 }

```

### 9.10.3.18 int AbstractDiagram::datasetDimension () const [inherited]

The dataset dimension of a diagram determines, how many value dimensions it expects each datapoint to have.

For each dimension it will expect one column of values in the model. If the dimensionality is 1, automatic values will be used for the abscissa.

For example a diagram with the default dimension of 1, will have one column per datapoint (the y values) and will use automatic values for the x axis (1, 2, 3, ... n). If the dimension is 2, the diagram will use the first, (and the third, fifth, etc) columns as X values, and the second, (and the fourth, sixth, etc) column as Y values.

#### Returns:

The dataset dimension of the diagram.

Definition at line 1072 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::brush(), KDChart::AbstractDiagram::compare(), KDChart::AbstractDiagram::datasetBrushes(), KDChart::AbstractDiagram::datasetLabels(), KDChart::AbstractDiagram::datasetMarkers(), KDChart::AbstractDiagram::datasetPens(), KDChart::LineDiagram::getCellValues(), KDChart::CartesianCoordinatePlane::getDataDimensionsList(), KDChart::TernaryPointDiagram::paint(), KDChart::TernaryLineDiagram::paint(), KDChart::AbstractDiagram::paintDataValueTexts(), KDChart::AbstractDiagram::paintMarkers(), KDChart::AbstractDiagram::pen(), KDChart::AbstractDiagram::setPen(), KDChart::Plotter::setType(), and KDChart::LineDiagram::setType().

```

1073 {
1074     return d->datasetDimension;
1075 }

```

### 9.10.3.19 QStringList AbstractDiagram::datasetLabels () const [inherited]

The set of dataset labels currently displayed, for use in legends, etc.

#### Returns:

The set of dataset labels currently displayed.

Definition at line 1005 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::attributesModelRootIndex(), KDChart::AttributesModel::columnCount(), KDChart::AbstractDiagram::datasetDimension(), and KDChart::AttributesModel::headerData().

```

1006 {
1007     QStringList ret;

```

```

1008     if( model() == 0 )
1009         return ret;
1010
1011     const int columnCount = attributesModel()->columnCount(attributesModelRootIndex());
1012     for( int i = 0; i < columnCount / datasetDimension(); ++i )
1013         ret << attributesModel()->headerData( i, Qt::Horizontal, Qt::DisplayRole ).toString();
1014
1015     return ret;
1016 }

```

### 9.10.3.20 `QList< MarkerAttributes > AbstractDiagram::datasetMarkers () const` [inherited]

The set of dataset markers currently used, for use in legends, etc.

#### Note:

Cell-level override markers, if set, take precedence over the dataset values, so you might need to check these too, in order to find the marker, that is shown for a single cell.

#### Returns:

The current set of dataset brushes.

Definition at line 1044 of file `KDChartAbstractDiagram.cpp`.

References `KDChart::AbstractDiagram::attributesModel()`, `KDChart::AbstractDiagram::attributesModelRootIndex()`, `KDChart::AttributesModel::columnCount()`, `KDChart::AbstractDiagram::datasetDimension()`, and `KDChart::DataValueLabelAttributesRole`.

```

1045 {
1046     QList<MarkerAttributes> ret;
1047     if( model() == 0 )
1048         return ret;
1049
1050     const int columnCount = attributesModel()->columnCount(attributesModelRootIndex());
1051     for( int i = 0; i < columnCount / datasetDimension(); ++i )
1052     {
1053         const DataValueAttributes a =
1054             qVariantValue<DataValueAttributes>( attributesModel()->headerData( i, Qt::Vertical, DataValueLabelAttributesRole ) );
1055         ret << a.markerAttributes();
1056     }
1057     return ret;
1058 }

```

### 9.10.3.21 `QList< QPen > AbstractDiagram::datasetPens () const` [inherited]

The set of dataset pens currently used, for use in legends, etc.

#### Note:

Cell-level override pens, if set, take precedence over the dataset values, so you might need to check these too, in order to find the pens, that is used for a single cell.

#### Returns:

The current set of dataset pens.

Definition at line 1031 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::attributesModelRootIndex(), KDChart::AttributesModel::columnCount(), KDChart::AbstractDiagram::datasetDimension(), and KDChart::DatasetPenRole.

```

1032 {
1033     QList<QPen> ret;
1034     if( model() == 0 )
1035         return ret;
1036
1037     const int columnCount = attributesModel()->columnCount(attributesModelRootIndex());
1038     for( int i = 0; i < columnCount / datasetDimension(); ++i )
1039         ret << qVariantValue<QPen>( attributesModel()->headerData( i, Qt::Vertical, DatasetPenRole ) );
1040
1041     return ret;
1042 }
```

### 9.10.3.22 [DataValueAttributes](#) AbstractDiagram::dataValueAttributes (const QModelIndex & index) const [inherited]

Retrieve the [DataValueAttributes](#) for the given index.

This will fall back automatically to what was set at dataset or model level, if there are no datapoint specific settings.

#### Parameters:

*index* The datapoint to retrieve the attributes for.

#### Returns:

The [DataValueAttributes](#) for the given index.

Definition at line 421 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AttributesModel::data(), and KDChart::DataValueLabelAttributesRole.

```

422 {
423     return qVariantValue<DataValueAttributes>(
424         attributesModel()->data( attributesModel()->mapFromSource(index),
425                                 KDChart::DataValueLabelAttributesRole ) );
426 }
```

### 9.10.3.23 [DataValueAttributes](#) AbstractDiagram::dataValueAttributes (int column) const [inherited]

Retrieve the [DataValueAttributes](#) for the given dataset.

This will fall back automatically to what was set at model level, if there are no dataset specific settings.

#### Parameters:

*column* The dataset to retrieve the attributes for.

#### Returns:

The [DataValueAttributes](#) for the given dataset.

Definition at line 414 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::columnToIndex(), KDChart::AttributesModel::data(), and KDChart::DataValueLabelAttributesRole.

```

415 {
416     return qVariantValue<DataValueAttributes>(
417         attributesModel()->data( attributesModel()->mapFromSource(columnToIndex( column )),
418         KDChart::DataValueLabelAttributesRole ) );
419 }
```

#### 9.10.3.24 [DataValueAttributes](#) AbstractDiagram::dataValueAttributes () const [inherited]

Retrieve the [DataValueAttributes](#) specified globally.

This will fall back automatically to the default settings, if there are no specific settings.

##### Returns:

The global [DataValueAttributes](#).

Definition at line 408 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DataValueLabelAttributesRole, and KDChart::AttributesModel::modelData().

Referenced by KDChart::AbstractDiagram::paintDataValueText(), KDChart::AbstractDiagram::paintMarker(), and KDChart::TernaryLineDiagram::TernaryLineDiagram().

```

409 {
410     return qVariantValue<DataValueAttributes>(
411         attributesModel()->modelData( KDChart::DataValueLabelAttributesRole ) );
412 }
```

#### 9.10.3.25 void AbstractDiagram::doItemsLayout () [virtual, inherited]

[reimplemented]

Definition at line 321 of file KDChartAbstractDiagram.cpp.

References d, and KDChart::AbstractDiagram::update().

```

322 {
323     if ( d->plane ) {
324         d->plane->layoutDiagrams();
325         update();
326     }
327     QAbstractItemView::doItemsLayout();
328 }
```

#### 9.10.3.26 int AbstractDiagram::horizontalOffset () const [virtual, inherited]

[reimplemented]

Definition at line 950 of file KDChartAbstractDiagram.cpp.

```

951 { return 0; }
```

### 9.10.3.27 QModelIndex AbstractDiagram::indexAt (const QPoint & *point*) const [virtual, inherited]

[reimplemented]

Definition at line 1098 of file KDChartAbstractDiagram.cpp.

References d.

```
1099 {
1100     return d->indexAt( point );
1101 }
```

### 9.10.3.28 QModelIndexList AbstractDiagram::indexesAt (const QPoint & *point*) const [inherited]

This method is added alongside with indexAt from QAIM, since in kdchart multiple indexes can be displayed at the same spot.

Definition at line 1103 of file KDChartAbstractDiagram.cpp.

References d.

```
1104 {
1105     return d->indexesAt( point );
1106 }
```

### 9.10.3.29 bool AbstractDiagram::isHidden (const QModelIndex & *index*) const [inherited]

Retrieve the hidden status for the given index.

This will fall back automatically to what was set at dataset or diagram level, if there are no datapoint specific settings.

#### Parameters:

***index*** The datapoint to retrieve the hidden status for.

#### Returns:

The hidden status for the given index.

Definition at line 380 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AttributesModel::data(), and KDChart::DataHiddenRole.

```
381 {
382     return qVariantValue<bool>(
383         attributesModel()->data(
384             attributesModel()->mapFromSource(index),
385             DataHiddenRole ) );
386 }
```

### 9.10.3.30 **bool AbstractDiagram::isHidden (int *column*) const** [inherited]

Retrieve the hidden status for the given dataset.

This will fall back automatically to what was set at diagram level, if there are no dataset specific settings.

#### Parameters:

*column* The dataset to retrieve the hidden status for.

#### Returns:

The hidden status for the given dataset.

Definition at line 373 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::columnToIndex(), KDChart::AttributesModel::data(), and KDChart::DataHiddenRole.

```

374 {
375     return qVariantValue<bool>(
376         attributesModel()->data(
377             attributesModel()->mapFromSource(columnToIndex( column )),
378             DataHiddenRole ) );
379 }
```

### 9.10.3.31 **bool AbstractDiagram::isHidden () const** [inherited]

Retrieve the hidden status specified globally.

This will fall back automatically to the default settings ( = not hidden), if there are no specific settings.

#### Returns:

The global hidden status.

Definition at line 367 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DataHiddenRole, and KDChart::AttributesModel::modelData().

Referenced by KDChart::LineDiagram::valueForCellTesting().

```

368 {
369     return qVariantValue<bool>(
370         attributesModel()->modelData( DataHiddenRole ) );
371 }
```

### 9.10.3.32 **bool AbstractDiagram::isIndexHidden (const QModelIndex & *index*) const** [virtual, inherited]

[reimplemented]

Definition at line 956 of file KDChartAbstractDiagram.cpp.

```

957 { return true; }
```

**9.10.3.33 QStringList AbstractDiagram::itemRowLabels () const** [inherited]

The set of item row labels currently displayed, for use in Abscissa axes, etc.

**Returns:**

The set of item row labels currently displayed.

Definition at line 989 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::attributesModelRootIndex(), KDChart::AttributesModel::rowCount(), KDChart::AbstractDiagram::unitPrefix(), and KDChart::AbstractDiagram::unitSuffix().

```

990 {
991     QStringList ret;
992     if( model() ){
993         //qDebug() << "AbstractDiagram::itemRowLabels(): " << attributesModel()->rowCount(attributesModelRootIndex());
994         const int rowCount = attributesModel()->rowCount(attributesModelRootIndex());
995         for( int i = 0; i < rowCount; ++i ){
996             //qDebug() << "item row label: " << attributesModel()->headerData( i, Qt::Vertical, Qt::DisplayRole ).toString();
997             ret << unitPrefix( i, Qt::Horizontal, true ) +
998                 attributesModel()->headerData( i, Qt::Vertical, Qt::DisplayRole ).toString() +
999                 unitSuffix( i, Qt::Horizontal, true );
1000         }
1001     }
1002     return ret;
1003 }
```

**9.10.3.34 void KDChart::AbstractDiagram::layoutChanged (AbstractDiagram \*)** [signal, inherited]

Diagrams are supposed to emit this signal, when the layout of one of their element changes.

Layouts can change, for example, when axes are added or removed, or when the configuration was changed in a way that the axes or the diagram itself are displayed in a different geometry. Changes in the diagrams coordinate system also result in the [layoutChanged\(\)](#) signal being emitted.

Referenced by KDChart::AbstractDiagram::setDatasetDimension(), KDChart::AbstractPieDiagram::setPieAttributes(), KDChart::AbstractPieDiagram::setThreeDPieAttributes(), KDChart::Plotter::setType(), KDChart::LineDiagram::setType(), and KDChart::BarDiagram::setType().

**9.10.3.35 void KDChart::AbstractDiagram::modelsChanged ()** [signal, inherited]

This signal is emitted, when either the model or the [AttributesModel](#) is replaced.

Referenced by KDChart::AbstractDiagram::setAttributesModel(), and KDChart::AbstractDiagram::setModel().

**9.10.3.36 QModelIndex AbstractDiagram::moveCursor (CursorAction cursorAction, Qt::KeyboardModifiers modifiers)** [virtual, inherited]

[reimplemented]

Definition at line 947 of file KDChartAbstractDiagram.cpp.

```

948 { return QModelIndex(); }
```

**9.10.3.37** `virtual double KDChart::AbstractPolarDiagram::numberOfGridRings () const` [pure virtual]

Implemented in [KDChart::PieDiagram](#), [KDChart::PolarDiagram](#), and [KDChart::RingDiagram](#).

**9.10.3.38** `virtual double KDChart::AbstractPolarDiagram::numberOfValuesPerDataset () const` [pure virtual]

Implemented in [KDChart::PieDiagram](#), [KDChart::PolarDiagram](#), and [KDChart::RingDiagram](#).

Referenced by `columnCount()`.

**9.10.3.39** `virtual void KDChart::AbstractDiagram::paint (PaintContext * paintContext)` [pure virtual, inherited]

Draw the diagram contents to the rectangle and painter, that are passed in as part of the paint context.

#### Parameters:

*paintContext* All information needed for painting.

Implemented in [KDChart::BarDiagram](#), [KDChart::LineDiagram](#), [KDChart::PieDiagram](#), [KDChart::Plotter](#), [KDChart::PolarDiagram](#), [KDChart::RingDiagram](#), [KDChart::TernaryLineDiagram](#), and [KDChart::TernaryPointDiagram](#).

**9.10.3.40** `void AbstractDiagram::paintDataValueText (QPainter * painter, const QModelIndex & index, const QPointF & pos, double value)` [inherited]

Definition at line 468 of file `KDChartAbstractDiagram.cpp`.

References `d`, `KDChart::DataValueAttributes::dataLabel()`, `KDChart::AbstractDiagram::dataValueAttributes()`, `KDChart::DataValueAttributes::decimalDigits()`, `KDChart::DataValueAttributes::isVisible()`, `KDChartEnums::MeasureOrientationMinimum`, `KDChart::DataValueAttributes::position()`, `KDChart::DataValueAttributes::prefix()`, `KDChart::DataValueAttributes::suffix()`, and `KDChart::DataValueAttributes::textAttributes()`.

Referenced by `KDChart::RingDiagram::paint()`, `KDChart::PolarDiagram::paint()`, and `KDChart::AbstractDiagram::paintDataValueTexts()`.

```

472 {
473     // paint one data series
474     const DataValueAttributes a( dataValueAttributes(index) );
475     if ( !a.isVisible() ) return;
476
477     // handle decimal digits
478     int decimalDigits = a.decimalDigits();
479     int decimalPos = QString::number( value ).indexOf( QLatin1Char( '.' ) );
480     QString roundedValue;
481     if ( a.dataLabel().isNull() ) {
482         if ( decimalPos > 0 && value != 0 )
483             roundedValue = roundValues ( value, decimalPos, decimalDigits );
484         else
485             roundedValue = QString::number( value );
486     } else
487         roundedValue = a.dataLabel();
488     // handle prefix and suffix
489     if ( !a.prefix().isNull() )

```



```

490         roundedValue.prepend( a.prefix() );
491
492     if ( !a.suffix().isNull() )
493         roundedValue.append( a.suffix() );
494
495     const TextAttributes ta( a.textAttributes() );
496     // FIXME draw the non-text bits, background, etc
497     if ( ta.isVisible() ) {
498
499         QPointF pt( pos );
500         /* for debugging:
501         PainterSaver painterSaver( painter );
502         painter->setPen( Qt::black );
503         painter->drawLine( pos - QPointF( 1,1), pos + QPointF( 1,1) );
504         painter->drawLine( pos - QPointF(-1,1), pos + QPointF(-1,1) );
505         */
506
507         QTextDocument doc;
508         if( Qt::mightBeRichText( roundedValue ) )
509             doc.setHtml( roundedValue );
510         else
511             doc.setPlainText( roundedValue );
512
513         const RelativePosition relPos( a.position( value >= 0.0 ) );
514         const Qt::Alignment alignBottomLeft = Qt::AlignBottom | Qt::AlignLeft;
515         const QFont calculatedFont( ta.calculatedFont( d->plane, KDChartEnums::MeasureOrientationMinimum ) );
516         const QRectF boundRect( d->cachedFontMetrics( calculatedFont, painter->device() )->boundingRect( roundedValue, calculatedFont ) );
517
518         // To place correctly
519         pt.ry() -= boundRect.height();
520
521         // qDebug() << "calculatedFont's point size:" << calculatedFont.pointSizeF();
522         // adjust the text start point position, if alignment is not Bottom/Left
523         if( (relPos.alignment() & alignBottomLeft) != alignBottomLeft ) {
524             if( relPos.alignment() & Qt::AlignRight )
525                 pt.rx() -= boundRect.width();
526             else if( relPos.alignment() & Qt::AlignHCenter )
527                 pt.rx() -= 0.5 * boundRect.width();
528
529             if( relPos.alignment() & Qt::AlignTop )
530                 pt.ry() += boundRect.height();
531             else if( relPos.alignment() & Qt::AlignVCenter )
532                 pt.ry() += 0.5 * boundRect.height();
533         }
534
535         // FIXME draw the non-text bits, background, etc
536
537         if ( a.showRepetitiveDataLabels() ||
538             pos.x() <= d->lastX ||
539             d->lastRoundedValue != roundedValue ) {
540             d->lastRoundedValue = roundedValue;
541             d->lastX = pos.x();
542             PainterSaver painterSaver( painter );
543
544             doc.setDefaultFont( calculatedFont );
545             QAbstractTextDocumentLayout::PaintContext context;
546             context.palette = palette();
547             context.palette.setColor( QPalette::Text, ta.pen().color() );
548
549             painter->translate( pt );
550             painter->rotate( ta.rotation() );
551
552             QAbstractTextDocumentLayout* layout = doc.documentLayout();
553             layout->draw( painter, context );
554         }
555     }
556 }

```

### 9.10.3.41 void AbstractDiagram::paintDataValueTexts (QPainter \* *painter*) [protected, virtual, inherited]

Definition at line 584 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::checkInvariants(), KDChart::AbstractDiagram::coordinatePlane(), KDChart::AbstractDiagram::datasetDimension(), KDChart::AbstractDiagram::paintDataValueText(), and KDChart::AbstractCoordinatePlane::translate().

```

585 {
586     if ( !checkInvariants() ) return;
587     const int rowCount = model()->rowCount(rootIndex());
588     const int columnCount = model()->columnCount(rootIndex());
589     for ( int i=datasetDimension()-1; i<columnCount; i += datasetDimension() ) {
590         for ( int j=0; j<rowCount; ++j ) {
591             const QModelIndex index = model()->index( j, i, rootIndex() );
592             double value = model()->data( index ).toDouble();
593             const QPointF pos = coordinatePlane()->translate( QPointF( j, value ) );
594             paintDataValueText( painter, index, pos, value );
595         }
596     }
597 }
```

### 9.10.3.42 void AbstractDiagram::paintMarker (QPainter \* *painter*, const QModelIndex & *index*, const QPointF & *pos*) [inherited]

Definition at line 600 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::brush(), KDChart::AbstractDiagram::checkInvariants(), d, KDChart::AbstractDiagram::dataValueAttributes(), KDChart::MarkerAttributes::isVisible(), KDChart::DataValueAttributes::isVisible(), KDChart::DataValueAttributes::markerAttributes(), KDChart::MarkerAttributes::markerColor(), KDChart::MarkerAttributes::markerSize(), KDChart::AbstractDiagram::paintMarker(), and KDChart::MarkerAttributes::pen().

```

603 {
604     if ( !checkInvariants() ) return;
605     DataValueAttributes a = dataValueAttributes(index);
606     if ( !a.isVisible() ) return;
607     const MarkerAttributes &ma = a.markerAttributes();
608     if ( !ma.isVisible() ) return;
609
610     PainterSaver painterSaver( painter );
611     QSizeF maSize( ma.markerSize() );
612     QBrush indexBrush( brush( index ) );
613     QPen indexPen( ma.pen() );
614     if ( ma.markerColor().isValid() )
615         indexBrush.setColor( ma.markerColor() );
616
617     paintMarker( painter, ma, indexBrush, indexPen, pos, maSize );
618
619     // workaround: BC cannot be changed, otherwise we would pass the
620     // index down to next-lower paintMarker function. So far, we
621     // basically save a circle of radius maSize at pos in the
622     // reverseMapper. This means that ^^^ this version of paintMarker
623     // needs to be called to reverse-map the marker.
624     d->reverseMapper.addCircle( index.row(), index.column(), pos, 2 * maSize );
625 }
```

### 9.10.3.43 void AbstractDiagram::paintMarker (QPainter \* *painter*, const MarkerAttributes & *markerAttributes*, const QBrush & *brush*, const QPen & *pen*, const QPointF & *point*, const QSizeF & *size*) [virtual, inherited]

Definition at line 627 of file KDChartAbstractDiagram.cpp.

References KDChart::MarkerAttributes::Marker1Pixel, KDChart::MarkerAttributes::Marker4Pixels, KDChart::MarkerAttributes::MarkerCircle, KDChart::MarkerAttributes::MarkerCross, KDChart::MarkerAttributes::MarkerDiamond, KDChart::MarkerAttributes::MarkerFastCross, KDChart::MarkerAttributes::MarkerRing, KDChart::MarkerAttributes::MarkerSquare, and KDChart::MarkerAttributes::markerStyle().

Referenced by KDChart::TernaryPointDiagram::paint(), KDChart::TernaryLineDiagram::paint(), KDChart::MarkerLayoutItem::paintIntoRect(), KDChart::AbstractDiagram::paintMarker(), and KDChart::AbstractDiagram::paintMarkers().

```

633 {
634     const QPen oldPen( painter->pen() );
635     // Pen is used to paint 4Pixels - 1 Pixel - Ring and FastCross types.
636     // make sure to use the brush color - see above in those cases.
637     const bool isFourPixels = (markerAttributes.markerStyle() == MarkerAttributes::Marker4Pixels);
638     if( isFourPixels || (markerAttributes.markerStyle() == MarkerAttributes::Marker1Pixel) ){
639         // for high-performance point charts with tiny point markers:
640         painter->setPen( QPen( brush.color().light() ) );
641         if( isFourPixels ){
642             const qreal x = pos.x();
643             const qreal y = pos.y();
644             painter->drawLine( QPointF(x-1.0,y-1.0),
645                             QPointF(x+1.0,y-1.0) );
646             painter->drawLine( QPointF(x-1.0,y),
647                             QPointF(x+1.0,y) );
648             painter->drawLine( QPointF(x-1.0,y+1.0),
649                             QPointF(x+1.0,y+1.0) );
650         }
651         painter->drawPoint( pos );
652     }else{
653         PainterSaver painterSaver( painter );
654         // we only a solid line surrounding the markers
655         QPen painterPen( pen );
656         painterPen.setStyle( Qt::SolidLine );
657         painter->setPen( painterPen );
658         painter->setBrush( brush );
659         painter->setRenderHint ( QPainter::Antialiasing );
660         painter->translate( pos );
661         switch ( markerAttributes.markerStyle() ) {
662             case MarkerAttributes::MarkerCircle:
663                 painter->drawEllipse( QRectF( 0 - maSize.height()/2, 0 - maSize.width()/2,
664                                             maSize.height(), maSize.width() ) );
665                 break;
666             case MarkerAttributes::MarkerSquare:
667                 {
668                     QRectF rect( 0 - maSize.width()/2, 0 - maSize.height()/2,
669                               maSize.width(), maSize.height() );
670                     painter->drawRect( rect );
671                     break;
672                 }
673             case MarkerAttributes::MarkerDiamond:
674                 {
675                     QVector <QPointF > diamondPoints;
676                     QPointF top, left, bottom, right;
677                     top = QPointF( 0, 0 - maSize.height()/2 );
678                     left = QPointF( 0 - maSize.width()/2, 0 );
679                     bottom = QPointF( 0, maSize.height()/2 );
680                     right = QPointF( maSize.width()/2, 0 );
681                     diamondPoints << top << left << bottom << right;

```

```

682         painter->drawPolygon( diamondPoints );
683         break;
684     }
685     // both handled on top of the method:
686     case MarkerAttributes::Marker1Pixel:
687     case MarkerAttributes::Marker4Pixels:
688         break;
689     case MarkerAttributes::MarkerRing:
690     {
691         painter->setPen( QPen( brush.color() ) );
692         painter->setBrush( Qt::NoBrush );
693         painter->drawEllipse( QRectF( 0 - maSize.height()/2, 0 - maSize.width()/2,
694                                     maSize.height(), maSize.width() ) );
695         break;
696     }
697     case MarkerAttributes::MarkerCross:
698     {
699         QRectF rect( maSize.width()*-0.5, maSize.height()*-0.2,
700                     maSize.width(), maSize.height()*0.4 );
701         painter->drawRect( rect );
702         rect.setTopLeft( QPointF( maSize.width()*-0.2, maSize.height()*-0.5 ) );
703         rect.setSize( QSizeF( maSize.width()*0.4, maSize.height() ) );
704         painter->drawRect( rect );
705         break;
706     }
707     case MarkerAttributes::MarkerFastCross:
708     {
709         QPointF left, right, top, bottom;
710         left = QPointF( -maSize.width()/2, 0 );
711         right = QPointF( maSize.width()/2, 0 );
712         top = QPointF( 0, -maSize.height()/2 );
713         bottom = QPointF( 0, maSize.height()/2 );
714         painter->setPen( QPen( brush.color() ) );
715         painter->drawLine( left, right );
716         painter->drawLine( top, bottom );
717         break;
718     }
719     default:
720         Q_ASSERT_X ( false, "paintMarkers()",
721                     "Type item does not match a defined Marker Type." );
722 }
723 }
724 painter->setPen( oldPen );
725 }

```

### 9.10.3.44 void AbstractDiagram::paintMarkers (QPainter \* *painter*) [protected, virtual, inherited]

Definition at line 727 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::checkInvariants(), KDChart::AbstractDiagram::coordinatePlane(), KDChart::AbstractDiagram::datasetDimension(), KDChart::AbstractDiagram::paintMarker(), and KDChart::AbstractCoordinatePlane::translate().

```

728 {
729     if ( !checkInvariants() ) return;
730     const int rowCount = model()->rowCount( rootIndex() );
731     const int columnCount = model()->columnCount( rootIndex() );
732     for ( int i=datasetDimension()-1; i<columnCount; i += datasetDimension() ) {
733         for ( int j=0; j<rowCount; ++j ) {
734             const QModelIndex index = model()->index( j, i, rootIndex() );
735             double value = model()->data( index ).toDouble();
736             const QPointF pos = coordinatePlane()->translate( QPointF( j, value ) );
737             paintMarker( painter, index, pos );

```

```

738     }
739 }
740 }
```

#### 9.10.3.45 QPen AbstractDiagram::pen (const QModelIndex & *index*) const [inherited]

Retrieve the pen to be used, for painting the datapoint at the given index in the model.

##### Parameters:

*index* The index of the datapoint in the model.

##### Returns:

The pen to use for painting.

Definition at line 788 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AttributesModel::data(), KDChart::AbstractDiagram::datasetDimension(), KDChart::DatasetPenRole, and KDChart::AbstractDiagram::pen().

```

789 {
790     if( datasetDimension() > 1 )
791         return pen( index.column() );
792     return qVariantValue<QPen>(
793         attributesModel()->data(
794             attributesModel()->mapFromSource( index ),
795             DatasetPenRole ) );
796 }
```

#### 9.10.3.46 QPen AbstractDiagram::pen (int *dataset*) const [inherited]

Retrieve the pen to be used for the given dataset.

This will fall back automatically to what was set at model level, if there are no dataset specific settings.

##### Parameters:

*dataset* The dataset to retrieve the pen for.

##### Returns:

The pen to use for painting.

Definition at line 777 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::columnToIndex(), KDChart::AttributesModel::data(), KDChart::AbstractDiagram::datasetDimension(), and KDChart::DatasetPenRole.

```

778 {
779     if( datasetDimension() > 1 )
780         dataset /= datasetDimension();
781
782     return qVariantValue<QPen>(
783         attributesModel()->data(
784             attributesModel()->mapFromSource( columnToIndex( dataset ) ),
785             DatasetPenRole ) );
786 }
```

**9.10.3.47 QPen AbstractDiagram::pen () const** [inherited]

Retrieve the pen to be used for painting datapoints globally.

This will fall back automatically to the default settings, if there are no specific settings.

**Returns:**

The pen to use for painting.

Definition at line 771 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AttributesModel::data(), and KDChart::DatasetPenRole.

Referenced by KDChart::TernaryPointDiagram::paint(), KDChart::TernaryLineDiagram::paint(), and KDChart::AbstractDiagram::pen().

```

772 {
773     return qVariantValue<QPen>(
774         attributesModel()->data( DatasetPenRole ) );
775 }
```

**9.10.3.48 bool AbstractDiagram::percentMode () const** [inherited]

Definition at line 462 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::compare(), and KDChart::CartesianCoordinatePlane::getDataDimensionsList().

```

463 {
464     return d->percent;
465 }
```

**9.10.3.49 const [PolarCoordinatePlane](#) \* AbstractPolarDiagram::polarCoordinatePlane () const**

Definition at line 55 of file KDChartAbstractPolarDiagram.cpp.

References KDChart::AbstractDiagram::coordinatePlane().

Referenced by KDChart::PieDiagram::paint().

```

56 {
57     return dynamic_cast<const PolarCoordinatePlane*>( coordinatePlane() );
58 }
```

**9.10.3.50 void KDChart::AbstractDiagram::propertiesChanged ()** [signal, inherited]

Emitted upon change of a property of the Diagram.

Referenced by KDChart::Plotter::resetLineAttributes(), KDChart::LineDiagram::resetLineAttributes(), KDChart::AbstractDiagram::setAllowOverlappingDataValueTexts(), KDChart::AbstractDiagram::setAntiAliasing(), KDChart::BarDiagram::setBarAttributes(), KDChart::AbstractDiagram::setBrush(),

KDChart::AbstractDiagram::setDataValueAttributes(), KDChart::Plotter::setLineAttributes(), KDChart::LineDiagram::setLineAttributes(), KDChart::AbstractDiagram::setPen(), KDChart::AbstractDiagram::setPercentMode(), KDChart::BarDiagram::setThreeDBarAttributes(), KDChart::Plotter::setThreeDLineAttributes(), KDChart::LineDiagram::setThreeDLineAttributes(), KDChart::Plotter::setType(), KDChart::LineDiagram::setType(), KDChart::BarDiagram::setType(), KDChart::Plotter::setValueTrackerAttributes(), and KDChart::LineDiagram::setValueTrackerAttributes().

### 9.10.3.51 virtual void KDChart::AbstractDiagram::resize (const QSizeF & *area*) [pure virtual, inherited]

Called by the widget's sizeEvent.

Adjust all internal structures, that are calculated, depending on the size of the widget.

#### Parameters:

*area*

Implemented in [KDChart::BarDiagram](#), [KDChart::LineDiagram](#), [KDChart::PieDiagram](#), [KDChart::Plotter](#), [KDChart::PolarDiagram](#), [KDChart::RingDiagram](#), [KDChart::TernaryLineDiagram](#), and [KDChart::TernaryPointDiagram](#).

Referenced by KDChart::CartesianCoordinatePlane::setGeometry().

### 9.10.3.52 void AbstractDiagram::scrollTo (const QModelIndex & *index*, ScrollHint *hint* = EnsureVisible) [virtual, inherited]

[reimplemented]

Definition at line 942 of file KDChartAbstractDiagram.cpp.

```
943 {}
```

### 9.10.3.53 void AbstractDiagram::setAllowOverlappingDataValueTexts (bool *allow*) [inherited]

Set whether data value labels are allowed to overlap.

#### Parameters:

*allow* True means that overlapping labels are allowed.

Definition at line 434 of file KDChartAbstractDiagram.cpp.

References [d](#), and KDChart::AbstractDiagram::propertiesChanged().

```
435 {
436     d->allowOverlappingDataValueTexts = allow;
437     emit propertiesChanged();
438 }
```

### 9.10.3.54 void AbstractDiagram::setAntiAliasing (bool *enabled*) [inherited]

Set whether anti-aliasing is to be used while rendering this diagram.

#### Parameters:

*enabled* True means that AA is enabled.

Definition at line 445 of file KDChartAbstractDiagram.cpp.

References `d`, and `KDChart::AbstractDiagram::propertiesChanged()`.

```
446 {
447     d->antiAliasing = enabled;
448     emit propertiesChanged();
449 }
```

### 9.10.3.55 void AbstractDiagram::setAttributesModel ([AttributesModel](#) \* *model*) [virtual, inherited]

Associate an [AttributesModel](#) with this diagram.

Note that the diagram does `_not_` take ownership of the [AttributesModel](#). This should thus only be used with [AttributesModels](#) that have been explicitly created by the user, and are owned by her. Setting an [AttributesModel](#) that is internal to another diagram is an error.

Correct:

```
AttributesModel *am = new AttributesModel( model, 0 );
diagram1->setAttributesModel( am );
diagram2->setAttributesModel( am );
```

Wrong:

```
diagram1->setAttributesModel( diagram2->attributesModel() );
```

#### Parameters:

*model* The [AttributesModel](#) to use for this diagram.

See also:

[AttributesModel](#), [usesExternalAttributesModel](#)

Reimplemented in [KDChart::AbstractCartesianDiagram](#).

Definition at line 255 of file KDChartAbstractDiagram.cpp.

References `d`, `KDChart::AbstractDiagram::modelsChanged()`, and `KDChart::AbstractDiagram::setDataBoundariesDirty()`.

Referenced by `KDChart::AbstractCartesianDiagram::setAttributesModel()`.

```
256 {
257     if( amodel->sourceModel() != model() ) {
258         qWarning("KDChart::AbstractDiagram::setAttributesModel() failed: "
259                 "Trying to set an attributesmodel which works on a different "
260                 "model than the diagram.");
261     }
```



```

261         return;
262     }
263     if( qobject_cast<PrivateAttributesModel*>(amodel) ) {
264         qWarning("KDChart::AbstractDiagram::setAttributesModel() failed: "
265             "Trying to set an attributesmodel that is private to another diagram.");
266         return;
267     }
268     d->setAttributesModel(amodel);
269     scheduleDelayedItemsLayout();
270     setDataBoundariesDirty();
271     emit modelsChanged();
272 }

```

### 9.10.3.56 void AbstractDiagram::setAttributesModelRootIndex (const QModelIndex &) [protected, inherited]

Definition at line 293 of file KDChartAbstractDiagram.cpp.

References `d`, and `KDChart::AbstractDiagram::setDataBoundariesDirty()`.

Referenced by `KDChart::AbstractDiagram::setRootIndex()`.

```

294 {
295     d->attributesModelRootIndex=idx;
296     setDataBoundariesDirty();
297     scheduleDelayedItemsLayout();
298 }

```

### 9.10.3.57 void AbstractDiagram::setBrush (const QBrush & *brush*) [inherited]

Set the brush to be used, for painting all datasets in the model.

#### Parameters:

***brush*** The brush to use.

Definition at line 806 of file KDChartAbstractDiagram.cpp.

References `KDChart::AbstractDiagram::attributesModel()`, `KDChart::DatasetBrushRole`, `KDChart::AbstractDiagram::propertiesChanged()`, and `KDChart::AttributesModel::setModelData()`.

```

807 {
808     attributesModel()->setModelData(
809         qVariantFromValue( brush ), DatasetBrushRole );
810     emit propertiesChanged();
811 }

```

### 9.10.3.58 void AbstractDiagram::setBrush (int *dataset*, const QBrush & *brush*) [inherited]

Set the brush to be used, for painting the given dataset.

#### Parameters:

***dataset*** The dataset's column in the model.

***brush*** The brush to use.

Definition at line 813 of file KDChartAbstractDiagram.cpp.

References [KDChart::AbstractDiagram::attributesModel\(\)](#), [KDChart::DatasetBrushRole](#), [KDChart::AbstractDiagram::propertiesChanged\(\)](#), and [KDChart::AttributesModel::setHeaderData\(\)](#).

```
814 {
815     attributesModel()->setHeaderData(
816         column, Qt::Vertical,
817         qVariantFromValue( brush ),
818         DatasetBrushRole );
819     emit propertiesChanged();
820 }
```

### 9.10.3.59 void AbstractDiagram::setBrush (const QModelIndex & *index*, const QBrush & *brush*) [inherited]

Set the brush to be used, for painting the datapoint at the given index.

#### Parameters:

***index*** The datapoint's index in the model.

***brush*** The brush to use.

Definition at line 798 of file KDChartAbstractDiagram.cpp.

References [KDChart::AbstractDiagram::attributesModel\(\)](#), [KDChart::DatasetBrushRole](#), [KDChart::AbstractDiagram::propertiesChanged\(\)](#), and [KDChart::AttributesModel::setData\(\)](#).

```
799 {
800     attributesModel()->setData(
801         attributesModel()->mapFromSource( index ),
802         qVariantFromValue( brush ), DatasetBrushRole );
803     emit propertiesChanged();
804 }
```

### 9.10.3.60 void AbstractDiagram::setCoordinatePlane ([AbstractCoordinatePlane](#) \* *plane*) [virtual, inherited]

Set the coordinate plane associated with the diagram.

This determines how coordinates in value space are mapped into pixel space. The chart takes ownership.

#### Returns:

The coordinate plane associated with the diagram.

Reimplemented in [KDChart::AbstractCartesianDiagram](#).

Definition at line 316 of file KDChartAbstractDiagram.cpp.

References [d](#).

Referenced by [KDChart::AbstractCoordinatePlane::addDiagram\(\)](#), [KDChart::AbstractCartesianDiagram::setCoordinatePlane\(\)](#), and [KDChart::AbstractCoordinatePlane::takeDiagram\(\)](#).

```
317 {
318     d->plane = parent;
319 }
```

### 9.10.3.61 void AbstractDiagram::setDataBoundariesDirty () const [protected, inherited]

Definition at line 234 of file KDChartAbstractDiagram.cpp.

References `d`.

Referenced by `KDChart::AbstractDiagram::dataChanged()`, `KDChart::Plotter::resize()`, `KDChart::LineDiagram::resize()`, `KDChart::BarDiagram::resize()`, `KDChart::AbstractDiagram::setAttributesModel()`, `KDChart::AbstractDiagram::setAttributesModelRootIndex()`, `KDChart::AbstractDiagram::setDatasetDimension()`, `KDChart::AbstractDiagram::setModel()`, `KDChart::BarDiagram::setThreeDBarAttributes()`, `KDChart::Plotter::setThreeDLineAttributes()`, `KDChart::LineDiagram::setThreeDLineAttributes()`, `KDChart::Plotter::setType()`, `KDChart::LineDiagram::setType()`, and `KDChart::BarDiagram::setType()`.

```
235 {
236     d->ataboundariesDirty = true;
237 }
```

### 9.10.3.62 void AbstractDiagram::setDatasetDimension (int *dimension*) [inherited]

Sets the dataset dimension of the diagram.

See also:

[datasetDimension](#).

**Parameters:**

*dimension*

Definition at line 1077 of file KDChartAbstractDiagram.cpp.

References `d`, `KDChart::AbstractDiagram::layoutChanged()`, and `KDChart::AbstractDiagram::setDataBoundariesDirty()`.

Referenced by `KDChart::Widget::setType()`, `KDChart::TernaryLineDiagram::TernaryLineDiagram()`, and `KDChart::TernaryPointDiagram::TernaryPointDiagram()`.

```
1078 {
1079     if ( d->datasetDimension == dimension ) return;
1080     d->datasetDimension = dimension;
1081     setDataBoundariesDirty();
1082     emit layoutChanged( this );
1083 }
```

### 9.10.3.63 void AbstractDiagram::setDataValueAttributes (const [DataValueAttributes](#) & *a*) [inherited]

Set the [DataValueAttributes](#) for all datapoints in the model.

**Parameters:**

*a* The attributes to set.

Definition at line 428 of file KDChartAbstractDiagram.cpp.

References `d`, `KDChart::DataValueLabelAttributesRole`, and `KDChart::AbstractDiagram::propertiesChanged()`.

```

429 {
430     d->attributesModel->setModelData( qVariantFromValue( a ), DataValueLabelAttributesRole );
431     emit propertiesChanged();
432 }

```

#### 9.10.3.64 void AbstractDiagram::setDataValueAttributes (int *dataset*, const [DataValueAttributes](#) & *a*) [inherited]

Set the [DataValueAttributes](#) for the given dataset.

##### Parameters:

*dataset* The dataset to set the attributes for.

*a* The attributes to set.

Definition at line 400 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::DataValueLabelAttributesRole](#), and [KDChart::AbstractDiagram::propertiesChanged\(\)](#).

```

401 {
402     d->attributesModel->setHeaderData(
403         column, Qt::Vertical,
404         qVariantFromValue( a ), DataValueLabelAttributesRole );
405     emit propertiesChanged();
406 }

```

#### 9.10.3.65 void AbstractDiagram::setDataValueAttributes (const QModelIndex & *index*, const [DataValueAttributes](#) & *a*) [inherited]

Set the [DataValueAttributes](#) for the given index.

##### Parameters:

*index* The datapoint to set the attributes for.

*a* The attributes to set.

Definition at line 389 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::DataValueLabelAttributesRole](#), and [KDChart::AbstractDiagram::propertiesChanged\(\)](#).

```

391 {
392     d->attributesModel->setData(
393         d->attributesModel->mapFromSource( index ),
394         qVariantFromValue( a ),
395         DataValueLabelAttributesRole );
396     emit propertiesChanged();
397 }

```

**9.10.3.66 void AbstractDiagram::setHidden (bool *hidden*)** [inherited]

Hide (or unhide, resp.

) all datapoints in the model.

**Note:**

Hidden data are still taken into account by the coordinate plane, so neither the grid nor your axes' ranges will change, when you hide data. For totally removing data from KD Chart's view you can use another approach: e.g. you could define a proxy model on top of your data model, and register the proxy model calling [setModel\(\)](#) instead of registering your real data model.

**Parameters:**

***hidden*** The hidden status to set.

Definition at line 359 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::AbstractDiagram::dataHidden\(\)](#), and [KDChart::DataHiddenRole](#).

```
360 {  
361     d->attributesModel->setModelData(  
362         qVariantFromValue( hidden ),  
363         DataHiddenRole );  
364     emit dataHidden();  
365 }
```

**9.10.3.67 void AbstractDiagram::setHidden (int *column*, bool *hidden*)** [inherited]

Hide (or unhide, resp.

) a dataset.

**Note:**

Hidden data are still taken into account by the coordinate plane, so neither the grid nor your axes' ranges will change, when you hide data. For totally removing data from KD Chart's view you can use another approach: e.g. you could define a proxy model on top of your data model, and register the proxy model calling [setModel\(\)](#) instead of registering your real data model.

**Parameters:**

***column*** The dataset to set the hidden status for.

***hidden*** The hidden status to set.

Definition at line 350 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::AbstractDiagram::dataHidden\(\)](#), and [KDChart::DataHiddenRole](#).

```
351 {  
352     d->attributesModel->setHeaderData(  
353         column, Qt::Vertical,  
354         qVariantFromValue( hidden ),  
355         DataHiddenRole );  
356     emit dataHidden();  
357 }
```

### 9.10.3.68 void AbstractDiagram::setHidden (const QModelIndex & *index*, bool *hidden*) [inherited]

Hide (or unhide, resp.

) a data cell.

#### Note:

Hidden data are still taken into account by the coordinate plane, so neither the grid nor your axes' ranges will change, when you hide data. For totally removing data from KD Chart's view you can use another approach: e.g. you could define a proxy model on top of your data model, and register the proxy model calling [setModel\(\)](#) instead of registering your real data model.

#### Parameters:

***index*** The datapoint to set the hidden status for.

***hidden*** The hidden status to set.

Definition at line 341 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::AbstractDiagram::dataHidden\(\)](#), and [KDChart::DataHiddenRole](#).

```
342 {
343     d->attributesModel->setData(
344         d->attributesModel->mapFromSource( index ),
345         qVariantFromValue( hidden ),
346         DataHiddenRole );
347     emit dataHidden();
348 }
```

### 9.10.3.69 void AbstractDiagram::setModel (QAbstractItemModel \* *model*) [virtual, inherited]

Associate a model with the diagram.

Reimplemented in [KDChart::AbstractCartesianDiagram](#).

Definition at line 239 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::AttributesModel::initFrom\(\)](#), [KDChart::AbstractDiagram::modelsChanged\(\)](#), and [KDChart::AbstractDiagram::setDataBoundariesDirty\(\)](#).

Referenced by [KDChart::AbstractCartesianDiagram::setModel\(\)](#), and [KDChart::Widget::setType\(\)](#).

```
240 {
241     QAbstractItemView::setModel( newModel );
242     AttributesModel* amodel = new PrivateAttributesModel( newModel, this );
243     amodel->initFrom( d->attributesModel );
244     d->setAttributesModel( amodel );
245     scheduleDelayedItemsLayout();
246     setDataBoundariesDirty();
247     emit modelsChanged();
248 }
```

### 9.10.3.70 void AbstractDiagram::setPen (const QPen & *pen*) [inherited]

Set the pen to be used, for painting all datasets in the model.

**Parameters:**

*pen* The pen to use.

Definition at line 753 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DatasetPenRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setModelData().

```
754 {
755     attributesModel()->setModelData(
756         qVariantFromValue( pen ), DatasetPenRole );
757     emit propertiesChanged();
758 }
```

**9.10.3.71 void AbstractDiagram::setPen (int *dataset*, const QPen & *pen*)** [inherited]

Set the pen to be used, for painting the given dataset.

**Parameters:**

*dataset* The dataset's row in the model.

*pen* The pen to use.

Definition at line 760 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::datasetDimension(), KDChart::DatasetPenRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setHeaderData().

```
761 {
762     if( datasetDimension() > 1 )
763         column *= datasetDimension();
764     attributesModel()->setHeaderData(
765         column, Qt::Vertical,
766         qVariantFromValue( pen ),
767         DatasetPenRole );
768     emit propertiesChanged();
769 }
```

**9.10.3.72 void AbstractDiagram::setPen (const QModelIndex & *index*, const QPen & *pen*)** [inherited]

Set the pen to be used, for painting the datapoint at the given index.

**Parameters:**

*index* The datapoint's index in the model.

*pen* The pen to use.

Definition at line 743 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::datasetDimension(), KDChart::DatasetPenRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setData().

```

744 {
745     if( datasetDimension() > 1 )
746         return setPen( index.column(), pen );
747     attributesModel()->setData(
748         attributesModel()->mapFromSource( index ),
749         qVariantFromValue( pen ), DatasetPenRole );
750     emit propertiesChanged();
751 }

```

### 9.10.3.73 void AbstractDiagram::setPercentMode (bool *percent*) [inherited]

Definition at line 456 of file KDChartAbstractDiagram.cpp.

References [d](#), and [KDChart::AbstractDiagram::propertiesChanged\(\)](#).

Referenced by [KDChart::LineDiagram::setType\(\)](#), and [KDChart::BarDiagram::setType\(\)](#).

```

457 {
458     d->percent = percent;
459     emit propertiesChanged();
460 }

```

### 9.10.3.74 void AbstractDiagram::setRootIndex (const QModelIndex & *idx*) [virtual, inherited]

Set the root index in the model, where the diagram starts referencing data for display.

[reimplemented]

Reimplemented in [KDChart::AbstractCartesianDiagram](#).

Definition at line 286 of file KDChartAbstractDiagram.cpp.

References [d](#), and [KDChart::AbstractDiagram::setAttributesModelRootIndex\(\)](#).

Referenced by [KDChart::AbstractCartesianDiagram::setRootIndex\(\)](#).

```

287 {
288     QAbstractItemView::setRootIndex(idx);
289     setAttributesModelRootIndex( d->attributesModel->mapFromSource(idx) );
290 }

```

### 9.10.3.75 void AbstractDiagram::setSelection (const QRect & *rect*, QItemSelectionModel::SelectionFlags *command*) [virtual, inherited]

[reimplemented]

Definition at line 959 of file KDChartAbstractDiagram.cpp.

References [d](#).

```

960 {
961     const QModelIndexList indexes = d->indexesIn( rect );
962     QItemSelection selection;
963     KDAB_FOREACH( const QModelIndex& index, indexes )
964     {
965         selection.append( QItemSelectionRange( index ) );
966     }
967     selectionModel()->select( selection, command );
968 }

```



**9.10.3.76 void AbstractDiagram::setUnitPrefix (const QString & *prefix*, Qt::Orientation *orientation*)** [inherited]

Sets the unit prefix for all values.

**Parameters:**

*prefix* the prefix to be set

*orientation* the orientation of the axis to set

Definition at line 864 of file KDChartAbstractDiagram.cpp.

References d.

```
865 {  
866     d->unitPrefix[ orientation ] = prefix;  
867 }
```

**9.10.3.77 void AbstractDiagram::setUnitPrefix (const QString & *prefix*, int *column*, Qt::Orientation *orientation*)** [inherited]

Sets the unit prefix for one value.

**Parameters:**

*prefix* the prefix to be set

*column* the value using that prefix

*orientation* the orientation of the axis to set

Definition at line 854 of file KDChartAbstractDiagram.cpp.

References d.

```
855 {  
856     d->unitPrefixMap[ column ][ orientation ] = prefix;  
857 }
```

**9.10.3.78 void AbstractDiagram::setUnitSuffix (const QString & *suffix*, Qt::Orientation *orientation*)** [inherited]

Sets the unit suffix for all values.

**Parameters:**

*suffix* the suffix to be set

*orientation* the orientation of the axis to set

Definition at line 885 of file KDChartAbstractDiagram.cpp.

References d.

```
886 {  
887     d->unitSuffix[ orientation ] = suffix;  
888 }
```

### 9.10.3.79 void AbstractDiagram::setUnitSuffix (const QString & *suffix*, int *column*, Qt::Orientation *orientation*) [inherited]

Sets the unit suffix for one value.

#### Parameters:

*suffix* the suffix to be set  
*column* the value using that suffix  
*orientation* the orientation of the axis to set

Definition at line 875 of file KDChartAbstractDiagram.cpp.

References d.

```
876 {
877     d->unitSuffixMap[ column ][ orientation ]= suffix;
878 }
```

### 9.10.3.80 QString AbstractDiagram::unitPrefix (Qt::Orientation *orientation*) const [inherited]

Returns the global unit prefix.

#### Parameters:

*orientation* the orientation of the axis

#### Returns:

the unit prefix

Definition at line 908 of file KDChartAbstractDiagram.cpp.

References d.

```
909 {
910     return d->unitPrefix[ orientation ];
911 }
```

### 9.10.3.81 QString AbstractDiagram::unitPrefix (int *column*, Qt::Orientation *orientation*, bool *fallback* = false) const [inherited]

Returns the unit prefix for a special value.

#### Parameters:

*column* the value which's prefix is requested  
*orientation* the orientation of the axis  
*fallback* if true, the global prefix is return when no specific one is set for that value

#### Returns:

the unit prefix

Definition at line 897 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::itemRowLabels(), and KDChart::CartesianAxis::paintCtx().

```

898 {
899     if( !fallback || d->unitPrefixMap[ column ].contains( orientation ) )
900         return d->unitPrefixMap[ column ][ orientation ];
901     return d->unitPrefix[ orientation ];
902 }
```

### 9.10.3.82 QString AbstractDiagram::unitSuffix (Qt::Orientation *orientation*) const [inherited]

Returns the global unit suffix.

#### Parameters:

*orientation* the orientation of the axis

#### Returns:

the unit suffix

Definition at line 931 of file KDChartAbstractDiagram.cpp.

References d.

```

932 {
933     return d->unitSuffix[ orientation ];
934 }
```

### 9.10.3.83 QString AbstractDiagram::unitSuffix (int *column*, Qt::Orientation *orientation*, bool *fallback* = false) const [inherited]

Returns the unit suffix for a special value.

#### Parameters:

*column* the value which's suffix is requested

*orientation* the orientation of the axis

*fallback* if true, the global suffix is return when no specific one is set for that value

#### Returns:

the unit suffix

Definition at line 920 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::itemRowLabels(), and KDChart::CartesianAxis::paintCtx().

```

921 {
922     if( !fallback || d->unitSuffixMap[ column ].contains( orientation ) )
923         return d->unitSuffixMap[ column ][ orientation ];
924     return d->unitSuffix[ orientation ];
925 }
```

**9.10.3.84 void AbstractDiagram::update () const** [inherited]

Definition at line 1091 of file KDChartAbstractDiagram.cpp.

References [d](#).

Referenced by [KDChart::AbstractDiagram::doItemsLayout\(\)](#).

```
1092 {  
1093     //qDebug("KDChart::AbstractDiagram::update() called");  
1094     if( d->plane )  
1095         d->plane->update();  
1096 }
```

**9.10.3.85 void KDChart::AbstractDiagram::useDefaultColors ()** [inherited]

Set the palette to be used, for painting datasets to the default palette.

**See also:**

[KDChart::Palette](#). FIXME: fold into one usePalette ([KDChart::Palette&](#)) method

Definition at line 974 of file KDChartAbstractDiagram.cpp.

References [d](#), and [KDChart::AttributesModel::PaletteTypeDefault](#).

```
975 {  
976     d->attributesModel->setPaletteType( AttributesModel::PaletteTypeDefault );  
977 }
```

**9.10.3.86 void KDChart::AbstractDiagram::useRainbowColors ()** [inherited]

Set the palette to be used, for painting datasets to the rainbow palette.

**See also:**

[KDChart::Palette](#).

Definition at line 984 of file KDChartAbstractDiagram.cpp.

References [d](#), and [KDChart::AttributesModel::PaletteTypeRainbow](#).

```
985 {  
986     d->attributesModel->setPaletteType( AttributesModel::PaletteTypeRainbow );  
987 }
```

**9.10.3.87 bool AbstractDiagram::usesExternalAttributesModel () const** [virtual, inherited]

Returns whether the diagram is using its own built-in attributes model or an attributes model that was set via [setAttributesModel](#).

**See also:**

[setAttributesModel](#)

Definition at line 274 of file KDChartAbstractDiagram.cpp.

References [d](#).

```
275 {  
276     return d->usesExternalAttributesModel();  
277 }
```

### 9.10.3.88 void KDChart::AbstractDiagram::useSubduedColors () [inherited]

Set the palette to be used, for painting datasets to the subdued palette.

**See also:**

[KDChart::Palette](#).

Definition at line 979 of file KDChartAbstractDiagram.cpp.

References [d](#), and [KDChart::AttributesModel::PaletteTypeSubdued](#).

```
980 {  
981     d->attributesModel->setPaletteType( AttributesModel::PaletteTypeSubdued );  
982 }
```

### 9.10.3.89 double AbstractDiagram::valueForCell (int row, int column) const [protected, inherited]

Helper method, retrieving the data value (DisplayRole) for a given row and column.

**Parameters:**

*row* The row to query.

*column* The column to query.

**Returns:**

The value of the display role at the given row and column as a double.

**Deprecated**

Definition at line 1085 of file KDChartAbstractDiagram.cpp.

References [KDChart::AbstractDiagram::attributesModelRootIndex\(\)](#), and [d](#).

```
1086 {  
1087     return d->attributesModel->data(  
1088         d->attributesModel->index( row, column, attributesModelRootIndex() ) ).toDouble();  
1089 }
```

**9.10.3.90** `virtual double KDChart::AbstractPolarDiagram::valueTotals () const` [pure virtual]

Implemented in [KDChart::PieDiagram](#), [KDChart::PolarDiagram](#), and [KDChart::RingDiagram](#).

Referenced by [KDChart::PolarCoordinatePlane::layoutDiagrams\(\)](#).

**9.10.3.91** `int AbstractDiagram::verticalOffset () const` [virtual, inherited]

[reimplemented]

Definition at line 953 of file [KDChartAbstractDiagram.cpp](#).

```
954 { return 0; }
```

**9.10.3.92** `QRect AbstractDiagram::visualRect (const QModelIndex & index) const` [virtual, inherited]

[reimplemented]

Definition at line 937 of file [KDChartAbstractDiagram.cpp](#).

```
938 {  
939     return QRect();  
940 }
```

**9.10.3.93** `QRegion AbstractDiagram::visualRegionForSelection (const QItemSelection & selection) const` [virtual, inherited]

[reimplemented]

Definition at line 970 of file [KDChartAbstractDiagram.cpp](#).

```
971 { return QRegion(); }
```

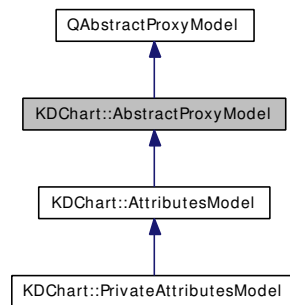
The documentation for this class was generated from the following files:

- [KDChartAbstractPolarDiagram.h](#)
- [KDChartAbstractPolarDiagram.cpp](#)

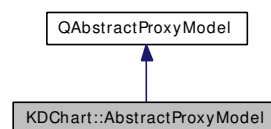
## 9.11 KChart::AbstractProxyModel Class Reference

```
#include <KChartAbstractProxyModel.h>
```

Inheritance diagram for KChart::AbstractProxyModel:



Collaboration diagram for KChart::AbstractProxyModel:



### 9.11.1 Detailed Description

Base class for all proxy models used inside KD Chart.

Definition at line 43 of file KChartAbstractProxyModel.h.

### Public Member Functions

- [AbstractProxyModel](#) ([QObject](#) \*parent=0)  
*This is basically KDAbstractProxyModel, but only the bits that we really need from it.*
- QModelIndex [index](#) (int row, int col, const QModelIndex &index) const
- QModelIndex [mapFromSource](#) (const QModelIndex &sourceIndex) const
- QModelIndex [mapToSource](#) (const QModelIndex &proxyIndex) const
- QModelIndex [parent](#) (const QModelIndex &index) const

### 9.11.2 Constructor & Destructor Documentation

#### 9.11.2.1 KChart::AbstractProxyModel::AbstractProxyModel ([QObject](#) \* parent = 0) [explicit]

This is basically KDAbstractProxyModel, but only the bits that we really need from it.

Definition at line 41 of file KChartAbstractProxyModel.cpp.

```
42 : QAbstractProxyModel(parent) {}
```

### 9.11.3 Member Function Documentation

#### 9.11.3.1 QModelIndex KDChart::AbstractProxyModel::index (int *row*, int *col*, const QModelIndex & *index*) const

[reimplemented]

Definition at line 84 of file KDChartAbstractProxyModel.cpp.

References mapFromSource(), and mapToSource().

Referenced by KDChart::AttributesModel::setHeaderData(), and KDChart::AttributesModel::setModelData().

```
85 {
86     Q_ASSERT(sourceModel());
87     return mapFromSource(sourceModel()->index( row, col, mapToSource(index) ));
88 }
```

#### 9.11.3.2 QModelIndex KDChart::AbstractProxyModel::mapFromSource (const QModelIndex & *sourceIndex*) const

[reimplemented]

Definition at line 52 of file KDChartAbstractProxyModel.cpp.

Referenced by index(), and parent().

```
53 {
54     if ( !sourceIndex.isValid() )
55         return QModelIndex();
56     //qDebug() << "sourceIndex.model()="<<sourceIndex.model();
57     //qDebug() << "model()="<<sourceModel();
58     Q_ASSERT( sourceIndex.model() == sourceModel() );
59
60     // Create an index that preserves the internal pointer from the source;
61     // this way AbstractProxyModel preserves the structure of the source model
62     return createIndex( sourceIndex.row(), sourceIndex.column(), sourceIndex.internalPointer() );
63 }
```

#### 9.11.3.3 QModelIndex KDChart::AbstractProxyModel::mapToSource (const QModelIndex & *proxyIndex*) const

[reimplemented]

Definition at line 65 of file KDChartAbstractProxyModel.cpp.

Referenced by KDChart::AttributesModel::columnCount(), KDChart::AttributesModel::data(), index(), parent(), KDChart::AttributesModel::rowCount(), and KDChart::AttributesModel::setData().

```
66 {
67     if ( !proxyIndex.isValid() )
68         return QModelIndex();
69     if( proxyIndex.model() != this )
70         qDebug() << proxyIndex.model() << this;
71     Q_ASSERT( proxyIndex.model() == this );
72     // So here we need to create a source index which holds that internal pointer.
73     // No way to pass it to sourceModel()->index... so we have to do the ugly way:
```



```
74 QModelIndex sourceIndex;  
75 KDPrivateModelIndex* hack = reinterpret_cast<KDPrivateModelIndex*>(&sourceIndex);  
76 hack->r = proxyIndex.row();  
77 hack->c = proxyIndex.column();  
78 hack->p = proxyIndex.internalPointer();  
79 hack->m = sourceModel();  
80 Q_ASSERT( sourceIndex.isValid() );  
81 return sourceIndex;  
82 }
```

#### 9.11.3.4 QModelIndex KDChart::AbstractProxyModel::parent (const QModelIndex & *index*) const

[reimplemented]

Definition at line 90 of file KDChartAbstractProxyModel.cpp.

References `mapFromSource()`, and `mapToSource()`.

```
91 {  
92     Q_ASSERT( sourceModel() );  
93     return mapFromSource( sourceModel()->parent( mapToSource( index ) ) );  
94 }
```

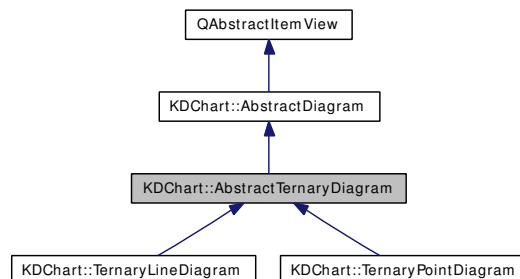
The documentation for this class was generated from the following files:

- [KDChartAbstractProxyModel.h](#)
- [KDChartAbstractProxyModel.cpp](#)

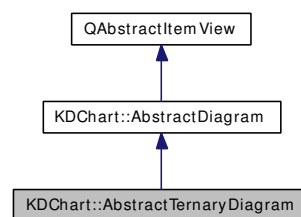
## 9.12 KDChart::AbstractTernaryDiagram Class Reference

```
#include <KDChartAbstractTernaryDiagram.h>
```

Inheritance diagram for KDChart::AbstractTernaryDiagram:



Collaboration diagram for KDChart::AbstractTernaryDiagram:



### 9.12.1 Detailed Description

Base class for diagrams based on a ternary coordinate plane.

Definition at line 44 of file KDChartAbstractTernaryDiagram.h.

### Signals

- void [dataHidden](#) ()  
*This signal is emitted, when the hidden status of at least one data cell was (un)set.*
- void [layoutChanged](#) ([AbstractDiagram](#) \*)  
*Diagrams are supposed to emit this signal, when the layout of one of their element changes.*
- void [modelsChanged](#) ()  
*This signal is emitted, when either the model or the [AttributesModel](#) is replaced.*
- void [propertiesChanged](#) ()  
*Emitted upon change of a property of the Diagram.*

## Public Member Functions

- bool [allowOverlappingDataValueTexts](#) () const  
*Returns:*  
*Whether data value labels are allowed to overlap.*
- bool [antiAliasing](#) () const  
*Returns:*  
*Whether anti-aliasing is to be used for rendering this diagram.*
- virtual [AttributesModel](#) \* [attributesModel](#) () const  
*Returns the [AttributesModel](#), that is used by this diagram.*
- [QBrush](#) [brush](#) (const [QModelIndex](#) &index) const  
*Retrieve the brush to be used, for painting the datapoint at the given index in the model.*
- [QBrush](#) [brush](#) (int dataset) const  
*Retrieve the brush to be used for the given dataset.*
- [QBrush](#) [brush](#) () const  
*Retrieve the brush to be used for painting datapoints globally.*
- bool [compare](#) (const [AbstractDiagram](#) \*other) const  
*Returns true if both diagrams have the same settings.*
- [AbstractCoordinatePlane](#) \* [coordinatePlane](#) () const  
*The coordinate plane associated with the diagram.*
- const [QPair](#)< [QPointF](#), [QPointF](#) > [dataBoundaries](#) () const  
*Return the bottom left and top right data point, that the diagram will display (unless the grid adjusts these values).*
- virtual void [dataChanged](#) (const [QModelIndex](#) &topLeft, const [QModelIndex](#) &bottomRight)  
*[reimplemented]*
- [QList](#)< [QBrush](#) > [datasetBrushes](#) () const  
*The set of dataset brushes currently used, for use in legends, etc.*
- int [datasetDimension](#) () const  
*The dataset dimension of a diagram determines, how many value dimensions it expects each datapoint to have.*
- [QStringList](#) [datasetLabels](#) () const  
*The set of dataset labels currently displayed, for use in legends, etc.*
- [QList](#)< [MarkerAttributes](#) > [datasetMarkers](#) () const  
*The set of dataset markers currently used, for use in legends, etc.*
- [QList](#)< [QPen](#) > [datasetPens](#) () const  
*The set of dataset pens currently used, for use in legends, etc.*

- [DataValueAttributes dataValueAttributes](#) (const QModelIndex &index) const  
*Retrieve the [DataValueAttributes](#) for the given index.*
- [DataValueAttributes dataValueAttributes](#) (int column) const  
*Retrieve the [DataValueAttributes](#) for the given dataset.*
- [DataValueAttributes dataValueAttributes](#) () const  
*Retrieve the [DataValueAttributes](#) specified globally.*
- virtual void [doItemsLayout](#) ()  
*[reimplemented]*
- virtual int [horizontalOffset](#) () const  
*[reimplemented]*
- virtual QModelIndex [indexAt](#) (const QPoint &point) const  
*[reimplemented]*
- QModelIndexList [indexesAt](#) (const QPoint &point) const  
*This method is added alongside with [indexAt](#) from QAIM, since in kdchart multiple indexes can be displayed at the same spot.*
- bool [isHidden](#) (const QModelIndex &index) const  
*Retrieve the hidden status for the given index.*
- bool [isHidden](#) (int column) const  
*Retrieve the hidden status for the given dataset.*
- bool [isHidden](#) () const  
*Retrieve the hidden status specified globally.*
- virtual bool [isIndexHidden](#) (const QModelIndex &index) const  
*[reimplemented]*
- QStringList [itemRowLabels](#) () const  
*The set of item row labels currently displayed, for use in Abscissa axes, etc.*
- virtual QModelIndex [moveCursor](#) (CursorAction cursorAction, Qt::KeyboardModifiers modifiers)  
*[reimplemented]*
- void [paintDataValueText](#) (QPainter \*painter, const QModelIndex &index, const QPointF &pos, double value)
- void [paintMarker](#) (QPainter \*painter, const QModelIndex &index, const QPointF &pos)
- virtual void [paintMarker](#) (QPainter \*painter, const [MarkerAttributes](#) &markerAttributes, const QBrush &brush, const QPen &, const QPointF &point, const QSizeF &size)
- QPen [pen](#) (const QModelIndex &index) const  
*Retrieve the pen to be used, for painting the datapoint at the given index in the model.*
- QPen [pen](#) (int dataset) const

*Retrieve the pen to be used for the given dataset.*

- QPen [pen](#) () const

*Retrieve the pen to be used for painting datapoints globally.*

- bool [percentMode](#) () const
- virtual void [scrollTo](#) (const QModelIndex &index, ScrollHint hint=EnsureVisible)  
*[reimplemented]*

- void [setAllowOverlappingDataValueTexts](#) (bool allow)

*Set whether data value labels are allowed to overlap.*

- void [setAntiAliasing](#) (bool enabled)

*Set whether anti-aliasing is to be used while rendering this diagram.*

- virtual void [setAttributesModel](#) ([AttributesModel](#) \*model)

*Associate an [AttributesModel](#) with this diagram.*

- void [setBrush](#) (const QBrush &brush)

*Set the brush to be used, for painting all datasets in the model.*

- void [setBrush](#) (int dataset, const QBrush &brush)

*Set the brush to be used, for painting the given dataset.*

- void [setBrush](#) (const QModelIndex &index, const QBrush &brush)

*Set the brush to be used, for painting the datapoint at the given index.*

- virtual void [setCoordinatePlane](#) ([AbstractCoordinatePlane](#) \*plane)

*Set the coordinate plane associated with the diagram.*

- void [setDatasetDimension](#) (int dimension)

*Sets the dataset dimension of the diagram.*

- void [setDataValueAttributes](#) (const [DataValueAttributes](#) &a)

*Set the [DataValueAttributes](#) for all datapoints in the model.*

- void [setDataValueAttributes](#) (int dataset, const [DataValueAttributes](#) &a)

*Set the [DataValueAttributes](#) for the given dataset.*

- void [setDataValueAttributes](#) (const QModelIndex &index, const [DataValueAttributes](#) &a)

*Set the [DataValueAttributes](#) for the given index.*

- void [setHidden](#) (bool hidden)

*Hide (or unhide, resp.)*

- void [setHidden](#) (int column, bool hidden)

*Hide (or unhide, resp.)*

- void [setHidden](#) (const QModelIndex &index, bool hidden)

*Hide (or unhide, resp.)*

- virtual void [setModel](#) (QAbstractItemModel \*model)  
*Associate a model with the diagram.*
- void [setPen](#) (const QPen &pen)  
*Set the pen to be used, for painting all datasets in the model.*
- void [setPen](#) (int dataset, const QPen &pen)  
*Set the pen to be used, for painting the given dataset.*
- void [setPen](#) (const QModelIndex &index, const QPen &pen)  
*Set the pen to be used, for painting the datapoint at the given index.*
- void [setPercentMode](#) (bool percent)
- virtual void [setRootIndex](#) (const QModelIndex &idx)  
*Set the root index in the model, where the diagram starts referencing data for display.*
- virtual void [setSelection](#) (const QRect &rect, QItemSelectionModel::SelectionFlags command)  
*[reimplemented]*
- void [setUnitPrefix](#) (const QString &prefix, Qt::Orientation orientation)  
*Sets the unit prefix for all values.*
- void [setUnitPrefix](#) (const QString &prefix, int column, Qt::Orientation orientation)  
*Sets the unit prefix for one value.*
- void [setUnitSuffix](#) (const QString &suffix, Qt::Orientation orientation)  
*Sets the unit suffix for all values.*
- void [setUnitSuffix](#) (const QString &suffix, int column, Qt::Orientation orientation)  
*Sets the unit suffix for one value.*
- QString [unitPrefix](#) (Qt::Orientation orientation) const  
*Returns the global unit prefix.*
- QString [unitPrefix](#) (int column, Qt::Orientation orientation, bool fallback=false) const  
*Returns the unit prefix for a special value.*
- QString [unitSuffix](#) (Qt::Orientation orientation) const  
*Returns the global unit suffix.*
- QString [unitSuffix](#) (int column, Qt::Orientation orientation, bool fallback=false) const  
*Returns the unit suffix for a special value.*
- void [update](#) () const
- void [useDefaultColors](#) ()  
*Set the palette to be used, for painting datasets to the default palette.*
- void [useRainbowColors](#) ()  
*Set the palette to be used, for painting datasets to the rainbow palette.*

- virtual bool [usesExternalAttributesModel](#) () const  
*Returns whether the diagram is using its own built-in attributes model or an attributes model that was set via `setAttributesModel`.*
- void [useSubduedColors](#) ()  
*Set the palette to be used, for painting datasets to the subdued palette.*
- virtual int [verticalOffset](#) () const  
*[reimplemented]*
- virtual QRect [visualRect](#) (const QModelIndex &index) const  
*[reimplemented]*
- virtual QRegion [visualRegionForSelection](#) (const QItemSelection &selection) const  
*[reimplemented]*

## Protected Member Functions

- QModelIndex [attributesModelRootIndex](#) () const
- virtual const QPair< QPointF, QPointF > [calculateDataBoundaries](#) () const=0
- virtual bool [checkInvariants](#) (bool justReturnTheStatus=false) const
- QModelIndex [columnToIndex](#) (int column) const
- virtual void [paintDataValueTexts](#) (QPainter \*painter)
- virtual void [paintMarkers](#) (QPainter \*painter)
- void [setAttributesModelRootIndex](#) (const QModelIndex &)
- void [setDataBoundariesDirty](#) () const
- double [valueForCell](#) (int row, int column) const  
*Helper method, retrieving the data value (`DisplayRole`) for a given row and column.*

## 9.12.2 Member Function Documentation

### 9.12.2.1 bool AbstractDiagram::allowOverlappingDataValueTexts () const [inherited]

#### Returns:

Whether data value labels are allowed to overlap.

Definition at line 440 of file `KDChartAbstractDiagram.cpp`.

References `d`.

Referenced by `KDChart::AbstractDiagram::compare()`.

```

441 {
442     return d->allowOverlappingDataValueTexts;
443 }
```

### 9.12.2.2 `bool AbstractDiagram::antiAliasing () const` [inherited]

#### Returns:

Whether anti-aliasing is to be used for rendering this diagram.

Definition at line 451 of file `KDChartAbstractDiagram.cpp`.

References d.

Referenced by `KDChart::AbstractDiagram::compare()`.

```
452 {
453     return d->antiAliasing;
454 }
```

### 9.12.2.3 `AttributesModel * AbstractDiagram::attributesModel () const` [virtual, inherited]

Returns the [AttributesModel](#), that is used by this diagram.

By default each diagram owns its own [AttributesModel](#), which should never be deleted. Only if a user-supplied [AttributesModel](#) has been set does the pointer returned here not belong to the diagram.

#### Returns:

The [AttributesModel](#) associated with the diagram.

#### See also:

[setAttributesModel](#)

Definition at line 280 of file `KDChartAbstractDiagram.cpp`.

References d.

Referenced by `KDChart::AbstractDiagram::brush()`, `KDChart::AbstractDiagram::compare()`, `KDChart::AbstractDiagram::datasetBrushes()`, `KDChart::AbstractDiagram::datasetLabels()`, `KDChart::AbstractDiagram::datasetMarkers()`, `KDChart::AbstractDiagram::datasetPens()`, `KDChart::AbstractDiagram::dataValueAttributes()`, `KDChart::AbstractDiagram::isHidden()`, `KDChart::AbstractDiagram::itemRowLabels()`, `KDChart::RingDiagram::paint()`, `KDChart::PolarDiagram::paint()`, `KDChart::AbstractDiagram::pen()`, `KDChart::AbstractCartesianDiagram::setAttributesModel()`, `KDChart::BarDiagram::setBarAttributes()`, `KDChart::AbstractDiagram::setBrush()`, `KDChart::AbstractCartesianDiagram::setModel()`, `KDChart::AbstractDiagram::setPen()`, and `KDChart::TernaryLineDiagram::TernaryLineDiagram()`.

```
281 {
282     return d->attributesModel;
283 }
```

### 9.12.2.4 `QModelIndex AbstractDiagram::attributesModelRootIndex () const` [protected, inherited]

returns a `QModelIndex` pointing into the [AttributesModel](#) that corresponds to the root index of the diagram.

Definition at line 302 of file `KDChartAbstractDiagram.cpp`.



References d.

Referenced by KDChart::AbstractDiagram::datasetBrushes(), KDChart::AbstractDiagram::datasetLabels(), KDChart::AbstractDiagram::datasetMarkers(), KDChart::AbstractDiagram::datasetPens(), KDChart::AbstractDiagram::itemRowLabels(), KDChart::Plotter::numberOfAbscissaSegments(), KDChart::LineDiagram::numberOfAbscissaSegments(), KDChart::BarDiagram::numberOfAbscissaSegments(), KDChart::Plotter::numberOfOrdinateSegments(), KDChart::LineDiagram::numberOfOrdinateSegments(), KDChart::BarDiagram::numberOfOrdinateSegments(), KDChart::AbstractDiagram::valueForCell(), and KDChart::LineDiagram::valueForCellTesting().

```

303 {
304     if ( !d->attributesModelRootIndex.isValid() )
305         d->attributesModelRootIndex = d->attributesModel->mapFromSource( rootIndex() );
306     return d->attributesModelRootIndex;
307 }
```

#### 9.12.2.5 QBrush AbstractDiagram::brush (const QModelIndex & *index*) const [inherited]

Retrieve the brush to be used, for painting the datapoint at the given index in the model.

##### Parameters:

*index* The index of the datapoint in the model.

##### Returns:

The brush to use for painting.

Definition at line 838 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::brush(), KDChart::AttributesModel::data(), KDChart::DatasetBrushRole, and KDChart::AbstractDiagram::datasetDimension().

```

839 {
840     if( datasetDimension() > 1 )
841         return brush( index.column() );
842     return QVariantValue<QBrush>(
843         attributesModel()->data(
844             attributesModel()->mapFromSource( index ),
845             DatasetBrushRole ) );
846 }
```

#### 9.12.2.6 QBrush AbstractDiagram::brush (int *dataset*) const [inherited]

Retrieve the brush to be used for the given dataset.

This will fall back automatically to what was set at model level, if there are no dataset specific settings.

##### Parameters:

*dataset* The dataset to retrieve the brush for.

##### Returns:

The brush to use for painting.

Definition at line 828 of file KDChartAbstractDiagram.cpp.

References [KDChart::AbstractDiagram::attributesModel\(\)](#), [KDChart::AbstractDiagram::columnToIndex\(\)](#), [KDChart::AttributesModel::data\(\)](#), [KDChart::DatasetBrushRole](#), and [KDChart::AbstractDiagram::datasetDimension\(\)](#).

```

829 {
830     if( datasetDimension() > 1 )
831         dataset /= datasetDimension();
832     return qVariantValue<QBrush>(
833         attributesModel()->data(
834             attributesModel()->mapFromSource( columnToIndex( dataset ) ),
835             DatasetBrushRole ) );
836 }
```

### 9.12.2.7 **QBrush AbstractDiagram::brush () const** [inherited]

Retrieve the brush to be used for painting datapoints globally.

This will fall back automatically to the default settings, if there are no specific settings.

#### Returns:

The brush to use for painting.

Definition at line 822 of file KDChartAbstractDiagram.cpp.

References [KDChart::AbstractDiagram::attributesModel\(\)](#), [KDChart::AttributesModel::data\(\)](#), and [KDChart::DatasetBrushRole](#).

Referenced by [KDChart::AbstractDiagram::brush\(\)](#), [KDChart::TernaryPointDiagram::paint\(\)](#), [KDChart::TernaryLineDiagram::paint\(\)](#), [KDChart::RingDiagram::paint\(\)](#), [KDChart::PolarDiagram::paint\(\)](#), and [KDChart::AbstractDiagram::paintMarker\(\)](#).

```

823 {
824     return qVariantValue<QBrush>(
825         attributesModel()->data( DatasetBrushRole ) );
826 }
```

### 9.12.2.8 **virtual const QPair< QPointF, QPointF > KDChart::AbstractTernaryDiagram::calculateDataBoundaries () const** [protected, pure virtual]

Implements [KDChart::AbstractDiagram](#).

Implemented in [KDChart::TernaryLineDiagram](#), and [KDChart::TernaryPointDiagram](#).

### 9.12.2.9 **bool AbstractDiagram::checkInvariants (bool *justReturnTheStatus* = false) const** [protected, virtual, inherited]

Definition at line 1060 of file KDChartAbstractDiagram.cpp.

References [KDChart::AbstractDiagram::coordinatePlane\(\)](#).

Referenced by KDCart::RingDiagram::calculateDataBoundaries(), KDCart::PolarDiagram::calculateDataBoundaries(), KDCart::Plotter::calculateDataBoundaries(), KDCart::PieDiagram::calculateDataBoundaries(), KDCart::LineDiagram::calculateDataBoundaries(), KDCart::BarDiagram::calculateDataBoundaries(), KDCart::RingDiagram::paint(), KDCart::PolarDiagram::paint(), KDCart::Plotter::paint(), KDCart::PieDiagram::paint(), KDCart::LineDiagram::paint(), KDCart::BarDiagram::paint(), KDCart::AbstractDiagram::paintDataValueTexts(), KDCart::AbstractDiagram::paintMarker(), and KDCart::AbstractDiagram::paintMarkers().

```

1061 {
1062     if( ! justReturnTheStatus ){
1063         Q_ASSERT_X ( model(), "AbstractDiagram::checkInvariants()",
1064                     "There is no usable model set, for the diagram." );
1065
1066         Q_ASSERT_X ( coordinatePlane(), "AbstractDiagram::checkInvariants()",
1067                     "There is no usable coordinate plane set, for the diagram." );
1068     }
1069     return model() && coordinatePlane();
1070 }

```

#### 9.12.2.10 QModelIndex AbstractDiagram::columnToIndex (int *column*) const [protected, inherited]

Definition at line 309 of file KDCartAbstractDiagram.cpp.

Referenced by KDCart::BarDiagram::barAttributes(), KDCart::AbstractDiagram::brush(), KDCart::AbstractDiagram::dataValueAttributes(), KDCart::AbstractDiagram::isHidden(), KDCart::Plotter::lineAttributes(), KDCart::LineDiagram::lineAttributes(), KDCart::AbstractDiagram::pen(), KDCart::AbstractPieDiagram::pieAttributes(), KDCart::BarDiagram::threeDBarAttributes(), KDCart::Plotter::threeDLineAttributes(), KDCart::LineDiagram::threeDLineAttributes(), and KDCart::AbstractPieDiagram::threeDPieAttributes().

```

310 { // FIXME (Mirko): shouldn't this be headerData? instead of the index for the first row?
311     if( model() )
312         return QModelIndex( model()->index( 0, column, rootIndex() ) );
313     return QModelIndex();
314 }

```

#### 9.12.2.11 bool AbstractDiagram::compare (const [AbstractDiagram](#) \* *other*) const [inherited]

Returns true if both diagrams have the same settings.

Definition at line 129 of file KDCartAbstractDiagram.cpp.

References KDCart::AbstractDiagram::allowOverlappingDataValueTexts(), KDCart::AbstractDiagram::antiAliasing(), KDCart::AbstractDiagram::attributesModel(), KDCart::AttributesModel::compare(), KDCart::AbstractDiagram::datasetDimension(), and KDCart::AbstractDiagram::percentMode().

```

130 {
131     if( other == this ) return true;
132     if( ! other ){
133         //qDebug() << "AbstractDiagram::compare() cannot compare to Null pointer";
134         return false;
135     }
136     /*

```

```

137     qDebug() << "\n                AbstractDiagram::compare() QAbstractScrollArea:";
138         // compare QAbstractScrollArea properties
139     qDebug() <<
140         ((horizontalScrollBarPolicy() == other->horizontalScrollBarPolicy()) &&
141         (verticalScrollBarPolicy() == other->verticalScrollBarPolicy()));
142     qDebug() << "AbstractDiagram::compare() QFrame:";
143         // compare QFrame properties
144     qDebug() <<
145         ((frameShadow() == other->frameShadow()) &&
146         (frameShape() == other->frameShape()) &&
147         (frameWidth() == other->frameWidth()) &&
148         (lineWidth() == other->lineWidth()) &&
149         (midLineWidth() == other->midLineWidth()));
150     qDebug() << "AbstractDiagram::compare() QAbstractItemView:";
151         // compare QAbstractItemView properties
152     qDebug() <<
153         ((alternatingRowColors() == other->alternatingRowColors()) &&
154         (hasAutoScroll() == other->hasAutoScroll()) &&
155 #if QT_VERSION > 0x040199
156         (dragDropMode() == other->dragDropMode()) &&
157         (dragDropOverwriteMode() == other->dragDropOverwriteMode()) &&
158         (horizontalScrollMode() == other->horizontalScrollMode()) &&
159         (verticalScrollMode() == other->verticalScrollMode()) &&
160 #endif
161         (dragEnabled() == other->dragEnabled()) &&
162         (editTriggers() == other->editTriggers()) &&
163         (iconSize() == other->iconSize()) &&
164         (selectionBehavior() == other->selectionBehavior()) &&
165         (selectionMode() == other->selectionMode()) &&
166         (showDropIndicator() == other->showDropIndicator()) &&
167         (tabKeyNavigation() == other->tabKeyNavigation()) &&
168         (textElideMode() == other->textElideMode()));
169     qDebug() << "AbstractDiagram::compare() AttributesModel: ";
170         // compare all of the properties stored in the attributes model
171     qDebug() << attributesModel()->compare( other->attributesModel() );
172     qDebug() << "AbstractDiagram::compare() own:";
173         // compare own properties
174     qDebug() <<
175         ((rootIndex().column() == other->rootIndex().column()) &&
176         (rootIndex().row() == other->rootIndex().row()) &&
177         (allowOverlappingDataValueTexts() == other->allowOverlappingDataValueTexts()) &&
178         (antiAliasing() == other->antiAliasing()) &&
179         (percentMode() == other->percentMode()) &&
180         (datasetDimension() == other->datasetDimension()));
181     */
182     return // compare QAbstractScrollArea properties
183         (horizontalScrollBarPolicy() == other->horizontalScrollBarPolicy()) &&
184         (verticalScrollBarPolicy() == other->verticalScrollBarPolicy()) &&
185         // compare QFrame properties
186         (frameShadow() == other->frameShadow()) &&
187         (frameShape() == other->frameShape()) &&
188     // frameWidth is a read-only property defined by the style, it should not be in here:
189         // (frameWidth() == other->frameWidth()) &&
190         (lineWidth() == other->lineWidth()) &&
191         (midLineWidth() == other->midLineWidth()) &&
192         // compare QAbstractItemView properties
193         (alternatingRowColors() == other->alternatingRowColors()) &&
194         (hasAutoScroll() == other->hasAutoScroll()) &&
195 #if QT_VERSION > 0x040199
196         (dragDropMode() == other->dragDropMode()) &&
197         (dragDropOverwriteMode() == other->dragDropOverwriteMode()) &&
198         (horizontalScrollMode() == other->horizontalScrollMode()) &&
199         (verticalScrollMode() == other->verticalScrollMode()) &&
200 #endif
201         (dragEnabled() == other->dragEnabled()) &&
202         (editTriggers() == other->editTriggers()) &&
203         (iconSize() == other->iconSize()) &&

```

```

204         (selectionBehavior()      == other->selectionBehavior()) &&
205         (selectionMode()          == other->selectionMode()) &&
206         (showDropIndicator()      == other->showDropIndicator()) &&
207         (tabKeyNavigation()       == other->tabKeyNavigation()) &&
208         (textElideMode()          == other->textElideMode()) &&
209         // compare all of the properties stored in the attributes model
210         attributesModel()->compare( other->attributesModel() ) &&
211         // compare own properties
212         (rootIndex().column()     == other->rootIndex().column()) &&
213         (rootIndex().row()        == other->rootIndex().row()) &&
214         (allowOverlappingDataValueTexts() == other->allowOverlappingDataValueTexts()) &&
215         (antiAliasing()           == other->antiAliasing()) &&
216         (percentMode()            == other->percentMode()) &&
217         (datasetDimension()       == other->datasetDimension());
218     }

```

### 9.12.2.12 [AbstractCoordinatePlane](#) \* [AbstractDiagram::coordinatePlane](#) () const [inherited]

The coordinate plane associated with the diagram.

This determines how coordinates in value space are mapped into pixel space. By default this is a [Cartesian-CoordinatePlane](#).

#### Returns:

The coordinate plane associated with the diagram.

Definition at line 220 of file `KDChartAbstractDiagram.cpp`.

References `d`.

Referenced by `KDChart::AbstractDiagram::checkInvariants()`, `KDChart::AbstractCartesianDiagram::layoutPlanes()`, `KDChart::PolarDiagram::paint()`, `KDChart::AbstractDiagram::paintDataValueTexts()`, `KDChart::AbstractDiagram::paintMarkers()`, `KDChart::AbstractPolarDiagram::polarCoordinatePlane()`, and `KDChart::AbstractCartesianDiagram::setCoordinatePlane()`.

```

221 {
222     return d->plane;
223 }

```

### 9.12.2.13 `const QPair< QPointF, QPointF >` [AbstractDiagram::dataBoundaries](#) () const [inherited]

Return the bottom left and top right data point, that the diagram will display (unless the grid adjusts these values).

This method returns a cached result of calculations done by `calculateDataBoundaries`. Classes derived from [AbstractDiagram](#) must implement the `calculateDataBoundaries` function, to specify their own way of calculating the data boundaries. If derived classes want to force recalculation of the data boundaries, they can call [setDataBoundariesDirty\(\)](#)

Returned value is in diagram coordinates.

Definition at line 225 of file `KDChartAbstractDiagram.cpp`.

References `KDChart::AbstractDiagram::calculateDataBoundaries()`, and `d`.

Referenced by `KDChart::CartesianCoordinatePlane::getRawDataBoundingRectFromDiagrams()`, `KDChart::PolarCoordinatePlane::layoutDiagrams()`, `KDChart::Plotter::paint()`, `KDChart::LineDiagram::paint()`, and `KDChart::BarDiagram::paint()`.

```

226 {
227     if( d->databoundariesDirty ){
228         d->databoundaries = calculateDataBoundaries ();
229         d->databoundariesDirty = false;
230     }
231     return d->databoundaries;
232 }
```

#### 9.12.2.14 void AbstractDiagram::dataChanged (const QModelIndex & *topLeft*, const QModelIndex & *bottomRight*) [virtual, inherited]

[reimplemented]

Definition at line 330 of file `KDChartAbstractDiagram.cpp`.

References `KDChart::AbstractDiagram::setDataBoundariesDirty()`.

```

332 {
333     Q_UNUSED( topLeft );
334     Q_UNUSED( bottomRight );
335     // We are still too dumb to do intelligent updates...
336     setDataBoundariesDirty();
337     scheduleDelayedItemsLayout();
338 }
```

#### 9.12.2.15 void KDChart::AbstractDiagram::dataHidden () [signal, inherited]

This signal is emitted, when the hidden status of at least one data cell was (un)set.

Referenced by `KDChart::AbstractDiagram::setHidden()`.

#### 9.12.2.16 QList< QBrush > AbstractDiagram::datasetBrushes () const [inherited]

The set of dataset brushes currently used, for use in legends, etc.

##### Note:

Cell-level override brushes, if set, take precedence over the dataset values, so you might need to check these too, in order to find the brush, that is used for a single cell.

##### Returns:

The current set of dataset brushes.

Definition at line 1018 of file `KDChartAbstractDiagram.cpp`.

References `KDChart::AbstractDiagram::attributesModel()`, `KDChart::AbstractDiagram::attributesModelRootIndex()`, `KDChart::AttributesModel::columnCount()`, `KDChart::DatasetBrushRole`, and `KDChart::AbstractDiagram::datasetDimension()`.

Referenced by `KDChart::Legend::setBrushesFromDiagram()`.

```

1019 {
1020     QList<QBrush> ret;
1021     if( model() == 0 )
1022         return ret;
1023
1024     const int columnCount = attributesModel()->columnCount(attributesModelRootIndex());
1025     for( int i = 0; i < columnCount / datasetDimension(); ++i )
1026         ret << QVariantValue<QBrush>( attributesModel()->headerData( i, Qt::Vertical, DatasetBrushRole() ));
1027
1028     return ret;
1029 }

```

#### 9.12.2.17 int AbstractDiagram::datasetDimension () const [inherited]

The dataset dimension of a diagram determines, how many value dimensions it expects each datapoint to have.

For each dimension it will expect one column of values in the model. If the dimensionality is 1, automatic values will be used for the abscissa.

For example a diagram with the default dimension of 1, will have one column per datapoint (the y values) and will use automatic values for the x axis (1, 2, 3, ... n). If the dimension is 2, the diagram will use the first, (and the third, fifth, etc) columns as X values, and the second, (and the fourth, sixth, etc) column as Y values.

#### Returns:

The dataset dimension of the diagram.

Definition at line 1072 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::brush(), KDChart::AbstractDiagram::compare(), KDChart::AbstractDiagram::datasetBrushes(), KDChart::AbstractDiagram::datasetLabels(), KDChart::AbstractDiagram::datasetMarkers(), KDChart::AbstractDiagram::datasetPens(), KDChart::LineDiagram::getCellValues(), KDChart::CartesianCoordinatePlane::getDataDimensionsList(), KDChart::TernaryPointDiagram::paint(), KDChart::TernaryLineDiagram::paint(), KDChart::AbstractDiagram::paintDataValueTexts(), KDChart::AbstractDiagram::paintMarkers(), KDChart::AbstractDiagram::pen(), KDChart::AbstractDiagram::setPen(), KDChart::Plotter::setType(), and KDChart::LineDiagram::setType().

```

1073 {
1074     return d->datasetDimension;
1075 }

```

#### 9.12.2.18 QStringList AbstractDiagram::datasetLabels () const [inherited]

The set of dataset labels currently displayed, for use in legends, etc.

#### Returns:

The set of dataset labels currently displayed.

Definition at line 1005 of file KDChartAbstractDiagram.cpp.

References `KDChart::AbstractDiagram::attributesModel()`, `KDChart::AbstractDiagram::attributesModelRootIndex()`, `KDChart::AttributesModel::columnCount()`, `KDChart::AbstractDiagram::datasetDimension()`, and `KDChart::AttributesModel::headerData()`.

```

1006 {
1007     QStringList ret;
1008     if( model() == 0 )
1009         return ret;
1010
1011     const int columnCount = attributesModel()->columnCount(attributesModelRootIndex());
1012     for( int i = 0; i < columnCount / datasetDimension(); ++i )
1013         ret << attributesModel()->headerData( i, Qt::Horizontal, Qt::DisplayRole ).toString();
1014
1015     return ret;
1016 }
```

#### 9.12.2.19 `QList< MarkerAttributes > AbstractDiagram::datasetMarkers () const` [inherited]

The set of dataset markers currently used, for use in legends, etc.

##### Note:

Cell-level override markers, if set, take precedence over the dataset values, so you might need to check these too, in order to find the marker, that is shown for a single cell.

##### Returns:

The current set of dataset brushes.

Definition at line 1044 of file `KDChartAbstractDiagram.cpp`.

References `KDChart::AbstractDiagram::attributesModel()`, `KDChart::AbstractDiagram::attributesModelRootIndex()`, `KDChart::AttributesModel::columnCount()`, `KDChart::AbstractDiagram::datasetDimension()`, and `KDChart::DataValueLabelAttributesRole`.

```

1045 {
1046     QList<MarkerAttributes> ret;
1047     if( model() == 0 )
1048         return ret;
1049
1050     const int columnCount = attributesModel()->columnCount(attributesModelRootIndex());
1051     for( int i = 0; i < columnCount / datasetDimension(); ++i )
1052     {
1053         const DataValueAttributes a =
1054             qVariantValue<DataValueAttributes>( attributesModel()->headerData( i, Qt::Vertical, DataValueLabelAttributesRole ) );
1055         ret << a.markerAttributes();
1056     }
1057     return ret;
1058 }
```

#### 9.12.2.20 `QList< QPen > AbstractDiagram::datasetPens () const` [inherited]

The set of dataset pens currently used, for use in legends, etc.

##### Note:

Cell-level override pens, if set, take precedence over the dataset values, so you might need to check these too, in order to find the pens, that is used for a single cell.



**Returns:**

The current set of dataset pens.

Definition at line 1031 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::attributesModelRootIndex(), KDChart::AttributesModel::columnCount(), KDChart::AbstractDiagram::datasetDimension(), and KDChart::DatasetPenRole.

```

1032 {
1033     QList<QPen> ret;
1034     if( model() == 0 )
1035         return ret;
1036
1037     const int columnCount = attributesModel()->columnCount(attributesModelRootIndex());
1038     for( int i = 0; i < columnCount / datasetDimension(); ++i )
1039         ret << qVariantValue<QPen>( attributesModel()->headerData( i, Qt::Vertical, DatasetPenRole ) );
1040
1041     return ret;
1042 }
```

### 9.12.2.21 [DataValueAttributes](#) AbstractDiagram::dataValueAttributes (const QModelIndex & index) const [inherited]

Retrieve the [DataValueAttributes](#) for the given index.

This will fall back automatically to what was set at dataset or model level, if there are no datapoint specific settings.

**Parameters:**

*index* The datapoint to retrieve the attributes for.

**Returns:**

The [DataValueAttributes](#) for the given index.

Definition at line 421 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AttributesModel::data(), and KDChart::DataValueLabelAttributesRole.

```

422 {
423     return qVariantValue<DataValueAttributes>(
424         attributesModel()->data( attributesModel()->mapFromSource(index),
425                                 KDChart::DataValueLabelAttributesRole ) );
426 }
```

### 9.12.2.22 [DataValueAttributes](#) AbstractDiagram::dataValueAttributes (int column) const [inherited]

Retrieve the [DataValueAttributes](#) for the given dataset.

This will fall back automatically to what was set at model level, if there are no dataset specific settings.

**Parameters:**

*column* The dataset to retrieve the attributes for.

**Returns:**

The [DataValueAttributes](#) for the given dataset.

Definition at line 414 of file KDChartAbstractDiagram.cpp.

References [KDChart::AbstractDiagram::attributesModel\(\)](#), [KDChart::AbstractDiagram::columnToIndex\(\)](#), [KDChart::AttributesModel::data\(\)](#), and [KDChart::DataValueLabelAttributesRole](#).

```
415 {
416     return qVariantValue<DataValueAttributes>(
417         attributesModel()->data( attributesModel()->mapFromSource(columnToIndex( column )),
418         KDChart::DataValueLabelAttributesRole ) );
419 }
```

**9.12.2.23 [DataValueAttributes](#) AbstractDiagram::dataValueAttributes () const** [inherited]

Retrieve the [DataValueAttributes](#) specified globally.

This will fall back automatically to the default settings, if there are no specific settings.

**Returns:**

The global [DataValueAttributes](#).

Definition at line 408 of file KDChartAbstractDiagram.cpp.

References [KDChart::AbstractDiagram::attributesModel\(\)](#), [KDChart::DataValueLabelAttributesRole](#), and [KDChart::AttributesModel::modelData\(\)](#).

Referenced by [KDChart::AbstractDiagram::paintDataValueText\(\)](#), [KDChart::AbstractDiagram::paintMarker\(\)](#), and [KDChart::TernaryLineDiagram::TernaryLineDiagram\(\)](#).

```
409 {
410     return qVariantValue<DataValueAttributes>(
411         attributesModel()->modelData( KDChart::DataValueLabelAttributesRole ) );
412 }
```

**9.12.2.24 void AbstractDiagram::doItemsLayout ()** [virtual, inherited]

[reimplemented]

Definition at line 321 of file KDChartAbstractDiagram.cpp.

References [d](#), and [KDChart::AbstractDiagram::update\(\)](#).

```
322 {
323     if ( d->plane ) {
324         d->plane->layoutDiagrams();
325         update();
326     }
327     QAbstractItemView::doItemsLayout();
328 }
```

**9.12.2.25 int AbstractDiagram::horizontalOffset () const** [virtual, inherited]

[reimplemented]

Definition at line 950 of file KDChartAbstractDiagram.cpp.

```
951 { return 0; }
```

**9.12.2.26 QModelIndex AbstractDiagram::indexAt (const QPoint & point) const** [virtual, inherited]

[reimplemented]

Definition at line 1098 of file KDChartAbstractDiagram.cpp.

References d.

```
1099 {
1100     return d->indexAt( point );
1101 }
```

**9.12.2.27 QModelIndexList AbstractDiagram::indexesAt (const QPoint & point) const** [inherited]

This method is added alongside with indexAt from QAIM, since in kdchart multiple indexes can be displayed at the same spot.

Definition at line 1103 of file KDChartAbstractDiagram.cpp.

References d.

```
1104 {
1105     return d->indexesAt( point );
1106 }
```

**9.12.2.28 bool AbstractDiagram::isHidden (const QModelIndex & index) const** [inherited]

Retrieve the hidden status for the given index.

This will fall back automatically to what was set at dataset or diagram level, if there are no datapoint specific settings.

**Parameters:**

*index* The datapoint to retrieve the hidden status for.

**Returns:**

The hidden status for the given index.

Definition at line 380 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AttributesModel::data(), and KDChart::DataHiddenRole.

```

381 {
382     return qVariantValue<bool>(
383         attributesModel()->data(
384             attributesModel()->mapFromSource(index),
385             DataHiddenRole ) );
386 }

```

#### 9.12.2.29 bool AbstractDiagram::isHidden (int *column*) const [inherited]

Retrieve the hidden status for the given dataset.

This will fall back automatically to what was set at diagram level, if there are no dataset specific settings.

##### Parameters:

*column* The dataset to retrieve the hidden status for.

##### Returns:

The hidden status for the given dataset.

Definition at line 373 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::columnToIndex(), KDChart::AttributesModel::data(), and KDChart::DataHiddenRole.

```

374 {
375     return qVariantValue<bool>(
376         attributesModel()->data(
377             attributesModel()->mapFromSource(columnToIndex( column )),
378             DataHiddenRole ) );
379 }

```

#### 9.12.2.30 bool AbstractDiagram::isHidden () const [inherited]

Retrieve the hidden status specified globally.

This will fall back automatically to the default settings ( = not hidden), if there are no specific settings.

##### Returns:

The global hidden status.

Definition at line 367 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DataHiddenRole, and KDChart::AttributesModel::modelData().

Referenced by KDChart::LineDiagram::valueForCellTesting().

```

368 {
369     return qVariantValue<bool>(
370         attributesModel()->modelData( DataHiddenRole ) );
371 }

```

### 9.12.2.31 bool AbstractDiagram::isIndexHidden (const QModelIndex & *index*) const

[virtual, inherited]

[reimplemented]

Definition at line 956 of file KDChartAbstractDiagram.cpp.

```
957 { return true; }
```

### 9.12.2.32 QStringList AbstractDiagram::itemRowLabels () const

[inherited]

The set of item row labels currently displayed, for use in Abscissa axes, etc.

#### Returns:

The set of item row labels currently displayed.

Definition at line 989 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::attributesModelRootIndex(), KDChart::AttributesModel::rowCount(), KDChart::AbstractDiagram::unitPrefix(), and KDChart::AbstractDiagram::unitSuffix().

```
990 {
991     QStringList ret;
992     if( model() ){
993         //qDebug() << "AbstractDiagram::itemRowLabels(): " << attributesModel()->rowCount(attributesModelRootIndex());
994         const int rowCount = attributesModel()->rowCount(attributesModelRootIndex());
995         for( int i = 0; i < rowCount; ++i ){
996             //qDebug() << "item row label: " << attributesModel()->headerData( i, Qt::Vertical, Qt::DisplayRole );
997             ret << unitPrefix( i, Qt::Horizontal, true ) +
998                 attributesModel()->headerData( i, Qt::Vertical, Qt::DisplayRole ).toString() +
999                 unitSuffix( i, Qt::Horizontal, true );
1000         }
1001     }
1002     return ret;
1003 }
```

### 9.12.2.33 void KDChart::AbstractDiagram::layoutChanged ([AbstractDiagram \\*](#))

[signal, inherited]

Diagrams are supposed to emit this signal, when the layout of one of their element changes.

Layouts can change, for example, when axes are added or removed, or when the configuration was changed in a way that the axes or the diagram itself are displayed in a different geometry. Changes in the diagrams coordinate system also result in the [layoutChanged\(\)](#) signal being emitted.

Referenced by KDChart::AbstractDiagram::setDatasetDimension(), KDChart::AbstractPieDiagram::setPieAttributes(), KDChart::AbstractPieDiagram::setThreeDPieAttributes(), KDChart::Plotter::setType(), KDChart::LineDiagram::setType(), and KDChart::BarDiagram::setType().

### 9.12.2.34 void KDChart::AbstractDiagram::modelsChanged ()

[signal, inherited]

This signal is emitted, when either the model or the [AttributesModel](#) is replaced.

Referenced by KDChart::AbstractDiagram::setAttributesModel(), and KDChart::AbstractDiagram::setModel().

### 9.12.2.35 QModelIndex AbstractDiagram::moveCursor (CursorAction *cursorAction*, Qt::KeyboardModifiers *modifiers*) [virtual, inherited]

[reimplemented]

Definition at line 947 of file KDChartAbstractDiagram.cpp.

```
948 { return QModelIndex(); }
```

### 9.12.2.36 void AbstractDiagram::paintDataValueText (QPainter \* *painter*, const QModelIndex & *index*, const QPointF & *pos*, double *value*) [inherited]

Definition at line 468 of file KDChartAbstractDiagram.cpp.

References `d`, `KDChart::DataValueAttributes::dataLabel()`, `KDChart::AbstractDiagram::dataValueAttributes()`, `KDChart::DataValueAttributes::decimalDigits()`, `KDChart::DataValueAttributes::isVisible()`, `KDChartEnums::MeasureOrientationMinimum`, `KDChart::DataValueAttributes::position()`, `KDChart::DataValueAttributes::prefix()`, `KDChart::DataValueAttributes::suffix()`, and `KDChart::DataValueAttributes::textAttributes()`.

Referenced by `KDChart::RingDiagram::paint()`, `KDChart::PolarDiagram::paint()`, and `KDChart::AbstractDiagram::paintDataValueTexts()`.

```
472 {
473     // paint one data series
474     const DataValueAttributes a( dataValueAttributes(index) );
475     if ( !a.isVisible() ) return;
476
477     // handle decimal digits
478     int decimalDigits = a.decimalDigits();
479     int decimalPos = QString::number( value ).indexOf( QLatin1Char( '.' ) );
480     QString roundedValue;
481     if ( a.dataLabel().isNull() ) {
482         if ( decimalPos > 0 && value != 0 )
483             roundedValue = roundValues( value, decimalPos, decimalDigits );
484         else
485             roundedValue = QString::number( value );
486     } else
487         roundedValue = a.dataLabel();
488     // handle prefix and suffix
489     if ( !a.prefix().isNull() )
490         roundedValue.prepend( a.prefix() );
491
492     if ( !a.suffix().isNull() )
493         roundedValue.append( a.suffix() );
494
495     const TextAttributes ta( a.textAttributes() );
496     // FIXME draw the non-text bits, background, etc
497     if ( ta.isVisible() ) {
498
499         QPointF pt( pos );
500         /* for debugging:
501         PainterSaver painterSaver( painter );
502         painter->setPen( Qt::black );
503         painter->drawLine( pos - QPointF( 1,1), pos + QPointF( 1,1) );
504         painter->drawLine( pos - QPointF(-1,1), pos + QPointF(-1,1) );
505         */
506
507         QTextDocument doc;
508         if( Qt::mightBeRichText( roundedValue ) )
509             doc.setHtml( roundedValue );
510         else
```

```

511         doc.setPlainText( roundedValue );
512
513         const RelativePosition relPos( a.position( value >= 0.0 ) );
514         const Qt::Alignment alignBottomLeft = Qt::AlignBottom | Qt::AlignLeft;
515         const QFont calculatedFont( ta.calculatedFont( d->plane, KDChartEnums::MeasureOrientationMinimum );
516         const QRectF boundRect( d->cachedFontMetrics( calculatedFont, painter->device() )->boundingRect( calculatedFont );
517
518         // To place correctly
519         pt.ry() -= boundRect.height();
520
521         // qDebug() << "calculatedFont's point size:" << calculatedFont.pointSizeF();
522         // adjust the text start point position, if alignment is not Bottom/Left
523         if( (relPos.alignment() & alignBottomLeft) != alignBottomLeft ){
524             if( relPos.alignment() & Qt::AlignRight )
525                 pt.rx() -= boundRect.width();
526             else if( relPos.alignment() & Qt::AlignHCenter )
527                 pt.rx() -= 0.5 * boundRect.width();
528
529             if( relPos.alignment() & Qt::AlignTop )
530                 pt.ry() += boundRect.height();
531             else if( relPos.alignment() & Qt::AlignVCenter )
532                 pt.ry() += 0.5 * boundRect.height();
533         }
534
535         // FIXME draw the non-text bits, background, etc
536
537         if ( a.showRepetitiveDataLabels() ||
538             pos.x() <= d->lastX ||
539             d->lastRoundedValue != roundedValue ) {
540             d->lastRoundedValue = roundedValue;
541             d->lastX = pos.x();
542             PainterSaver painterSaver( painter );
543
544             doc.setDefaultFont( calculatedFont );
545             QAbstractTextDocumentLayout::PaintContext context;
546             context.palette = palette();
547             context.palette.setColor( QPalette::Text, ta.pen().color() );
548
549             painter->translate( pt );
550             painter->rotate( ta.rotation() );
551
552             QAbstractTextDocumentLayout* layout = doc.documentLayout();
553             layout->draw( painter, context );
554         }
555     }
556 }

```

### 9.12.237 void AbstractDiagram::paintDataValueTexts (QPainter \* *painter*) [protected, virtual, inherited]

Definition at line 584 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::checkInvariants(), KDChart::AbstractDiagram::coordinatePlane(), KDChart::AbstractDiagram::datasetDimension(), KDChart::AbstractDiagram::paintDataValueText(), and KDChart::AbstractCoordinatePlane::translate().

```

585 {
586     if ( !checkInvariants() ) return;
587     const int rowCount = model()->rowCount( rootIndex() );
588     const int columnCount = model()->columnCount( rootIndex() );
589     for ( int i=datasetDimension()-1; i<columnCount; i += datasetDimension() ) {
590         for ( int j=0; j<rowCount; ++j ) {
591             const QModelIndex index = model()->index( j, i, rootIndex() );
592             double value = model()->data( index ).toDouble();

```

```

593         const QPointF pos = coordinatePlane()->translate( QPointF( j, value ) );
594         paintDataValueText( painter, index, pos, value );
595     }
596 }
597 }

```

### 9.12.2.38 void AbstractDiagram::paintMarker (QPainter \* *painter*, const QModelIndex & *index*, const QPointF & *pos*) [inherited]

Definition at line 600 of file KDChartAbstractDiagram.cpp.

References [KDChart::AbstractDiagram::brush\(\)](#), [KDChart::AbstractDiagram::checkInvariants\(\)](#), [d](#), [KDChart::AbstractDiagram::dataValueAttributes\(\)](#), [KDChart::MarkerAttributes::isVisible\(\)](#), [KDChart::DataValueAttributes::isVisible\(\)](#), [KDChart::DataValueAttributes::markerAttributes\(\)](#), [KDChart::MarkerAttributes::markerColor\(\)](#), [KDChart::MarkerAttributes::markerSize\(\)](#), [KDChart::AbstractDiagram::paintMarker\(\)](#), and [KDChart::MarkerAttributes::pen\(\)](#).

```

603 {
604     if ( !checkInvariants() ) return;
605     DataValueAttributes a = dataValueAttributes(index);
606     if ( !a.isVisible() ) return;
607     const MarkerAttributes &ma = a.markerAttributes();
608     if ( !ma.isVisible() ) return;
609
610     PainterSaver painterSaver( painter );
611     QSizeF maSize( ma.markerSize() );
612     QBrush indexBrush( brush( index ) );
613     QPen indexPen( ma.pen() );
614     if ( ma.markerColor().isValid() )
615         indexBrush.setColor( ma.markerColor() );
616
617     paintMarker( painter, ma, indexBrush, indexPen, pos, maSize );
618
619     // workaround: BC cannot be changed, otherwise we would pass the
620     // index down to next-lower paintMarker function. So far, we
621     // basically save a circle of radius maSize at pos in the
622     // reverseMapper. This means that ^^ this version of paintMarker
623     // needs to be called to reverse-map the marker.
624     d->reverseMapper.addCircle( index.row(), index.column(), pos, 2 * maSize );
625 }

```

### 9.12.2.39 void AbstractDiagram::paintMarker (QPainter \* *painter*, const [MarkerAttributes](#) & *markerAttributes*, const QBrush & *brush*, const QPen & *pen*, const QPointF & *point*, const QSizeF & *size*) [virtual, inherited]

Definition at line 627 of file KDChartAbstractDiagram.cpp.

References [KDChart::MarkerAttributes::Marker1Pixel](#), [KDChart::MarkerAttributes::Marker4Pixels](#), [KDChart::MarkerAttributes::MarkerCircle](#), [KDChart::MarkerAttributes::MarkerCross](#), [KDChart::MarkerAttributes::MarkerDiamond](#), [KDChart::MarkerAttributes::MarkerFastCross](#), [KDChart::MarkerAttributes::MarkerRing](#), [KDChart::MarkerAttributes::MarkerSquare](#), and [KDChart::MarkerAttributes::markerStyle\(\)](#).

Referenced by [KDChart::TernaryPointDiagram::paint\(\)](#), [KDChart::TernaryLineDiagram::paint\(\)](#), [KDChart::MarkerLayoutItem::paintIntoRect\(\)](#), [KDChart::AbstractDiagram::paintMarker\(\)](#), and [KDChart::AbstractDiagram::paintMarkers\(\)](#).

```

633 {

```



```

634     const QPen oldPen( painter->pen() );
635     // Pen is used to paint 4Pixels - 1 Pixel - Ring and FastCross types.
636     // make sure to use the brush color - see above in those cases.
637     const bool isFourPixels = (markerAttributes.markerStyle() == MarkerAttributes::Marker4Pixels);
638     if( isFourPixels || (markerAttributes.markerStyle() == MarkerAttributes::Marker1Pixel) ){
639         // for high-performance point charts with tiny point markers:
640         painter->setPen( QPen( brush.color().light() ) );
641         if( isFourPixels ){
642             const qreal x = pos.x();
643             const qreal y = pos.y();
644             painter->drawLine( QPointF(x-1.0,y-1.0),
645                               QPointF(x+1.0,y-1.0) );
646             painter->drawLine( QPointF(x-1.0,y),
647                               QPointF(x+1.0,y) );
648             painter->drawLine( QPointF(x-1.0,y+1.0),
649                               QPointF(x+1.0,y+1.0) );
650         }
651         painter->drawPoint( pos );
652     }else{
653         PainterSaver painterSaver( painter );
654         // we only a solid line surrounding the markers
655         QPen painterPen( pen );
656         painterPen.setStyle( Qt::SolidLine );
657         painter->setPen( painterPen );
658         painter->setBrush( brush );
659         painter->setRenderHint ( QPainter::Antialiasing );
660         painter->translate( pos );
661         switch ( markerAttributes.markerStyle() ) {
662             case MarkerAttributes::MarkerCircle:
663                 painter->drawEllipse( QRectF( 0 - maSize.height()/2, 0 - maSize.width()/2,
664                                                maSize.height(), maSize.width() ) );
665                 break;
666             case MarkerAttributes::MarkerSquare:
667                 {
668                     QRectF rect( 0 - maSize.width()/2, 0 - maSize.height()/2,
669                                 maSize.width(), maSize.height() );
670                     painter->drawRect( rect );
671                     break;
672                 }
673             case MarkerAttributes::MarkerDiamond:
674                 {
675                     QVector<QPointF> diamondPoints;
676                     QPointF top, left, bottom, right;
677                     top = QPointF( 0, 0 - maSize.height()/2 );
678                     left = QPointF( 0 - maSize.width()/2, 0 );
679                     bottom = QPointF( 0, maSize.height()/2 );
680                     right = QPointF( maSize.width()/2, 0 );
681                     diamondPoints << top << left << bottom << right;
682                     painter->drawPolygon( diamondPoints );
683                     break;
684                 }
685             // both handled on top of the method:
686             case MarkerAttributes::Marker1Pixel:
687             case MarkerAttributes::Marker4Pixels:
688                 break;
689             case MarkerAttributes::MarkerRing:
690                 {
691                     painter->setPen( QPen( brush.color() ) );
692                     painter->setBrush( Qt::NoBrush );
693                     painter->drawEllipse( QRectF( 0 - maSize.height()/2, 0 - maSize.width()/2,
694                                                    maSize.height(), maSize.width() ) );
695                     break;
696                 }
697             case MarkerAttributes::MarkerCross:
698                 {
699                     QRectF rect( maSize.width()*-0.5, maSize.height()*-0.2,
700                                 maSize.width(), maSize.height()*0.4 );

```

```

701         painter->drawRect( rect );
702         rect.setTopLeft(QPointF( maSize.width()*-0.2, maSize.height()*-0.5 ));
703         rect.setSize(QSizeF( maSize.width()*0.4, maSize.height() ));
704         painter->drawRect( rect );
705         break;
706     }
707     case MarkerAttributes::MarkerFastCross:
708     {
709         QPointF left, right, top, bottom;
710         left = QPointF( -maSize.width()/2, 0 );
711         right = QPointF( maSize.width()/2, 0 );
712         top = QPointF( 0, -maSize.height()/2 );
713         bottom= QPointF( 0, maSize.height()/2 );
714         painter->setPen( QPen( brush.color() ) );
715         painter->drawLine( left, right );
716         painter->drawLine( top, bottom );
717         break;
718     }
719     default:
720         Q_ASSERT_X ( false, "paintMarkers()",
721                     "Type item does not match a defined Marker Type." );
722 }
723 }
724 painter->setPen( oldPen );
725 }

```

**9.12.2.40 void AbstractDiagram::paintMarkers (QPainter \* *painter*)** [protected, virtual, inherited]

Definition at line 727 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::checkInvariants(), KDChart::AbstractDiagram::coordinatePlane(), KDChart::AbstractDiagram::datasetDimension(), KDChart::AbstractDiagram::paintMarker(), and KDChart::AbstractCoordinatePlane::translate().

```

728 {
729     if ( !checkInvariants() ) return;
730     const int rowCount = model()->rowCount( rootIndex() );
731     const int columnCount = model()->columnCount( rootIndex() );
732     for ( int i=datasetDimension()-1; i<columnCount; i += datasetDimension() ) {
733         for ( int j=0; j<rowCount; ++j ) {
734             const QModelIndex index = model()->index( j, i, rootIndex() );
735             double value = model()->data( index ).toDouble();
736             const QPointF pos = coordinatePlane()->translate( QPointF( j, value ) );
737             paintMarker( painter, index, pos );
738         }
739     }
740 }

```

**9.12.2.41 QPen AbstractDiagram::pen (const QModelIndex & *index*) const** [inherited]

Retrieve the pen to be used, for painting the datapoint at the given index in the model.

#### Parameters:

*index* The index of the datapoint in the model.

#### Returns:

The pen to use for painting.

Definition at line 788 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AttributesModel::data(), KDChart::AbstractDiagram::datasetDimension(), KDChart::DatasetPenRole, and KDChart::AbstractDiagram::pen().

```

789 {
790     if( datasetDimension() > 1 )
791         return pen( index.column() );
792     return qVariantValue<QPen>(
793         attributesModel()->data(
794             attributesModel()->mapFromSource( index ),
795             DatasetPenRole ) );
796 }
```

#### 9.12.2.42 QPen AbstractDiagram::pen (int *dataset*) const [inherited]

Retrieve the pen to be used for the given dataset.

This will fall back automatically to what was set at model level, if there are no dataset specific settings.

##### Parameters:

*dataset* The dataset to retrieve the pen for.

##### Returns:

The pen to use for painting.

Definition at line 777 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::columnToIndex(), KDChart::AttributesModel::data(), KDChart::AbstractDiagram::datasetDimension(), and KDChart::DatasetPenRole.

```

778 {
779     if( datasetDimension() > 1 )
780         dataset /= datasetDimension();
781     return qVariantValue<QPen>(
782         attributesModel()->data(
783             attributesModel()->mapFromSource( columnToIndex( dataset ) ),
784             DatasetPenRole ) );
785 }
786 }
```

#### 9.12.2.43 QPen AbstractDiagram::pen () const [inherited]

Retrieve the pen to be used for painting datapoints globally.

This will fall back automatically to the default settings, if there are no specific settings.

##### Returns:

The pen to use for painting.

Definition at line 771 of file KDChartAbstractDiagram.cpp.

References `KDChart::AbstractDiagram::attributesModel()`, `KDChart::AttributesModel::data()`, and `KDChart::DatasetPenRole`.

Referenced by `KDChart::TernaryPointDiagram::paint()`, `KDChart::TernaryLineDiagram::paint()`, and `KDChart::AbstractDiagram::pen()`.

```

772 {
773     return qVariantValue<QPen>(
774         attributesModel()->data( DatasetPenRole ) );
775 }
```

#### 9.12.2.44 `bool AbstractDiagram::percentMode () const` [inherited]

Definition at line 462 of file `KDChartAbstractDiagram.cpp`.

References `d`.

Referenced by `KDChart::AbstractDiagram::compare()`, and `KDChart::CartesianCoordinatePlane::getDataDimensionsList()`.

```

463 {
464     return d->percent;
465 }
```

#### 9.12.2.45 `void KDChart::AbstractDiagram::propertiesChanged ()` [signal, inherited]

Emitted upon change of a property of the Diagram.

Referenced by `KDChart::Plotter::resetLineAttributes()`, `KDChart::LineDiagram::resetLineAttributes()`, `KDChart::AbstractDiagram::setAllowOverlappingDataValueTexts()`, `KDChart::AbstractDiagram::setAntiAliasing()`, `KDChart::BarDiagram::setBarAttributes()`, `KDChart::AbstractDiagram::setBrush()`, `KDChart::AbstractDiagram::setDataValueAttributes()`, `KDChart::Plotter::setLineAttributes()`, `KDChart::LineDiagram::setLineAttributes()`, `KDChart::AbstractDiagram::setPen()`, `KDChart::AbstractDiagram::setPercentMode()`, `KDChart::BarDiagram::setThreeDBarAttributes()`, `KDChart::Plotter::setThreeDLineAttributes()`, `KDChart::LineDiagram::setThreeDLineAttributes()`, `KDChart::Plotter::setType()`, `KDChart::LineDiagram::setType()`, `KDChart::BarDiagram::setType()`, `KDChart::Plotter::setValueTrackerAttributes()`, and `KDChart::LineDiagram::setValueTrackerAttributes()`.

#### 9.12.2.46 `void AbstractDiagram::scrollTo (const QModelIndex & index, ScrollHint hint = EnsureVisible)` [virtual, inherited]

[reimplemented]

Definition at line 942 of file `KDChartAbstractDiagram.cpp`.

```

943 {}
```

#### 9.12.2.47 `void AbstractDiagram::setAllowOverlappingDataValueTexts (bool allow)` [inherited]

Set whether data value labels are allowed to overlap.

**Parameters:**

*allow* True means that overlapping labels are allowed.

Definition at line 434 of file KDChartAbstractDiagram.cpp.

References `d`, and `KDChart::AbstractDiagram::propertiesChanged()`.

```
435 {  
436     d->allowOverlappingDataValueTexts = allow;  
437     emit propertiesChanged();  
438 }
```

**9.12.2.48 void AbstractDiagram::setAntiAliasing (bool *enabled*)** [inherited]

Set whether anti-aliasing is to be used while rendering this diagram.

**Parameters:**

*enabled* True means that AA is enabled.

Definition at line 445 of file KDChartAbstractDiagram.cpp.

References `d`, and `KDChart::AbstractDiagram::propertiesChanged()`.

```
446 {  
447     d->antiAliasing = enabled;  
448     emit propertiesChanged();  
449 }
```

**9.12.2.49 void AbstractDiagram::setAttributesModel ([AttributesModel](#) \* *model*)** [virtual, inherited]

Associate an [AttributesModel](#) with this diagram.

Note that the diagram does `_not_` take ownership of the [AttributesModel](#). This should thus only be used with [AttributesModels](#) that have been explicitly created by the user, and are owned by her. Setting an [AttributesModel](#) that is internal to another diagram is an error.

Correct:

```
AttributesModel *am = new AttributesModel( model, 0 );  
diagram1->setAttributesModel( am );  
diagram2->setAttributesModel( am );
```

Wrong:

```
diagram1->setAttributesModel( diagram2->attributesModel() );
```

**Parameters:**

*model* The [AttributesModel](#) to use for this diagram.

**See also:**

[AttributesModel](#), [usesExternalAttributesModel](#)

Reimplemented in [KDChart::AbstractCartesianDiagram](#).

Definition at line 255 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::AbstractDiagram::modelsChanged\(\)](#), and [KDChart::AbstractDiagram::setDataBoundariesDirty\(\)](#).

Referenced by [KDChart::AbstractCartesianDiagram::setAttributesModel\(\)](#).

```

256 {
257     if( amodel->sourceModel() != model() ) {
258         qWarning("KDChart::AbstractDiagram::setAttributesModel() failed: "
259                "Trying to set an attributesmodel which works on a different "
260                "model than the diagram.");
261         return;
262     }
263     if( qobject_cast<PrivateAttributesModel*>(amodel) ) {
264         qWarning("KDChart::AbstractDiagram::setAttributesModel() failed: "
265                "Trying to set an attributesmodel that is private to another diagram.");
266         return;
267     }
268     d->setAttributesModel(amodel);
269     scheduleDelayedItemsLayout();
270     setDataBoundariesDirty();
271     emit modelsChanged();
272 }
```

#### 9.12.2.50 void AbstractDiagram::setAttributesModelRootIndex (const QModelIndex &) [protected, inherited]

Definition at line 293 of file KDChartAbstractDiagram.cpp.

References [d](#), and [KDChart::AbstractDiagram::setDataBoundariesDirty\(\)](#).

Referenced by [KDChart::AbstractDiagram::setRootIndex\(\)](#).

```

294 {
295     d->attributesModelRootIndex=idx;
296     setDataBoundariesDirty();
297     scheduleDelayedItemsLayout();
298 }
```

#### 9.12.2.51 void AbstractDiagram::setBrush (const QBrush & brush) [inherited]

Set the brush to be used, for painting all datasets in the model.

##### Parameters:

**brush** The brush to use.

Definition at line 806 of file KDChartAbstractDiagram.cpp.

References [KDChart::AbstractDiagram::attributesModel\(\)](#), [KDChart::DatasetBrushRole](#), [KDChart::AbstractDiagram::propertiesChanged\(\)](#), and [KDChart::AttributesModel::setModelData\(\)](#).

```

807 {
808     attributesModel()->setModelData(
809         qVariantFromValue( brush ), DatasetBrushRole );
810     emit propertiesChanged();
811 }
```

**9.12.2.52 void AbstractDiagram::setBrush (int *dataset*, const QBrush & *brush*)** [inherited]

Set the brush to be used, for painting the given dataset.

**Parameters:**

*dataset* The dataset's column in the model.

*brush* The brush to use.

Definition at line 813 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DatasetBrushRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setHeaderData().

```

814 {
815     attributesModel()->setHeaderData(
816         column, Qt::Vertical,
817         qVariantFromValue( brush ),
818         DatasetBrushRole );
819     emit propertiesChanged();
820 }
```

**9.12.2.53 void AbstractDiagram::setBrush (const QModelIndex & *index*, const QBrush & *brush*)** [inherited]

Set the brush to be used, for painting the datapoint at the given index.

**Parameters:**

*index* The datapoint's index in the model.

*brush* The brush to use.

Definition at line 798 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DatasetBrushRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setData().

```

799 {
800     attributesModel()->setData(
801         attributesModel()->mapFromSource( index ),
802         qVariantFromValue( brush ), DatasetBrushRole );
803     emit propertiesChanged();
804 }
```

**9.12.2.54 void AbstractDiagram::setCoordinatePlane (AbstractCoordinatePlane \* *plane*)** [virtual, inherited]

Set the coordinate plane associated with the diagram.

This determines how coordinates in value space are mapped into pixel space. The chart takes ownership.

**Returns:**

The coordinate plane associated with the diagram.

Reimplemented in [KDChart::AbstractCartesianDiagram](#).

Definition at line 316 of file KDChartAbstractDiagram.cpp.

References [d](#).

Referenced by [KDChart::AbstractCoordinatePlane::addDiagram\(\)](#), [KDChart::AbstractCartesianDiagram::setCoordinatePlane\(\)](#), and [KDChart::AbstractCoordinatePlane::takeDiagram\(\)](#).

```
317 {
318     d->plane = parent;
319 }
```

#### 9.12.2.55 void AbstractDiagram::setDataBoundariesDirty () const [protected, inherited]

Definition at line 234 of file KDChartAbstractDiagram.cpp.

References [d](#).

Referenced by [KDChart::AbstractDiagram::dataChanged\(\)](#), [KDChart::Plotter::resize\(\)](#), [KDChart::LineDiagram::resize\(\)](#), [KDChart::BarDiagram::resize\(\)](#), [KDChart::AbstractDiagram::setAttributesModel\(\)](#), [KDChart::AbstractDiagram::setAttributesModelRootIndex\(\)](#), [KDChart::AbstractDiagram::setDatasetDimension\(\)](#), [KDChart::AbstractDiagram::setModel\(\)](#), [KDChart::BarDiagram::setThreeDBarAttributes\(\)](#), [KDChart::Plotter::setThreeDLineAttributes\(\)](#), [KDChart::LineDiagram::setThreeDLineAttributes\(\)](#), [KDChart::Plotter::setType\(\)](#), [KDChart::LineDiagram::setType\(\)](#), and [KDChart::BarDiagram::setType\(\)](#).

```
235 {
236     d->databoundariesDirty = true;
237 }
```

#### 9.12.2.56 void AbstractDiagram::setDatasetDimension (int *dimension*) [inherited]

Sets the dataset dimension of the diagram.

See also:

[datasetDimension](#).

Parameters:

*dimension*

Definition at line 1077 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::AbstractDiagram::layoutChanged\(\)](#), and [KDChart::AbstractDiagram::setDataBoundariesDirty\(\)](#).

Referenced by [KDChart::Widget::setType\(\)](#), [KDChart::TernaryLineDiagram::TernaryLineDiagram\(\)](#), and [KDChart::TernaryPointDiagram::TernaryPointDiagram\(\)](#).

```
1078 {
1079     if ( d->datasetDimension == dimension ) return;
1080     d->datasetDimension = dimension;
1081     setDataBoundariesDirty();
1082     emit layoutChanged( this );
1083 }
```



**9.12.2.57 void AbstractDiagram::setDataValueAttributes (const [DataValueAttributes](#) & *a*)**  
[inherited]

Set the [DataValueAttributes](#) for all datapoints in the model.

**Parameters:**

*a* The attributes to set.

Definition at line 428 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::DataValueLabelAttributesRole](#), and [KDChart::AbstractDiagram::propertiesChanged\(\)](#).

```
429 {  
430     d->attributesModel->setModelData( qVariantFromValue( a ), DataValueLabelAttributesRole );  
431     emit propertiesChanged();  
432 }
```

**9.12.2.58 void AbstractDiagram::setDataValueAttributes (int *dataset*, const [DataValueAttributes](#) & *a*)**  
[inherited]

Set the [DataValueAttributes](#) for the given dataset.

**Parameters:**

*dataset* The dataset to set the attributes for.

*a* The attributes to set.

Definition at line 400 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::DataValueLabelAttributesRole](#), and [KDChart::AbstractDiagram::propertiesChanged\(\)](#).

```
401 {  
402     d->attributesModel->setHeaderData(  
403         column, Qt::Vertical,  
404         qVariantFromValue( a ), DataValueLabelAttributesRole );  
405     emit propertiesChanged();  
406 }
```

**9.12.2.59 void AbstractDiagram::setDataValueAttributes (const [QModelIndex](#) & *index*, const [DataValueAttributes](#) & *a*)**  
[inherited]

Set the [DataValueAttributes](#) for the given index.

**Parameters:**

*index* The datapoint to set the attributes for.

*a* The attributes to set.

Definition at line 389 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::DataValueLabelAttributesRole](#), and [KDChart::AbstractDiagram::propertiesChanged\(\)](#).

```

391 {
392     d->attributesModel->setData(
393         d->attributesModel->mapFromSource( index ),
394         qVariantFromValue( a ),
395         DataValueLabelAttributesRole );
396     emit propertiesChanged();
397 }

```

### 9.12.2.60 void AbstractDiagram::setHidden (bool *hidden*) [inherited]

Hide (or unhide, resp.

) all datapoints in the model.

#### Note:

Hidden data are still taken into account by the coordinate plane, so neither the grid nor your axes' ranges will change, when you hide data. For totally removing data from KD Chart's view you can use another approach: e.g. you could define a proxy model on top of your data model, and register the proxy model calling [setModel\(\)](#) instead of registering your real data model.

#### Parameters:

***hidden*** The hidden status to set.

Definition at line 359 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::AbstractDiagram::dataHidden\(\)](#), and [KDChart::DataHiddenRole](#).

```

360 {
361     d->attributesModel->setModelData(
362         qVariantFromValue( hidden ),
363         DataHiddenRole );
364     emit dataHidden();
365 }

```

### 9.12.2.61 void AbstractDiagram::setHidden (int *column*, bool *hidden*) [inherited]

Hide (or unhide, resp.

) a dataset.

#### Note:

Hidden data are still taken into account by the coordinate plane, so neither the grid nor your axes' ranges will change, when you hide data. For totally removing data from KD Chart's view you can use another approach: e.g. you could define a proxy model on top of your data model, and register the proxy model calling [setModel\(\)](#) instead of registering your real data model.

#### Parameters:

***column*** The dataset to set the hidden status for.

***hidden*** The hidden status to set.

Definition at line 350 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::AbstractDiagram::dataHidden\(\)](#), and [KDChart::DataHiddenRole](#).

```

351 {
352     d->attributesModel->setHeaderData(
353         column, Qt::Vertical,
354         qVariantFromValue( hidden ),
355         DataHiddenRole );
356     emit dataHidden();
357 }

```

### 9.12.2.62 void AbstractDiagram::setHidden (const QModelIndex & *index*, bool *hidden*) [inherited]

Hide (or unhide, resp.

) a data cell.

#### Note:

Hidden data are still taken into account by the coordinate plane, so neither the grid nor your axes' ranges will change, when you hide data. For totally removing data from KD Chart's view you can use another approach: e.g. you could define a proxy model on top of your data model, and register the proxy model calling [setModel\(\)](#) instead of registering your real data model.

#### Parameters:

***index*** The datapoint to set the hidden status for.

***hidden*** The hidden status to set.

Definition at line 341 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::AbstractDiagram::dataHidden\(\)](#), and [KDChart::DataHiddenRole](#).

```

342 {
343     d->attributesModel->setData(
344         d->attributesModel->mapFromSource( index ),
345         qVariantFromValue( hidden ),
346         DataHiddenRole );
347     emit dataHidden();
348 }

```

### 9.12.2.63 void AbstractDiagram::setModel (QAbstractItemModel \* *model*) [virtual, inherited]

Associate a model with the diagram.

Reimplemented in [KDChart::AbstractCartesianDiagram](#).

Definition at line 239 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::AttributesModel::initFrom\(\)](#), [KDChart::AbstractDiagram::modelsChanged\(\)](#), and [KDChart::AbstractDiagram::setDataBoundariesDirty\(\)](#).

Referenced by [KDChart::AbstractCartesianDiagram::setModel\(\)](#), and [KDChart::Widget::setType\(\)](#).

```

240 {
241     QAbstractItemView::setModel( newModel );
242     AttributesModel* amodel = new PrivateAttributesModel( newModel, this );
243     amodel->initFrom( d->attributesModel );

```

```

244     d->setAttributesModel(amodel);
245     scheduleDelayedItemsLayout();
246     setDataBoundariesDirty();
247     emit modelsChanged();
248 }

```

#### 9.12.2.64 void AbstractDiagram::setPen (const QPen & *pen*) [inherited]

Set the pen to be used, for painting all datasets in the model.

##### Parameters:

*pen* The pen to use.

Definition at line 753 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DatasetPenRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setModelData().

```

754 {
755     attributesModel()->setModelData(
756         qVariantFromValue( pen ), DatasetPenRole );
757     emit propertiesChanged();
758 }

```

#### 9.12.2.65 void AbstractDiagram::setPen (int *dataset*, const QPen & *pen*) [inherited]

Set the pen to be used, for painting the given dataset.

##### Parameters:

*dataset* The dataset's row in the model.

*pen* The pen to use.

Definition at line 760 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::datasetDimension(), KDChart::DatasetPenRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setHeaderData().

```

761 {
762     if( datasetDimension() > 1 )
763         column *= datasetDimension();
764     attributesModel()->setHeaderData(
765         column, Qt::Vertical,
766         qVariantFromValue( pen ),
767         DatasetPenRole );
768     emit propertiesChanged();
769 }

```

#### 9.12.2.66 void AbstractDiagram::setPen (const QModelIndex & *index*, const QPen & *pen*) [inherited]

Set the pen to be used, for painting the datapoint at the given index.

**Parameters:**

*index* The datapoint's index in the model.

*pen* The pen to use.

Definition at line 743 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::datasetDimension(), KDChart::DatasetPenRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setData().

```

744 {
745     if( datasetDimension() > 1 )
746         return setPen( index.column(), pen );
747     attributesModel()->setData(
748         attributesModel()->mapFromSource( index ),
749         QVariantFromValue( pen ), DatasetPenRole );
750     emit propertiesChanged();
751 }
```

**9.12.2.67 void AbstractDiagram::setPercentMode (bool *percent*)** [inherited]

Definition at line 456 of file KDChartAbstractDiagram.cpp.

References d, and KDChart::AbstractDiagram::propertiesChanged().

Referenced by KDChart::LineDiagram::setType(), and KDChart::BarDiagram::setType().

```

457 {
458     d->percent = percent;
459     emit propertiesChanged();
460 }
```

**9.12.2.68 void AbstractDiagram::setRootIndex (const QModelIndex & *idx*)** [virtual, inherited]

Set the root index in the model, where the diagram starts referencing data for display.

[reimplemented]

Reimplemented in [KDChart::AbstractCartesianDiagram](#).

Definition at line 286 of file KDChartAbstractDiagram.cpp.

References d, and KDChart::AbstractDiagram::setAttributesModelRootIndex().

Referenced by KDChart::AbstractCartesianDiagram::setRootIndex().

```

287 {
288     QAbstractItemView::setRootIndex(idx);
289     setAttributesModelRootIndex( d->attributesModel->mapFromSource(idx) );
290 }
```

**9.12.2.69 void AbstractDiagram::setSelection (const QRect & *rect*, QItemSelectionModel::SelectionFlags *command*)** [virtual, inherited]

[reimplemented]

Definition at line 959 of file KDChartAbstractDiagram.cpp.

References d.

```

960 {
961     const QModelIndexList indexes = d->indexesIn( rect );
962     QItemSelection selection;
963     KDAB_FOREACH( const QModelIndex& index, indexes )
964     {
965         selection.append( QItemSelectionRange( index ) );
966     }
967     selectionModel()->select( selection, command );
968 }
```

### 9.12.2.70 void AbstractDiagram::setUnitPrefix (const QString & *prefix*, Qt::Orientation *orientation*) [inherited]

Sets the unit prefix for all values.

#### Parameters:

*prefix* the prefix to be set

*orientation* the orientation of the axis to set

Definition at line 864 of file KDChartAbstractDiagram.cpp.

References d.

```

865 {
866     d->unitPrefix[ orientation ] = prefix;
867 }
```

### 9.12.2.71 void AbstractDiagram::setUnitPrefix (const QString & *prefix*, int *column*, Qt::Orientation *orientation*) [inherited]

Sets the unit prefix for one value.

#### Parameters:

*prefix* the prefix to be set

*column* the value using that prefix

*orientation* the orientation of the axis to set

Definition at line 854 of file KDChartAbstractDiagram.cpp.

References d.

```

855 {
856     d->unitPrefixMap[ column ][ orientation ]= prefix;
857 }
```

**9.12.2.72 void AbstractDiagram::setUnitSuffix (const QString & *suffix*, Qt::Orientation *orientation*)** [inherited]

Sets the unit suffix for all values.

**Parameters:**

*suffix* the suffix to be set

*orientation* the orientation of the axis to set

Definition at line 885 of file KDChartAbstractDiagram.cpp.

References d.

```
886 {  
887     d->unitSuffix[ orientation ] = suffix;  
888 }
```

**9.12.2.73 void AbstractDiagram::setUnitSuffix (const QString & *suffix*, int *column*, Qt::Orientation *orientation*)** [inherited]

Sets the unit suffix for one value.

**Parameters:**

*suffix* the suffix to be set

*column* the value using that suffix

*orientation* the orientation of the axis to set

Definition at line 875 of file KDChartAbstractDiagram.cpp.

References d.

```
876 {  
877     d->unitSuffixMap[ column ][ orientation ]= suffix;  
878 }
```

**9.12.2.74 QString AbstractDiagram::unitPrefix (Qt::Orientation *orientation*)** const [inherited]

Returns the global unit prefix.

**Parameters:**

*orientation* the orientation of the axis

**Returns:**

the unit prefix

Definition at line 908 of file KDChartAbstractDiagram.cpp.

References d.

```
909 {  
910     return d->unitPrefix[ orientation ];  
911 }
```

### 9.12.2.75 QString AbstractDiagram::unitPrefix (int *column*, Qt::Orientation *orientation*, bool *fallback* = false) const [inherited]

Returns the unit prefix for a special value.

#### Parameters:

*column* the value which's prefix is requested

*orientation* the orientation of the axis

*fallback* if true, the global prefix is return when no specific one is set for that value

#### Returns:

the unit prefix

Definition at line 897 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::itemRowLabels(), and KDChart::CartesianAxis::paintCtx().

```
898 {  
899     if( !fallback || d->unitPrefixMap[ column ].contains( orientation ) )  
900         return d->unitPrefixMap[ column ][ orientation ];  
901     return d->unitPrefix[ orientation ];  
902 }
```

### 9.12.2.76 QString AbstractDiagram::unitSuffix (Qt::Orientation *orientation*) const [inherited]

Returns the global unit suffix.

#### Parameters:

*orientation* the orientation of the axis

#### Returns:

the unit siffix

Definition at line 931 of file KDChartAbstractDiagram.cpp.

References d.

```
932 {  
933     return d->unitSuffix[ orientation ];  
934 }
```

### 9.12.2.77 QString AbstractDiagram::unitSuffix (int *column*, Qt::Orientation *orientation*, bool *fallback* = false) const [inherited]

Returns the unit suffix for a special value.

#### Parameters:

*column* the value which's suffix is requested



*orientation* the orientation of the axis

*fallback* if true, the global suffix is return when no specific one is set for that value

#### Returns:

the unit suffix

Definition at line 920 of file KDChartAbstractDiagram.cpp.

References [d](#).

Referenced by [KDChart::AbstractDiagram::itemRowLabels\(\)](#), and [KDChart::CartesianAxis::paintCtx\(\)](#).

```

921 {
922     if( !fallback || d->unitSuffixMap[ column ].contains( orientation ) )
923         return d->unitSuffixMap[ column ][ orientation ];
924     return d->unitSuffix[ orientation ];
925 }
```

#### 9.12.2.78 void AbstractDiagram::update () const [inherited]

Definition at line 1091 of file KDChartAbstractDiagram.cpp.

References [d](#).

Referenced by [KDChart::AbstractDiagram::doItemsLayout\(\)](#).

```

1092 {
1093     //QDebug("KDChart::AbstractDiagram::update() called");
1094     if( d->plane )
1095         d->plane->update();
1096 }
```

#### 9.12.2.79 void KDChart::AbstractDiagram::useDefaultColors () [inherited]

Set the palette to be used, for painting datasets to the default palette.

#### See also:

[KDChart::Palette](#). FIXME: fold into one usePalette ([KDChart::Palette&](#)) method

Definition at line 974 of file KDChartAbstractDiagram.cpp.

References [d](#), and [KDChart::AttributesModel::PaletteTypeDefault](#).

```

975 {
976     d->attributesModel->setPaletteType( AttributesModel::PaletteTypeDefault );
977 }
```

#### 9.12.2.80 void KDChart::AbstractDiagram::useRainbowColors () [inherited]

Set the palette to be used, for painting datasets to the rainbow palette.

See also:

[KDChart::Palette](#).

Definition at line 984 of file KDChartAbstractDiagram.cpp.

References [d](#), and [KDChart::AttributesModel::PaletteTypeRainbow](#).

```
985 {  
986     d->attributesModel->setPaletteType( AttributesModel::PaletteTypeRainbow );  
987 }
```

**9.12.2.81** `bool AbstractDiagram::usesExternalAttributesModel () const` [virtual, inherited]

Returns whether the diagram is using its own built-in attributes model or an attributes model that was set via [setAttributesModel](#).

See also:

[setAttributesModel](#)

Definition at line 274 of file KDChartAbstractDiagram.cpp.

References [d](#).

```
275 {  
276     return d->usesExternalAttributesModel();  
277 }
```

**9.12.2.82** `void KDChart::AbstractDiagram::useSubduedColors ()` [inherited]

Set the palette to be used, for painting datasets to the subdued palette.

See also:

[KDChart::Palette](#).

Definition at line 979 of file KDChartAbstractDiagram.cpp.

References [d](#), and [KDChart::AttributesModel::PaletteTypeSubdued](#).

```
980 {  
981     d->attributesModel->setPaletteType( AttributesModel::PaletteTypeSubdued );  
982 }
```

**9.12.2.83** `double AbstractDiagram::valueForCell (int row, int column) const` [protected, inherited]

Helper method, retrieving the data value ([DisplayRole](#)) for a given row and column.

Parameters:

*row* The row to query.

*column* The column to query.

**Returns:**

The value of the display role at the given row and column as a double.

**Deprecated**

Definition at line 1085 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModelRootIndex(), and d.

```
1086 {  
1087     return d->attributesModel->data(  
1088         d->attributesModel->index( row, column, attributesModelRootIndex() ) ).toDouble();  
1089 }
```

**9.12.2.84 int AbstractDiagram::verticalOffset () const** [virtual, inherited]

[reimplemented]

Definition at line 953 of file KDChartAbstractDiagram.cpp.

```
954 { return 0; }
```

**9.12.2.85 QRect AbstractDiagram::visualRect (const QModelIndex & *index*) const** [virtual, inherited]

[reimplemented]

Definition at line 937 of file KDChartAbstractDiagram.cpp.

```
938 {  
939     return QRect();  
940 }
```

**9.12.2.86 QRegion AbstractDiagram::visualRegionForSelection (const QItemSelection & *selection*) const** [virtual, inherited]

[reimplemented]

Definition at line 970 of file KDChartAbstractDiagram.cpp.

```
971 { return QRegion(); }
```

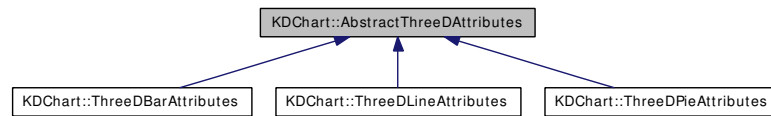
The documentation for this class was generated from the following files:

- [KDChartAbstractTernaryDiagram.h](#)
- [KDChartAbstractTernaryDiagram.cpp](#)

## 9.13 KDChart::AbstractThreeDAttributes Class Reference

```
#include <KDChartAbstractThreeDAttributes.h>
```

Inheritance diagram for KDChart::AbstractThreeDAttributes:



### 9.13.1 Detailed Description

Base class for 3D attributes.

Definition at line 41 of file KDChartAbstractThreeDAttributes.h.

#### Public Member Functions

- [AbstractThreeDAttributes](#) (const [AbstractThreeDAttributes](#) &)
- [AbstractThreeDAttributes](#) ()
- double [depth](#) () const
- bool [isEnabled](#) () const
- bool [operator!=](#) (const [AbstractThreeDAttributes](#) &other) const
- [AbstractThreeDAttributes](#) & [operator=](#) (const [AbstractThreeDAttributes](#) &)
- bool [operator==](#) (const [AbstractThreeDAttributes](#) &) const
- void [setDepth](#) (double depth)
- void [setEnabled](#) (bool enabled)
- double [validDepth](#) () const
- virtual [~AbstractThreeDAttributes](#) ()=0

### 9.13.2 Constructor & Destructor Documentation

#### 9.13.2.1 AbstractThreeDAttributes::AbstractThreeDAttributes ()

Definition at line 46 of file KDChartAbstractThreeDAttributes.cpp.

```

47      : _d( new Private() )
48  {
49  }
```

#### 9.13.2.2 AbstractThreeDAttributes::AbstractThreeDAttributes (const [AbstractThreeDAttributes](#) &)

Definition at line 51 of file KDChartAbstractThreeDAttributes.cpp.

```

52      : _d( new Private( *r.d ) )
53  {
54  }
```

### 9.13.2.3 AbstractThreeDAttributes::~~AbstractThreeDAttributes () [pure virtual]

Definition at line 66 of file KDChartAbstractThreeDAttributes.cpp.

```
67 {  
68     delete _d; _d = 0;  
69 }
```

## 9.13.3 Member Function Documentation

### 9.13.3.1 double AbstractThreeDAttributes::depth () const

Definition at line 103 of file KDChartAbstractThreeDAttributes.cpp.

References d.

Referenced by operator<<(), operator==(), and KDChart::PieDiagram::paint().

```
104 {  
105     return d->depth;  
106 }
```

### 9.13.3.2 bool AbstractThreeDAttributes::isEnabled () const

Definition at line 92 of file KDChartAbstractThreeDAttributes.cpp.

References d.

Referenced by operator<<(), operator==(), KDChart::PieDiagram::paint(), and validDepth().

```
93 {  
94     return d->enabled;  
95 }
```

### 9.13.3.3 bool KDChart::AbstractThreeDAttributes::operator!= (const [AbstractThreeDAttributes](#) & other) const

Definition at line 60 of file KDChartAbstractThreeDAttributes.h.

```
60 { return !operator==(other); }
```

### 9.13.3.4 [AbstractThreeDAttributes](#) & AbstractThreeDAttributes::operator= (const [AbstractThreeDAttributes](#) &)

Definition at line 56 of file KDChartAbstractThreeDAttributes.cpp.

References d.

```
57 {  
58     if( this == &r )  
59         return *this;  
60  
61     *d = *r.d;  
62  
63     return *this;  
64 }
```

### 9.13.3.5 bool AbstractThreeDAttributes::operator==(const AbstractThreeDAttributes &) const

Definition at line 72 of file KDChartAbstractThreeDAttributes.cpp.

References `depth()`, and `isEnabled()`.

Referenced by `KDChart::ThreeDPieAttributes::operator==()`, `KDChart::ThreeDLineAttributes::operator==()`, and `KDChart::ThreeDBarAttributes::operator==()`.

```
73 {  
74     if( isEnabled() == r.isEnabled() &&  
75         depth() == r.depth() )  
76         return true;  
77     else  
78         return false;  
79 }
```

### 9.13.3.6 void AbstractThreeDAttributes::setDepth (double *depth*)

Definition at line 97 of file KDChartAbstractThreeDAttributes.cpp.

References `d`.

```
98 {  
99     d->depth = depth;  
100 }
```

### 9.13.3.7 void AbstractThreeDAttributes::setEnabled (bool *enabled*)

Definition at line 87 of file KDChartAbstractThreeDAttributes.cpp.

References `d`.

```
88 {  
89     d->enabled = enabled;  
90 }
```

### 9.13.3.8 double AbstractThreeDAttributes::validDepth () const

Definition at line 109 of file KDChartAbstractThreeDAttributes.cpp.

References `d`, and `isEnabled()`.

Referenced by `KDChart::Plotter::threeDItemDepth()`, `KDChart::LineDiagram::threeDItemDepth()`, and `KDChart::BarDiagram::threeDItemDepth()`.

```
110 {  
111     return isEnabled() ? d->depth : 0.0;  
112 }
```

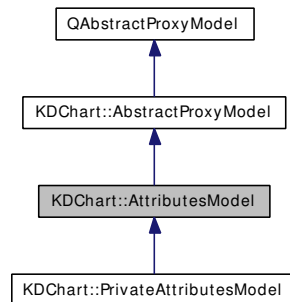
The documentation for this class was generated from the following files:

- [KDChartAbstractThreeDAttributes.h](#)
- [KDChartAbstractThreeDAttributes.cpp](#)

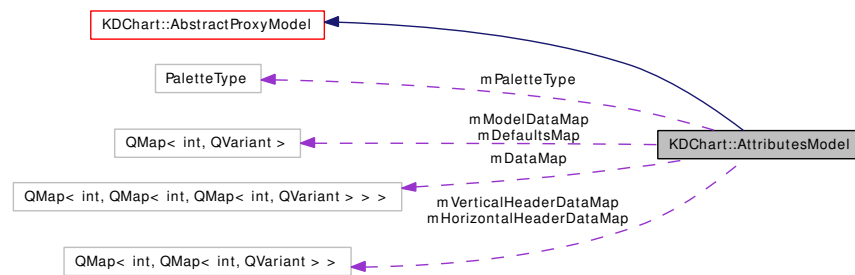
## 9.14 KDChart::AttributesModel Class Reference

```
#include <KDChartAttributesModel.h>
```

Inheritance diagram for KDChart::AttributesModel:



Collaboration diagram for KDChart::AttributesModel:



### 9.14.1 Detailed Description

A proxy model used for storing attributes.

Definition at line 43 of file `KDChartAttributesModel.h`.

### Public Types

- enum `PaletteType` {  
`PaletteTypeDefault` = 0,  
`PaletteTypeRainbow` = 1,  
`PaletteTypeSubdued` = 2 }

### Signals

- void `attributesChanged` (const `QModelIndex` &, const `QModelIndex` &)

## Public Member Functions

- [AttributesModel](#) (QAbstractItemModel \*model, [QObject](#) \*parent=0)
- int [columnCount](#) (const QModelIndex &) const  
*[reimplemented]*
- bool [compare](#) (const [AttributesModel](#) \*other) const
- bool [compareAttributes](#) (int role, const QVariant &a, const QVariant &b) const
- QVariant [data](#) (const QModelIndex &, int role=Qt::DisplayRole) const  
*[reimplemented]*
- QVariant [data](#) (int column, int role) const  
*Returns the data that were specified at per column level, or the globally set data, or the default data, or QVariant().*
- QVariant [data](#) (int role) const  
*Returns the data that were specified at global level, or the default data, or QVariant().*
- QVariant [headerData](#) (int section, Qt::Orientation orientation, int role=Qt::DisplayRole) const  
*[reimplemented]*
- QModelIndex [index](#) (int row, int col, const QModelIndex &index) const
- void [initFrom](#) (const [AttributesModel](#) \*other)
- bool [isKnownAttributesRole](#) (int role) const  
*Returns whether the given role corresponds to one of the known internally used ones.*
- QModelIndex [mapFromSource](#) (const QModelIndex &sourceIndex) const
- QModelIndex [mapToSource](#) (const QModelIndex &proxyIndex) const
- QVariant [modelData](#) (int role) const
- [PaletteType](#) [paletteType](#) () const
- QModelIndex [parent](#) (const QModelIndex &index) const
- bool [resetData](#) (const QModelIndex &index, int role=Qt::DisplayRole)  
*Remove any explicit attributes settings that might have been specified before.*
- bool [resetHeaderData](#) (int section, Qt::Orientation orientation, int role=Qt::DisplayRole)  
*Remove any explicit attributes settings that might have been specified before.*
- int [rowCount](#) (const QModelIndex &) const  
*[reimplemented]*
- bool [setData](#) (const QModelIndex &index, const QVariant &value, int role=Qt::DisplayRole)  
*[reimplemented]*
- void [setDefaultForRole](#) (int role, const QVariant &value)  
*Define the default value for a certain role.*
- bool [setHeaderData](#) (int section, Qt::Orientation orientation, const QVariant &value, int role=Qt::DisplayRole)  
*[reimplemented]*



- bool [setModelData](#) (const QVariant value, int role)
- void [setPaletteType](#) ([PaletteType](#) type)  
*Sets the palettetype used by this attributesmodel.*
- void [setSourceModel](#) (QAbstractItemModel \*sourceModel)  
*[reimplemented]*
- [~AttributesModel](#) ()

## Protected Member Functions

- const QMap< int, QMap< int, QMap< int, QVariant > > > [dataMap](#) () const  
*needed for serialization*
- const QMap< int, QMap< int, QVariant > > [horizontalHeaderDataMap](#) () const  
*needed for serialization*
- const QMap< int, QVariant > [modelDataMap](#) () const  
*needed for serialization*
- void [setDataMap](#) (const QMap< int, QMap< int, QMap< int, QVariant > > > map)  
*needed for serialization*
- void [setHorizontalHeaderDataMap](#) (const QMap< int, QMap< int, QVariant > > map)  
*needed for serialization*
- void [setModelDataMap](#) (const QMap< int, QVariant > map)  
*needed for serialization*
- void [setVerticalHeaderDataMap](#) (const QMap< int, QMap< int, QVariant > > map)  
*needed for serialization*
- const QMap< int, QMap< int, QVariant > > [verticalHeaderDataMap](#) () const  
*needed for serialization*

## 9.14.2 Member Enumeration Documentation

### 9.14.2.1 enum [KDChart::AttributesModel::PaletteType](#)

Enumerator:

*PaletteTypeDefault*  
*PaletteTypeRainbow*  
*PaletteTypeSubdued*

Definition at line 50 of file KDChartAttributesModel.h.

```

50         {
51             PaletteTypeDefault = 0,
52             PaletteTypeRainbow = 1,
53             PaletteTypeSubdued = 2
54         };

```

### 9.14.3 Constructor & Destructor Documentation

#### 9.14.3.1 `AttributesModel::AttributesModel (QAbstractItemModel * model, QObject * parent = 0)` [explicit]

Definition at line 57 of file `KDChartAttributesModel.cpp`.

References `KDChart::DataValueLabelAttributesRole`, `KDChart::DataValueAttributes::defaultAttributesAsVariant()`, `setDefaultForRole()`, and `setSourceModel()`.

```

58 : AbstractProxyModel( parent ),
59   mPaletteType( PaletteTypeDefault )
60 {
61     setSourceModel(model);
62     setDefaultForRole( KDChart::DataValueLabelAttributesRole,
63                       DataValueAttributes::defaultAttributesAsVariant() );
64 }
```

#### 9.14.3.2 `AttributesModel::~AttributesModel ()`

Definition at line 66 of file `KDChartAttributesModel.cpp`.

```

67 {
68 }
```

### 9.14.4 Member Function Documentation

#### 9.14.4.1 `void KDChart::AttributesModel::attributesChanged (const QModelIndex &, const QModelIndex &) [signal]`

Referenced by `setData()`, `setHeaderData()`, and `setModelData()`.

#### 9.14.4.2 `int AttributesModel::columnCount (const QModelIndex &) const`

[reimplemented]

Definition at line 523 of file `KDChartAttributesModel.cpp`.

References `KDChart::AbstractProxyModel::mapToSource()`.

Referenced by `KDChart::AbstractDiagram::datasetBrushes()`, `KDChart::AbstractDiagram::datasetLabels()`, `KDChart::AbstractDiagram::datasetMarkers()`, `KDChart::AbstractDiagram::datasetPens()`, and `setModelData()`.

```

524 {
525     if ( sourceModel() ) {
526         return sourceModel()->columnCount( mapToSource(index) );
527     } else {
528         return 0;
529     }
530 }
```

**9.14.4.3 bool AttributesModel::compare (const [AttributesModel](#) \* other) const**

Definition at line 83 of file KDChartAttributesModel.cpp.

References `compareAttributes()`, `mDataMap`, `mHorizontalHeaderDataMap`, `mModelDataMap`, `mVerticalHeaderDataMap`, and `paletteType()`.

Referenced by `KDChart::AbstractDiagram::compare()`.

```

84 {
85     if( other == this ) return true;
86     if( ! other ){
87         //qDebug() << "AttributesModel::compare() cannot compare to Null pointer";
88         return false;
89     }
90
91     {
92         if (mDataMap.count() != other->mDataMap.count()){
93             //qDebug() << "AttributesModel::compare() dataMap have different sizes";
94             return false;
95         }
96         QMap<int, QMap<int, QMap<int, QVariant> > >::const_iterator itA = mDataMap.constBegin();
97         QMap<int, QMap<int, QMap<int, QVariant> > >::const_iterator itB = other->mDataMap.constBegin();
98         while (itA != mDataMap.constEnd()) {
99             if ((*itA).count() != (*itB).count()){
100                 //qDebug() << "AttributesModel::compare() dataMap/map have different sizes";
101                 return false;
102             }
103             QMap<int, QMap<int, QVariant> >::const_iterator it2A = (*itA).constBegin();
104             QMap<int, QMap<int, QVariant> >::const_iterator it2B = (*itB).constBegin();
105             while (it2A != itA->constEnd()) {
106                 if ((*it2A).count() != (*it2B).count()){
107                     //qDebug() << "AttributesModel::compare() dataMap/map/map have different sizes:"
108                     // << (*it2A).count() << (*it2B).count();
109                     return false;
110                 }
111                 QMap<int, QVariant>::const_iterator it3A = (*it2A).constBegin();
112                 QMap<int, QVariant>::const_iterator it3B = (*it2B).constBegin();
113                 while (it3A != it2A->constEnd()) {
114                     if ( it3A.key() != it3B.key() ){
115                         //qDebug( "AttributesModel::compare()\n"
116                         //      "    dataMap[%i, %i] values have different types.  A: %x  B: %x",
117                         //      itA.key(), it2A.key(), it3A.key(), it3B.key());
118                         return false;
119                     }
120                     if ( ! compareAttributes( it3A.key(), it3A.value(), it3B.value() ) ){
121                         //qDebug( "AttributesModel::compare()\n"
122                         //      "    dataMap[%i, %i] values are different. Role: %x", itA.key(), it2A
123                         return false;
124                     }
125                     ++it3A;
126                     ++it3B;
127                 }
128                 ++it2A;
129                 ++it2B;
130             }
131             ++itA;
132             ++itB;
133         }
134     }
135
136     if (mHorizontalHeaderDataMap.count() != other->mHorizontalHeaderDataMap.count()){
137         //qDebug() << "AttributesModel::compare() horizontalHeaderDataMap have different sizes";
138         return false;
139     }
140     QMap<int, QMap<int, QVariant> >::const_iterator itA = mHorizontalHeaderDataMap.constBegin();
141     QMap<int, QMap<int, QVariant> >::const_iterator itB = other->mHorizontalHeaderDataMap.constBeg

```

```

142     while (itA != mHorizontalHeaderDataMap.constEnd()) {
143         if ((*itA).count() != (*itB).count()){
144             //qDebug() << "AttributesModel::compare() horizontalHeaderDataMap/map have different sizes";
145             return false;
146         }
147         QMap<int, QVariant>::const_iterator it2A = (*itA).constBegin();
148         QMap<int, QVariant>::const_iterator it2B = (*itB).constBegin();
149         while (it2A != itA->constEnd()) {
150             if ( it2A.key() != it2B.key() ){
151                 //qDebug( "AttributesModel::compare()\n"
152                 //      " horizontalHeaderDataMap[ %i ] values have different types. A: %x B: %x",
153                 //      itA.key(), it2A.key(), it2B.key());
154                 return false;
155             }
156             if ( ! compareAttributes( it2A.key(), it2A.value(), it2B.value() ) ){
157                 //qDebug( "AttributesModel::compare()\n"
158                 //      " horizontalHeaderDataMap[ %i ] values are different. Role: %x", itA.key(), itB.key());
159                 return false;
160             }
161             ++it2A;
162             ++it2B;
163         }
164         ++itA;
165         ++itB;
166     }
167 }
168 {
169     if (mVerticalHeaderDataMap.count() != other->mVerticalHeaderDataMap.count()){
170         //qDebug() << "AttributesModel::compare() verticalHeaderDataMap have different sizes";
171         return false;
172     }
173     QMap<int, QMap<int, QVariant> >::const_iterator itA = mVerticalHeaderDataMap.constBegin();
174     QMap<int, QMap<int, QVariant> >::const_iterator itB = other->mVerticalHeaderDataMap.constBegin();
175     while (itA != mVerticalHeaderDataMap.constEnd()) {
176         if ((*itA).count() != (*itB).count()){
177             //qDebug() << "AttributesModel::compare() verticalHeaderDataMap/map have different sizes";
178             return false;
179         }
180         QMap<int, QVariant>::const_iterator it2A = (*itA).constBegin();
181         QMap<int, QVariant>::const_iterator it2B = (*itB).constBegin();
182         while (it2A != itA->constEnd()) {
183             if ( it2A.key() != it2B.key() ){
184                 //qDebug( "AttributesModel::compare()\n"
185                 //      " verticalHeaderDataMap[ %i ] values have different types. A: %x B: %x",
186                 //      itA.key(), it2A.key(), it2B.key());
187                 return false;
188             }
189             if ( ! compareAttributes( it2A.key(), it2A.value(), it2B.value() ) ){
190                 //qDebug( "AttributesModel::compare()\n"
191                 //      " verticalHeaderDataMap[ %i ] values are different. Role: %x", itA.key(), itB.key());
192                 return false;
193             }
194             ++it2A;
195             ++it2B;
196         }
197         ++itA;
198         ++itB;
199     }
200 }
201 {
202     if (mModelDataMap.count() != other->mModelDataMap.count()){
203         //qDebug() << "AttributesModel::compare() modelDataMap have different sizes:" << mModelDataMap.count();
204         return false;
205     }
206     QMap<int, QVariant>::const_iterator itA = mModelDataMap.constBegin();
207     QMap<int, QVariant>::const_iterator itB = other->mModelDataMap.constBegin();
208     while (itA != mModelDataMap.constEnd()) {

```

```

209         if ( itA.key() != itB.key() ){
210             //qDebug( "AttributesModel::compare()\n"
211             //      "      modelDataMap values have different types.  A: %x  B: %x",
212             //      itA.key(), itB.key());
213             return false;
214         }
215         if ( ! compareAttributes( itA.key(), itA.value(), itB.value() ) ){
216             //qDebug( "AttributesModel::compare()\n"
217             //      "      modelDataMap values are different. Role: %x", itA.key() );
218             return false;
219         }
220         ++itA;
221         ++itB;
222     }
223 }
224 if (paletteType() != other->paletteType()){
225     //qDebug() << "AttributesModel::compare() palette types are different";
226     return false;
227 }
228 return true;
229 }

```

#### 9.14.4.4 bool AttributesModel::compareAttributes (int role, const QVariant &a, const QVariant &b) const

Definition at line 231 of file KDChartAttributesModel.cpp.

References KDChart::BarAttributesRole, KDChart::DataHiddenRole, KDChart::DatasetBrushRole, KDChart::DatasetPenRole, KDChart::DataValueLabelAttributesRole, isKnownAttributesRole(), KDChart::LineAttributesRole, KDChart::PieAttributesRole, KDChart::ThreeDAttributesRole, KDChart::ThreeDBarAttributesRole, KDChart::ThreeDLineAttributesRole, KDChart::ThreeDPieAttributesRole, and KDChart::ValueTrackerAttributesRole.

Referenced by compare().

```

233 {
234     if( isKnownAttributesRole( role ) ){
235         switch( role ) {
236             case DataValueLabelAttributesRole:
237                 return (qVariantValue<DataValueAttributes>( a ) ==
238                     qVariantValue<DataValueAttributes>( b ));
239             case DatasetBrushRole:
240                 return (qVariantValue<QBrush>( a ) ==
241                     qVariantValue<QBrush>( b ));
242             case DatasetPenRole:
243                 return (qVariantValue<QPen>( a ) ==
244                     qVariantValue<QPen>( b ));
245             case ThreeDAttributesRole:
246                 // As of yet there is no ThreeDAttributes class,
247                 // and the AbstractThreeDAttributes class is pure virtual,
248                 // so we ignore this role for now.
249                 // (khz, 04.04.2007)
250                 /*
251                 return (qVariantValue<ThreeDAttributes>( a ) ==
252                     qVariantValue<ThreeDAttributes>( b ));
253                 */
254                 break;
255             case LineAttributesRole:
256                 return (qVariantValue<LineAttributes>( a ) ==
257                     qVariantValue<LineAttributes>( b ));
258             case ThreeDLineAttributesRole:
259                 return (qVariantValue<ThreeDLineAttributes>( a ) ==
260                     qVariantValue<ThreeDLineAttributes>( b ));

```

```

261         case BarAttributesRole:
262             return (qVariantValue<BarAttributes>( a ) ==
263                     qVariantValue<BarAttributes>( b ));
264         case ThreeDBarAttributesRole:
265             return (qVariantValue<ThreeDBarAttributes>( a ) ==
266                     qVariantValue<ThreeDBarAttributes>( b ));
267         case PieAttributesRole:
268             return (qVariantValue<PieAttributes>( a ) ==
269                     qVariantValue<PieAttributes>( b ));
270         case ThreeDPieAttributesRole:
271             return (qVariantValue<ThreeDPieAttributes>( a ) ==
272                     qVariantValue<ThreeDPieAttributes>( b ));
273         case ValueTrackerAttributesRole:
274             return (qVariantValue<ValueTrackerAttributes>( a ) ==
275                     qVariantValue<ValueTrackerAttributes>( b ));
276         case DataHiddenRole:
277             return (qVariantValue<bool>( a ) ==
278                     qVariantValue<bool>( b ));
279         default:
280             Q_ASSERT( false ); // all of our own roles need to be handled
281             break;
282     }
283 }else{
284     return (a == b);
285 }
286 return true;
287 }

```

#### 9.14.4.5 QVariant AttributesModel::data (const QModelIndex &, int role = Qt::DisplayRole) const

[reimplemented]

Definition at line 373 of file KDChartAttributesModel.cpp.

References data(), dataMap(), and KDChart::AbstractProxyModel::mapToSource().

```

374 {
375     //qDebug() << "AttributesModel::data(" << index << role << ")";
376     if( index.isValid() ) {
377         Q_ASSERT( index.model() == this );
378     }
379
380     if( sourceModel() == 0 )
381         return QVariant();
382
383     if( index.isValid() )
384     {
385         const QVariant sourceData = sourceModel()->data( mapToSource(index), role );
386         if( sourceData.isValid() )
387             return sourceData;
388     }
389
390     // check if we are storing a value for this role at this cell index
391     if( mDataMap.contains( index.column() ) )
392     {
393         const QMap< int, QMap< int, QVariant > >& colDataMap = mDataMap[ index.column() ];
394         if( colDataMap.contains( index.row() ) )
395         {
396             const QMap< int, QVariant >& dataMap = colDataMap[ index.row() ];
397             if( dataMap.contains( role ) )
398             {
399                 const QVariant v = dataMap[ role ];
400                 if( v.isValid() )

```

```

401             return v;
402         }
403     }
404 }
405 // check if there is something set for the column (dataset), or at global level
406 if( index.isValid() )
407     return data( index.column(), role ); // includes automatic fallback to default
408
409 return QVariant();
410 }

```

#### 9.14.4.6 QVariant AttributesModel::data (int *column*, int *role*) const

Returns the data that were specified at per column level, or the globally set data, or the default data, or QVariant().

Definition at line 357 of file KDChartAttributesModel.cpp.

References data(), headerData(), and isKnownAttributesRole().

```

358 {
359     if ( isKnownAttributesRole( role ) ) {
360         // check if there is something set for the column (dataset)
361         QVariant v;
362         v = headerData( column, Qt::Vertical, role );
363
364         // check if there is something set at global level
365         if ( !v.isValid() )
366             v = data( role ); // includes automatic fallback to default
367         return v;
368     }
369     return QVariant();
370 }

```

#### 9.14.4.7 QVariant AttributesModel::data (int *role*) const

Returns the data that were specified at global level, or the default data, or QVariant().

Definition at line 339 of file KDChartAttributesModel.cpp.

References isKnownAttributesRole(), and modelData().

Referenced by KDChart::AbstractDiagram::brush(), data(), KDChart::AbstractDiagram::dataValueAttributes(), KDChart::AbstractDiagram::isHidden(), and KDChart::AbstractDiagram::pen().

```

340 {
341     if ( isKnownAttributesRole( role ) ) {
342         // check if there is something set at global level
343         QVariant v = modelData( role );
344
345         // else return the default setting, if any
346         if ( !v.isValid() )
347             v = defaultsForRole( role );
348         return v;
349     }
350     return QVariant();
351 }

```

#### 9.14.4.8 `const QMap< int, QMap< int, QMap< int, QVariant > > > AttributesModel::dataMap () const` [protected]

needed for serialization

Definition at line 564 of file KDChartAttributesModel.cpp.

Referenced by `data()`, `headerData()`, `setData()`, and `setHeaderData()`.

```
565 {
566     return mDataMap;
567 }
```

#### 9.14.4.9 `QVariant AttributesModel::headerData (int section, Qt::Orientation orientation, int role = Qt::DisplayRole) const`

[reimplemented]

Definition at line 290 of file KDChartAttributesModel.cpp.

References `dataMap()`, `KDChart::DatasetBrushRole`, `KDChart::DatasetPenRole`, `KDChart::Palette::defaultPalette()`, `KDChart::Palette::getBrush()`, `modelData()`, `paletteType()`, `PaletteTypeDefault`, `PaletteTypeRainbow`, `PaletteTypeSubdued`, `KDChart::Palette::rainbowPalette()`, and `KDChart::Palette::subduedPalette()`.

Referenced by `data()`, `KDChart::AbstractDiagram::datasetLabels()`, and `setHeaderData()`.

```
293 {
294     QVariant sourceData = sourceModel()->headerData( section, orientation, role );
295     if ( sourceData.isValid() ) return sourceData;
296     // the source model didn't have data set, let's use our stored values
297     const QMap<int, QMap<int, QVariant> >& map = orientation == Qt::Horizontal ? mHorizontalHeaderDataMa
298     if ( map.contains( section ) ) {
299         const QMap<int, QVariant> &dataMap = map[ section ];
300         if ( dataMap.contains( role ) ) {
301             return dataMap[ role ];
302         }
303     }
304     // Default values if nothing else matches
305     switch ( role ) {
306     case Qt::DisplayRole:
307         return QLatin1String( orientation == Qt::Vertical ? "Series " : "Item " ) + QString::number( se
308     case KDChart::DatasetBrushRole: {
309         if ( paletteType() == PaletteTypeSubdued )
310             return Palette::subduedPalette().getBrush( section );
311         else if ( paletteType() == PaletteTypeRainbow )
312             return Palette::rainbowPalette().getBrush( section );
313         else if ( paletteType() == PaletteTypeDefault )
314             return Palette::defaultPalette().getBrush( section );
315         else
316             qWarning("Unknown type of fallback palette!");
317     }
318     case KDChart::DatasetPenRole: {
319         // default to the color set for the brush (or it's defaults)
320         // but only if no per model override was set
321         if ( !modelData( role ).isValid() ) {
322             QBrush brush = qVariantValue<QBrush>( headerData( section, orientation, DatasetBrushRole ) )
323             return QPen( brush.color() );
324         }
325     }
326 }
```



```

328     default:
329         break;
330     }
331
332     return QVariant();
333 }

```

#### 9.14.4.10 `const QMap< int, QMap< int, QVariant > > AttributesModel::horizontalHeaderDataMap () const` [protected]

needed for serialization

Definition at line 569 of file KDChartAttributesModel.cpp.

```

570 {
571     return mHorizontalHeaderDataMap;
572 }

```

#### 9.14.4.11 `QModelIndex KDChart::AbstractProxyModel::index (int row, int col, const QModelIndex & index) const` [inherited]

[reimplemented]

Definition at line 84 of file KDChartAbstractProxyModel.cpp.

References KDChart::AbstractProxyModel::mapFromSource(), and KDChart::AbstractProxyModel::mapToSource().

Referenced by setHeaderData(), and setModelData().

```

85 {
86     Q_ASSERT(sourceModel());
87     return mapFromSource(sourceModel()->index( row, col, mapToSource(index) ));
88 }

```

#### 9.14.4.12 `void AttributesModel::initFrom (const AttributesModel * other)`

Definition at line 70 of file KDChartAttributesModel.cpp.

References mDataMap, mDefaultsMap, mHorizontalHeaderDataMap, mModelDataMap, mVerticalHeaderDataMap, paletteType(), and setPaletteType().

Referenced by KDChart::AbstractDiagram::setModel().

```

71 {
72     if( other == this || ! other ) return;
73
74     mDataMap = other->mDataMap;
75     mHorizontalHeaderDataMap = other->mHorizontalHeaderDataMap;
76     mVerticalHeaderDataMap = other->mVerticalHeaderDataMap;
77     mModelDataMap = other->mModelDataMap;
78     mDefaultsMap = other->mDefaultsMap;
79
80     setPaletteType( other->paletteType() );
81 }

```

#### 9.14.4.13 bool AttributesModel::isKnownAttributesRole (int *role*) const

Returns whether the given role corresponds to one of the known internally used ones.

Definition at line 413 of file KDChartAttributesModel.cpp.

References KDChart::BarAttributesRole, KDChart::DataHiddenRole, KDChart::DatasetBrushRole, KDChart::DatasetPenRole, KDChart::DataValueLabelAttributesRole, KDChart::LineAttributesRole, KDChart::PieAttributesRole, KDChart::ThreeDAttributesRole, KDChart::ThreeDBarAttributesRole, KDChart::ThreeDLineAttributesRole, KDChart::ThreeDPieAttributesRole, and KDChart::ValueTrackerAttributesRole.

Referenced by compareAttributes(), data(), setData(), and setHeaderData().

```

414 {
415     bool oneOfOurs = false;
416     switch( role ) {
417         // fallthrough intended
418         case DataValueLabelAttributesRole:
419         case DatasetBrushRole:
420         case DatasetPenRole:
421         case ThreeDAttributesRole:
422         case LineAttributesRole:
423         case ThreeDLineAttributesRole:
424         case BarAttributesRole:
425         case ThreeDBarAttributesRole:
426         case PieAttributesRole:
427         case ThreeDPieAttributesRole:
428         case ValueTrackerAttributesRole:
429         case DataHiddenRole:
430         oneOfOurs = true;
431     default:
432         break;
433     }
434     return oneOfOurs;
435 }
```

#### 9.14.4.14 QModelIndex KDChart::AbstractProxyModel::mapFromSource (const QModelIndex & *sourceIndex*) const [inherited]

[reimplemented]

Definition at line 52 of file KDChartAbstractProxyModel.cpp.

Referenced by KDChart::AbstractProxyModel::index(), and KDChart::AbstractProxyModel::parent().

```

53 {
54     if ( !sourceIndex.isValid() )
55         return QModelIndex();
56     //qDebug() << "sourceIndex.model()="<<sourceIndex.model();
57     //qDebug() << "model()="<<sourceModel();
58     Q_ASSERT( sourceIndex.model() == sourceModel() );
59
60     // Create an index that preserves the internal pointer from the source;
61     // this way AbstractProxyModel preserves the structure of the source model
62     return createIndex( sourceIndex.row(), sourceIndex.column(), sourceIndex.internalPointer() );
63 }
```

#### 9.14.4.15 QModelIndex KDChart::AbstractProxyModel::mapToSource (const QModelIndex & proxyIndex) const [inherited]

[reimplemented]

Definition at line 65 of file KDChartAbstractProxyModel.cpp.

Referenced by columnCount(), data(), KDChart::AbstractProxyModel::index(), KDChart::AbstractProxyModel::parent(), rowCount(), and setData().

```

66 {
67     if ( !proxyIndex.isValid() )
68         return QModelIndex();
69     if( proxyIndex.model() != this )
70         qDebug() << proxyIndex.model() << this;
71     Q_ASSERT( proxyIndex.model() == this );
72     // So here we need to create a source index which holds that internal pointer.
73     // No way to pass it to sourceModel()->index... so we have to do the ugly way:
74     QModelIndex sourceIndex;
75     KDPrivateModelIndex* hack = reinterpret_cast<KDPrivateModelIndex*>(&sourceIndex);
76     hack->r = proxyIndex.row();
77     hack->c = proxyIndex.column();
78     hack->p = proxyIndex.internalPointer();
79     hack->m = sourceModel();
80     Q_ASSERT( sourceIndex.isValid() );
81     return sourceIndex;
82 }
```

#### 9.14.4.16 QVariant KDChart::AttributesModel::modelData (int role) const

Definition at line 509 of file KDChartAttributesModel.cpp.

Referenced by data(), KDChart::AbstractDiagram::dataValueAttributes(), headerData(), and KDChart::AbstractDiagram::isHidden().

```

510 {
511     return mModelDataMap.value( role, QVariant() );
512 }
```

#### 9.14.4.17 const QMap< int, QVariant > AttributesModel::modelDataMap () const [protected]

needed for serialization

Definition at line 579 of file KDChartAttributesModel.cpp.

```

580 {
581     return mModelDataMap;
582 }
```

#### 9.14.4.18 AttributesModel::PaletteType AttributesModel::paletteType () const

Definition at line 493 of file KDChartAttributesModel.cpp.

Referenced by compare(), headerData(), and initFrom().

```

494 {
495     return mPaletteType;
496 }

```

#### 9.14.4.19 QModelIndex KDChart::AbstractProxyModel::parent (const QModelIndex & *index*) const [inherited]

[reimplemented]

Definition at line 90 of file KDChartAbstractProxyModel.cpp.

References KDChart::AbstractProxyModel::mapFromSource(), and KDChart::AbstractProxyModel::mapToSource().

```

91 {
92     Q_ASSERT(sourceModel());
93     return mapFromSource(sourceModel()->parent( mapToSource(index) ));
94 }

```

#### 9.14.4.20 bool AttributesModel::resetData (const QModelIndex & *index*, int *role* = Qt::DisplayRole)

Remove any explicit attributes settings that might have been specified before.

Definition at line 457 of file KDChartAttributesModel.cpp.

References setData().

```

458 {
459     return setData ( index, QVariant(), role );
460 }

```

#### 9.14.4.21 bool AttributesModel::resetHeaderData (int *section*, Qt::Orientation *orientation*, int *role* = Qt::DisplayRole)

Remove any explicit attributes settings that might have been specified before.

Definition at line 483 of file KDChartAttributesModel.cpp.

References setHeaderData().

```

484 {
485     return setHeaderData ( section, orientation, QVariant(), role );
486 }

```

#### 9.14.4.22 int AttributesModel::rowCount (const QModelIndex &) const

[reimplemented]

Definition at line 514 of file KDChartAttributesModel.cpp.

References KDChart::AbstractProxyModel::mapToSource().

Referenced by KDChart::AbstractDiagram::itemRowLabels(), setHeaderData(), and setModelData().

```

515 {
516     if ( sourceModel() ) {
517         return sourceModel()->rowCount( mapToSource(index) );
518     } else {
519         return 0;
520     }
521 }

```

#### 9.14.4.23 bool AttributesModel::setData (const QModelIndex & *index*, const QVariant & *value*, int *role* = Qt::DisplayRole)

[reimplemented]

Definition at line 443 of file KDChartAttributesModel.cpp.

References attributesChanged(), dataMap(), isKnownAttributesRole(), and KDChart::AbstractProxyModel::mapToSource().

Referenced by resetData(), KDChart::BarDiagram::setBarAttributes(), KDChart::AbstractDiagram::setBrush(), and KDChart::AbstractDiagram::setPen().

```

444 {
445     if ( !isKnownAttributesRole( role ) ) {
446         return sourceModel()->setData( mapToSource(index), value, role );
447     } else {
448         QMap< int, QMap< int, QVariant> > &colDataMap = mDataMap[ index.column() ];
449         QMap<int, QVariant> &dataMap = colDataMap[ index.row() ];
450         //qDebug() << "AttributesModel::setData" <<"role" << role << "value" << value;
451         dataMap.insert( role, value );
452         emit attributesChanged( index, index );
453         return true;
454     }
455 }

```

#### 9.14.4.24 void AttributesModel::setDataMap (const QMap< int, QMap< int, QMap< int, QVariant > > > *map*) [protected]

needed for serialization

Definition at line 585 of file KDChartAttributesModel.cpp.

```

586 {
587     mDataMap = map;
588 }

```

#### 9.14.4.25 void AttributesModel::setDefaultForRole (int *role*, const QVariant & *value*)

Define the default value for a certain role.

Passing a default-constructed QVariant is equivalent to removing the default.

Definition at line 605 of file KDChartAttributesModel.cpp.

Referenced by AttributesModel(), and KDChart::TernaryLineDiagram::TernaryLineDiagram().

```

606 {
607     if ( value.isValid() ) {

```

```

608         mDefaultsMap.insert( role, value );
609     } else {
610         // erase the possibly existing value to not let the map grow:
611         QMap<int, QVariant>::iterator it = mDefaultsMap.find( role );
612         if ( it != mDefaultsMap.end() ) {
613             mDefaultsMap.erase( it );
614         }
615     }
616
617     Q_ASSERT( defaultsForRole( role ) == value );
618 }

```

#### 9.14.4.26 bool AttributesModel::setHeaderData (int section, Qt::Orientation orientation, const QVariant & value, int role = Qt::DisplayRole)

[reimplemented]

Definition at line 462 of file KDChartAttributesModel.cpp.

References attributesChanged(), dataMap(), headerData(), KDChart::AbstractProxyModel::index(), isKnownAttributesRole(), and rowCount().

Referenced by resetHeaderData(), KDChart::AbstractDiagram::setBrush(), and KDChart::AbstractDiagram::setPen().

```

464 {
465     if( sourceModel() != 0 && headerData( section, orientation, role ) == value )
466         return true;
467     if ( !isKnownAttributesRole( role ) ) {
468         return sourceModel()->setHeaderData( section, orientation, value, role );
469     } else {
470         QMap<int, QMap<int, QVariant> > &sectionDataMap
471             = orientation == Qt::Horizontal ? mHorizontalHeaderDataMap : mVerticalHeaderDataMap;
472         QMap<int, QVariant> &dataMap = sectionDataMap[ section ];
473         dataMap.insert( role, value );
474         if( sourceModel() ){
475             emit attributesChanged( index( 0, section, QModelIndex() ),
476                                     index( rowCount( QModelIndex() ), section, QModelIndex() ) );
477             emit headerDataChanged( orientation, section, section );
478         }
479         return true;
480     }
481 }

```

#### 9.14.4.27 void AttributesModel::setHorizontalHeaderDataMap (const QMap< int, QMap< int, QVariant > > map) [protected]

needed for serialization

Definition at line 590 of file KDChartAttributesModel.cpp.

```

591 {
592     mHorizontalHeaderDataMap = map;
593 }

```

#### 9.14.4.28 bool KDChart::AttributesModel::setModelData (const QVariant value, int role)

Definition at line 498 of file KDChartAttributesModel.cpp.

References `attributesChanged()`, `columnCount()`, `KDChart::AbstractProxyModel::index()`, and `rowCount()`.

Referenced by `KDChart::AbstractDiagram::setBrush()`, and `KDChart::AbstractDiagram::setPen()`.

```

499 {
500     mModelDataMap.insert( role, value );
501     if( sourceModel() ){
502         emit attributesChanged( index( 0, 0, QModelIndex() ),
503                                 index( rowCount( QModelIndex() ),
504                                     columnCount( QModelIndex() ), QModelIndex() ) );
505     }
506     return true;
507 }

```

#### 9.14.4.29 void AttributesModel::setModelDataMap (const QMap< int, QVariant > map) [protected]

needed for serialization

Definition at line 600 of file `KDChartAttributesModel.cpp`.

```

601 {
602     mModelDataMap = map;
603 }

```

#### 9.14.4.30 void AttributesModel::setPaletteType (PaletteType type)

Sets the palettetype used by this attributesmodel.

Definition at line 488 of file `KDChartAttributesModel.cpp`.

Referenced by `initFrom()`.

```

489 {
490     mPaletteType = type;
491 }

```

#### 9.14.4.31 void AttributesModel::setSourceModel (QAbstractItemModel \* sourceModel)

[reimplemented]

Definition at line 532 of file `KDChartAttributesModel.cpp`.

Referenced by `AttributesModel()`.

```

533 {
534     if( this->sourceModel() != 0 )
535     {
536         disconnect( this->sourceModel(), SIGNAL( dataChanged( const QModelIndex&, const QModelIndex&))
537                     this, SIGNAL( dataChanged( const QModelIndex&, const QModelIndex&)) );
538         disconnect( this->sourceModel(), SIGNAL( rowsInserted( const QModelIndex&, int, int ) ) ,
539                     this, SIGNAL( rowsInserted( const QModelIndex&, int, int ) ) );
540         disconnect( this->sourceModel(), SIGNAL( rowsRemoved( const QModelIndex&, int, int ) ) ,
541                     this, SIGNAL( rowsRemoved( const QModelIndex&, int, int ) ) );
542         disconnect( this->sourceModel(), SIGNAL( columnsInserted( const QModelIndex&, int, int ) ) ,
543                     this, SIGNAL( columnsInserted( const QModelIndex&, int, int ) ) );

```

```

544         disconnect( this->sourceModel(), SIGNAL( columnsRemoved( const QModelIndex&, int, int ) ),
545                     this, SIGNAL( columnsRemoved( const QModelIndex&, int, int ) ) );
546     }
547     QAbstractProxyModel::setSourceModel( sourceModel );
548     if( this->sourceModel() != NULL )
549     {
550         connect( this->sourceModel(), SIGNAL( dataChanged( const QModelIndex&, const QModelIndex&)),
551                 this, SIGNAL( dataChanged( const QModelIndex&, const QModelIndex&)));
552         connect( this->sourceModel(), SIGNAL( rowsInserted( const QModelIndex&, int, int ) ),
553                 this, SIGNAL( rowsInserted( const QModelIndex&, int, int ) ) );
554         connect( this->sourceModel(), SIGNAL( rowsRemoved( const QModelIndex&, int, int ) ),
555                 this, SIGNAL( rowsRemoved( const QModelIndex&, int, int ) ) );
556         connect( this->sourceModel(), SIGNAL( columnsInserted( const QModelIndex&, int, int ) ),
557                 this, SIGNAL( columnsInserted( const QModelIndex&, int, int ) ) );
558         connect( this->sourceModel(), SIGNAL( columnsRemoved( const QModelIndex&, int, int ) ),
559                 this, SIGNAL( columnsRemoved( const QModelIndex&, int, int ) ) );
560     }
561 }

```

#### 9.14.4.32 void AttributesModel::setVerticalHeaderDataMap (const QMap< int, QMap< int, QVariant > > *map*) [protected]

needed for serialization

Definition at line 595 of file KDChartAttributesModel.cpp.

```

596 {
597     mVerticalHeaderDataMap = map;
598 }

```

#### 9.14.4.33 const QMap< int, QMap< int, QVariant > > AttributesModel::verticalHeaderDataMap () const [protected]

needed for serialization

Definition at line 574 of file KDChartAttributesModel.cpp.

```

575 {
576     return mVerticalHeaderDataMap;
577 }

```

The documentation for this class was generated from the following files:

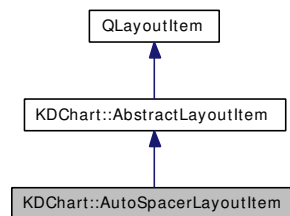
- [KDChartAttributesModel.h](#)
- [KDChartAttributesModel.cpp](#)



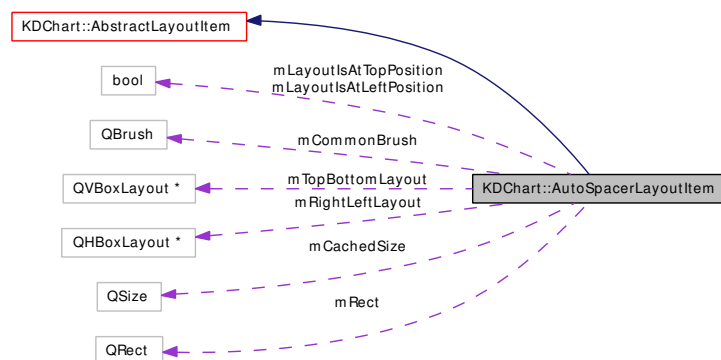
## 9.15 KDChart::AutoSpacerLayoutItem Class Reference

```
#include <KDChartLayoutItems.h>
```

Inheritance diagram for KDChart::AutoSpacerLayoutItem:



Collaboration diagram for KDChart::AutoSpacerLayoutItem:



### 9.15.1 Detailed Description

An empty layout item.

Definition at line 383 of file KDChartLayoutItems.h.

### Public Member Functions

- [AutoSpacerLayoutItem](#) (bool layoutIsAtTopPosition, QHBoxLayout \*rightToLeftLayout, bool layoutIsAtLeftPosition, QVBoxLayout \*topBottomLayout)
- virtual Qt::Orientations [expandingDirections](#) () const
- virtual QRect [geometry](#) () const
- virtual bool [isEmpty](#) () const
- virtual QSize [maximumSize](#) () const
- virtual QSize [minimumSize](#) () const
- virtual void [paint](#) (QPainter \*)
- virtual void [paintAll](#) (QPainter &painter)

*Default impl: just call paint.*

- virtual void [paintCtx](#) ([PaintContext](#) \*context)

*Default impl: Paint the complete item using its layouted position and size.*

- `QLayout * parentLayout ()`
- `void removeFromParentLayout ()`
- `virtual void setGeometry (const QRect &r)`
- `void setParentLayout (QLayout *lay)`
- `virtual void setParentWidget (QWidget *widget)`

*Inform the item about its widget: This enables the item, to trigger that widget's update, whenever the size of the item's contents has changed.*

- `virtual QSize sizeHint () const`
- `virtual void sizeHintChanged () const`

*Report changed size hint: ask the parent widget to recalculate the layout.*

## Protected Attributes

- `QWidget * mParent`
- `QLayout * mParentLayout`

## 9.15.2 Constructor & Destructor Documentation

### 9.15.2.1 `KDChart::AutoSpacerLayoutItem::AutoSpacerLayoutItem (bool layoutIsAtTopPosition, QHBoxLayout * rightToLeftLayout, bool layoutIsAtLeftPosition, QVBoxLayout * topBottomLayout)`

Definition at line 760 of file `KDChartLayoutItems.cpp`.

```

763      : AbstractLayoutItem( Qt::AlignCenter )
764      , mLayoutIsAtTopPosition( layoutIsAtTopPosition )
765      , mRightToLeftLayout( rightToLeftLayout )
766      , mLayoutIsAtLeftPosition( layoutIsAtLeftPosition )
767      , mTopBottomLayout( topBottomLayout )
768  {
769  }
```

## 9.15.3 Member Function Documentation

### 9.15.3.1 `Qt::Orientations KDChart::AutoSpacerLayoutItem::expandingDirections () const` [virtual]

Definition at line 771 of file `KDChartLayoutItems.cpp`.

```

772  {
773      return 0; // Grow neither vertically nor horizontally
774  }
```

### 9.15.3.2 `QRect KDChart::AutoSpacerLayoutItem::geometry () const` [virtual]

Definition at line 776 of file `KDChartLayoutItems.cpp`.

```

777  {
778      return mRect;
779  }
```

### 9.15.3.3 bool KdChart::AutoSpacerLayoutItem::isEmpty () const [virtual]

Definition at line 781 of file KdChartLayoutItems.cpp.

```
782 {  
783     return true; // never empty, otherwise the layout item would not exist  
784 }
```

### 9.15.3.4 QSize KdChart::AutoSpacerLayoutItem::maximumSize () const [virtual]

Definition at line 786 of file KdChartLayoutItems.cpp.

References [sizeHint\(\)](#).

```
787 {  
788     return sizeHint();  
789 }
```

### 9.15.3.5 QSize KdChart::AutoSpacerLayoutItem::minimumSize () const [virtual]

Definition at line 791 of file KdChartLayoutItems.cpp.

References [sizeHint\(\)](#).

```
792 {  
793     return sizeHint();  
794 }
```

### 9.15.3.6 void KdChart::AutoSpacerLayoutItem::paint (QPainter \*) [virtual]

Implements [KdChart::AbstractLayoutItem](#).

Definition at line 865 of file KdChartLayoutItems.cpp.

References [KdChart::AbstractLayoutItem::mParentLayout](#).

```
866 {  
867     if( mParentLayout && mRect.isValid() && mCachedSize.isValid() &&  
868         mCommonBrush.style() != Qt::NoBrush )  
869     {  
870         QPoint p1( mRect.topLeft() );  
871         QPoint p2( mRect.bottomRight() );  
872         if( mLayoutIsAtLeftPosition )  
873             p1.rx() += mCachedSize.width() - mParentLayout->spacing();  
874         else  
875             p2.rx() -= mCachedSize.width() - mParentLayout->spacing();  
876         if( mLayoutIsAtTopPosition ){  
877             p1.ry() += mCachedSize.height() - mParentLayout->spacing() - 1;  
878             p2.ry() -= 1;  
879         }else  
880             p2.ry() -= mCachedSize.height() - mParentLayout->spacing() - 1;  
881         //qDebug() << mLayoutIsAtTopPosition << mLayoutIsAtLeftPosition;  
882         //qDebug() << mRect;  
883         //qDebug() << mParentLayout->margin();  
884         //qDebug() << QRect( p1, p2 );  
885         const QPoint oldBrushOrigin( painter->brushOrigin() );
```

```

886         const QBrush oldBrush( painter->brush() );
887         const QPen oldPen( painter->pen() );
888         const QPointF newTopLeft( painter->deviceMatrix().map( p1 ) );
889         painter->setBrushOrigin( newTopLeft );
890         painter->setBrush( mCommonBrush );
891         painter->setPen( Qt::NoPen );
892         painter->drawRect( QRect( p1, p2 ) );
893         painter->setBrushOrigin( oldBrushOrigin );
894         painter->setBrush( oldBrush );
895         painter->setPen( oldPen );
896     }
897     // debug code:
898     #if 0
899     //qDebug() << "KDChart::AutoSpacerLayoutItem::paint()";
900     if( !mRect.isValid() )
901         return;
902
903     painter->drawRect( mRect );
904     painter->drawLine( QPointF( mRect.x(), mRect.top() ),
905                      QPointF( mRect.right(), mRect.bottom() ) );
906     painter->drawLine( QPointF( mRect.right(), mRect.top() ),
907                      QPointF( mRect.x(), mRect.bottom() ) );
908     #endif
909 }

```

### 9.15.3.7 void KDChart::AbstractLayoutItem::paintAll (QPainter & *painter*) [virtual, inherited]

Default impl: just call paint.

Derived classes like [KDChart::AbstractArea](#) are providing additional action here.

Reimplemented in [KDChart::AbstractArea](#), [KDChart::TextArea](#), and [KDChart::TernaryAxis](#).

Definition at line 69 of file KDChartLayoutItems.cpp.

References [KDChart::AbstractLayoutItem::paint\(\)](#).

```

70 {
71     paint( &painter );
72 }

```

### 9.15.3.8 void KDChart::AbstractLayoutItem::paintCtx ([PaintContext](#) \* *context*) [virtual, inherited]

Default impl: Paint the complete item using its layouted position and size.

Reimplemented in [KDChart::CartesianAxis](#), and [KDChart::TernaryAxis](#).

Definition at line 77 of file KDChartLayoutItems.cpp.

References [KDChart::AbstractLayoutItem::paint\(\)](#), and [KDChart::PaintContext::painter\(\)](#).

```

78 {
79     if( context )
80         paint( context->painter() );
81 }

```

**9.15.3.9** `QLayout* KDChart::AbstractLayoutItem::parentLayout ()` [inherited]

Definition at line 76 of file KDChartLayoutItems.h.

```

77         {
78             return mParentLayout;
79         }

```

**9.15.3.10** `void KDChart::AbstractLayoutItem::removeFromParentLayout ()` [inherited]

Definition at line 80 of file KDChartLayoutItems.h.

Referenced by KDChart::Chart::takeCoordinatePlane().

```

81         {
82             if( mParentLayout ){
83                 if( widget() )
84                     mParentLayout->removeWidget( widget() );
85                 else
86                     mParentLayout->removeItem( this );
87             }
88         }

```

**9.15.3.11** `void KDChart::AutoSpacerLayoutItem::setGeometry (const QRect & r)` [virtual]

Definition at line 796 of file KDChartLayoutItems.cpp.

```

797 {
798     mRect = r;
799 }

```

**9.15.3.12** `void KDChart::AbstractLayoutItem::setParentLayout (QLayout * lay)` [inherited]

Definition at line 72 of file KDChartLayoutItems.h.

```

73         {
74             mParentLayout = lay;
75         }

```

**9.15.3.13** `void KDChart::AbstractLayoutItem::setParentWidget (QWidget * widget)`  
[virtual, inherited]

Inform the item about its widget: This enables the item, to trigger that widget's update, whenever the size of the item's contents has changed.

Thus, you need to call setParentWidget on every item, that has a non-fixed size.

Definition at line 64 of file KDChartLayoutItems.cpp.

References KDChart::AbstractLayoutItem::mParent.

Referenced by KDChart::HeaderFooter::setParent(), and KDChart::AbstractCartesianDiagram::takeAxis().

```

65 {
66     mParent = widget;
67 }

```

#### 9.15.3.14 QSize KDChart::AutoSpacerLayoutItem::sizeHint () const [virtual]

Definition at line 821 of file KDChartLayoutItems.cpp.

References `updateCommonBrush()`.

Referenced by `maximumSize()`, and `minimumSize()`.

```

822 {
823     QBrush commonBrush;
824     bool bStart=true;
825     // calculate the maximal overlap of the top/bottom axes:
826     int topBottomOverlap = 0;
827     if( mTopBottomLayout ){
828         for (int i = 0; i < mTopBottomLayout->count(); ++i){
829             AbstractArea* area = dynamic_cast<AbstractArea*>(mTopBottomLayout->itemAt(i));
830             if( area ){
831                 //qDebug() << "AutoSpacerLayoutItem testing" << area;
832                 topBottomOverlap =
833                     mLayoutIsAtLeftPosition
834                     ? qMax( topBottomOverlap, area->rightOverlap() )
835                     : qMax( topBottomOverlap, area->leftOverlap() );
836                 updateCommonBrush( commonBrush, bStart, *area );
837             }
838         }
839     }
840     // calculate the maximal overlap of the left/right axes:
841     int leftRightOverlap = 0;
842     if( mRightToLeftLayout ){
843         for (int i = 0; i < mRightToLeftLayout->count(); ++i){
844             AbstractArea* area = dynamic_cast<AbstractArea*>(mRightToLeftLayout->itemAt(i));
845             if( area ){
846                 //qDebug() << "AutoSpacerLayoutItem testing" << area;
847                 leftRightOverlap =
848                     mLayoutIsAtTopPosition
849                     ? qMax( leftRightOverlap, area->bottomOverlap() )
850                     : qMax( leftRightOverlap, area->topOverlap() );
851                 updateCommonBrush( commonBrush, bStart, *area );
852             }
853         }
854     }
855     if( topBottomOverlap > 0 && leftRightOverlap > 0 )
856         mCommonBrush = commonBrush;
857     else
858         mCommonBrush = QBrush();
859     mCachedSize = QSize( topBottomOverlap, leftRightOverlap );
860     //qDebug() << mCachedSize;
861     return mCachedSize;
862 }

```

#### 9.15.3.15 void KDChart::AbstractLayoutItem::sizeHintChanged () const [virtual, inherited]

Report changed size hint: ask the parent widget to recalculate the layout.

Definition at line 86 of file KDChartLayoutItems.cpp.

References `KDChart::AbstractLayoutItem::mParent`.

Referenced by KDChart::TextLayoutItem::sizeHint().

```
87 {  
88     // This is exactly like what QWidget::updateGeometry does.  
89     // qDebug("KDChart::AbstractLayoutItem::sizeHintChanged() called");  
90     if( mParent ) {  
91         if ( mParent->layout() )  
92             mParent->layout()->invalidate();  
93         else  
94             QApplication::postEvent( mParent, new QEvent( QEvent::LayoutRequest ) );  
95     }  
96 }
```

## 9.15.4 Member Data Documentation

### 9.15.4.1 [QWidget\\*](#) [KDChart::AbstractLayoutItem::mParent](#) [protected, inherited]

Definition at line 90 of file KDChartLayoutItems.h.

Referenced by KDChart::AbstractLayoutItem::setParentWidget(), KDChart::TextLayoutItem::setText(), KDChart::TextLayoutItem::setTextAttributes(), and KDChart::AbstractLayoutItem::sizeHintChanged().

### 9.15.4.2 [QLayout\\*](#) [KDChart::AbstractLayoutItem::mParentLayout](#) [protected, inherited]

Definition at line 91 of file KDChartLayoutItems.h.

Referenced by paint().

The documentation for this class was generated from the following files:

- [KDChartLayoutItems.h](#)
- [KDChartLayoutItems.cpp](#)

## 9.16 KDCart::BackgroundAttributes Class Reference

```
#include <KDCartBackgroundAttributes.h>
```

### 9.16.1 Detailed Description

Set of attributes usable for background pixmaps.

Definition at line 43 of file KDCartBackgroundAttributes.h.

### Public Types

- enum [BackgroundPixmapMode](#) {  
    [BackgroundPixmapModeNone](#),  
    [BackgroundPixmapModeCentered](#),  
    [BackgroundPixmapModeScaled](#),  
    [BackgroundPixmapModeStretched](#) }

### Public Member Functions

- [BackgroundAttributes](#) (const [BackgroundAttributes](#) &)
- [BackgroundAttributes](#) ()
- [QBrush brush](#) () const
- bool [isEqualTo](#) (const [BackgroundAttributes](#) &other, bool ignorePixmap=false) const
- bool [isVisible](#) () const
- bool [operator!=](#) (const [BackgroundAttributes](#) &other) const
- [BackgroundAttributes](#) & [operator=](#) (const [BackgroundAttributes](#) &)
- bool [operator==](#) (const [BackgroundAttributes](#) &) const
- [QPixmap pixmap](#) () const
- [BackgroundPixmapMode pixmapMode](#) () const
- void [setBrush](#) (const [QBrush](#) &brush)
- void [setPixmap](#) (const [QPixmap](#) &backPixmap)
- void [setPixmapMode](#) ([BackgroundPixmapMode](#) mode)
- void [setVisible](#) (bool visible)
- [~BackgroundAttributes](#) ()

### 9.16.2 Member Enumeration Documentation

#### 9.16.2.1 enum [KDCart::BackgroundAttributes::BackgroundPixmapMode](#)

Enumerator:

*[BackgroundPixmapModeNone](#)*  
*[BackgroundPixmapModeCentered](#)*  
*[BackgroundPixmapModeScaled](#)*  
*[BackgroundPixmapModeStretched](#)*

Definition at line 52 of file KDCartBackgroundAttributes.h.



```

52             { BackgroundPixmapModeNone,
53               BackgroundPixmapModeCentered,
54               BackgroundPixmapModeScaled,
55               BackgroundPixmapModeStretched };

```

### 9.16.3 Constructor & Destructor Documentation

#### 9.16.3.1 BackgroundAttributes::BackgroundAttributes ()

Definition at line 55 of file KDCartBackgroundAttributes.cpp.

```

56     : _d( new Private() )
57 {
58 }

```

#### 9.16.3.2 BackgroundAttributes::BackgroundAttributes (const [BackgroundAttributes](#) &)

Definition at line 60 of file KDCartBackgroundAttributes.cpp.

```

61     : _d( new Private( *r.d ) )
62 {
63 }

```

#### 9.16.3.3 BackgroundAttributes::~~BackgroundAttributes ()

Definition at line 100 of file KDCartBackgroundAttributes.cpp.

```

101 {
102     delete _d; _d = 0;
103 }

```

### 9.16.4 Member Function Documentation

#### 9.16.4.1 QBrush BackgroundAttributes::brush () const

Definition at line 124 of file KDCartBackgroundAttributes.cpp.

References [d](#).

Referenced by [isEqualTo\(\)](#), [operator<<\(\)](#), and [KDCart::AbstractAreaBase::paintBackgroundAttributes\(\)](#).

```

125 {
126     return d->brush;
127 }

```

#### 9.16.4.2 bool BackgroundAttributes::isEqualTo (const [BackgroundAttributes](#) & *other*, bool *ignorePixmap* = false) const

Definition at line 81 of file KDCartBackgroundAttributes.cpp.

References [brush\(\)](#), [isVisible\(\)](#), [pixmap\(\)](#), and [pixmapMode\(\)](#).

Referenced by [operator==\(\)](#).

```

83 {
84     /*
85     qDebug() << "BackgroundAttributes::operator==";
86     qDebug() << "isVisible" << (isVisible() == other.isVisible());
87     qDebug() << "brush" << (brush() == other.brush());
88     qDebug() << "pixmapMode" << (pixmapMode() == other.pixmapMode());
89     qDebug() << "pixmap" << (pixmap().serialNumber() == other.pixmap().serialNumber());
90     */
91     return (
92         isVisible() == other.isVisible() &&
93         brush() == other.brush() &&
94         pixmapMode() == other.pixmapMode() &&
95         (ignorePixmap ||
96          pixmap().serialNumber() == other.pixmap().serialNumber()) );
97 }

```

#### 9.16.4.3 bool BackgroundAttributes::isVisible () const

Definition at line 114 of file KDChartBackgroundAttributes.cpp.

References d.

Referenced by isEqualTo(), operator<<(), and KDChart::AbstractAreaBase::paintBackgroundAttributes().

```

115 {
116     return d->visible;
117 }

```

#### 9.16.4.4 bool KDChart::BackgroundAttributes::operator!=(const BackgroundAttributes & other) const

Definition at line 70 of file KDChartBackgroundAttributes.h.

```

70 { return !operator==(other); }

```

#### 9.16.4.5 BackgroundAttributes & BackgroundAttributes::operator= (const BackgroundAttributes &)

Definition at line 65 of file KDChartBackgroundAttributes.cpp.

References d.

```

66 {
67     if( this == &r )
68         return *this;
69
70     *d = *r.d;
71
72     return *this;
73 }

```

#### 9.16.4.6 bool BackgroundAttributes::operator==(const BackgroundAttributes &) const

Definition at line 75 of file KDCartBackgroundAttributes.cpp.

References `isEqualTo()`.

```
76 {  
77     return isEqualTo( r );  
78 }
```

#### 9.16.4.7 QPixmap BackgroundAttributes::pixmap () const

Definition at line 144 of file KDCartBackgroundAttributes.cpp.

References `d`.

Referenced by `isEqualTo()`, `operator<<()`, and `KDCart::AbstractAreaBase::paintBackgroundAttributes()`.

```
145 {  
146     return d->pixmap;  
147 }
```

#### 9.16.4.8 BackgroundAttributes::BackgroundPixmapMode BackgroundAttributes::pixmapMode () const

Definition at line 134 of file KDCartBackgroundAttributes.cpp.

References `d`.

Referenced by `isEqualTo()`, `operator<<()`, and `KDCart::AbstractAreaBase::paintBackgroundAttributes()`.

```
135 {  
136     return d->pixmapMode;  
137 }
```

#### 9.16.4.9 void BackgroundAttributes::setBrush (const QBrush & brush)

Definition at line 119 of file KDCartBackgroundAttributes.cpp.

References `d`.

```
120 {  
121     d->brush = brush;  
122 }
```

#### 9.16.4.10 void BackgroundAttributes::setPixmap (const QPixmap & backPixmap)

Definition at line 139 of file KDCartBackgroundAttributes.cpp.

References `d`.

```
140 {  
141     d->pixmap = backPixmap;  
142 }
```

**9.16.4.11 void BackgroundAttributes::setPixmapMode ([BackgroundPixmapMode](#) *mode*)**

Definition at line 129 of file `KDChartBackgroundAttributes.cpp`.

References [d](#).

```
130 {  
131     d->pixmapMode = mode;  
132 }
```

**9.16.4.12 void BackgroundAttributes::setVisible (bool *visible*)**

Definition at line 108 of file `KDChartBackgroundAttributes.cpp`.

References [d](#).

```
109 {  
110     d->visible = visible;  
111 }
```

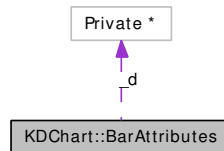
The documentation for this class was generated from the following files:

- [KDChartBackgroundAttributes.h](#)
- [KDChartBackgroundAttributes.cpp](#)

## 9.17 KDChart::BarAttributes Class Reference

```
#include <KDChartBarAttributes.h>
```

Collaboration diagram for KDChart::BarAttributes:



### 9.17.1 Detailed Description

Set of attributes for changing the appearance of bar charts.

Definition at line 37 of file KDChartBarAttributes.h.

#### Public Member Functions

- [BarAttributes](#) (const [BarAttributes](#) &)
- [BarAttributes](#) ()
- qreal [barGapFactor](#) () const
- bool [drawSolidExcessArrows](#) () const
- qreal [fixedBarWidth](#) () const
- qreal [fixedDataValueGap](#) () const
- qreal [fixedValueBlockGap](#) () const
- qreal [groupGapFactor](#) () const
- bool [operator!=](#) (const [BarAttributes](#) &other) const
- [BarAttributes](#) & [operator=](#) (const [BarAttributes](#) &)
- bool [operator==](#) (const [BarAttributes](#) &) const
- void [setBarGapFactor](#) (qreal gapFactor)
- void [setDrawSolidExcessArrows](#) (bool solidArrows)
- void [setFixedBarWidth](#) (qreal width)
- void [setFixedDataValueGap](#) (qreal gap)
- void [setFixedValueBlockGap](#) (qreal gap)
- void [setGroupGapFactor](#) (qreal gapFactor)
- void [setUseFixedBarWidth](#) (bool useFixedBarWidth)
- void [setUseFixedDataValueGap](#) (bool gapIsFixed)
- void [setUseFixedValueBlockGap](#) (bool gapIsFixed)
- bool [useFixedBarWidth](#) () const
- bool [useFixedDataValueGap](#) () const
- bool [useFixedValueBlockGap](#) () const
- [~BarAttributes](#) ()

## 9.17.2 Constructor & Destructor Documentation

### 9.17.2.1 BarAttributes::BarAttributes ()

Definition at line 69 of file KDChartBarAttributes.cpp.

```
70      : _d( new Private() )
71  {
72  }
```

### 9.17.2.2 BarAttributes::BarAttributes (const BarAttributes &)

Definition at line 74 of file KDChartBarAttributes.cpp.

```
75      : _d( new Private( *r.d ) )
76  {
77  }
```

### 9.17.2.3 BarAttributes::~~BarAttributes ()

Definition at line 89 of file KDChartBarAttributes.cpp.

```
90  {
91      delete _d; _d = 0;
92  }
```

## 9.17.3 Member Function Documentation

### 9.17.3.1 qreal BarAttributes::barGapFactor () const

Definition at line 188 of file KDChartBarAttributes.cpp.

References d.

Referenced by operator==( ).

```
189  {
190      return d->barGapFactor;
191  }
```

### 9.17.3.2 bool BarAttributes::drawSolidExcessArrows () const

Definition at line 198 of file KDChartBarAttributes.cpp.

References d.

Referenced by operator==( ).

```
199  {
200      return d->drawSolidExcessArrows;
201  }
```

### 9.17.3.3 qreal BarAttributes::fixedBarWidth () const

Definition at line 157 of file KDChartBarAttributes.cpp.

References [d](#).

Referenced by [operator==\(\(\)\)](#).

```
158 {  
159  
160     return d->barWidth;  
161 }
```

### 9.17.3.4 qreal BarAttributes::fixedDataValueGap () const

Definition at line 117 of file KDChartBarAttributes.cpp.

References [d](#).

Referenced by [operator==\(\(\)\)](#).

```
118 {  
119     return d->datasetGap;  
120 }
```

### 9.17.3.5 qreal BarAttributes::fixedValueBlockGap () const

Definition at line 137 of file KDChartBarAttributes.cpp.

References [d](#).

Referenced by [operator==\(\(\)\)](#).

```
138 {  
139     return d->valueBlockGap;  
140 }
```

### 9.17.3.6 qreal BarAttributes::groupGapFactor () const

Definition at line 178 of file KDChartBarAttributes.cpp.

References [d](#).

Referenced by [operator==\(\(\)\)](#).

```
179 {  
180     return d->groupGapFactor;  
181 }
```

### 9.17.3.7 bool KDChart::BarAttributes::operator!= (const [BarAttributes](#) & *other*) const

Definition at line 74 of file KDChartBarAttributes.h.

```
74 { return !operator==(other); }
```

### 9.17.3.8 **BarAttributes** & BarAttributes::operator= (const **BarAttributes** &)

Definition at line 79 of file KDChartBarAttributes.cpp.

References `d`.

```

80 {
81     if( this == &r )
82         return *this;
83
84     *d = *r.d;
85
86     return *this;
87 }
```

### 9.17.3.9 **bool** BarAttributes::operator== (const **BarAttributes** &) const

Definition at line 95 of file KDChartBarAttributes.cpp.

References `barGapFactor()`, `drawSolidExcessArrows()`, `fixedBarWidth()`, `fixedDataValueGap()`, `fixedValueBlockGap()`, `groupGapFactor()`, `useFixedBarWidth()`, `useFixedDataValueGap()`, and `useFixedValueBlockGap()`.

```

96 {
97     if( fixedDataValueGap() == r.fixedDataValueGap() &&
98         useFixedDataValueGap() == r.useFixedDataValueGap() &&
99         fixedValueBlockGap() == r.fixedValueBlockGap() &&
100        useFixedValueBlockGap() == r.useFixedValueBlockGap() &&
101        fixedBarWidth() == r.fixedBarWidth() &&
102        useFixedBarWidth() == r.useFixedBarWidth() &&
103        groupGapFactor() == r.groupGapFactor() &&
104        barGapFactor() == r.barGapFactor() &&
105        drawSolidExcessArrows() == r.drawSolidExcessArrows() )
106        return true;
107     else
108         return false;
109 }
```

### 9.17.3.10 **void** BarAttributes::setBarGapFactor (qreal *gapFactor*)

Definition at line 183 of file KDChartBarAttributes.cpp.

References `d`.

```

184 {
185     d->barGapFactor = gapFactor;
186 }
```

### 9.17.3.11 **void** BarAttributes::setDrawSolidExcessArrows (bool *solidArrows*)

Definition at line 193 of file KDChartBarAttributes.cpp.

References `d`.

```

194 {
195     d->drawSolidExcessArrows = solidArrows;
196 }
```



**9.17.3.12 void BarAttributes::setFixedBarWidth (qreal *width*)**

Definition at line 152 of file KDChartBarAttributes.cpp.

References [d](#).

```
153 {  
154     d->barWidth = width;  
155 }
```

**9.17.3.13 void BarAttributes::setFixedDataValueGap (qreal *gap*)**

Definition at line 112 of file KDChartBarAttributes.cpp.

References [d](#).

```
113 {  
114     d->datasetGap = gap;  
115 }
```

**9.17.3.14 void BarAttributes::setFixedValueBlockGap (qreal *gap*)**

Definition at line 132 of file KDChartBarAttributes.cpp.

References [d](#).

```
133 {  
134     d->valueBlockGap = gap;  
135 }
```

**9.17.3.15 void BarAttributes::setGroupGapFactor (qreal *gapFactor*)**

Definition at line 173 of file KDChartBarAttributes.cpp.

References [d](#).

```
174 {  
175     d->groupGapFactor = gapFactor;  
176 }
```

**9.17.3.16 void BarAttributes::setUseFixedBarWidth (bool *useFixedBarWidth*)**

Definition at line 163 of file KDChartBarAttributes.cpp.

References [d](#).

```
164 {  
165     d->useFixedBarWidth = useFixedBarWidth;  
166 }
```

**9.17.3.17 void BarAttributes::setUseFixedDataValueGap (bool *gapIsFixed*)**

Definition at line 122 of file KDChartBarAttributes.cpp.

References [d](#).

```
123 {  
124     d->useFixedDatasetGap = gapIsFixed;  
125 }
```

**9.17.3.18 void BarAttributes::setUseFixedValueBlockGap (bool *gapIsFixed*)**

Definition at line 142 of file KDChartBarAttributes.cpp.

References [d](#).

```
143 {  
144     d->useFixedValueBlockGap = gapIsFixed;  
145 }
```

**9.17.3.19 bool BarAttributes::useFixedBarWidth () const**

Definition at line 168 of file KDChartBarAttributes.cpp.

References [d](#).

Referenced by [operator==\(\(\)\)](#).

```
169 {  
170     return d->useFixedBarWidth;  
171 }
```

**9.17.3.20 bool BarAttributes::useFixedDataValueGap () const**

Definition at line 127 of file KDChartBarAttributes.cpp.

References [d](#).

Referenced by [operator==\(\(\)\)](#).

```
128 {  
129     return d->useFixedDatasetGap;  
130 }
```

**9.17.3.21 bool BarAttributes::useFixedValueBlockGap () const**

Definition at line 147 of file KDChartBarAttributes.cpp.

References [d](#).

Referenced by [operator==\(\(\)\)](#).

```
148 {  
149     return d->useFixedValueBlockGap;  
150 }
```

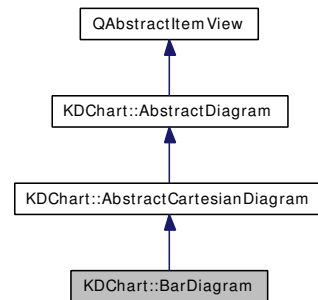
The documentation for this class was generated from the following files:

- [KDChartBarAttributes.h](#)
- [KDChartBarAttributes.cpp](#)

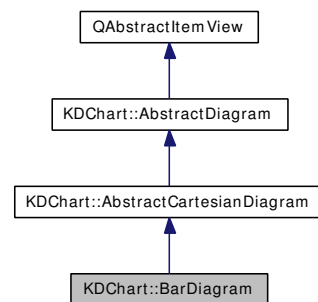
## 9.18 KDChart::BarDiagram Class Reference

```
#include <KDChartBarDiagram.h>
```

Inheritance diagram for KDChart::BarDiagram:



Collaboration diagram for KDChart::BarDiagram:



### 9.18.1 Detailed Description

[BarDiagram](#) defines a common bar diagram.

It provides different subtypes which are set using *setType*.

Definition at line 48 of file KDChartBarDiagram.h.

#### Public Types

- enum [BarType](#) {  
    [Normal](#),  
    [Stacked](#),  
    [Percent](#),  
    [Rows](#) }

#### Signals

- void [dataHidden](#) ()

*This signal is emitted, when the hidden status of at least one data cell was (un)set.*

- void [layoutChanged](#) ([AbstractDiagram](#) \*)  
*Diagrams are supposed to emit this signal, when the layout of one of their element changes.*
- void [modelsChanged](#) ()  
*This signal is emitted, when either the model or the [AttributesModel](#) is replaced.*
- void [propertiesChanged](#) ()  
*Emitted upon change of a property of the Diagram.*

## Public Member Functions

- virtual void [addAxis](#) ([CartesianAxis](#) \*axis)  
*Add the axis to the diagram.*
- bool [allowOverlappingDataValueTexts](#) () const  
**Returns:**  
*Whether data value labels are allowed to overlap.*
- bool [antiAliasing](#) () const  
**Returns:**  
*Whether anti-aliasing is to be used for rendering this diagram.*
- virtual [AttributesModel](#) \* [attributesModel](#) () const  
*Returns the [AttributesModel](#), that is used by this diagram.*
- virtual [KDChart::CartesianAxisList](#) [axes](#) () const  
**Returns:**  
*a list of all axes added to the diagram*
- [BarAttributes](#) [barAttributes](#) (const [QModelIndex](#) &index) const  
**Returns:**  
*the bar attribute set of the model index index*
- [BarAttributes](#) [barAttributes](#) (int column) const  
**Returns:**  
*the bar attribute set of data set column*
- [BarAttributes](#) [barAttributes](#) () const  
**Returns:**  
*the global bar attribute set*
- [BarDiagram](#) ([QWidget](#) \*parent=0, [CartesianCoordinatePlane](#) \*plane=0)
- [QBrush](#) [brush](#) (const [QModelIndex](#) &index) const  
*Retrieve the brush to be used, for painting the datapoint at the given index in the model.*
- [QBrush](#) [brush](#) (int dataset) const

*Retrieve the brush to be used for the given dataset.*

- `QBrush brush () const`  
*Retrieve the brush to be used for painting datapoints globally.*
- `virtual BarDiagram * clone () const`  
*Creates an exact copy of this diagram.*
- `bool compare (const AbstractDiagram *other) const`  
*Returns true if both diagrams have the same settings.*
- `bool compare (const AbstractCartesianDiagram *other) const`  
*Returns true if both diagrams have the same settings.*
- `bool compare (const BarDiagram *other) const`  
*Returns true if both diagrams have the same settings.*
- `AbstractCoordinatePlane * coordinatePlane () const`  
*The coordinate plane associated with the diagram.*
- `const QPair< QPointF, QPointF > dataBoundaries () const`  
*Return the bottom left and top right data point, that the diagram will display (unless the grid adjusts these values).*
- `virtual void dataChanged (const QModelIndex &topLeft, const QModelIndex &bottomRight)`  
*[reimplemented]*
- `QList< QBrush > datasetBrushes () const`  
*The set of dataset brushes currently used, for use in legends, etc.*
- `int datasetDimension () const`  
*The dataset dimension of a diagram determines, how many value dimensions it expects each datapoint to have.*
- `QStringList datasetLabels () const`  
*The set of dataset labels currently displayed, for use in legends, etc.*
- `QList< MarkerAttributes > datasetMarkers () const`  
*The set of dataset markers currently used, for use in legends, etc.*
- `QList< QPen > datasetPens () const`  
*The set of dataset pens currently used, for use in legends, etc.*
- `DataValueAttributes dataValueAttributes (const QModelIndex &index) const`  
*Retrieve the DataValueAttributes for the given index.*
- `DataValueAttributes dataValueAttributes (int column) const`  
*Retrieve the DataValueAttributes for the given dataset.*
- `DataValueAttributes dataValueAttributes () const`

Retrieve the [DataValueAttributes](#) specified globally.

- virtual void [doItemsLayout](#) ()  
*[reimplemented]*
- virtual int [horizontalOffset](#) () const  
*[reimplemented]*
- virtual QModelIndex [indexAt](#) (const QPoint &point) const  
*[reimplemented]*
- QModelIndexList [indexesAt](#) (const QPoint &point) const  
*This method is added alongside with indexAt from QAIM, since in kdchart multiple indexes can be displayed at the same spot.*
- bool [isHidden](#) (const QModelIndex &index) const  
*Retrieve the hidden status for the given index.*
- bool [isHidden](#) (int column) const  
*Retrieve the hidden status for the given dataset.*
- bool [isHidden](#) () const  
*Retrieve the hidden status specified globally.*
- virtual bool [isIndexHidden](#) (const QModelIndex &index) const  
*[reimplemented]*
- QStringList [itemRowLabels](#) () const  
*The set of item row labels currently displayed, for use in Abscissa axes, etc.*
- virtual void [layoutPlanes](#) ()  
*Triggers layouting of all coordinate planes on the current chart.*
- virtual QModelIndex [moveCursor](#) (CursorAction cursorAction, Qt::KeyboardModifiers modifiers)  
*[reimplemented]*
- const int [numberOfAbscissaSegments](#) () const  
*[reimplemented]*
- const int [numberOfOrdinateSegments](#) () const  
*[reimplemented]*
- void [paintDataValueText](#) (QPainter \*painter, const QModelIndex &index, const QPointF &pos, double value)
- void [paintMarker](#) (QPainter \*painter, const QModelIndex &index, const QPointF &pos)
- virtual void [paintMarker](#) (QPainter \*painter, const [MarkerAttributes](#) &markerAttributes, const QBrush &brush, const QPen &, const QPointF &point, const QSizeF &size)
- QPen [pen](#) (const QModelIndex &index) const  
*Retrieve the pen to be used, for painting the datapoint at the given index in the model.*

- QPen [pen](#) (int dataset) const  
*Retrieve the pen to be used for the given dataset.*
- QPen [pen](#) () const  
*Retrieve the pen to be used for painting datapoints globally.*
- bool [percentMode](#) () const
- virtual [AbstractCartesianDiagram](#) \* [referenceDiagram](#) () const  
**Returns:**  
*this diagram's reference diagram*
- virtual QPointF [referenceDiagramOffset](#) () const  
**Returns:**  
*the relative offset of this diagram's reference diagram*
- void [resize](#) (const QSizeF &area)  
*Called by the widget's sizeEvent.*
- virtual void [scrollTo](#) (const QModelIndex &index, ScrollHint hint=EnsureVisible)  
*[reimplemented]*
- void [setAllowOverlappingDataValueTexts](#) (bool allow)  
*Set whether data value labels are allowed to overlap.*
- void [setAntiAliasing](#) (bool enabled)  
*Set whether anti-aliasing is to be used while rendering this diagram.*
- void [setAttributesModel](#) ([AttributesModel](#) \*model)  
*Associate an [AttributesModel](#) with this diagram.*
- void [setBarAttributes](#) (const QModelIndex &index, const [BarAttributes](#) &a)  
*Sets the line attributes for the model index index to ba.*
- void [setBarAttributes](#) (int column, const [BarAttributes](#) &a)  
*Sets the bar attributes of data set column to ba.*
- void [setBarAttributes](#) (const [BarAttributes](#) &a)  
*Sets the global bar attributes to ba.*
- void [setBrush](#) (const QBrush &brush)  
*Set the brush to be used, for painting all datasets in the model.*
- void [setBrush](#) (int dataset, const QBrush &brush)  
*Set the brush to be used, for painting the given dataset.*
- void [setBrush](#) (const QModelIndex &index, const QBrush &brush)  
*Set the brush to be used, for painting the datapoint at the given index.*
- virtual void [setCoordinatePlane](#) ([AbstractCoordinatePlane](#) \*plane)



*[reimplemented]*

- void [setDatasetDimension](#) (int dimension)  
*Sets the dataset dimension of the diagram.*
- void [setDataValueAttributes](#) (const [DataValueAttributes](#) &a)  
*Set the [DataValueAttributes](#) for all datapoints in the model.*
- void [setDataValueAttributes](#) (int dataset, const [DataValueAttributes](#) &a)  
*Set the [DataValueAttributes](#) for the given dataset.*
- void [setDataValueAttributes](#) (const QModelIndex &index, const [DataValueAttributes](#) &a)  
*Set the [DataValueAttributes](#) for the given index.*
- void [setHidden](#) (bool hidden)  
*Hide (or unhide, resp.)*
- void [setHidden](#) (int column, bool hidden)  
*Hide (or unhide, resp.)*
- void [setHidden](#) (const QModelIndex &index, bool hidden)  
*Hide (or unhide, resp.)*
- void [setModel](#) (QAbstractItemModel \*model)  
*Associate a model with the diagram.*
- void [setPen](#) (const QPen &pen)  
*Set the pen to be used, for painting all datasets in the model.*
- void [setPen](#) (int dataset, const QPen &pen)  
*Set the pen to be used, for painting the given dataset.*
- void [setPen](#) (const QModelIndex &index, const QPen &pen)  
*Set the pen to be used, for painting the datapoint at the given index.*
- void [setPercentMode](#) (bool percent)
- virtual void [setReferenceDiagram](#) ([AbstractCartesianDiagram](#) \*diagram, const QPointF &offset=QPointF())  
*Makes this diagram use another diagram diagram as reference diagram with relative offset offset.*
- void [setRootIndex](#) (const QModelIndex &index)  
*Set the root index in the model, where the diagram starts referencing data for display.*
- virtual void [setSelection](#) (const QRect &rect, QItemSelectionModel::SelectionFlags command)  
*[reimplemented]*
- void [setThreeDBarAttributes](#) (const QModelIndex &index, const [ThreeDBarAttributes](#) &a)  
*Sets the 3D line attributes of model index index to threeDAttrs.*
- void [setThreeDBarAttributes](#) (int column, const [ThreeDBarAttributes](#) &a)

*Sets the 3D bar attributes of dataset column to threeDAttrs.*

- void [setThreeDBarAttributes](#) (const [ThreeDBarAttributes](#) &a)  
*Sets the global 3D bar attributes to threeDAttrs.*
- void [setType](#) (const [BarType](#) type)  
*Sets the bar diagram's type to type.*
- void [setUnitPrefix](#) (const QString &prefix, Qt::Orientation orientation)  
*Sets the unit prefix for all values.*
- void [setUnitPrefix](#) (const QString &prefix, int column, Qt::Orientation orientation)  
*Sets the unit prefix for one value.*
- void [setUnitSuffix](#) (const QString &suffix, Qt::Orientation orientation)  
*Sets the unit suffix for all values.*
- void [setUnitSuffix](#) (const QString &suffix, int column, Qt::Orientation orientation)  
*Sets the unit suffix for one value.*
- virtual void [takeAxis](#) ([CartesianAxis](#) \*axis)  
*Removes the axis from the diagram, without deleting it.*
- [ThreeDBarAttributes](#) [threeDBarAttributes](#) (const QModelIndex &index) const  
**Returns:**  
*the 3D bar attributes of the model index index*
- [ThreeDBarAttributes](#) [threeDBarAttributes](#) (int column) const  
**Returns:**  
*the 3D bar attributes of data set column*
- [ThreeDBarAttributes](#) [threeDBarAttributes](#) () const  
**Returns:**  
*the global 3D bar attributes*
- [BarType](#) [type](#) () const  
**Returns:**  
*the type of the line diagram*
- QString [unitPrefix](#) (Qt::Orientation orientation) const  
*Returns the global unit prefix.*
- QString [unitPrefix](#) (int column, Qt::Orientation orientation, bool fallback=false) const  
*Returns the unit prefix for a special value.*
- QString [unitSuffix](#) (Qt::Orientation orientation) const  
*Returns the global unit suffix.*
- QString [unitSuffix](#) (int column, Qt::Orientation orientation, bool fallback=false) const

*Returns the unit suffix for a special value.*

- void [update](#) () const
- void [useDefaultColors](#) ()  
*Set the palette to be used, for painting datasets to the default palette.*
- void [useRainbowColors](#) ()  
*Set the palette to be used, for painting datasets to the rainbow palette.*
- virtual bool [usesExternalAttributesModel](#) () const  
*Returns whether the diagram is using its own built-in attributes model or an attributes model that was set via [setAttributesModel](#).*
- void [useSubduedColors](#) ()  
*Set the palette to be used, for painting datasets to the subdued palette.*
- virtual int [verticalOffset](#) () const  
*[reimplemented]*
- virtual QRect [visualRect](#) (const QModelIndex &index) const  
*[reimplemented]*
- virtual QRegion [visualRegionForSelection](#) (const QItemSelection &selection) const  
*[reimplemented]*
- virtual [~BarDiagram](#) ()

## Protected Member Functions

- QModelIndex [attributesModelRootIndex](#) () const
- const QPair< QPointF, QPointF > [calculateDataBoundaries](#) () const  
*[reimplemented]*
- virtual bool [checkInvariants](#) (bool justReturnTheStatus=false) const
- QModelIndex [columnToIndex](#) (int column) const
- void [paint](#) (PaintContext \*paintContext)  
*Draw the diagram contents to the rectangle and painter, that are passed in as part of the paint context.*
- virtual void [paintDataValueTexts](#) (QPainter \*painter)
- void [paintEvent](#) (QPaintEvent \*)
- virtual void [paintMarkers](#) (QPainter \*painter)
- void [resizeEvent](#) (QResizeEvent \*)
- void [setAttributesModelRootIndex](#) (const QModelIndex &)
- void [setDataBoundariesDirty](#) () const
- virtual double [threeDItemDepth](#) (int column) const  
**Returns:**  
*the 3D item depth of the data set column*
- virtual double [threeDItemDepth](#) (const QModelIndex &index) const

**Returns:**

*the 3D item depth of the model index index*

- double [valueForCell](#) (int row, int column) const

*Helper method, retrieving the data value (DisplayRole) for a given row and column.*

## 9.18.2 Member Enumeration Documentation

### 9.18.2.1 enum [KDChart::BarDiagram::BarType](#)

#### Enumerator:

*Normal*

*Stacked*

*Percent*

*Rows*

Definition at line 70 of file KDChartBarDiagram.h.

```
70             { Normal,
71               Stacked,
72               Percent,
73               Rows };
```

## 9.18.3 Constructor & Destructor Documentation

### 9.18.3.1 [BarDiagram::BarDiagram](#) ([QWidget](#) \* *parent* = 0, [CartesianCoordinatePlane](#) \* *plane* = 0) [explicit]

Definition at line 55 of file KDChartBarDiagram.cpp.

Referenced by [clone\(\)](#).

```
55                                     :
56     AbstractCartesianDiagram( new Private(), parent, plane )
57 {
58     init();
59 }
```

### 9.18.3.2 [BarDiagram::~~BarDiagram](#) () [virtual]

Definition at line 71 of file KDChartBarDiagram.cpp.

```
72 {
73 }
```

## 9.18.4 Member Function Documentation

### 9.18.4.1 void AbstractCartesianDiagram::addAxis ([CartesianAxis](#) \* *axis*) [virtual, inherited]

Add the axis to the diagram.

The diagram takes ownership of the axis and will delete it.

To gain back ownership (e.g. for assigning the axis to another diagram) use the takeAxis method, before calling addAxis on the other diagram.

**See also:**

[takeAxis](#)

Definition at line 91 of file KDChartAbstractCartesianDiagram.cpp.

References [d](#), and [KDChart::AbstractCartesianDiagram::layoutPlanes\(\)](#).

```
92 {
93     if ( !d->axesList.contains( axis ) ) {
94         d->axesList.append( axis );
95         axis->createObserver( this );
96         layoutPlanes();
97     }
98 }
```

### 9.18.4.2 bool AbstractDiagram::allowOverlappingDataValueTexts () const [inherited]

**Returns:**

Whether data value labels are allowed to overlap.

Definition at line 440 of file KDChartAbstractDiagram.cpp.

References [d](#).

Referenced by [KDChart::AbstractDiagram::compare\(\)](#).

```
441 {
442     return d->allowOverlappingDataValueTexts;
443 }
```

### 9.18.4.3 bool AbstractDiagram::antiAliasing () const [inherited]

**Returns:**

Whether anti-aliasing is to be used for rendering this diagram.

Definition at line 451 of file KDChartAbstractDiagram.cpp.

References [d](#).

Referenced by [KDChart::AbstractDiagram::compare\(\)](#).

```
452 {
453     return d->antiAliasing;
454 }
```

#### 9.18.4.4 [AttributesModel](#) \* [AbstractDiagram::attributesModel](#) () const [virtual, inherited]

Returns the [AttributesModel](#), that is used by this diagram.

By default each diagram owns its own [AttributesModel](#), which should never be deleted. Only if a user-supplied [AttributesModel](#) has been set does the pointer returned here not belong to the diagram.

#### Returns:

The [AttributesModel](#) associated with the diagram.

#### See also:

[setAttributesModel](#)

Definition at line 280 of file [KDChartAbstractDiagram.cpp](#).

References d.

Referenced by [KDChart::AbstractDiagram::brush\(\)](#), [KDChart::AbstractDiagram::compare\(\)](#), [KDChart::AbstractDiagram::datasetBrushes\(\)](#), [KDChart::AbstractDiagram::datasetLabels\(\)](#), [KDChart::AbstractDiagram::datasetMarkers\(\)](#), [KDChart::AbstractDiagram::datasetPens\(\)](#), [KDChart::AbstractDiagram::dataValueAttributes\(\)](#), [KDChart::AbstractDiagram::isHidden\(\)](#), [KDChart::AbstractDiagram::itemRowLabels\(\)](#), [KDChart::RingDiagram::paint\(\)](#), [KDChart::PolarDiagram::paint\(\)](#), [KDChart::AbstractDiagram::pen\(\)](#), [KDChart::AbstractCartesianDiagram::setAttributesModel\(\)](#), [setBarAttributes\(\)](#), [KDChart::AbstractDiagram::setBrush\(\)](#), [KDChart::AbstractCartesianDiagram::setModel\(\)](#), [KDChart::AbstractDiagram::setPen\(\)](#), and [KDChart::TernaryLineDiagram::TernaryLineDiagram\(\)](#).

```
281 {
282     return d->attributesModel;
283 }
```

#### 9.18.4.5 [QModelIndex](#) [AbstractDiagram::attributesModelRootIndex](#) () const [protected, inherited]

returns a [QModelIndex](#) pointing into the [AttributesModel](#) that corresponds to the root index of the diagram.

Definition at line 302 of file [KDChartAbstractDiagram.cpp](#).

References d.

Referenced by [KDChart::AbstractDiagram::datasetBrushes\(\)](#), [KDChart::AbstractDiagram::datasetLabels\(\)](#), [KDChart::AbstractDiagram::datasetMarkers\(\)](#), [KDChart::AbstractDiagram::datasetPens\(\)](#), [KDChart::AbstractDiagram::itemRowLabels\(\)](#), [KDChart::Plotter::numberOfAbscissaSegments\(\)](#), [KDChart::LineDiagram::numberOfAbscissaSegments\(\)](#), [numberOfAbscissaSegments\(\)](#), [KDChart::Plotter::numberOfOrdinateSegments\(\)](#), [KDChart::LineDiagram::numberOfOrdinateSegments\(\)](#), [numberOfOrdinateSegments\(\)](#), [KDChart::AbstractDiagram::valueForCell\(\)](#), and [KDChart::LineDiagram::valueForCellTesting\(\)](#).

```
303 {
304     if ( !d->attributesModelRootIndex.isValid() )
305         d->attributesModelRootIndex = d->attributesModel->mapFromSource( rootIndex() );
306     return d->attributesModelRootIndex;
307 }
```

**9.18.4.6** [KDChart::CartesianAxisList](#) **AbstractCartesianDiagram::axes () const** [virtual, inherited]

**Returns:**

a list of all axes added to the diagram

Definition at line 110 of file KDChartAbstractCartesianDiagram.cpp.

References d.

Referenced by KDChart::Widget::setType(), and KDChart::CartesianCoordinatePlane::sharedAxisMasterPlane().

```
111 {  
112     return d->axesList;  
113 }
```

**9.18.4.7** [BarAttributes](#) **BarDiagram::barAttributes (const QModelIndex & index) const**

**Returns:**

the bar attribute set of the model index *index*

Definition at line 196 of file KDChartBarDiagram.cpp.

References KDChart::BarAttributesRole, and d.

```
197 {  
198     return qVariantValue<BarAttributes>(   
199         d->attributesModel->data(   
200             d->attributesModel->mapFromSource( index ),  
201             KDChart::BarAttributesRole ) );  
202 }
```

**9.18.4.8** [BarAttributes](#) **BarDiagram::barAttributes (int column) const**

**Returns:**

the bar attribute set of data set *column*

Definition at line 185 of file KDChartBarDiagram.cpp.

References KDChart::BarAttributesRole, KDChart::AbstractDiagram::columnToIndex(), and d.

```
186 {  
187     return qVariantValue<BarAttributes>(   
188         d->attributesModel->data(   
189             d->attributesModel->mapFromSource( columnToIndex( column ) ),  
190             KDChart::BarAttributesRole ) );  
191 }
```

#### 9.18.4.9 **BarAttributes** `BarDiagram::barAttributes () const`

##### Returns:

the global bar attribute set

Definition at line 176 of file `KDChartBarDiagram.cpp`.

References `KDChart::BarAttributesRole`, and `d`.

```
177 {
178     return qVariantValue<BarAttributes>(
179         d->attributesModel->data( KDChart::BarAttributesRole ) );
180 }
```

#### 9.18.4.10 **QBrush** `AbstractDiagram::brush (const QModelIndex & index) const` [inherited]

Retrieve the brush to be used, for painting the datapoint at the given index in the model.

##### Parameters:

*index* The index of the datapoint in the model.

##### Returns:

The brush to use for painting.

Definition at line 838 of file `KDChartAbstractDiagram.cpp`.

References `KDChart::AbstractDiagram::attributesModel()`, `KDChart::AbstractDiagram::brush()`, `KDChart::AttributesModel::data()`, `KDChart::DatasetBrushRole`, and `KDChart::AbstractDiagram::datasetDimension()`.

```
839 {
840     if( datasetDimension() > 1 )
841         return brush( index.column() );
842     return qVariantValue<QBrush>(
843         attributesModel()->data(
844             attributesModel()->mapFromSource( index ),
845             DatasetBrushRole ) );
846 }
```

#### 9.18.4.11 **QBrush** `AbstractDiagram::brush (int dataset) const` [inherited]

Retrieve the brush to be used for the given dataset.

This will fall back automatically to what was set at model level, if there are no dataset specific settings.

##### Parameters:

*dataset* The dataset to retrieve the brush for.

##### Returns:

The brush to use for painting.



Definition at line 828 of file KDChartAbstractDiagram.cpp.

References [KDChart::AbstractDiagram::attributesModel\(\)](#), [KDChart::AbstractDiagram::columnToIndex\(\)](#), [KDChart::AttributesModel::data\(\)](#), [KDChart::DatasetBrushRole](#), and [KDChart::AbstractDiagram::datasetDimension\(\)](#).

```
829 {
830     if( datasetDimension() > 1 )
831         dataset /= datasetDimension();
832     return qVariantValue<QBrush>(
833         attributesModel()->data(
834             attributesModel()->mapFromSource( columnToIndex( dataset ) ),
835             DatasetBrushRole ) );
836 }
```

#### 9.18.4.12 QBrush AbstractDiagram::brush () const [inherited]

Retrieve the brush to be used for painting datapoints globally.

This will fall back automatically to the default settings, if there are no specific settings.

##### Returns:

The brush to use for painting.

Definition at line 822 of file KDChartAbstractDiagram.cpp.

References [KDChart::AbstractDiagram::attributesModel\(\)](#), [KDChart::AttributesModel::data\(\)](#), and [KDChart::DatasetBrushRole](#).

Referenced by [KDChart::AbstractDiagram::brush\(\)](#), [KDChart::TernaryPointDiagram::paint\(\)](#), [KDChart::TernaryLineDiagram::paint\(\)](#), [KDChart::RingDiagram::paint\(\)](#), [KDChart::PolarDiagram::paint\(\)](#), and [KDChart::AbstractDiagram::paintMarker\(\)](#).

```
823 {
824     return qVariantValue<QBrush>(
825         attributesModel()->data( DatasetBrushRole ) );
826 }
```

#### 9.18.4.13 const QPair< QPointF, QPointF > BarDiagram::calculateDataBoundaries () const [protected, virtual]

[reimplemented]

Implements [KDChart::AbstractDiagram](#).

Definition at line 294 of file KDChartBarDiagram.cpp.

References [KDChart::AbstractDiagram::checkInvariants\(\)](#), and [d](#).

```
295 {
296     if ( !checkInvariants(true) ) return QPair<QPointF, QPointF>( QPointF( 0, 0 ), QPointF( 0, 0 ) );
297
298     // note: calculateDataBoundaries() is ignoring the hidden flags.
299     // That's not a bug but a feature: Hiding data does not mean removing them.
300     // For totally removing data from KD Chart's view people can use e.g. a proxy model
301     // calculate boundaries for different line types Normal - Stacked - Percent - Default Normal
302     return d->implementor->calculateDataBoundaries();
303 }
```

#### 9.18.4.14 **bool AbstractDiagram::checkInvariants (bool *justReturnTheStatus* = false) const** [protected, virtual, inherited]

Definition at line 1060 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::coordinatePlane().

Referenced by KDChart::RingDiagram::calculateDataBoundaries(), KDChart::PolarDiagram::calculateDataBoundaries(), KDChart::Plotter::calculateDataBoundaries(), KDChart::PieDiagram::calculateDataBoundaries(), KDChart::LineDiagram::calculateDataBoundaries(), calculateDataBoundaries(), KDChart::RingDiagram::paint(), KDChart::PolarDiagram::paint(), KDChart::Plotter::paint(), KDChart::PieDiagram::paint(), KDChart::LineDiagram::paint(), paint(), KDChart::AbstractDiagram::paintDataValueTexts(), KDChart::AbstractDiagram::paintMarker(), and KDChart::AbstractDiagram::paintMarkers().

```

1061 {
1062     if( ! justReturnTheStatus ){
1063         Q_ASSERT_X ( model(), "AbstractDiagram::checkInvariants()",
1064                     "There is no usable model set, for the diagram." );
1065
1066         Q_ASSERT_X ( coordinatePlane(), "AbstractDiagram::checkInvariants()",
1067                     "There is no usable coordinate plane set, for the diagram." );
1068     }
1069     return model() && coordinatePlane();
1070 }
```

#### 9.18.4.15 **BarDiagram \* BarDiagram::clone () const** [virtual]

Creates an exact copy of this diagram.

Definition at line 78 of file KDChartBarDiagram.cpp.

References BarDiagram(), d, setType(), and type().

```

79 {
80
81     BarDiagram* newDiagram = new BarDiagram( new Private( *d ) );
82     newDiagram->setType( type() );
83     return newDiagram;
84 }
```

#### 9.18.4.16 **QModelIndex AbstractDiagram::columnToIndex (int *column*) const** [protected, inherited]

Definition at line 309 of file KDChartAbstractDiagram.cpp.

Referenced by barAttributes(), KDChart::AbstractDiagram::brush(), KDChart::AbstractDiagram::dataValueAttributes(), KDChart::AbstractDiagram::isHidden(), KDChart::Plotter::lineAttributes(), KDChart::LineDiagram::lineAttributes(), KDChart::AbstractDiagram::pen(), KDChart::AbstractPieDiagram::pieAttributes(), threeDBarAttributes(), KDChart::Plotter::threeDLineAttributes(), KDChart::LineDiagram::threeDLineAttributes(), and KDChart::AbstractPieDiagram::threeDPieAttributes().

```

310 {    // FIXME (Mirko): shouldn't this be headerData? instead of the index for the first row?
311     if( model() )
312         return QModelIndex( model()->index( 0, column, rootIndex() ) );
313     return QModelIndex();
314 }
```

#### 9.18.4.17 bool AbstractDiagram::compare (const [AbstractDiagram](#) \* other) const

[inherited]

Returns true if both diagrams have the same settings.

Definition at line 129 of file KDChartAbstractDiagram.cpp.

References [KDChart::AbstractDiagram::allowOverlappingDataValueTexts\(\)](#), [KDChart::AbstractDiagram::antiAliasing\(\)](#), [KDChart::AbstractDiagram::attributesModel\(\)](#), [KDChart::AttributesModel::compare\(\)](#), [KDChart::AbstractDiagram::datasetDimension\(\)](#), and [KDChart::AbstractDiagram::percentMode\(\)](#).

```

130 {
131     if( other == this ) return true;
132     if( ! other ){
133         //qDebug() << "AbstractDiagram::compare() cannot compare to Null pointer";
134         return false;
135     }
136     /*
137     qDebug() << "\n                AbstractDiagram::compare() QAbstractScrollArea:";
138         // compare QAbstractScrollArea properties
139     qDebug() <<
140         ((horizontalScrollBarPolicy() == other->horizontalScrollBarPolicy()) &&
141         (verticalScrollBarPolicy() == other->verticalScrollBarPolicy()));
142     qDebug() << "AbstractDiagram::compare() QFrame:";
143         // compare QFrame properties
144     qDebug() <<
145         ((frameShadow() == other->frameShadow()) &&
146         (frameShape() == other->frameShape()) &&
147         (frameWidth() == other->frameWidth()) &&
148         (lineWidth() == other->lineWidth()) &&
149         (midLineWidth() == other->midLineWidth()));
150     qDebug() << "AbstractDiagram::compare() QAbstractItemView:";
151         // compare QAbstractItemView properties
152     qDebug() <<
153         ((alternatingRowColors() == other->alternatingRowColors()) &&
154         (hasAutoScroll() == other->hasAutoScroll()) &&
155 #if QT_VERSION > 0x040199
156         (dragDropMode() == other->dragDropMode()) &&
157         (dragDropOverwriteMode() == other->dragDropOverwriteMode()) &&
158         (horizontalScrollMode() == other->horizontalScrollMode()) &&
159         (verticalScrollMode() == other->verticalScrollMode()) &&
160 #endif
161         (dragEnabled() == other->dragEnabled()) &&
162         (editTriggers() == other->editTriggers()) &&
163         (iconSize() == other->iconSize()) &&
164         (selectionBehavior() == other->selectionBehavior()) &&
165         (selectionMode() == other->selectionMode()) &&
166         (showDropIndicator() == other->showDropIndicator()) &&
167         (tabKeyNavigation() == other->tabKeyNavigation()) &&
168         (textElideMode() == other->textElideMode()));
169     qDebug() << "AbstractDiagram::compare() AttributesModel: ";
170         // compare all of the properties stored in the attributes model
171     qDebug() << attributesModel()->compare( other->attributesModel() );
172     qDebug() << "AbstractDiagram::compare() own:";
173         // compare own properties
174     qDebug() <<
175         ((rootIndex().column() == other->rootIndex().column()) &&
176         (rootIndex().row() == other->rootIndex().row()) &&
177         (allowOverlappingDataValueTexts() == other->allowOverlappingDataValueTexts()) &&
178         (antiAliasing() == other->antiAliasing()) &&
179         (percentMode() == other->percentMode()) &&
180         (datasetDimension() == other->datasetDimension()));
181     */
182     return // compare QAbstractScrollArea properties
183         (horizontalScrollBarPolicy() == other->horizontalScrollBarPolicy()) &&

```

```

184         (verticalScrollBarPolicy() == other->verticalScrollBarPolicy()) &&
185         // compare QFrame properties
186         (frameShadow() == other->frameShadow()) &&
187         (frameShape() == other->frameShape()) &&
188 // frameWidth is a read-only property defined by the style, it should not be in here:
189         // (frameWidth() == other->frameWidth()) &&
190         (lineWidth() == other->lineWidth()) &&
191         (midLineWidth() == other->midLineWidth()) &&
192         // compare QAbstractItemView properties
193         (alternatingRowColors() == other->alternatingRowColors()) &&
194         (hasAutoScroll() == other->hasAutoScroll()) &&
195 #if QT_VERSION > 0x040199
196         (dragDropMode() == other->dragDropMode()) &&
197         (dragDropOverwriteMode() == other->dragDropOverwriteMode()) &&
198         (horizontalScrollMode() == other->horizontalScrollMode()) &&
199         (verticalScrollMode() == other->verticalScrollMode()) &&
200 #endif
201         (dragEnabled() == other->dragEnabled()) &&
202         (editTriggers() == other->editTriggers()) &&
203         (iconSize() == other->iconSize()) &&
204         (selectionBehavior() == other->selectionBehavior()) &&
205         (selectionMode() == other->selectionMode()) &&
206         (showDropIndicator() == other->showDropIndicator()) &&
207         (tabKeyNavigation() == other->tabKeyNavigation()) &&
208         (textElideMode() == other->textElideMode()) &&
209         // compare all of the properties stored in the attributes model
210         attributesModel()->compare( other->attributesModel() ) &&
211         // compare own properties
212         (rootIndex().column() == other->rootIndex().column()) &&
213         (rootIndex().row() == other->rootIndex().row()) &&
214         (allowOverlappingDataValueTexts() == other->allowOverlappingDataValueTexts()) &&
215         (antiAliasing() == other->antiAliasing()) &&
216         (percentMode() == other->percentMode()) &&
217         (datasetDimension() == other->datasetDimension());
218 }

```

#### 9.18.4.18 bool AbstractCartesianDiagram::compare (const [AbstractCartesianDiagram](#) \* other) const [inherited]

Returns true if both diagrams have the same settings.

Definition at line 46 of file KDChartAbstractCartesianDiagram.cpp.

References [KDChart::AbstractCartesianDiagram::referenceDiagram\(\)](#), and [KDChart::AbstractCartesianDiagram::referenceDiagramOffset\(\)](#).

```

47 {
48     if( other == this ) return true;
49     if( ! other ){
50         //qDebug() << "AbstractCartesianDiagram::compare() cannot compare to Null pointer";
51         return false;
52     }
53     /*
54     qDebug() << "\n                AbstractCartesianDiagram::compare() : ";
55         // compare own properties
56     qDebug() <<
57         ((referenceDiagram() == other->referenceDiagram()) &&
58         ((! referenceDiagram()) || (referenceDiagramOffset() == other->referenceDiagramOffset())));
59     */
60     return // compare the base class
61         ( static_cast<const AbstractDiagram*>(this)->compare( other ) ) &&
62         // compare own properties
63         (referenceDiagram() == other->referenceDiagram()) &&
64         ((! referenceDiagram()) || (referenceDiagramOffset() == other->referenceDiagramOffset()));

```

```
65 }
```

#### 9.18.4.19 bool BarDiagram::compare (const [BarDiagram](#) \* other) const

Returns true if both diagrams have the same settings.

Definition at line 86 of file KDChartBarDiagram.cpp.

References [type\(\)](#).

```
87 {
88     if( other == this ) return true;
89     if( ! other ){
90         return false;
91     }
92
93     return // compare the base class
94           ( static_cast<const AbstractCartesianDiagram*>(this)->compare( other ) ) &&
95           // compare own properties
96           (type() == other->type());
97 }
```

#### 9.18.4.20 [AbstractCoordinatePlane](#) \* AbstractDiagram::coordinatePlane () const [inherited]

The coordinate plane associated with the diagram.

This determines how coordinates in value space are mapped into pixel space. By default this is a [Cartesian-CoordinatePlane](#).

##### Returns:

The coordinate plane associated with the diagram.

Definition at line 220 of file KDChartAbstractDiagram.cpp.

References [d](#).

Referenced by [KDChart::AbstractDiagram::checkInvariants\(\)](#), [KDChart::AbstractCartesianDiagram::layoutPlanes\(\)](#), [KDChart::PolarDiagram::paint\(\)](#), [KDChart::AbstractDiagram::paintData-ValueTexts\(\)](#), [KDChart::AbstractDiagram::paintMarkers\(\)](#), [KDChart::AbstractPolarDiagram::polar-CoordinatePlane\(\)](#), and [KDChart::AbstractCartesianDiagram::setCoordinatePlane\(\)](#).

```
221 {
222     return d->plane;
223 }
```

#### 9.18.4.21 const QPair< QPointF, QPointF > AbstractDiagram::dataBoundaries () const [inherited]

Return the bottom left and top right data point, that the diagram will display (unless the grid adjusts these values).

This method returns a cached result of calculations done by [calculateDataBoundaries](#). Classes derived from [AbstractDiagram](#) must implement the [calculateDataBoundaries](#) function, to specify their own way of

calculating the data boundaries. If derived classes want to force recalculation of the data boundaries, they can call [setDataBoundariesDirty\(\)](#)

Returned value is in diagram coordinates.

Definition at line 225 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::calculateDataBoundaries(), and d.

Referenced by KDChart::CartesianCoordinatePlane::getRawDataBoundingRectFromDiagrams(), KDChart::PolarCoordinatePlane::layoutDiagrams(), KDChart::Plotter::paint(), KDChart::LineDiagram::paint(), and paint().

```

226 {
227     if( d->ataboundariesDirty ){
228         d->ataboundaries = calculateDataBoundaries ();
229         d->ataboundariesDirty = false;
230     }
231     return d->ataboundaries;
232 }
```

#### 9.18.4.22 void AbstractDiagram::dataChanged (const QModelIndex & *topLeft*, const QModelIndex & *bottomRight*) [virtual, inherited]

[reimplemented]

Definition at line 330 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::setDataBoundariesDirty().

```

332 {
333     Q_UNUSED( topLeft );
334     Q_UNUSED( bottomRight );
335     // We are still too dumb to do intelligent updates...
336     setDataBoundariesDirty();
337     scheduleDelayedItemsLayout();
338 }
```

#### 9.18.4.23 void KDChart::AbstractDiagram::dataHidden () [signal, inherited]

This signal is emitted, when the hidden status of at least one data cell was (un)set.

Referenced by KDChart::AbstractDiagram::setHidden().

#### 9.18.4.24 QList< QBrush > AbstractDiagram::datasetBrushes () const [inherited]

The set of dataset brushes currently used, for use in legends, etc.

#### Note:

Cell-level override brushes, if set, take precedence over the dataset values, so you might need to check these too, in order to find the brush, that is used for a single cell.

#### Returns:

The current set of dataset brushes.

Definition at line 1018 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::attributesModelRootIndex(), KDChart::AttributesModel::columnCount(), KDChart::DatasetBrushRole, and KDChart::AbstractDiagram::datasetDimension().

Referenced by KDChart::Legend::setBrushesFromDiagram().

```

1019 {
1020     QList<QBrush> ret;
1021     if( model() == 0 )
1022         return ret;
1023
1024     const int columnCount = attributesModel()->columnCount(attributesModelRootIndex());
1025     for( int i = 0; i < columnCount / datasetDimension(); ++i )
1026         ret << QVariantValue<QBrush>( attributesModel()->headerData( i, Qt::Vertical, DatasetBrushRole ));
1027
1028     return ret;
1029 }
```

#### 9.18.4.25 int AbstractDiagram::datasetDimension () const [inherited]

The dataset dimension of a diagram determines, how many value dimensions it expects each datapoint to have.

For each dimension it will expect one column of values in the model. If the dimensionality is 1, automatic values will be used for the abscissa.

For example a diagram with the default dimension of 1, will have one column per datapoint (the y values) and will use automatic values for the x axis (1, 2, 3, ... n). If the dimension is 2, the diagram will use the first, (and the third, fifth, etc) columns as X values, and the second, (and the fourth, sixth, etc) column as Y values.

##### Returns:

The dataset dimension of the diagram.

Definition at line 1072 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::brush(), KDChart::AbstractDiagram::compare(), KDChart::AbstractDiagram::datasetBrushes(), KDChart::AbstractDiagram::datasetLabels(), KDChart::AbstractDiagram::datasetMarkers(), KDChart::AbstractDiagram::datasetPens(), KDChart::LineDiagram::getCellValues(), KDChart::CartesianCoordinatePlane::getDataDimensionsList(), KDChart::TernaryPointDiagram::paint(), KDChart::TernaryLineDiagram::paint(), KDChart::AbstractDiagram::paintDataValueTexts(), KDChart::AbstractDiagram::paintMarkers(), KDChart::AbstractDiagram::pen(), KDChart::AbstractDiagram::setPen(), KDChart::Plotter::setType(), and KDChart::LineDiagram::setType().

```

1073 {
1074     return d->datasetDimension;
1075 }
```

#### 9.18.4.26 QStringList AbstractDiagram::datasetLabels () const [inherited]

The set of dataset labels currently displayed, for use in legends, etc.

**Returns:**

The set of dataset labels currently displayed.

Definition at line 1005 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::attributesModelRootIndex(), KDChart::AttributesModel::columnCount(), KDChart::AbstractDiagram::datasetDimension(), and KDChart::AttributesModel::headerData().

```

1006 {
1007     QStringList ret;
1008     if( model() == 0 )
1009         return ret;
1010
1011     const int columnCount = attributesModel()->columnCount(attributesModelRootIndex());
1012     for( int i = 0; i < columnCount / datasetDimension(); ++i )
1013         ret << attributesModel()->headerData( i, Qt::Horizontal, Qt::DisplayRole ).toString();
1014
1015     return ret;
1016 }
```

#### 9.18.4.27 QList< [MarkerAttributes](#) > AbstractDiagram::datasetMarkers () const

[inherited]

The set of dataset markers currently used, for use in legends, etc.

**Note:**

Cell-level override markers, if set, take precedence over the dataset values, so you might need to check these too, in order to find the marker, that is shown for a single cell.

**Returns:**

The current set of dataset brushes.

Definition at line 1044 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::attributesModelRootIndex(), KDChart::AttributesModel::columnCount(), KDChart::AbstractDiagram::datasetDimension(), and KDChart::DataValueLabelAttributesRole.

```

1045 {
1046     QList<MarkerAttributes> ret;
1047     if( model() == 0 )
1048         return ret;
1049
1050     const int columnCount = attributesModel()->columnCount(attributesModelRootIndex());
1051     for( int i = 0; i < columnCount / datasetDimension(); ++i )
1052     {
1053         const DataValueAttributes a =
1054             qVariantValue<DataValueAttributes>( attributesModel()->headerData( i, Qt::Vertical, DataValueLabelAttributesRole ) );
1055         ret << a.markerAttributes();
1056     }
1057     return ret;
1058 }
```



**9.18.4.28 QList< QPen > AbstractDiagram::datasetPens () const** [inherited]

The set of dataset pens currently used, for use in legends, etc.

**Note:**

Cell-level override pens, if set, take precedence over the dataset values, so you might need to check these too, in order to find the pens, that is used for a single cell.

**Returns:**

The current set of dataset pens.

Definition at line 1031 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::attributesModelRootIndex(), KDChart::AttributesModel::columnCount(), KDChart::AbstractDiagram::datasetDimension(), and KDChart::DatasetPenRole.

```

1032 {
1033     QList<QPen> ret;
1034     if( model() == 0 )
1035         return ret;
1036
1037     const int columnCount = attributesModel()->columnCount(attributesModelRootIndex());
1038     for( int i = 0; i < columnCount / datasetDimension(); ++i )
1039         ret << qVariantValue<QPen>( attributesModel()->headerData( i, Qt::Vertical, DatasetPenRole ) );
1040
1041     return ret;
1042 }
```

**9.18.4.29 DataValueAttributes AbstractDiagram::dataValueAttributes (const QModelIndex & index) const** [inherited]

Retrieve the [DataValueAttributes](#) for the given index.

This will fall back automatically to what was set at dataset or model level, if there are no datapoint specific settings.

**Parameters:**

*index* The datapoint to retrieve the attributes for.

**Returns:**

The [DataValueAttributes](#) for the given index.

Definition at line 421 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AttributesModel::data(), and KDChart::DataValueLabelAttributesRole.

```

422 {
423     return qVariantValue<DataValueAttributes>(
424         attributesModel()->data( attributesModel()->mapFromSource(index),
425                                 KDChart::DataValueLabelAttributesRole ) );
426 }
```

#### 9.18.4.30 [DataValueAttributes](#) AbstractDiagram::dataValueAttributes (int *column*) const [inherited]

Retrieve the [DataValueAttributes](#) for the given dataset.

This will fall back automatically to what was set at model level, if there are no dataset specific settings.

##### Parameters:

*column* The dataset to retrieve the attributes for.

##### Returns:

The [DataValueAttributes](#) for the given dataset.

Definition at line 414 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::columnToIndex(), KDChart::AttributesModel::data(), and KDChart::DataValueLabelAttributesRole.

```
415 {
416     return qVariantValue<DataValueAttributes>(
417         attributesModel()->data( attributesModel()->mapFromSource(columnToIndex( column )),
418         KDChart::DataValueLabelAttributesRole ) );
419 }
```

#### 9.18.4.31 [DataValueAttributes](#) AbstractDiagram::dataValueAttributes () const [inherited]

Retrieve the [DataValueAttributes](#) specified globally.

This will fall back automatically to the default settings, if there are no specific settings.

##### Returns:

The global [DataValueAttributes](#).

Definition at line 408 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DataValueLabelAttributesRole, and KDChart::AttributesModel::modelData().

Referenced by KDChart::AbstractDiagram::paintDataValueText(), KDChart::AbstractDiagram::paintMarker(), and KDChart::TernaryLineDiagram::TernaryLineDiagram().

```
409 {
410     return qVariantValue<DataValueAttributes>(
411         attributesModel()->modelData( KDChart::DataValueLabelAttributesRole ) );
412 }
```

#### 9.18.4.32 void AbstractDiagram::doItemsLayout () [virtual, inherited]

[reimplemented]

Definition at line 321 of file KDChartAbstractDiagram.cpp.

References d, and KDChart::AbstractDiagram::update().

```

322 {
323     if ( d->plane ) {
324         d->plane->layoutDiagrams();
325         update();
326     }
327     QAbstractItemView::doItemsLayout();
328 }

```

#### 9.18.4.33 int AbstractDiagram::horizontalOffset () const [virtual, inherited]

[reimplemented]

Definition at line 950 of file KDChartAbstractDiagram.cpp.

```

951 { return 0; }

```

#### 9.18.4.34 QModelIndex AbstractDiagram::indexAt (const QPoint & *point*) const [virtual, inherited]

[reimplemented]

Definition at line 1098 of file KDChartAbstractDiagram.cpp.

References d.

```

1099 {
1100     return d->indexAt( point );
1101 }

```

#### 9.18.4.35 QModelIndexList AbstractDiagram::indexesAt (const QPoint & *point*) const [inherited]

This method is added alongside with indexAt from QAIM, since in kdchart multiple indexes can be displayed at the same spot.

Definition at line 1103 of file KDChartAbstractDiagram.cpp.

References d.

```

1104 {
1105     return d->indexesAt( point );
1106 }

```

#### 9.18.4.36 bool AbstractDiagram::isHidden (const QModelIndex & *index*) const [inherited]

Retrieve the hidden status for the given index.

This will fall back automatically to what was set at dataset or diagram level, if there are no datapoint specific settings.

##### Parameters:

*index* The datapoint to retrieve the hidden status for.

**Returns:**

The hidden status for the given index.

Definition at line 380 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AttributesModel::data(), and KDChart::DataHiddenRole.

```

381 {
382     return qVariantValue<bool>(
383         attributesModel()->data(
384             attributesModel()->mapFromSource(index),
385             DataHiddenRole ) );
386 }
```

**9.18.4.37 bool AbstractDiagram::isHidden (int *column*) const** [inherited]

Retrieve the hidden status for the given dataset.

This will fall back automatically to what was set at diagram level, if there are no dataset specific settings.

**Parameters:**

***column*** The dataset to retrieve the hidden status for.

**Returns:**

The hidden status for the given dataset.

Definition at line 373 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::columnToIndex(), KDChart::AttributesModel::data(), and KDChart::DataHiddenRole.

```

374 {
375     return qVariantValue<bool>(
376         attributesModel()->data(
377             attributesModel()->mapFromSource(columnToIndex( column )),
378             DataHiddenRole ) );
379 }
```

**9.18.4.38 bool AbstractDiagram::isHidden () const** [inherited]

Retrieve the hidden status specified globally.

This will fall back automatically to the default settings (= not hidden), if there are no specific settings.

**Returns:**

The global hidden status.

Definition at line 367 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DataHiddenRole, and KDChart::AttributesModel::modelData().

Referenced by KDChart::LineDiagram::valueForCellTesting().

```

368 {
369     return qVariantValue<bool>(
370         attributesModel()->modelData( DataHiddenRole ) );
371 }

```

#### 9.18.4.39 bool AbstractDiagram::isIndexHidden (const QModelIndex & *index*) const

[virtual, inherited]

[reimplemented]

Definition at line 956 of file KDChartAbstractDiagram.cpp.

```

957 { return true; }

```

#### 9.18.4.40 QStringList AbstractDiagram::itemRowLabels () const

[inherited]

The set of item row labels currently displayed, for use in Abscissa axes, etc.

##### Returns:

The set of item row labels currently displayed.

Definition at line 989 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::attributesModelRootIndex(), KDChart::AttributesModel::rowCount(), KDChart::AbstractDiagram::unitPrefix(), and KDChart::AbstractDiagram::unitSuffix().

```

990 {
991     QStringList ret;
992     if( model() ){
993         //qDebug() << "AbstractDiagram::itemRowLabels(): " << attributesModel()->rowCount(attributesModelRootIndex());
994         const int rowCount = attributesModel()->rowCount(attributesModelRootIndex());
995         for( int i = 0; i < rowCount; ++i ){
996             //qDebug() << "item row label: " << attributesModel()->headerData( i, Qt::Vertical, Qt::DisplayRole ).toString();
997             ret << unitPrefix( i, Qt::Horizontal, true ) +
998                 attributesModel()->headerData( i, Qt::Vertical, Qt::DisplayRole ).toString() +
999                 unitSuffix( i, Qt::Horizontal, true );
1000         }
1001     }
1002     return ret;
1003 }

```

#### 9.18.4.41 void KDChart::AbstractDiagram::layoutChanged ([AbstractDiagram](#) \*)

[signal, inherited]

Diagrams are supposed to emit this signal, when the layout of one of their element changes.

Layouts can change, for example, when axes are added or removed, or when the configuration was changed in a way that the axes or the diagram itself are displayed in a different geometry. Changes in the diagrams coordinate system also result in the [layoutChanged\(\)](#) signal being emitted.

Referenced by KDChart::AbstractDiagram::setDatasetDimension(), KDChart::AbstractPieDiagram::setPieAttributes(), KDChart::AbstractPieDiagram::setThreeDPieAttributes(), KDChart::Plotter::setType(), KDChart::LineDiagram::setType(), and setType().

#### 9.18.4.42 void KDChart::AbstractCartesianDiagram::layoutPlanes () [virtual, inherited]

Triggers layouting of all coordinate planes on the current chart.

Normally you don't need to call this method. It's handled automatically for you.

Definition at line 115 of file KDChartAbstractCartesianDiagram.cpp.

References KDChart::AbstractDiagram::coordinatePlane(), and KDChart::AbstractCoordinatePlane::layoutPlanes().

Referenced by KDChart::AbstractCartesianDiagram::addAxis(), and KDChart::AbstractCartesianDiagram::takeAxis().

```

116 {
117     //qDebug() << "KDChart::AbstractCartesianDiagram::layoutPlanes()";
118     AbstractCoordinatePlane* plane = coordinatePlane();
119     if( plane ){
120         plane->layoutPlanes();
121         //qDebug() << "KDChart::AbstractCartesianDiagram::layoutPlanes() OK";
122     }
123 }
```

#### 9.18.4.43 void KDChart::AbstractDiagram::modelsChanged () [signal, inherited]

This signal is emitted, when either the model or the [AttributesModel](#) is replaced.

Referenced by KDChart::AbstractDiagram::setAttributesModel(), and KDChart::AbstractDiagram::setModel().

#### 9.18.4.44 QModelIndex AbstractDiagram::moveCursor (CursorAction *cursorAction*, Qt::KeyboardModifiers *modifiers*) [virtual, inherited]

[reimplemented]

Definition at line 947 of file KDChartAbstractDiagram.cpp.

```

948 { return QModelIndex(); }
```

#### 9.18.4.45 const int BarDiagram::numberOfAbscissaSegments () const [virtual]

[reimplemented]

Implements [KDChart::AbstractCartesianDiagram](#).

Definition at line 338 of file KDChartBarDiagram.cpp.

References KDChart::AbstractDiagram::attributesModelRootIndex(), and d.

```

339 {
340     return d->attributesModel->rowCount(attributesModelRootIndex());
341 }
```

**9.18.4.46** `const int BarDiagram::numberOfOrdinateSegments () const` [virtual]

[reimplemented]

Implements [KDChart::AbstractCartesianDiagram](#).

Definition at line 343 of file KDChartBarDiagram.cpp.

References [KDChart::AbstractDiagram::attributesModelRootIndex\(\)](#), and [d](#).

```

344 {
345     return d->attributesModel->columnCount (attributesModelRootIndex());
346 }
```

**9.18.4.47** `void BarDiagram::paint (PaintContext * paintContext)` [protected, virtual]

Draw the diagram contents to the rectangle and painter, that are passed in as part of the paint context.

**Parameters:***paintContext* All information needed for painting.Implements [KDChart::AbstractDiagram](#).

Definition at line 314 of file KDChartBarDiagram.cpp.

References [KDChart::AbstractDiagram::checkInvariants\(\)](#), [KDChart::PaintContext::coordinatePlane\(\)](#), [d](#), [KDChart::AbstractDiagram::dataBoundaries\(\)](#), [KDChart::PaintContext::painter\(\)](#), [KDChart::PaintContext::setCoordinatePlane\(\)](#), and [KDChart::AbstractCoordinatePlane::sharedAxisMasterPlane\(\)](#).Referenced by [paintEvent\(\)](#).

```

315 {
316     if ( !checkInvariants( true ) ) return;
317     if ( !AbstractGrid::isBoundariesValid(dataBoundaries()) ) return;
318     const PainterSaver p( ctx->painter() );
319     if( model()->rowCount() == 0 || model()->columnCount() == 0 )
320         return; // nothing to paint for us
321
322     AbstractCoordinatePlane* const plane = ctx->coordinatePlane();
323     ctx->setCoordinatePlane( plane->sharedAxisMasterPlane( ctx->painter() ) );
324
325     // paint different bar types Normal - Stacked - Percent - Default Normal
326     d->implementor->paint( ctx );
327
328     ctx->setCoordinatePlane( plane );
329 }
```

**9.18.4.48** `void AbstractDiagram::paintDataValueText (QPainter * painter, const QModelIndex & index, const QPointF & pos, double value)` [inherited]

Definition at line 468 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::DataValueAttributes::dataLabel\(\)](#), [KDChart::AbstractDiagram::dataValueAttributes\(\)](#), [KDChart::DataValueAttributes::decimalDigits\(\)](#), [KDChart::DataValueAttributes::isVisible\(\)](#), [KDChartEnums::MeasureOrientationMinimum](#), [KDChart::DataValueAttributes::position\(\)](#), [KDChart::DataValueAttributes::prefix\(\)](#), [KDChart::DataValueAttributes::suffix\(\)](#), and [KDChart::DataValueAttributes::textAttributes\(\)](#).

Referenced by `KDChart::RingDiagram::paint()`, `KDChart::PolarDiagram::paint()`, and `KDChart::AbstractDiagram::paintDataValueTexts()`.

```

472 {
473     // paint one data series
474     const DataValueAttributes a( dataValueAttributes(index) );
475     if ( !a.isVisible() ) return;
476
477     // handle decimal digits
478     int decimalDigits = a.decimalDigits();
479     int decimalPos = QString::number( value ).indexOf( QLatin1Char( '.' ) );
480     QString roundedValue;
481     if ( a.dataLabel().isNull() ) {
482         if ( decimalPos > 0 && value != 0 )
483             roundedValue = roundValues( value, decimalPos, decimalDigits );
484         else
485             roundedValue = QString::number( value );
486     } else
487         roundedValue = a.dataLabel();
488     // handle prefix and suffix
489     if ( !a.prefix().isNull() )
490         roundedValue.prepend( a.prefix() );
491
492     if ( !a.suffix().isNull() )
493         roundedValue.append( a.suffix() );
494
495     const TextAttributes ta( a.textAttributes() );
496     // FIXME draw the non-text bits, background, etc
497     if ( ta.isVisible() ) {
498
499         QPointF pt( pos );
500         /* for debugging:
501         PainterSaver painterSaver( painter );
502         painter->setPen( Qt::black );
503         painter->drawLine( pos - QPointF( 1,1), pos + QPointF( 1,1) );
504         painter->drawLine( pos - QPointF(-1,1), pos + QPointF(-1,1) );
505         */
506
507         QTextDocument doc;
508         if( Qt::mightBeRichText( roundedValue ) )
509             doc.setHtml( roundedValue );
510         else
511             doc.setPlainText( roundedValue );
512
513         const RelativePosition relPos( a.position( value >= 0.0 ) );
514         const Qt::Alignment alignBottomLeft = Qt::AlignBottom | Qt::AlignLeft;
515         const QFont calculatedFont( ta.calculatedFont( d->plane, KDChartEnums::MeasureOrientationMinimum ) );
516         const QRectF boundRect( d->cachedFontMetrics( calculatedFont, painter->device() )->boundingRect( roundedValue, calculatedFont ) );
517
518         // To place correctly
519         pt.ry() -= boundRect.height();
520
521         //qDebug() << "calculatedFont's point size:" << calculatedFont.pointSizeF();
522         // adjust the text start point position, if alignment is not Bottom/Left
523         if( (relPos.alignment() & alignBottomLeft) != alignBottomLeft ){
524             if( relPos.alignment() & Qt::AlignRight )
525                 pt.rx() -= boundRect.width();
526             else if( relPos.alignment() & Qt::AlignHCenter )
527                 pt.rx() -= 0.5 * boundRect.width();
528
529             if( relPos.alignment() & Qt::AlignTop )
530                 pt.ry() += boundRect.height();
531             else if( relPos.alignment() & Qt::AlignVCenter )
532                 pt.ry() += 0.5 * boundRect.height();
533         }
534
535         // FIXME draw the non-text bits, background, etc

```



```

536
537         if ( a.showRepetitiveDataLabels() ||
538             pos.x() <= d->lastX ||
539             d->lastRoundedValue != roundedValue ) {
540             d->lastRoundedValue = roundedValue;
541             d->lastX = pos.x();
542             PainterSaver painterSaver( painter );
543
544             doc.setDefaultFont( calculatedFont );
545             QAbstractTextDocumentLayout::PaintContext context;
546             context.palette = palette();
547             context.palette.setColor( QPalette::Text, ta.pen().color() );
548
549             painter->translate( pt );
550             painter->rotate( ta.rotation() );
551
552             QAbstractTextDocumentLayout* layout = doc.documentLayout();
553             layout->draw( painter, context );
554         }
555     }
556 }

```

#### 9.18.4.49 void AbstractDiagram::paintDataValueTexts (QPainter \*painter) [protected, virtual, inherited]

Definition at line 584 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::checkInvariants(), KDChart::AbstractDiagram::coordinatePlane(), KDChart::AbstractDiagram::datasetDimension(), KDChart::AbstractDiagram::paintDataValueText(), and KDChart::AbstractCoordinatePlane::translate().

```

585 {
586     if ( !checkInvariants() ) return;
587     const int rowCount = model()->rowCount( rootIndex() );
588     const int columnCount = model()->columnCount( rootIndex() );
589     for ( int i=datasetDimension()-1; i<columnCount; i += datasetDimension() ) {
590         for ( int j=0; j< rowCount; ++j ) {
591             const QModelIndex index = model()->index( j, i, rootIndex() );
592             double value = model()->data( index ).toDouble();
593             const QPointF pos = coordinatePlane()->translate( QPointF( j, value ) );
594             paintDataValueText( painter, index, pos, value );
595         }
596     }
597 }

```

#### 9.18.4.50 void BarDiagram::paintEvent (QPaintEvent \*) [protected]

Definition at line 305 of file KDChartBarDiagram.cpp.

References paint(), KDChart::PaintContext::setPainter(), and KDChart::PaintContext::setRectangle().

```

306 {
307     QPainter painter ( viewport() );
308     PaintContext ctx;
309     ctx.setPainter ( &painter );
310     ctx.setRectangle( QRectF ( 0, 0, width(), height() ) );
311     paint ( &ctx );
312 }

```

#### 9.18.4.51 void AbstractDiagram::paintMarker (QPainter \* *painter*, const QModelIndex & *index*, const QPointF & *pos*) [inherited]

Definition at line 600 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::brush(), KDChart::AbstractDiagram::checkInvariants(), d, KDChart::AbstractDiagram::dataValueAttributes(), KDChart::MarkerAttributes::isVisible(), KDChart::DataValueAttributes::isVisible(), KDChart::DataValueAttributes::markerAttributes(), KDChart::MarkerAttributes::markerColor(), KDChart::MarkerAttributes::markerSize(), KDChart::AbstractDiagram::paintMarker(), and KDChart::MarkerAttributes::pen().

```

603 {
604     if ( !checkInvariants() ) return;
605     DataValueAttributes a = dataValueAttributes(index);
606     if ( !a.isVisible() ) return;
607     const MarkerAttributes &ma = a.markerAttributes();
608     if ( !ma.isVisible() ) return;
609
610     PainterSaver painterSaver( painter );
611     QSizeF maSize( ma.markerSize() );
612     QBrush indexBrush( brush( index ) );
613     QPen indexPen( ma.pen() );
614     if ( ma.markerColor().isValid() )
615         indexBrush.setColor( ma.markerColor() );
616
617     paintMarker( painter, ma, indexBrush, indexPen, pos, maSize );
618
619     // workaround: BC cannot be changed, otherwise we would pass the
620     // index down to next-lower paintMarker function. So far, we
621     // basically save a circle of radius maSize at pos in the
622     // reverseMapper. This means that ^^^ this version of paintMarker
623     // needs to be called to reverse-map the marker.
624     d->reverseMapper.addCircle( index.row(), index.column(), pos, 2 * maSize );
625 }
```

#### 9.18.4.52 void AbstractDiagram::paintMarker (QPainter \* *painter*, const MarkerAttributes & *markerAttributes*, const QBrush & *brush*, const QPen & *pen*, const QPointF & *point*, const QSizeF & *size*) [virtual, inherited]

Definition at line 627 of file KDChartAbstractDiagram.cpp.

References KDChart::MarkerAttributes::Marker1Pixel, KDChart::MarkerAttributes::Marker4Pixels, KDChart::MarkerAttributes::MarkerCircle, KDChart::MarkerAttributes::MarkerCross, KDChart::MarkerAttributes::MarkerDiamond, KDChart::MarkerAttributes::MarkerFastCross, KDChart::MarkerAttributes::MarkerRing, KDChart::MarkerAttributes::MarkerSquare, and KDChart::MarkerAttributes::markerStyle().

Referenced by KDChart::TernaryPointDiagram::paint(), KDChart::TernaryLineDiagram::paint(), KDChart::MarkerLayoutItem::paintIntoRect(), KDChart::AbstractDiagram::paintMarker(), and KDChart::AbstractDiagram::paintMarkers().

```

633 {
634     const QPen oldPen( painter->pen() );
635     // Pen is used to paint 4Pixels - 1 Pixel - Ring and FastCross types.
636     // make sure to use the brush color - see above in those cases.
637     const bool isFourPixels = (markerAttributes.markerStyle() == MarkerAttributes::Marker4Pixels);
638     if( isFourPixels || (markerAttributes.markerStyle() == MarkerAttributes::Marker1Pixel) ){
639         // for high-performance point charts with tiny point markers:
640         painter->setPen( QPen( brush.color().light() ) );
641         if( isFourPixels ){
```

```

642         const qreal x = pos.x();
643         const qreal y = pos.y();
644         painter->drawLine( QPointF(x-1.0,y-1.0),
645                           QPointF(x+1.0,y-1.0) );
646         painter->drawLine( QPointF(x-1.0,y),
647                           QPointF(x+1.0,y) );
648         painter->drawLine( QPointF(x-1.0,y+1.0),
649                           QPointF(x+1.0,y+1.0) );
650     }
651     painter->drawPoint( pos );
652 }else{
653     PainterSaver painterSaver( painter );
654     // we only a solid line surrounding the markers
655     QPen painterPen( pen );
656     painterPen.setStyle( Qt::SolidLine );
657     painter->setPen( painterPen );
658     painter->setBrush( brush );
659     painter->setRenderHint ( QPainter::Antialiasing );
660     painter->translate( pos );
661     switch ( markerAttributes.markerStyle() ) {
662     case MarkerAttributes::MarkerCircle:
663         painter->drawEllipse( QRectF( 0 - maSize.height()/2, 0 - maSize.width()/2,
664                                     maSize.height(), maSize.width() ) );
665         break;
666     case MarkerAttributes::MarkerSquare:
667     {
668         QRectF rect( 0 - maSize.width()/2, 0 - maSize.height()/2,
669                     maSize.width(), maSize.height() );
670         painter->drawRect( rect );
671         break;
672     }
673     case MarkerAttributes::MarkerDiamond:
674     {
675         QVector <QPointF > diamondPoints;
676         QPointF top, left, bottom, right;
677         top = QPointF( 0, 0 - maSize.height()/2 );
678         left = QPointF( 0 - maSize.width()/2, 0 );
679         bottom = QPointF( 0, maSize.height()/2 );
680         right = QPointF( maSize.width()/2, 0 );
681         diamondPoints << top << left << bottom << right;
682         painter->drawPolygon( diamondPoints );
683         break;
684     }
685     // both handled on top of the method:
686     case MarkerAttributes::Marker1Pixel:
687     case MarkerAttributes::Marker4Pixels:
688         break;
689     case MarkerAttributes::MarkerRing:
690     {
691         painter->setPen( QPen( brush.color() ) );
692         painter->setBrush( Qt::NoBrush );
693         painter->drawEllipse( QRectF( 0 - maSize.height()/2, 0 - maSize.width()/2,
694                                     maSize.height(), maSize.width() ) );
695         break;
696     }
697     case MarkerAttributes::MarkerCross:
698     {
699         QRectF rect( maSize.width()*-0.5, maSize.height()*-0.2,
700                     maSize.width(), maSize.height()*0.4 );
701         painter->drawRect( rect );
702         rect.setTopLeft( QPointF( maSize.width()*-0.2, maSize.height()*-0.5 ) );
703         rect.setSize( QSizeF( maSize.width()*0.4, maSize.height() ) );
704         painter->drawRect( rect );
705         break;
706     }
707     case MarkerAttributes::MarkerFastCross:
708     {

```

```

709             QPointF left, right, top, bottom;
710             left = QPointF( -maSize.width()/2, 0 );
711             right = QPointF( maSize.width()/2, 0 );
712             top = QPointF( 0, -maSize.height()/2 );
713             bottom= QPointF( 0, maSize.height()/2 );
714             painter->setPen( QPen( brush.color() ) );
715             painter->drawLine( left, right );
716             painter->drawLine( top, bottom );
717             break;
718         }
719         default:
720             Q_ASSERT_X ( false, "paintMarkers()",
721                 "Type item does not match a defined Marker Type." );
722     }
723 }
724 painter->setPen( oldPen );
725 }

```

#### 9.18.4.53 void AbstractDiagram::paintMarkers (QPainter \* *painter*) [protected, virtual, inherited]

Definition at line 727 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::checkInvariants(), KDChart::AbstractDiagram::coordinatePlane(), KDChart::AbstractDiagram::datasetDimension(), KDChart::AbstractDiagram::paintMarker(), and KDChart::AbstractCoordinatePlane::translate().

```

728 {
729     if ( !checkInvariants() ) return;
730     const int rowCount = model()->rowCount( rootIndex() );
731     const int columnCount = model()->columnCount( rootIndex() );
732     for ( int i=datasetDimension()-1; i<columnCount; i += datasetDimension() ) {
733         for ( int j=0; j<rowCount; ++j ) {
734             const QModelIndex index = model()->index( j, i, rootIndex() );
735             double value = model()->data( index ).toDouble();
736             const QPointF pos = coordinatePlane()->translate( QPointF( j, value ) );
737             paintMarker( painter, index, pos );
738         }
739     }
740 }

```

#### 9.18.4.54 QPen AbstractDiagram::pen (const QModelIndex & *index*) const [inherited]

Retrieve the pen to be used, for painting the datapoint at the given index in the model.

##### Parameters:

*index* The index of the datapoint in the model.

##### Returns:

The pen to use for painting.

Definition at line 788 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AttributesModel::data(), KDChart::AbstractDiagram::datasetDimension(), KDChart::DatasetPenRole, and KDChart::AbstractDiagram::pen().

```

789 {
790     if( datasetDimension() > 1 )
791         return pen( index.column() );
792     return qVariantValue<QPen>(
793         attributesModel()->data(
794             attributesModel()->mapFromSource( index ),
795             DatasetPenRole ) );
796 }

```

#### 9.18.4.55 QPen AbstractDiagram::pen (int *dataset*) const [inherited]

Retrieve the pen to be used for the given dataset.

This will fall back automatically to what was set at model level, if there are no dataset specific settings.

##### Parameters:

*dataset* The dataset to retrieve the pen for.

##### Returns:

The pen to use for painting.

Definition at line 777 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::columnToIndex(), KDChart::AttributesModel::data(), KDChart::AbstractDiagram::datasetDimension(), and KDChart::DatasetPenRole.

```

778 {
779     if( datasetDimension() > 1 )
780         dataset /= datasetDimension();
781
782     return qVariantValue<QPen>(
783         attributesModel()->data(
784             attributesModel()->mapFromSource( columnToIndex( dataset ) ),
785             DatasetPenRole ) );
786 }

```

#### 9.18.4.56 QPen AbstractDiagram::pen () const [inherited]

Retrieve the pen to be used for painting datapoints globally.

This will fall back automatically to the default settings, if there are no specific settings.

##### Returns:

The pen to use for painting.

Definition at line 771 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AttributesModel::data(), and KDChart::DatasetPenRole.

Referenced by KDChart::TernaryPointDiagram::paint(), KDChart::TernaryLineDiagram::paint(), and KDChart::AbstractDiagram::pen().

```

772 {
773     return qVariantValue<QPen>(
774         attributesModel()->data( DatasetPenRole ) );
775 }

```

**9.18.4.57 bool AbstractDiagram::percentMode () const** [inherited]

Definition at line 462 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::compare(), and KDChart::CartesianCoordinatePlane::getDataDimensionsList().

```
463 {
464     return d->percent;
465 }
```

**9.18.4.58 void KDChart::AbstractDiagram::propertiesChanged ()** [signal, inherited]

Emitted upon change of a property of the Diagram.

Referenced by KDChart::Plotter::resetLineAttributes(), KDChart::LineDiagram::resetLineAttributes(), KDChart::AbstractDiagram::setAllowOverlappingDataValueTexts(), KDChart::AbstractDiagram::setAntiAliasing(), setBarAttributes(), KDChart::AbstractDiagram::setBrush(), KDChart::AbstractDiagram::setDataValueAttributes(), KDChart::Plotter::setLineAttributes(), KDChart::LineDiagram::setLineAttributes(), KDChart::AbstractDiagram::setPen(), KDChart::AbstractDiagram::setPercentMode(), setThreeDBarAttributes(), KDChart::Plotter::setThreeDLineAttributes(), KDChart::LineDiagram::setThreeDLineAttributes(), KDChart::Plotter::setType(), KDChart::LineDiagram::setType(), setType(), KDChart::Plotter::setValueTrackerAttributes(), and KDChart::LineDiagram::setValueTrackerAttributes().

**9.18.4.59 AbstractCartesianDiagram \* AbstractCartesianDiagram::referenceDiagram () const** [virtual, inherited]**Returns:**

this diagram's reference diagram

**See also:**

[setReferenceDiagram](#)

Definition at line 148 of file KDChartAbstractCartesianDiagram.cpp.

References d.

Referenced by KDChart::AbstractCartesianDiagram::compare(), and referenceDiagramIsBarDiagram().

```
149 {
150     return d->referenceDiagram;
151 }
```

**9.18.4.60 QPointF AbstractCartesianDiagram::referenceDiagramOffset () const** [virtual, inherited]**Returns:**

the relative offset of this diagram's reference diagram

**See also:**

[setReferenceDiagram](#)

Definition at line 153 of file KDChartAbstractCartesianDiagram.cpp.

References [d](#).

Referenced by [KDChart::AbstractCartesianDiagram::compare\(\)](#).

```
154 {  
155     return d->referenceDiagramOffset;  
156 }
```

#### 9.18.4.61 void BarDiagram::resize (const QSizeF & *area*) [virtual]

Called by the widget's `sizeEvent`.

Adjust all internal structures, that are calculated, depending on the size of the widget.

##### Parameters:

*area*

Implements [KDChart::AbstractDiagram](#).

Definition at line 331 of file KDChartBarDiagram.cpp.

References [d](#), and [KDChart::AbstractDiagram::setDataBoundariesDirty\(\)](#).

```
332 {  
333     d->compressor.setResolution( static_cast< int >( size.width() ),  
334                               static_cast< int >( size.height() ) );  
335     setDataBoundariesDirty();  
336 }
```

#### 9.18.4.62 void BarDiagram::resizeEvent (QResizeEvent \*) [protected]

Definition at line 289 of file KDChartBarDiagram.cpp.

```
290 {  
291  
292 }
```

#### 9.18.4.63 void AbstractDiagram::scrollTo (const QModelIndex & *index*, ScrollHint *hint* = EnsureVisible) [virtual, inherited]

[reimplemented]

Definition at line 942 of file KDChartAbstractDiagram.cpp.

```
943 {}
```

#### 9.18.4.64 void AbstractDiagram::setAllowOverlappingDataValueTexts (bool *allow*) [inherited]

Set whether data value labels are allowed to overlap.

##### Parameters:

*allow* True means that overlapping labels are allowed.

Definition at line 434 of file KDChartAbstractDiagram.cpp.

References [d](#), and [KDChart::AbstractDiagram::propertiesChanged\(\)](#).

```
435 {  
436     d->allowOverlappingDataValueTexts = allow;  
437     emit propertiesChanged();  
438 }
```

#### 9.18.4.65 void AbstractDiagram::setAntiAliasing (bool *enabled*) [inherited]

Set whether anti-aliasing is to be used while rendering this diagram.

##### Parameters:

*enabled* True means that AA is enabled.

Definition at line 445 of file KDChartAbstractDiagram.cpp.

References [d](#), and [KDChart::AbstractDiagram::propertiesChanged\(\)](#).

```
446 {  
447     d->antiAliasing = enabled;  
448     emit propertiesChanged();  
449 }
```

#### 9.18.4.66 void AbstractCartesianDiagram::setAttributesModel ([AttributesModel](#) \* *model*) [virtual, inherited]

Associate an [AttributesModel](#) with this diagram.

Note that the diagram does `_not_` take ownership of the [AttributesModel](#). This should thus only be used with [AttributesModels](#) that have been explicitly created by the user, and are owned by her. Setting an [AttributesModel](#) that is internal to another diagram is an error.

Correct:

```
AttributesModel *am = new AttributesModel( model, 0 );  
diagram1->setAttributesModel( am );  
diagram2->setAttributesModel( am );
```

Wrong:

```
diagram1->setAttributesModel( diagram2->attributesModel() );
```



**Parameters:**

*model* The [AttributesModel](#) to use for this diagram.

**See also:**

[AttributesModel](#), [usesExternalAttributesModel](#)

Reimplemented from [KDChart::AbstractDiagram](#).

Definition at line 170 of file `KDChartAbstractCartesianDiagram.cpp`.

References `KDChart::AbstractDiagram::attributesModel()`, `d`, and `KDChart::AbstractDiagram::setAttributesModel()`.

```
171 {
172     AbstractDiagram::setAttributesModel( model );
173     d->compressor.setModel( attributesModel() );
174 }
```

#### 9.18.4.67 void AbstractDiagram::setAttributesModelRootIndex (const QModelIndex &) [protected, inherited]

Definition at line 293 of file `KDChartAbstractDiagram.cpp`.

References `d`, and `KDChart::AbstractDiagram::setDataBoundariesDirty()`.

Referenced by `KDChart::AbstractDiagram::setRootIndex()`.

```
294 {
295     d->attributesModelRootIndex=idx;
296     setDataBoundariesDirty();
297     scheduleDelayedItemsLayout();
298 }
```

#### 9.18.4.68 void BarDiagram::setBarAttributes (const QModelIndex & index, const BarAttributes & a)

Sets the line attributes for the model index *index* to *ba*.

Definition at line 164 of file `KDChartBarDiagram.cpp`.

References `KDChart::AbstractDiagram::attributesModel()`, `KDChart::BarAttributesRole`, `d`, `KDChart::AbstractDiagram::propertiesChanged()`, and `KDChart::AttributesModel::setData()`.

```
165 {
166     attributesModel()->setData(
167         d->attributesModel->mapFromSource( index ),
168         qVariantFromValue( ba ),
169         BarAttributesRole );
170     emit propertiesChanged();
171 }
```

#### 9.18.4.69 void BarDiagram::setBarAttributes (int column, const BarAttributes & a)

Sets the bar attributes of data set *column* to *ba*.

Definition at line 152 of file KDChartBarDiagram.cpp.

References KDChart::BarAttributesRole, d, and KDChart::AbstractDiagram::propertiesChanged().

```
153 {
154     d->attributesModel->setHeaderData(
155         column, Qt::Vertical,
156         qVariantFromValue( ba ),
157         BarAttributesRole );
158     emit propertiesChanged();
159 }
```

#### 9.18.4.70 void BarDiagram::setBarAttributes (const BarAttributes & a)

Sets the global bar attributes to *ba*.

Definition at line 143 of file KDChartBarDiagram.cpp.

References KDChart::BarAttributesRole, d, and KDChart::AbstractDiagram::propertiesChanged().

```
144 {
145     d->attributesModel->setModelData( qVariantFromValue( ba ), BarAttributesRole );
146     emit propertiesChanged();
147 }
```

#### 9.18.4.71 void AbstractDiagram::setBrush (const QBrush & brush) [inherited]

Set the brush to be used, for painting all datasets in the model.

##### Parameters:

**brush** The brush to use.

Definition at line 806 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DatasetBrushRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setModelData().

```
807 {
808     attributesModel()->setModelData(
809         qVariantFromValue( brush ), DatasetBrushRole );
810     emit propertiesChanged();
811 }
```

#### 9.18.4.72 void AbstractDiagram::setBrush (int dataset, const QBrush & brush) [inherited]

Set the brush to be used, for painting the given dataset.

##### Parameters:

**dataset** The dataset's column in the model.

**brush** The brush to use.

Definition at line 813 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DatasetBrushRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setHeaderData().

```

814 {
815     attributesModel()->setHeaderData(
816         column, Qt::Vertical,
817         qVariantFromValue( brush ),
818         DatasetBrushRole );
819     emit propertiesChanged();
820 }
```

#### 9.18.4.73 void AbstractDiagram::setBrush (const QModelIndex & *index*, const QBrush & *brush*) [inherited]

Set the brush to be used, for painting the datapoint at the given index.

##### Parameters:

***index*** The datapoint's index in the model.

***brush*** The brush to use.

Definition at line 798 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DatasetBrushRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setData().

```

799 {
800     attributesModel()->setData(
801         attributesModel()->mapFromSource( index ),
802         qVariantFromValue( brush ), DatasetBrushRole );
803     emit propertiesChanged();
804 }
```

#### 9.18.4.74 void KDChart::AbstractCartesianDiagram::setCoordinatePlane ([AbstractCoordinatePlane](#) \* *plane*) [virtual, inherited]

[reimplemented]

Reimplemented from [KDChart::AbstractDiagram](#).

Definition at line 125 of file KDChartAbstractCartesianDiagram.cpp.

References KDChart::AbstractDiagram::coordinatePlane(), and KDChart::AbstractDiagram::setCoordinatePlane().

```

126 {
127     if( coordinatePlane() ) disconnect( coordinatePlane() );
128     AbstractDiagram::setCoordinatePlane( plane );
129
130     // show the axes, after all have been adjusted
131     // (because they might be dependend on each other)
132     /*
133     if( plane )
134         Q_FOREACH( CartesianAxis* axis, d->axesList )
135             axis->show();
136     */
137 }
```

```

136     else
137         Q_FOREACH( CartesianAxis* axis, d->axesList )
138             axis->hide();
139     */
140 }

```

#### 9.18.4.75 void AbstractDiagram::setDataBoundariesDirty () const [protected, inherited]

Definition at line 234 of file KDChartAbstractDiagram.cpp.

References `d`.

Referenced by `KDChart::AbstractDiagram::dataChanged()`, `KDChart::Plotter::resize()`, `KDChart::LineDiagram::resize()`, `resize()`, `KDChart::AbstractDiagram::setAttributesModel()`, `KDChart::AbstractDiagram::setAttributesModelRootIndex()`, `KDChart::AbstractDiagram::setDatasetDimension()`, `KDChart::AbstractDiagram::setModel()`, `setThreeDBarAttributes()`, `KDChart::Plotter::setThreeDLineAttributes()`, `KDChart::LineDiagram::setThreeDLineAttributes()`, `KDChart::Plotter::setType()`, `KDChart::LineDiagram::setType()`, and `setType()`.

```

235 {
236     d->databoundariesDirty = true;
237 }

```

#### 9.18.4.76 void AbstractDiagram::setDatasetDimension (int *dimension*) [inherited]

Sets the dataset dimension of the diagram.

See also:

[datasetDimension](#).

Parameters:

*dimension*

Definition at line 1077 of file KDChartAbstractDiagram.cpp.

References `d`, `KDChart::AbstractDiagram::layoutChanged()`, and `KDChart::AbstractDiagram::setDataBoundariesDirty()`.

Referenced by `KDChart::Widget::setType()`, `KDChart::TernaryLineDiagram::TernaryLineDiagram()`, and `KDChart::TernaryPointDiagram::TernaryPointDiagram()`.

```

1078 {
1079     if ( d->datasetDimension == dimension ) return;
1080     d->datasetDimension = dimension;
1081     setDataBoundariesDirty();
1082     emit layoutChanged( this );
1083 }

```

#### 9.18.4.77 void AbstractDiagram::setDataValueAttributes (const [DataValueAttributes](#) & *a*) [inherited]

Set the [DataValueAttributes](#) for all datapoints in the model.

**Parameters:**

*a* The attributes to set.

Definition at line 428 of file KDChartAbstractDiagram.cpp.

References `d`, `KDChart::DataValueLabelAttributesRole`, and `KDChart::AbstractDiagram::propertiesChanged()`.

```
429 {  
430     d->attributesModel->setModelData( qVariantFromValue( a ), DataValueLabelAttributesRole );  
431     emit propertiesChanged();  
432 }
```

**9.18.4.78 void AbstractDiagram::setDataValueAttributes (int *dataset*, const [DataValueAttributes](#) & *a*)** [inherited]

Set the [DataValueAttributes](#) for the given dataset.

**Parameters:**

*dataset* The dataset to set the attributes for.

*a* The attributes to set.

Definition at line 400 of file KDChartAbstractDiagram.cpp.

References `d`, `KDChart::DataValueLabelAttributesRole`, and `KDChart::AbstractDiagram::propertiesChanged()`.

```
401 {  
402     d->attributesModel->setHeaderData(  
403         column, Qt::Vertical,  
404         qVariantFromValue( a ), DataValueLabelAttributesRole );  
405     emit propertiesChanged();  
406 }
```

**9.18.4.79 void AbstractDiagram::setDataValueAttributes (const QModelIndex & *index*, const [DataValueAttributes](#) & *a*)** [inherited]

Set the [DataValueAttributes](#) for the given index.

**Parameters:**

*index* The datapoint to set the attributes for.

*a* The attributes to set.

Definition at line 389 of file KDChartAbstractDiagram.cpp.

References `d`, `KDChart::DataValueLabelAttributesRole`, and `KDChart::AbstractDiagram::propertiesChanged()`.

```
391 {  
392     d->attributesModel->setData(  
393         d->attributesModel->mapFromSource( index ),  
394         qVariantFromValue( a ),  
395         DataValueLabelAttributesRole );  
396     emit propertiesChanged();  
397 }
```

**9.18.4.80 void AbstractDiagram::setHidden (bool *hidden*)** [inherited]

Hide (or unhide, resp.  
 ) all datapoints in the model.

**Note:**

Hidden data are still taken into account by the coordinate plane, so neither the grid nor your axes' ranges will change, when you hide data. For totally removing data from KD Chart's view you can use another approach: e.g. you could define a proxy model on top of your data model, and register the proxy model calling [setModel\(\)](#) instead of registering your real data model.

**Parameters:**

*hidden* The hidden status to set.

Definition at line 359 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::AbstractDiagram::dataHidden\(\)](#), and [KDChart::DataHiddenRole](#).

```

360 {
361     d->attributesModel->setModelData (
362         qVariantFromValue( hidden ),
363         DataHiddenRole );
364     emit dataHidden();
365 }
```

**9.18.4.81 void AbstractDiagram::setHidden (int *column*, bool *hidden*)** [inherited]

Hide (or unhide, resp.  
 ) a dataset.

**Note:**

Hidden data are still taken into account by the coordinate plane, so neither the grid nor your axes' ranges will change, when you hide data. For totally removing data from KD Chart's view you can use another approach: e.g. you could define a proxy model on top of your data model, and register the proxy model calling [setModel\(\)](#) instead of registering your real data model.

**Parameters:**

*column* The dataset to set the hidden status for.

*hidden* The hidden status to set.

Definition at line 350 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::AbstractDiagram::dataHidden\(\)](#), and [KDChart::DataHiddenRole](#).

```

351 {
352     d->attributesModel->setHeaderData (
353         column, Qt::Vertical,
354         qVariantFromValue( hidden ),
355         DataHiddenRole );
356     emit dataHidden();
357 }
```

#### 9.18.4.82 void AbstractDiagram::setHidden (const QModelIndex & *index*, bool *hidden*) [inherited]

Hide (or unhide, resp.

) a data cell.

##### Note:

Hidden data are still taken into account by the coordinate plane, so neither the grid nor your axes' ranges will change, when you hide data. For totally removing data from KD Chart's view you can use another approach: e.g. you could define a proxy model on top of your data model, and register the proxy model calling [setModel\(\)](#) instead of registering your real data model.

##### Parameters:

*index* The datapoint to set the hidden status for.

*hidden* The hidden status to set.

Definition at line 341 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::AbstractDiagram::dataHidden\(\)](#), and [KDChart::DataHiddenRole](#).

```
342 {
343     d->attributesModel->setData(
344         d->attributesModel->mapFromSource( index ),
345         QVariantFromValue( hidden ),
346         DataHiddenRole );
347     emit dataHidden();
348 }
```

#### 9.18.4.83 void AbstractCartesianDiagram::setModel (QAbstractItemModel \* *model*) [virtual, inherited]

Associate a model with the diagram.

Reimplemented from [KDChart::AbstractDiagram](#).

Definition at line 164 of file KDChartAbstractCartesianDiagram.cpp.

References [KDChart::AbstractDiagram::attributesModel\(\)](#), [d](#), and [KDChart::AbstractDiagram::setModel\(\)](#).

```
165 {
166     AbstractDiagram::setModel( model );
167     d->compressor.setModel( attributesModel() );
168 }
```

#### 9.18.4.84 void AbstractDiagram::setPen (const QPen & *pen*) [inherited]

Set the pen to be used, for painting all datasets in the model.

##### Parameters:

*pen* The pen to use.

Definition at line 753 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DatasetPenRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setModelData().

```

754 {
755     attributesModel()->setModelData(
756         qVariantFromValue( pen ), DatasetPenRole );
757     emit propertiesChanged();
758 }
```

#### 9.18.4.85 void AbstractDiagram::setPen (int *dataset*, const QPen & *pen*) [inherited]

Set the pen to be used, for painting the given dataset.

##### Parameters:

*dataset* The dataset's row in the model.

*pen* The pen to use.

Definition at line 760 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::datasetDimension(), KDChart::DatasetPenRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setHeaderData().

```

761 {
762     if( datasetDimension() > 1 )
763         column *= datasetDimension();
764     attributesModel()->setHeaderData(
765         column, Qt::Vertical,
766         qVariantFromValue( pen ),
767         DatasetPenRole );
768     emit propertiesChanged();
769 }
```

#### 9.18.4.86 void AbstractDiagram::setPen (const QModelIndex & *index*, const QPen & *pen*) [inherited]

Set the pen to be used, for painting the datapoint at the given index.

##### Parameters:

*index* The datapoint's index in the model.

*pen* The pen to use.

Definition at line 743 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::datasetDimension(), KDChart::DatasetPenRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setData().

```

744 {
745     if( datasetDimension() > 1 )
746         return setPen( index.column(), pen );
```



```

747     attributesModel()->setData(
748         attributesModel()->mapFromSource( index ),
749         qVariantFromValue( pen ), DatasetPenRole );
750     emit propertiesChanged();
751 }

```

#### 9.18.4.87 void AbstractDiagram::setPercentMode (bool *percent*) [inherited]

Definition at line 456 of file KDChartAbstractDiagram.cpp.

References `d`, and `KDChart::AbstractDiagram::propertiesChanged()`.

Referenced by `KDChart::LineDiagram::setType()`, and `setType()`.

```

457 {
458     d->percent = percent;
459     emit propertiesChanged();
460 }

```

#### 9.18.4.88 void AbstractCartesianDiagram::setReferenceDiagram ([AbstractCartesianDiagram](#) \* *diagram*, const QPointF & *offset* = QPointF()) [virtual, inherited]

Makes this diagram use another diagram *diagram* as reference diagram with relative offset *offset*.

To share cartesian axes between different diagrams there might be cases when you need that. Normally you don't.

**See also:**

`examples/SharedAbscissa`

Definition at line 142 of file KDChartAbstractCartesianDiagram.cpp.

References `d`.

```

143 {
144     d->referenceDiagram = diagram;
145     d->referenceDiagramOffset = offset;
146 }

```

#### 9.18.4.89 void AbstractCartesianDiagram::setRootIndex (const QModelIndex & *index*) [virtual, inherited]

Set the root index in the model, where the diagram starts referencing data for display.

[reimplemented]

Reimplemented from [KDChart::AbstractDiagram](#).

Definition at line 158 of file KDChartAbstractCartesianDiagram.cpp.

References `d`, and `KDChart::AbstractDiagram::setRootIndex()`.

```

159 {
160     d->compressor.setRootIndex( index );
161     AbstractDiagram::setRootIndex( index );
162 }

```

#### 9.18.4.90 void AbstractDiagram::setSelection (const QRect & *rect*, QItemSelectionModel::SelectionFlags *command*) [virtual, inherited]

[reimplemented]

Definition at line 959 of file KDChartAbstractDiagram.cpp.

References [d](#).

```

960 {
961     const QModelIndexList indexes = d->indexesIn( rect );
962     QItemSelection selection;
963     KDAB_FOREACH( const QModelIndex& index, indexes )
964     {
965         selection.append( QItemSelectionRange( index ) );
966     }
967     selectionModel()->select( selection, command );
968 }
```

#### 9.18.4.91 void BarDiagram::setThreeDBarAttributes (const QModelIndex & *index*, const ThreeDBarAttributes & *a*)

Sets the 3D line attributes of model index *index* to *threeDAttrs*.

Definition at line 233 of file KDChartBarDiagram.cpp.

References [d](#), [KDChart::AbstractDiagram::propertiesChanged\(\)](#), [KDChart::AbstractDiagram::setDataBoundariesDirty\(\)](#), and [KDChart::ThreeDBarAttributesRole](#).

```

234 {
235     setDataBoundariesDirty();
236     d->attributesModel->setData(
237         d->attributesModel->mapFromSource(index),
238         qVariantFromValue( threeDAttrs ),
239         ThreeDBarAttributesRole );
240     //emit layoutChanged( this );
241     emit propertiesChanged();
242 }
```

#### 9.18.4.92 void BarDiagram::setThreeDBarAttributes (int *column*, const ThreeDBarAttributes & *a*)

Sets the 3D bar attributes of dataset *column* to *threeDAttrs*.

Definition at line 218 of file KDChartBarDiagram.cpp.

References [d](#), [KDChart::AbstractDiagram::propertiesChanged\(\)](#), [KDChart::AbstractDiagram::setDataBoundariesDirty\(\)](#), and [KDChart::ThreeDBarAttributesRole](#).

```

219 {
220     setDataBoundariesDirty();
221     d->attributesModel->setHeaderData(
222         column, Qt::Vertical,
223         qVariantFromValue( threeDAttrs ),
224         ThreeDBarAttributesRole );
225     //emit layoutChanged( this );
226     emit propertiesChanged();
227
228 }
```

**9.18.4.93 void BarDiagram::setThreeDBarAttributes (const [ThreeDBarAttributes](#) & a)**

Sets the global 3D bar attributes to *threeDAttrs*.

Definition at line 207 of file KDChartBarDiagram.cpp.

References [d](#), [KDChart::AbstractDiagram::propertiesChanged\(\)](#), [KDChart::AbstractDiagram::setDataBoundariesDirty\(\)](#), and [KDChart::ThreeDBarAttributesRole](#).

```

208 {
209     setDataBoundariesDirty();
210     d->attributesModel->setModelData( qVariantFromValue( threeDAttrs ), ThreeDBarAttributesRole );
211     //emit layoutChanged( this );
212     emit propertiesChanged();
213 }
```

**9.18.4.94 void BarDiagram::setType (const [BarType](#) type)**

Sets the bar diagram's type to *type*.

See also:

[BarDiagram::BarType](#)

Definition at line 103 of file KDChartBarDiagram.cpp.

References [d](#), [KDChart::AbstractDiagram::layoutChanged\(\)](#), [Normal](#), [Percent](#), [KDChart::AbstractDiagram::propertiesChanged\(\)](#), [KDChart::AbstractDiagram::setDataBoundariesDirty\(\)](#), [KDChart::AbstractDiagram::setPercentMode\(\)](#), and [Stacked](#).

Referenced by [clone\(\)](#).

```

104 {
105     //if ( type == d->barType ) return;
106     if ( d->implementor->type() == type ) return;
107
108     switch( type ) {
109     case Normal:
110         d->implementor = d->normalDiagram;
111         break;
112     case Stacked:
113         d->implementor = d->stackedDiagram;
114         break;
115     case Percent:
116         d->implementor = d->percentDiagram;
117         break;
118     default:
119         Q_ASSERT_X( false, "BarDiagram::setType", "unknown diagram subtype" );
120     };
121
122     Q_ASSERT( d->implementor->type() == type );
123
124     //d->barType = type;
125     // AbstractAxis settings - see AbstractDiagram and CartesianAxis
126     setPercentMode( type == BarDiagram::Percent );
127     setDataBoundariesDirty();
128     emit layoutChanged( this );
129     emit propertiesChanged();
130 }
```

**9.18.4.95 void AbstractDiagram::setUnitPrefix (const QString & *prefix*, Qt::Orientation *orientation*)** [inherited]

Sets the unit prefix for all values.

**Parameters:**

*prefix* the prefix to be set  
*orientation* the orientation of the axis to set

Definition at line 864 of file KDChartAbstractDiagram.cpp.

References d.

```
865 {  
866     d->unitPrefix[ orientation ] = prefix;  
867 }
```

**9.18.4.96 void AbstractDiagram::setUnitPrefix (const QString & *prefix*, int *column*, Qt::Orientation *orientation*)** [inherited]

Sets the unit prefix for one value.

**Parameters:**

*prefix* the prefix to be set  
*column* the value using that prefix  
*orientation* the orientation of the axis to set

Definition at line 854 of file KDChartAbstractDiagram.cpp.

References d.

```
855 {  
856     d->unitPrefixMap[ column ][ orientation ] = prefix;  
857 }
```

**9.18.4.97 void AbstractDiagram::setUnitSuffix (const QString & *suffix*, Qt::Orientation *orientation*)** [inherited]

Sets the unit suffix for all values.

**Parameters:**

*suffix* the suffix to be set  
*orientation* the orientation of the axis to set

Definition at line 885 of file KDChartAbstractDiagram.cpp.

References d.

```
886 {  
887     d->unitSuffix[ orientation ] = suffix;  
888 }
```

#### 9.18.4.98 void AbstractDiagram::setUnitSuffix (const QString & *suffix*, int *column*, Qt::Orientation *orientation*) [inherited]

Sets the unit suffix for one value.

##### Parameters:

- suffix* the suffix to be set
- column* the value using that suffix
- orientation* the orientation of the axis to set

Definition at line 875 of file KDChartAbstractDiagram.cpp.

References [d](#).

```
876 {
877     d->unitSuffixMap[ column ][ orientation ]= suffix;
878 }
```

#### 9.18.4.99 void AbstractCartesianDiagram::takeAxis (CartesianAxis \* *axis*) [virtual, inherited]

Removes the axis from the diagram, without deleting it.

The diagram no longer owns the axis, so it is the caller's responsibility to delete the axis.

##### See also:

[addAxis](#)

Definition at line 100 of file KDChartAbstractCartesianDiagram.cpp.

References [d](#), [KDChart::AbstractAxis::deleteObserver\(\)](#), [KDChart::AbstractCartesianDiagram::layoutPlanes\(\)](#), and [KDChart::AbstractLayoutItem::setParentWidget\(\)](#).

Referenced by [KDChart::Widget::setType\(\)](#), and [KDChart::CartesianAxis::~~CartesianAxis\(\)](#).

```
101 {
102     const int idx = d->axesList.indexOf( axis );
103     if( idx != -1 )
104         d->axesList.takeAt( idx );
105     axis->deleteObserver( this );
106     axis->setParentWidget( 0 );
107     layoutPlanes();
108 }
```

#### 9.18.4.100 [ThreeDBarAttributes](#) BarDiagram::threeDBarAttributes (const QModelIndex & *index*) const

##### Returns:

the 3D bar attributes of the model index *index*

Definition at line 267 of file KDChartBarDiagram.cpp.

References [d](#), and [KDChart::ThreeDBarAttributesRole](#).

```

268 {
269     return qVariantValue<ThreeDBarAttributes>(
270         d->attributesModel->data(
271             d->attributesModel->mapFromSource(index),
272             KDChart::ThreeDBarAttributesRole ) );
273 }

```

#### 9.18.4.101 [ThreeDBarAttributes](#) BarDiagram::threeDBarAttributes (int *column*) const

##### Returns:

the 3D bar attributes of data set *column*

Definition at line 256 of file KDChartBarDiagram.cpp.

References KDChart::AbstractDiagram::columnToIndex(), d, and KDChart::ThreeDBarAttributesRole.

```

257 {
258     return qVariantValue<ThreeDBarAttributes>(
259         d->attributesModel->data(
260             d->attributesModel->mapFromSource( columnToIndex( column ) ),
261             KDChart::ThreeDBarAttributesRole ) );
262 }

```

#### 9.18.4.102 [ThreeDBarAttributes](#) BarDiagram::threeDBarAttributes () const

##### Returns:

the global 3D bar attributes

Definition at line 247 of file KDChartBarDiagram.cpp.

References d, and KDChart::ThreeDBarAttributesRole.

Referenced by threeDItemDepth().

```

248 {
249     return qVariantValue<ThreeDBarAttributes>(
250         d->attributesModel->data( KDChart::ThreeDBarAttributesRole ) );
251 }

```

#### 9.18.4.103 [double](#) BarDiagram::threeDItemDepth (int *column*) const [protected, virtual]

##### Returns:

the 3D item depth of the data set *column*

Implements [KDChart::AbstractCartesianDiagram](#).

Definition at line 280 of file KDChartBarDiagram.cpp.

References d, and KDChart::ThreeDBarAttributesRole.

```

281 {
282     return qVariantValue<ThreeDBarAttributes>(
283         d->attributesModel->headerData (
284             column,
285             Qt::Vertical,
286             KDChart::ThreeDBarAttributesRole ) ).validDepth();
287 }

```

#### 9.18.4.104 double BarDiagram::threeDItemDepth (const QModelIndex & *index*) const [protected, virtual]

##### Returns:

the 3D item depth of the model index *index*

Implements [KDChart::AbstractCartesianDiagram](#).

Definition at line 275 of file KDChartBarDiagram.cpp.

References [threeDBarAttributes\(\)](#), and [KDChart::AbstractThreeDAttributes::validDepth\(\)](#).

```

276 {
277     return threeDBarAttributes( index ).validDepth();
278 }

```

#### 9.18.4.105 [BarDiagram::BarType](#) BarDiagram::type () const

##### Returns:

the type of the line diagram

Definition at line 135 of file KDChartBarDiagram.cpp.

References [d](#).

Referenced by [clone\(\)](#), and [compare\(\)](#).

```

136 {
137     return d->implementor->type();
138 }

```

#### 9.18.4.106 QString AbstractDiagram::unitPrefix (Qt::Orientation *orientation*) const [inherited]

Returns the global unit prefix.

##### Parameters:

***orientation*** the orientation of the axis

##### Returns:

the unit prefix

Definition at line 908 of file KDChartAbstractDiagram.cpp.

References [d](#).

```
909 {  
910     return d->unitPrefix[ orientation ];  
911 }
```

#### 9.18.4.107 QString AbstractDiagram::unitPrefix (int *column*, Qt::Orientation *orientation*, bool *fallback* = false) const [inherited]

Returns the unit prefix for a special value.

##### Parameters:

*column* the value which's prefix is requested

*orientation* the orientation of the axis

*fallback* if true, the global prefix is return when no specific one is set for that value

##### Returns:

the unit prefix

Definition at line 897 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::itemRowLabels(), and KDChart::CartesianAxis::paintCtx().

```
898 {  
899     if( !fallback || d->unitPrefixMap[ column ].contains( orientation ) )  
900         return d->unitPrefixMap[ column ][ orientation ];  
901     return d->unitPrefix[ orientation ];  
902 }
```

#### 9.18.4.108 QString AbstractDiagram::unitSuffix (Qt::Orientation *orientation*) const [inherited]

Returns the global unit suffix.

##### Parameters:

*orientation* the orientation of the axis

##### Returns:

the unit suffix

Definition at line 931 of file KDChartAbstractDiagram.cpp.

References d.

```
932 {  
933     return d->unitSuffix[ orientation ];  
934 }
```



#### 9.18.4.109 QString AbstractDiagram::unitSuffix (int *column*, Qt::Orientation *orientation*, bool *fallback* = false) const [inherited]

Returns the unit suffix for a special value.

##### Parameters:

*column* the value which's suffix is requested

*orientation* the orientation of the axis

*fallback* if true, the global suffix is return when no specific one is set for that value

##### Returns:

the unit suffix

Definition at line 920 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::itemRowLabels(), and KDChart::CartesianAxis::paintCtx().

```

921 {
922     if( !fallback || d->unitSuffixMap[ column ].contains( orientation ) )
923         return d->unitSuffixMap[ column ][ orientation ];
924     return d->unitSuffix[ orientation ];
925 }
```

#### 9.18.4.110 void AbstractDiagram::update () const [inherited]

Definition at line 1091 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::doItemsLayout().

```

1092 {
1093     //qDebug("KDChart::AbstractDiagram::update() called");
1094     if( d->plane )
1095         d->plane->update();
1096 }
```

#### 9.18.4.111 void KDChart::AbstractDiagram::useDefaultColors () [inherited]

Set the palette to be used, for painting datasets to the default palette.

##### See also:

[KDChart::Palette](#). FIXME: fold into one usePalette ([KDChart::Palette&](#)) method

Definition at line 974 of file KDChartAbstractDiagram.cpp.

References d, and KDChart::AttributesModel::PaletteTypeDefault.

```

975 {
976     d->attributesModel->setPaletteType( AttributesModel::PaletteTypeDefault );
977 }
```

**9.18.4.112 void KDChart::AbstractDiagram::useRainbowColors ()** [inherited]

Set the palette to be used, for painting datasets to the rainbow palette.

**See also:**

[KDChart::Palette](#).

Definition at line 984 of file KDChartAbstractDiagram.cpp.

References [d](#), and [KDChart::AttributesModel::PaletteTypeRainbow](#).

```
985 {  
986     d->attributesModel->setPaletteType( AttributesModel::PaletteTypeRainbow );  
987 }
```

**9.18.4.113 bool AbstractDiagram::usesExternalAttributesModel () const** [virtual, inherited]

Returns whether the diagram is using its own built-in attributes model or an attributes model that was set via [setAttributesModel](#).

**See also:**

[setAttributesModel](#)

Definition at line 274 of file KDChartAbstractDiagram.cpp.

References [d](#).

```
275 {  
276     return d->usesExternalAttributesModel();  
277 }
```

**9.18.4.114 void KDChart::AbstractDiagram::useSubduedColors ()** [inherited]

Set the palette to be used, for painting datasets to the subdued palette.

**See also:**

[KDChart::Palette](#).

Definition at line 979 of file KDChartAbstractDiagram.cpp.

References [d](#), and [KDChart::AttributesModel::PaletteTypeSubdued](#).

```
980 {  
981     d->attributesModel->setPaletteType( AttributesModel::PaletteTypeSubdued );  
982 }
```

**9.18.4.115** `double AbstractDiagram::valueForCell (int row, int column) const` [protected, inherited]

Helper method, retrieving the data value (DisplayRole) for a given row and column.

**Parameters:**

*row* The row to query.

*column* The column to query.

**Returns:**

The value of the display role at the given row and column as a double.

**Deprecated**

Definition at line 1085 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModelRootIndex(), and d.

```
1086 {
1087     return d->attributesModel->data(
1088         d->attributesModel->index( row, column, attributesModelRootIndex() ) ).toDouble();
1089 }
```

**9.18.4.116** `int AbstractDiagram::verticalOffset () const` [virtual, inherited]

[reimplemented]

Definition at line 953 of file KDChartAbstractDiagram.cpp.

```
954 { return 0; }
```

**9.18.4.117** `QRect AbstractDiagram::visualRect (const QModelIndex & index) const` [virtual, inherited]

[reimplemented]

Definition at line 937 of file KDChartAbstractDiagram.cpp.

```
938 {
939     return QRect();
940 }
```

**9.18.4.118** `QRegion AbstractDiagram::visualRegionForSelection (const QItemSelection & selection) const` [virtual, inherited]

[reimplemented]

Definition at line 970 of file KDChartAbstractDiagram.cpp.

```
971 { return QRegion(); }
```

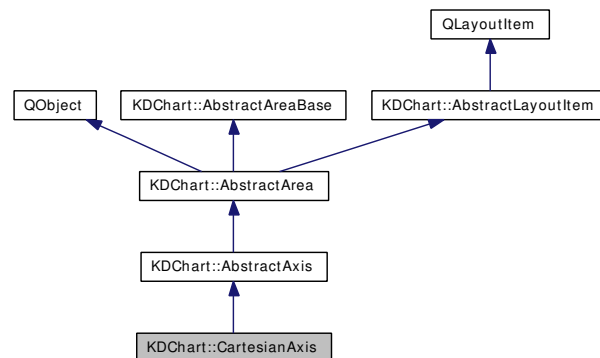
The documentation for this class was generated from the following files:

- [KDChartBarDiagram.h](#)
- [KDChartBarDiagram.cpp](#)

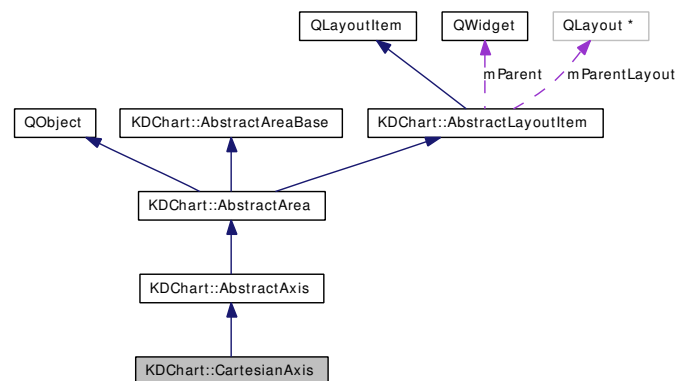
## 9.19 KDChart::CartesianAxis Class Reference

```
#include <KDChartCartesianAxis.h>
```

Inheritance diagram for KDChart::CartesianAxis:



Collaboration diagram for KDChart::CartesianAxis:



### 9.19.1 Detailed Description

The class for cartesian axes.

For being useful, axes need to be assigned to a diagram, see [AbstractCartesianDiagram::addAxis](#) and [AbstractCartesianDiagram::takeAxis](#).

**See also:**

PolarAxis, [AbstractCartesianDiagram](#)

Definition at line 48 of file `KDChartCartesianAxis.h`.

### Public Types

- enum [Position](#) {  
    [Bottom](#),

Top,  
Right,  
Left }

## Public Slots

- void `update` ()

## Signals

- void `positionChanged` (`AbstractArea *`)

## Public Member Functions

- void `alignToReferencePoint` (const `RelativePosition` &position)
- `BackgroundAttributes backgroundAttributes` () const
- virtual int `bottomOverlap` (bool doNotRecalculate=false) const  
*This is called at layout time by `KDChart::AutoSpacerLayoutItem::sizeHint()`.*
- `CartesianAxis` (`AbstractCartesianDiagram *diagram=0`)  
*C'tor of the class for cartesian axes.*
- bool `compare` (const `AbstractAreaBase *other`) const  
*Returns true if both areas have the same settings.*
- bool `compare` (const `AbstractAxis *other`) const  
*Returns true if both axes have the same settings.*
- bool `compare` (const `CartesianAxis *other`) const  
*Returns true if both axes have the same settings.*
- virtual void `connectSignals` ()  
*Wiring the signal/slot connections.*
- const `AbstractCoordinatePlane *coordinatePlane` () const  
*Convenience function, returns the coordinate plane, in which this axis is used.*
- void `createObserver` (`AbstractDiagram *diagram`)
- virtual const QString `customizedLabel` (const QString &label) const  
*Implement this method if you want to adjust axis labels before they are printed.*
- void `deleteObserver` (`AbstractDiagram *diagram`)
- const `AbstractDiagram *diagram` () const
- virtual Qt::Orientations `expandingDirections` () const  
*pure virtual in `QLayoutItem`*
- `FrameAttributes frameAttributes` () const
- virtual QRect `geometry` () const

*pure virtual in [QLayoutItem](#)*

- void [getFrameLeadings](#) (int &left, int &top, int &right, int &bottom) const
- bool [hasDefaultTitleTextAttributes](#) () const
- virtual bool [isAbscissa](#) () const
- virtual bool [isEmpty](#) () const

*pure virtual in [QLayoutItem](#)*

- virtual bool [isOrdinate](#) () const
- QStringList [labels](#) () const

*Returns a list of strings, that are used as axis labels, as set via [setLabels](#).*

- virtual void [layoutPlanes](#) ()
- virtual int [leftOverlap](#) (bool doNotRecalculate=false) const

*This is called at layout time by [KDChart::AutoSpacerLayoutItem::sizeHint\(\)](#).*

- virtual QSize [maximumSize](#) () const

*pure virtual in [QLayoutItem](#)*

- virtual QSize [minimumSize](#) () const

*pure virtual in [QLayoutItem](#)*

- bool [observedBy](#) ([AbstractDiagram](#) \*diagram) const
- virtual void [paint](#) (QPainter \*)

*reimpl*

- virtual void [paintAll](#) (QPainter &painter)

*Call [paintAll](#), if you want the background and the frame to be drawn before the normal [paint\(\)](#) is invoked automatically.*

- virtual void [paintBackground](#) (QPainter &painter, const QRect &rectangle)
- virtual void [paintCtx](#) ([PaintContext](#) \*)

*reimpl*

- virtual void [paintFrame](#) (QPainter &painter, const QRect &rectangle)
- virtual void [paintIntoRect](#) (QPainter &painter, const QRect &rect)

*Draws the background and frame, then calls [paint\(\)](#).*

- QLayout \* [parentLayout](#) ()
- virtual const [Position](#) [position](#) () const
- void [removeFromParentLayout](#) ()
- void [resetTitleTextAttributes](#) ()

*Reset the title text attributes to the built-in default:.*

- virtual int [rightOverlap](#) (bool doNotRecalculate=false) const

*This is called at layout time by [KDChart::AutoSpacerLayoutItem::sizeHint\(\)](#).*

- void [setBackgroundAttributes](#) (const [BackgroundAttributes](#) &a)
- void [setFrameAttributes](#) (const [FrameAttributes](#) &a)
- virtual void [setGeometry](#) (const QRect &r)

*pure virtual in [QLayoutItem](#)*

- void [setLabels](#) (const QStringList &list)

*Use this to specify your own set of strings, to be used as axis labels.*

- void [setParentLayout](#) (QLayout \*lay)
- virtual void [setParentWidget](#) (QWidget \*widget)

*Inform the item about its widget: This enables the item, to trigger that widget's update, whenever the size of the item's contents has changed.*

- virtual void [setPosition](#) (Position p)
- void [setShortLabels](#) (const QStringList &list)

*Use this to specify your own set of strings, to be used as axis labels, in case the normal labels are too long.*

- void [setTextAttributes](#) (const TextAttributes &a)

*Use this to specify the text attributes to be used for axis labels.*

- void [setTitleText](#) (const QString &text)
- void [setTitleTextAttributes](#) (const TextAttributes &a)
- QStringList [shortLabels](#) () const

*Returns a list of strings, that are used as axis labels, as set via [setShortLabels](#).*

- virtual QSize [sizeHint](#) () const

*pure virtual in [QLayoutItem](#)*

- virtual void [sizeHintChanged](#) () const

*Report changed size hint: ask the parent widget to recalculate the layout.*

- [TextAttributes](#) [textAttributes](#) () const

*Returns the text attributes to be used for axis labels.*

- int [tickLength](#) (bool subUnitTicks=false) const
- QString [titleText](#) () const
- [TextAttributes](#) [titleTextAttributes](#) () const

*Returns the text attributes that will be used for displaying the title text.*

- virtual int [topOverlap](#) (bool doNotRecalculate=false) const

*This is called at layout time by [KDChart::AutoSpacerLayoutItem::sizeHint\(\)](#).*

- [~CartesianAxis](#) ()

## Static Public Member Functions

- static void [paintBackgroundAttributes](#) (QPainter &painter, const QRect &rectangle, const [KDChart::BackgroundAttributes](#) &attributes)
- static void [paintFrameAttributes](#) (QPainter &painter, const QRect &rectangle, const [KDChart::FrameAttributes](#) &attributes)



## Protected Slots

- virtual void [delayedInit](#) ()  
*called for initializing after the c'tor has completed*

## Protected Member Functions

- virtual QRect [areaGeometry](#) () const
- QRect [innerRect](#) () const
- virtual void [positionHasChanged](#) ()

## Protected Attributes

- QWidget \* [mParent](#)
- QLayout \* [mParentLayout](#)

## 9.19.2 Member Enumeration Documentation

### 9.19.2.1 enum [KDChart::CartesianAxis::Position](#)

Enumerator:

*Bottom*

*Top*

*Right*

*Left*

Definition at line 56 of file KDChartCartesianAxis.h.

```
56         {
57             Bottom,
58             Top,
59             Right,
60             Left
61         };
```

## 9.19.3 Constructor & Destructor Documentation

### 9.19.3.1 CartesianAxis::CartesianAxis ([AbstractCartesianDiagram](#) \* *diagram* = 0) [explicit]

C'tor of the class for cartesian axes.

**Note:**

If using a zero parent for the constructor, you need to call your diagram's `addAxis` function to add your axis to the diagram. Otherwise, there is no need to call `addAxis`, since the constructor does that automatically for you, if you pass a diagram as parameter.

**See also:**

[AbstractCartesianDiagram::addAxis](#)

Definition at line 52 of file KDChartCartesianAxis.cpp.

```

53     : AbstractAxis ( new Private( diagram, this ), diagram )
54 {
55     init();
56 }
```

### 9.19.3.2 CartesianAxis::~~CartesianAxis ()

Definition at line 58 of file KDChartCartesianAxis.cpp.

References `d`, `KDChart::AbstractAxis::diagram()`, and `KDChart::AbstractCartesianDiagram::takeAxis()`.

```

59 {
60     // when we remove the first axis it will unregister itself and
61     // propagate the next one to the primary, thus the while loop
62     while ( d->mDiagram ) {
63         AbstractCartesianDiagram *cd = qobject_cast<AbstractCartesianDiagram*>( d->mDiagram );
64         cd->takeAxis( this );
65     }
66     Q_FOREACH( AbstractDiagram *diagram, d->secondaryDiagrams ) {
67         AbstractCartesianDiagram *cd = qobject_cast<AbstractCartesianDiagram*>( diagram );
68         cd->takeAxis( this );
69     }
70 }
```

## 9.19.4 Member Function Documentation

### 9.19.4.1 void AbstractAreaBase::alignToReferencePoint (const [RelativePosition](#) & position) [inherited]

Definition at line 90 of file KDChartAbstractAreaBase.cpp.

```

91 {
92     Q_UNUSED( position );
93     // PENDING(kalle) FIXME
94     qWarning( "Sorry, not implemented: void AbstractAreaBase::alignToReferencePoint( const RelativePosi
95 }
```

### 9.19.4.2 QRect AbstractArea::areaGeometry () const [protected, virtual, inherited]

Implements [KDChart::AbstractAreaBase](#).

Definition at line 150 of file KDChartAbstractArea.cpp.

Referenced by `KDChart::CartesianCoordinatePlane::drawingArea()`, `KDChart::TernaryCoordinatePlane::layoutDiagrams()`, `KDChart::PolarCoordinatePlane::layoutDiagrams()`, `KDChart::TernaryCoordinatePlane::paint()`, `paint()`, `KDChart::AbstractArea::paintAll()`, and `paintCtx()`.

```

151 {
152     return geometry();
153 }
```

#### 9.19.4.3 [BackgroundAttributes](#) AbstractAreaBase::backgroundAttributes () const [inherited]

Definition at line 120 of file KDChartAbstractAreaBase.cpp.

References d.

Referenced by KDChart::AbstractAreaBase::compare(), and updateCommonBrush().

```
121 {
122     return d->backgroundAttributes;
123 }
```

#### 9.19.4.4 int AbstractArea::bottomOverlap (bool *doNotRecalculate* = false) const [virtual, inherited]

This is called at layout time by KDChart::AutoSpacerLayoutItem::sizeHint().

The method triggers AbstractArea::sizeHint() to find out the amount of overlap at the bottom edge of the area.

##### Note:

The default implementation is not using any caching, it might make sense to implement a more sophisticated solution for derived classes that have complex work to do in sizeHint(). All we have here is a primitive flag to be set by the caller if it is sure that no sizeHint() needs to be called.

Definition at line 101 of file KDChartAbstractArea.cpp.

References d.

Referenced by maximumSize().

```
102 {
103     // Re-calculate the sizes,
104     // so we also get the amountOf..Overlap members set newly:
105     if( ! doNotRecalculate )
106         sizeHint();
107     return d->amountOfBottomOverlap;
108 }
```

#### 9.19.4.5 bool AbstractAreaBase::compare (const [AbstractAreaBase](#) \* *other*) const [inherited]

Returns true if both areas have the same settings.

Definition at line 75 of file KDChartAbstractAreaBase.cpp.

References KDChart::AbstractAreaBase::backgroundAttributes(), and KDChart::AbstractAreaBase::frameAttributes().

```
76 {
77     if( other == this ) return true;
78     if( ! other ){
79         //qDebug() << "CartesianAxis::compare() cannot compare to Null pointer";
80         return false;
81     }
```

```

82     /*
83     qDebug() << "AbstractAreaBase:" << (frameAttributes() == other->frameAttributes())
84         << (backgroundAttributes() == other->backgroundAttributes()) << "\n";
85     */
86     return (frameAttributes() == other->frameAttributes()) &&
87         (backgroundAttributes() == other->backgroundAttributes());
88 }

```

#### 9.19.4.6 bool AbstractAxis::compare (const AbstractAxis \* other) const [inherited]

Returns true if both axes have the same settings.

Definition at line 142 of file KDChartAbstractAxis.cpp.

References KDChart::AbstractAxis::labels(), KDChart::AbstractAxis::shortLabels(), and KDChart::AbstractAxis::textAttributes().

```

143 {
144     if( other == this ) return true;
145     if( ! other ){
146         //qDebug() << "CartesianAxis::compare() cannot compare to Null pointer";
147         return false;
148     }
149     /*
150     qDebug() << (textAttributes() == other->textAttributes());
151     qDebug() << (labels() == other->labels());
152     qDebug() << (shortLabels() == other->shortLabels());
153     */
154     return ( static_cast<const AbstractAreaBase*>(this)->compare( other ) ) &&
155         (textAttributes() == other->textAttributes()) &&
156         (labels() == other->labels()) &&
157         (shortLabels() == other->shortLabels());
158 }

```

#### 9.19.4.7 bool CartesianAxis::compare (const CartesianAxis \* other) const

Returns true if both axes have the same settings.

Definition at line 78 of file KDChartCartesianAxis.cpp.

References position(), titleText(), and titleTextAttributes().

```

79 {
80     if( other == this ) return true;
81     if( ! other ){
82         //qDebug() << "CartesianAxis::compare() cannot compare to Null pointer";
83         return false;
84     }
85     /*
86     qDebug() << (position() == other->position());
87     qDebug() << (titleText() == other->titleText());
88     qDebug() << (titleTextAttributes() == other->titleTextAttributes());
89     */
90     return ( static_cast<const AbstractAxis*>(this)->compare( other ) ) &&
91         ( position() == other->position() ) &&
92         ( titleText() == other->titleText() ) &&
93         ( titleTextAttributes() == other->titleTextAttributes() );
94 }

```

**9.19.4.8 void AbstractAxis::connectSignals ()** [virtual, inherited]

Wiring the signal/slot connections.

This method gets called automatically, each time, when you assign the axis to a diagram, either by passing a diagram\* to the c'tor, or by calling the diagram's setAxis method, resp.

If overwriting this method in derived classes, make sure to call this base method [AbstractAxis::connectSignals\(\)](#), so your axis gets connected to the diagram's built-in signals.

**See also:**

[AbstractCartesianDiagram::addAxis\(\)](#)

Definition at line 211 of file KDChartAbstractAxis.cpp.

References [d](#), and [KDChart::AbstractAxis::update\(\)](#).

Referenced by [KDChart::AbstractAxis::createObserver\(\)](#).

```

212 {
213     if( d->observer ) {
214         connect( d->observer, SIGNAL( diagramDataChanged( AbstractDiagram *) ),
215                 this, SLOT( update() ) );
216     }
217 }
```

**9.19.4.9 const [AbstractCoordinatePlane](#) \* AbstractAxis::coordinatePlane ()** const [inherited]

Convenience function, returns the coordinate plane, in which this axis is used.

If the axis is not used in a coordinate plane, the return value is Zero.

Definition at line 324 of file KDChartAbstractAxis.cpp.

References [d](#).

Referenced by [KDChart::CartesianCoordinatePlane::sharedAxisMasterPlane\(\)](#).

```

325 {
326     if( d->diagram() )
327         return d->diagram()->coordinatePlane();
328     return 0;
329 }
```

**9.19.4.10 void AbstractAxis::createObserver ([AbstractDiagram](#) \* diagram)** [inherited]

Definition at line 177 of file KDChartAbstractAxis.cpp.

References [KDChart::AbstractAxis::connectSignals\(\)](#), [d](#), and [KDChart::AbstractAxis::diagram\(\)](#).

```

178 {
179     if( d->setDiagram( diagram ) )
180         connectSignals();
181 }
```

#### 9.19.4.11 `const QString AbstractAxis::customizedLabel (const QString & label) const` [virtual, inherited]

Implement this method if you want to adjust axis labels before they are printed.

KD [Chart](#) is calling this method immediately before drawing the text, this means: What you return here will be drawn without further modifications.

##### Parameters:

*label* The text of the label as KD [Chart](#) has calculated it automatically (or as it was taken from a QStringList provided by you, resp.)

##### Returns:

The text to be drawn. By default this is the same as `label`.

Definition at line 161 of file `KDChartAbstractAxis.cpp`.

Referenced by `maximumSize()`, and `paintCtx()`.

```
162 {
163     return label;
164 }
```

#### 9.19.4.12 `void AbstractAxis::delayedInit ()` [protected, virtual, slot, inherited]

called for initializing after the c'tor has completed

Definition at line 134 of file `KDChartAbstractAxis.cpp`.

References `d`.

Referenced by `KDChart::AbstractAxis::AbstractAxis()`.

```
135 {
136     // We call setDiagram() here, because the c'tor of Private
137     // only has stored the pointers, but it did not call setDiagram().
138     if( d )
139         d->setDiagram( 0, true /* delayedInit */ );
140 }
```

#### 9.19.4.13 `void AbstractAxis::deleteObserver (AbstractDiagram * diagram)` [inherited]

Definition at line 193 of file `KDChartAbstractAxis.cpp`.

References `d`, and `KDChart::AbstractAxis::diagram()`.

Referenced by `KDChart::AbstractCartesianDiagram::takeAxis()`, and `KDChart::AbstractCartesianDiagram::~~AbstractCartesianDiagram()`.

```
194 {
195     d->unsetDiagram( diagram );
196 }
```

**9.19.4.14** `const AbstractDiagram * KDChart::AbstractAxis::diagram () const` [inherited]

Definition at line 331 of file KDChartAbstractAxis.cpp.

References [d](#).

Referenced by [KDChart::AbstractAxis::createObserver\(\)](#), [KDChart::AbstractAxis::deleteObserver\(\)](#), [KDChart::AbstractAxis::observedBy\(\)](#), [paintCtx\(\)](#), [KDChart::TernaryAxis::TernaryAxis\(\)](#), and [~CartesianAxis\(\)](#).

```
332 {  
333     return d->diagram();  
334 }
```

**9.19.4.15** `Qt::Orientations CartesianAxis::expandingDirections () const` [virtual]

pure virtual in [QLayoutItem](#)

Definition at line 991 of file KDChartCartesianAxis.cpp.

References [Bottom](#), [Left](#), [position\(\)](#), [Right](#), and [Top](#).

```
992 {  
993     Qt::Orientations ret;  
994     switch ( position() )  
995     {  
996     case Bottom:  
997     case Top:  
998         ret = Qt::Horizontal;  
999         break;  
1000    case Left:  
1001    case Right:  
1002        ret = Qt::Vertical;  
1003        break;  
1004    default:  
1005        Q_ASSERT( false ); // all positions need to be handled  
1006        break;  
1007    };  
1008    return ret;  
1009 }
```

**9.19.4.16** `FrameAttributes AbstractAreaBase::frameAttributes () const` [inherited]

Definition at line 106 of file KDChartAbstractAreaBase.cpp.

References [d](#).

Referenced by [KDChart::Legend::clone\(\)](#), [KDChart::AbstractAreaBase::compare\(\)](#), and [updateCommonBrush\(\)](#).

```
107 {  
108     return d->frameAttributes;  
109 }
```

**9.19.4.17** `QRect CartesianAxis::geometry () const` [virtual]

pure virtual in [QLayoutItem](#)

Implements [KDChart::AbstractAxis](#).

Definition at line 1234 of file KDChartCartesianAxis.cpp.

References [d](#).

Referenced by [paintCtx\(\)](#).

```
1235 {
1236     return d->geometry;
1237 }
```

#### 9.19.4.18 void AbstractAreaBase::getFrameLeadings (int & left, int & top, int & right, int & bottom) const [inherited]

Definition at line 212 of file KDChartAbstractAreaBase.cpp.

References [d](#).

Referenced by [KDChart::AbstractAreaBase::innerRect\(\)](#), and [KDChart::AbstractAreaWidget::paintAll\(\)](#).

```
213 {
214     if( d && d->frameAttributes.isVisible() ){
215         const int padding = qMax( d->frameAttributes.padding(), 0 );
216         left    = padding;
217         top     = padding;
218         right   = padding;
219         bottom  = padding;
220     }else{
221         left    = 0;
222         top     = 0;
223         right   = 0;
224         bottom  = 0;
225     }
226 }
```

#### 9.19.4.19 bool CartesianAxis::hasDefaultTitleTextAttributes () const

Definition at line 133 of file KDChartCartesianAxis.cpp.

References [d](#).

Referenced by [titleTextAttributes\(\)](#).

```
134 {
135     return d->useDefaultTextAttributes;
136 }
```

#### 9.19.4.20 QRect AbstractAreaBase::innerRect () const [protected, inherited]

Definition at line 228 of file KDChartAbstractAreaBase.cpp.

References [KDChart::AbstractAreaBase::areaGeometry\(\)](#), and [KDChart::AbstractAreaBase::getFrameLeadings\(\)](#).

Referenced by [KDChart::TextArea::paintAll\(\)](#), and [KDChart::AbstractArea::paintAll\(\)](#).



```
229 {
230     int left;
231     int top;
232     int right;
233     int bottom;
234     setFrameLeadings( left, top, right, bottom );
235     return
236         QRect( QPoint(0,0), areaGeometry().size() )
237             .adjusted( left, top, -right, -bottom );
238 }
```

#### 9.19.4.21 bool CartesianAxis::isAbscissa () const [virtual]

Definition at line 164 of file KDChartCartesianAxis.cpp.

References Bottom, position(), and Top.

Referenced by paintCtx(), and tickLength().

```
165 {
166     return position() == Bottom || position() == Top;
167 }
```

#### 9.19.4.22 bool CartesianAxis::isEmpty () const [virtual]

pure virtual in [QLayoutItem](#)

Definition at line 986 of file KDChartCartesianAxis.cpp.

Referenced by paintCtx().

```
987 {
988     return false; // if the axis exists, it has some (perhaps default) content
989 }
```

#### 9.19.4.23 bool CartesianAxis::isOrdinate () const [virtual]

Definition at line 169 of file KDChartCartesianAxis.cpp.

References Left, position(), and Right.

Referenced by paintCtx().

```
170 {
171     return position() == Left || position() == Right;
172 }
```

#### 9.19.4.24 QStringList AbstractAxis::labels () const [inherited]

Returns a list of strings, that are used as axis labels, as set via setLabels.

See also:

[setLabels](#)

Definition at line 281 of file KDChartAbstractAxis.cpp.

References [d](#).

Referenced by [KDChart::AbstractAxis::compare\(\)](#), [maximumSize\(\)](#), [KDChart::TernaryAxis::paintCtx\(\)](#), and [paintCtx\(\)](#).

```
282 {
283     return d->hardLabels;
284 }
```

#### 9.19.4.25 void CartesianAxis::layoutPlanes () [virtual]

Definition at line 150 of file KDChartCartesianAxis.cpp.

References [d](#), and [KDChart::AbstractCoordinatePlane::layoutPlanes\(\)](#).

Referenced by [resetTitleTextAttributes\(\)](#), [setPosition\(\)](#), [setTitleText\(\)](#), and [setTitleTextAttributes\(\)](#).

```
151 {
152     //qDebug() << "CartesianAxis::layoutPlanes()";
153     if( ! d->diagram() || ! d->diagram()->coordinatePlane() ) {
154         //qDebug() << "CartesianAxis::layoutPlanes(): Sorry, found no plane.";
155         return;
156     }
157     AbstractCoordinatePlane* plane = d->diagram()->coordinatePlane();
158     if( plane ){
159         plane->layoutPlanes();
160         //qDebug() << "CartesianAxis::layoutPlanes() OK";
161     }
162 }
```

#### 9.19.4.26 int AbstractArea::leftOverlap (bool *doNotRecalculate* = false) const [virtual, inherited]

This is called at layout time by [KDChart::AutoSpacerLayoutItem::sizeHint\(\)](#).

The method triggers [AbstractArea::sizeHint\(\)](#) to find out the amount of overlap at the left edge of the area.

##### Note:

The default implementation is not using any caching, it might make sense to implement a more sophisticated solution for derived classes that have complex work to do in [sizeHint\(\)](#). All we have here is a primitive flag to be set by the caller if it is sure that no [sizeHint\(\)](#) needs to be called.

Definition at line 77 of file KDChartAbstractArea.cpp.

References [d](#).

Referenced by [maximumSize\(\)](#).

```
78 {
79     // Re-calculate the sizes,
80     // so we also get the amountOf..Overlap members set newly:
81     if( ! doNotRecalculate )
82         sizeHint();
83     return d->amountOfLeftOverlap;
84 }
```

**9.19.4.27 QSize CartesianAxis::maximumSize () const** [virtual]

pure virtual in [QLayoutItem](#)

Definition at line 1038 of file KDChartCartesianAxis.cpp.

References [Bottom](#), [KDChart::AbstractArea::bottomOverlap\(\)](#), [calculateOverlap\(\)](#), [KDChart::AbstractAxis::customizedLabel\(\)](#), [d](#), [KDChart::AbstractCoordinatePlane::gridDimensionsList\(\)](#), [KDChart::TextAttributes::isVisible\(\)](#), [KDChart::AbstractAxis::labels\(\)](#), [Left](#), [KDChart::AbstractArea::leftOverlap\(\)](#), [KDChartEnums::MeasureOrientationMinimum](#), [KDChart::GlobalMeasureScaling::paintDevice\(\)](#), [KDChart::AbstractCoordinatePlane::parent\(\)](#), [position\(\)](#), [KDChart::TextLayoutItem::realFont\(\)](#), [referenceDiagramIsBarDiagram\(\)](#), [Right](#), [KDChart::AbstractArea::rightOverlap\(\)](#), [KDChart::TextLayoutItem::setText\(\)](#), [KDChart::TextLayoutItem::sizeHint\(\)](#), [KDChart::AbstractAxis::textAttributes\(\)](#), [tickLength\(\)](#), [titleText\(\)](#), [Top](#), and [KDChart::AbstractArea::topOverlap\(\)](#).

Referenced by [minimumSize\(\)](#), and [sizeHint\(\)](#).

```

1039 {
1040     QSize result;
1041     if ( !d->diagram() )
1042         return result;
1043
1044     const TextAttributes labelTA = textAttributes();
1045     const bool drawLabels = labelTA.isVisible();
1046
1047     const TextAttributes titleTA( d->titleTextAttributesWithAdjustedRotation() );
1048     const bool drawTitle = titleTA.isVisible() && ! titleText().isEmpty();
1049
1050     AbstractCoordinatePlane* plane = d->diagram()->coordinatePlane();
1051     //qDebug() << this<<":maximumSize() uses plane geometry" << plane->geometry();
1052     QObject* refArea = plane->parent();
1053     TextLayoutItem labelItem( QString::null, labelTA, refArea,
1054                             KDChartEnums::MeasureOrientationMinimum, Qt::AlignLeft );
1055     TextLayoutItem titleItem( titleText(), titleTA, refArea,
1056                             KDChartEnums::MeasureOrientationMinimum, Qt::AlignHCenter | Qt::AlignVCenter );
1057     const qreal labelGap =
1058         drawLabels
1059         ? (QFontMetricsF( labelItem.realFont(), GlobalMeasureScaling::paintDevice() ).height() / 3.0)
1060         : 0.0;
1061     const qreal titleGap =
1062         drawTitle
1063         ? (QFontMetricsF( titleItem.realFont(), GlobalMeasureScaling::paintDevice() ).height() / 3.0)
1064         : 0.0;
1065
1066     switch ( position() )
1067     {
1068     case Bottom:
1069     case Top: {
1070         const bool isBarDiagram = referenceDiagramIsBarDiagram(d->diagram());
1071         int leftOverlap = 0;
1072         int rightOverlap = 0;
1073
1074         qreal w = 10.0;
1075         qreal h = 0.0;
1076         if( drawLabels ){
1077             // if there're no label strings, we take the biggest needed number as height
1078             if ( labels().count() ){
1079                 // find the longest label text:
1080                 const int first=0;
1081                 const int last=labels().count()-1;
1082                 const QStringList labelsList( labels() );
1083                 for ( int i = first; i <= last; ++i )
1084                 {
1085                     labelItem.setText( customizedLabel(labelsList[ i ]) );
1086                     const QSize siz = labelItem.sizeHint();

```

```

1087         h = qMax( h, static_cast<qreal>(siz.height()) );
1088         calculateOverlap( i, first, last, siz.width(), isBarDiagram,
1089                         leftOverlap, rightOverlap );
1090     }
1091 }
1092 }else{
1093     QStringList headerLabels = d->diagram()->itemRowLabels();
1094     const int headerLabelsCount = headerLabels.count();
1095     if( headerLabelsCount ){
1096         const bool useFastCalcAlgorithm
1097             = (strcmp( metaObject()->className(), "KDChart::CartesianAxis" ) == 0);
1098         const int first=0;
1099         const int last=headerLabelsCount-1;
1100         for ( int i = first;
1101             i <= last;
1102             i = (useFastCalcAlgorithm && i < last) ? last : (i+1) )
1103         {
1104             labelItem.setText( customizedLabel(headerLabels[ i ]) );
1105             const QSize siz = labelItem.sizeHint();
1106             h = qMax( h, static_cast<qreal>(siz.height()) );
1107             calculateOverlap( i, first, last, siz.width(), isBarDiagram,
1108                             leftOverlap, rightOverlap );
1109         }
1110     }else{
1111         labelItem.setText(
1112             customizedLabel(
1113                 QString::number( plane->gridDimensionsList().first().end, 'f', 0
1114             const QSize siz = labelItem.sizeHint();
1115             h = siz.height();
1116             calculateOverlap( 0, 0, 0, siz.width(), isBarDiagram,
1117                             leftOverlap, rightOverlap );
1118         )
1119     }
1120     // we leave a little gap between axis labels and bottom (or top, resp.) side of axis
1121     h += labelGap;
1122 }
1123 // space for a possible title:
1124 if ( drawTitle ) {
1125     // we add the title height and leave a little gap between axis labels and axis title
1126     h += titleItem.sizeHint().height() + titleGap;
1127     w = titleItem.sizeHint().width() + 2.0;
1128 }
1129 // space for the ticks
1130 h += qAbs( tickLength() ) * 3.0;
1131 result = QSize ( static_cast<int>( w ), static_cast<int>( h ) );
1132
1133 // If necessary adjust the widths
1134 // of the left (or right, resp.) side neighboring columns:
1135 d->amountOfLeftOverlap = leftOverlap;
1136 d->amountOfRightOverlap = rightOverlap;
1137 /* Unused code for a push-model:
1138 if( leftOverlap || rightOverlap ){
1139     QTimer::singleShot(200, const_cast<CartesianAxis*>(this),
1140                         SLOT(adjustLeftRightGridColumnWidths()));
1141 }
1142 */
1143 }
1144 }
1145 break;
1146 case Left:
1147 case Right: {
1148     int topOverlap = 0;
1149     int bottomOverlap = 0;
1150
1151     qreal w = 0.0;
1152     qreal h = 10.0;
1153     if( drawLabels ){

```

```

1154         // if there're no label strings, we take the biggest needed number as width
1155         if ( labels().count() == 0 )
1156         {
1157             labelItem.setText(
1158                 customizedLabel(
1159                     QString::number( plane->gridDimensionsList().last().end, 'f', 0 ));
1160             const QSize siz = labelItem.sizeHint();
1161             w = siz.width();
1162             calculateOverlap( 0, 0, 0, siz.height(), false, // bar diagram flag is ignored for Ord
1163                             topOverlap, bottomOverlap );
1164         }else{
1165             // find the longest label text:
1166             const int first=0;
1167             const int last=labels().count()-1;
1168             const QStringList labelsList( labels() );
1169             for ( int i = first; i <= last; ++i )
1170             {
1171                 labelItem.setText( customizedLabel(labelsList[ i ]) );
1172                 const QSize siz = labelItem.sizeHint();
1173                 qreal lw = siz.width();
1174                 w = qMax( w, lw );
1175                 calculateOverlap( 0, 0, 0, siz.height(), false, // bar diagram flag is ignored for
1176                                 topOverlap, bottomOverlap );
1177             }
1178         }
1179         // we leave a little gap between axis labels and left (or right, resp.) side of axis
1180         w += labelGap;
1181     }
1182     // space for a possible title:
1183     if ( drawTitle ) {
1184         // we add the title height and leave a little gap between axis labels and axis title
1185         w += titleItem.sizeHint().width() + titleGap;
1186         h = titleItem.sizeHint().height() + 2.0;
1187         //qDebug() << "left/right axis title item size-hint:" << titleItem.sizeHint();
1188     }
1189     // space for the ticks
1190     w += qAbs( tickLength() ) * 3.0;
1191
1192     result = QSize ( static_cast<int>( w ), static_cast<int>( h ) );
1193     //qDebug() << "left/right axis width:" << result << "    w:" << w;
1194
1195     // If necessary adjust the heights
1196     // of the top (or bottom, resp.) side neighboring rows:
1197     d->amountOfTopOverlap = topOverlap;
1198     d->amountOfBottomOverlap = bottomOverlap;
1199     /* Unused code for a push-model:
1200     if( topOverlap || bottomOverlap ){
1201         QTimer::singleShot(200, const_cast<CartesianAxis*>(this),
1202                             SLOT(adjustTopBottomGridRowHeights()));
1203     }
1204     */
1205 }
1206 }
1207 break;
1208 default:
1209     Q_ASSERT( false ); // all positions need to be handled
1210     break;
1211 };
1212 //qDebug() << "*****" << result;
1213 //result=QSize(0,0);
1214 return result;
1215 }

```

**9.19.4.28 QSize CartesianAxis::minimumSize () const** [virtual]

pure virtual in [QLayoutItem](#)

Definition at line 1217 of file `KDChartCartesianAxis.cpp`.

References `maximumSize()`.

```
1218 {
1219     return maximumSize();
1220 }
```

**9.19.4.29 bool KDChart::AbstractAxis::observedBy ([AbstractDiagram](#) \* *diagram*) const**  
[inherited]

Definition at line 336 of file `KDChartAbstractAxis.cpp`.

References `d`, and `KDChart::AbstractAxis::diagram()`.

```
337 {
338     return d->hasDiagram( diagram );
339 }
```

**9.19.4.30 void CartesianAxis::paint (QPainter \*)** [virtual]

reimpl

Implements [KDChart::AbstractLayoutItem](#).

Definition at line 193 of file `KDChartCartesianAxis.cpp`.

References `KDChart::AbstractArea::areaGeometry()`, `d`, `paintCtx()`, `KDChart::PaintContext::setCoordinatePlane()`, `KDChart::PaintContext::setPainter()`, and `KDChart::PaintContext::setRectangle()`.

```
194 {
195     if( ! d->diagram() || ! d->diagram()->coordinatePlane() ) return;
196     PaintContext ctx;
197     ctx.setPainter ( painter );
198     ctx.setCoordinatePlane( d->diagram()->coordinatePlane() );
199     const QRect rect( areaGeometry() );
200
201     //qDebug() << "CartesianAxis::paint( QPainter* painter ) " << " areaGeometry()();" << rect << " s
202
203     ctx.setRectangle(
204         QRectF (
205             QPointF(0, 0),
206             QPointF(rect.left(), rect.top()),
207             QSizeF(rect.width(), rect.height() ) ) );
208     // enabling clipping so that we're not drawing outside
209     QRegion clipRegion( rect.adjusted( -1, -1, 1, 1 ) );
210     painter->save();
211     painter->setClipRegion( clipRegion );
212     paintCtx( &ctx );
213     painter->restore();
214     //qDebug() << "KDChart::CartesianAxis::paint() done.";
215 }
```

**9.19.4.31 void AbstractArea::paintAll (QPainter & *painter*)** [virtual, inherited]

Call paintAll, if you want the background and the frame to be drawn before the normal [paint\(\)](#) is invoked automatically.

Reimplemented from [KDChart::AbstractLayoutItem](#).

Reimplemented in [KDChart::TernaryAxis](#).

Definition at line 123 of file KDChartAbstractArea.cpp.

References [KDChart::AbstractArea::areaGeometry\(\)](#), [d](#), [KDChart::AbstractAreaBase::innerRect\(\)](#), [KDChart::AbstractLayoutItem::paint\(\)](#), [KDChart::AbstractAreaBase::paintBackground\(\)](#), and [KDChart::AbstractAreaBase::paintFrame\(\)](#).

Referenced by [KDChart::AbstractArea::paintIntoRect\(\)](#).

```

124 {
125     // Paint the background and frame
126     const QRect overlappingArea( geometry().adjusted(
127         -d->amountOfLeftOverlap,
128         -d->amountOfTopOverlap,
129         d->amountOfRightOverlap,
130         d->amountOfBottomOverlap ) );
131     paintBackground( painter, overlappingArea );
132     paintFrame(      painter, overlappingArea );
133
134     // temporarily adjust the widget size, to be sure all content gets calculated
135     // to fit into the inner rectangle
136     const QRect oldGeometry( areaGeometry() );
137     QRect inner( innerRect() );
138     inner.moveTo(
139         oldGeometry.left() + inner.left(),
140         oldGeometry.top()  + inner.top() );
141     const bool needAdjustGeometry = oldGeometry != inner;
142     if( needAdjustGeometry )
143         setGeometry( inner );
144     paint( &painter );
145     if( needAdjustGeometry )
146         setGeometry( oldGeometry );
147     //qDebug() << "AbstractAreaWidget::paintAll() done.";
148 }
```

**9.19.4.32 void AbstractAreaBase::paintBackground (QPainter & *painter*, const QRect & *rectangle*)** [virtual, inherited]

Definition at line 196 of file KDChartAbstractAreaBase.cpp.

References [d](#), and [KDChart::AbstractAreaBase::paintBackgroundAttributes\(\)](#).

Referenced by [KDChart::TextArea::paintAll\(\)](#), [KDChart::AbstractAreaWidget::paintAll\(\)](#), and [KDChart::AbstractArea::paintAll\(\)](#).

```

197 {
198     Q_ASSERT_X ( d != 0, "AbstractAreaBase::paintBackground()",
199                 "Private class was not initialized!" );
200     paintBackgroundAttributes( painter, rect, d->backgroundAttributes );
201 }
```

**19.19.4.33** `void AbstractAreaBase::paintBackgroundAttributes (QPainter & painter, const QRect & rectangle, const KDChart::BackgroundAttributes & attributes)` [static, inherited]

Definition at line 127 of file `KDChartAbstractAreaBase.cpp`.

References `KDChart::BackgroundAttributes::BackgroundPixmapModeCentered`, `KDChart::BackgroundAttributes::BackgroundPixmapModeNone`, `KDChart::BackgroundAttributes::BackgroundPixmapModeScaled`, `KDChart::BackgroundAttributes::BackgroundPixmapModeStretched`, `KDChart::BackgroundAttributes::brush()`, `KDChart::BackgroundAttributes::isVisible()`, `KDChart::BackgroundAttributes::pixmap()`, and `KDChart::BackgroundAttributes::pixmapMode()`.

Referenced by `KDChart::AbstractAreaBase::paintBackground()`.

```

129 {
130     if( !attributes.isVisible() ) return;
131
132     /* first draw the brush (may contain a pixmap)*/
133     if( Qt::NoBrush != attributes.brush().style() ) {
134         KDChart::PainterSaver painterSaver( &painter );
135         painter.setPen( Qt::NoPen );
136         const QPointF newTopLeft( painter.deviceMatrix().map( rect.topLeft() ) );
137         painter.setBrushOrigin( newTopLeft );
138         painter.setBrush( attributes.brush() );
139         painter.drawRect( rect.adjusted( 0, 0, -1, -1 ) );
140     }
141     /* next draw the backPixmap over the brush */
142     if( !attributes.pixmap().isNull() &&
143         attributes.pixmapMode() != BackgroundAttributes::BackgroundPixmapModeNone ) {
144         QPointF ol = rect.topLeft();
145         if( BackgroundAttributes::BackgroundPixmapModeCentered == attributes.pixmapMode() )
146         {
147             ol.setX( rect.center().x() - attributes.pixmap().width() / 2 );
148             ol.setY( rect.center().y() - attributes.pixmap().height() / 2 );
149             painter.drawPixmap( ol, attributes.pixmap() );
150         } else {
151             QMatrix m;
152             double zW = (double)rect.width() / (double)attributes.pixmap().width();
153             double zH = (double)rect.height() / (double)attributes.pixmap().height();
154             switch( attributes.pixmapMode() ) {
155                 case BackgroundAttributes::BackgroundPixmapModeScaled:
156                 {
157                     double z;
158                     z = qMin( zW, zH );
159                     m.scale( z, z );
160                 }
161                 break;
162                 case BackgroundAttributes::BackgroundPixmapModeStretched:
163                 {
164                     m.scale( zW, zH );
165                     break;
166                 }
167                 default:
168                 {
169                     ; // Cannot happen, previously checked
170                 }
171             }
172             QPixmap pm = attributes.pixmap().transformed( m );
173             ol.setX( rect.center().x() - pm.width() / 2 );
174             ol.setY( rect.center().y() - pm.height() / 2 );
175             painter.drawPixmap( ol, pm );
176         }
177     }
178 }
```



**9.19.4.34 void CartesianAxis::paintCtx (PaintContext \*) [virtual]**

reimpl

Reimplemented from [KDChart::AbstractLayoutItem](#).

Definition at line 374 of file KDChartCartesianAxis.cpp.

References [KDChart::AbstractArea::areaGeometry\(\)](#), [Bottom](#), [calculateNextLabel\(\)](#), [KDChart::PaintContext::coordinatePlane\(\)](#), [KDChart::AbstractAxis::customizedLabel\(\)](#), [d](#), [KDChart::AbstractAxis::diagram\(\)](#), [KDChart::TextLayoutItem::geometry\(\)](#), [geometry\(\)](#), [KDChart::TextLayoutItem::intersects\(\)](#), [isAbscissa\(\)](#), [isEmpty\(\)](#), [isOrdinate\(\)](#), [KDChart::TextAttributes::isVisible\(\)](#), [KDChart::AbstractAxis::labels\(\)](#), [Left](#), [KDChart::AbstractCoordinatePlane::Logarithmic](#), [KDChart::Enums::MeasureOrientationMinimum](#), [KDChart::TextLayoutItem::paint\(\)](#), [KDChart::GlobalMeasureScaling::paintDevice\(\)](#), [KDChart::PaintContext::painter\(\)](#), [KDChart::TextAttributes::pen\(\)](#), [position\(\)](#), [KDChart::TextLayoutItem::realFont\(\)](#), [referenceDiagramIsBarDiagram\(\)](#), [Right](#), [KDChart::TextLayoutItem::setGeometry\(\)](#), [KDChart::TextLayoutItem::setText\(\)](#), [KDChart::AbstractAxis::shortLabels\(\)](#), [KDChart::TextLayoutItem::sizeHint\(\)](#), [KDChart::TextLayoutItem::text\(\)](#), [KDChart::AbstractAxis::textAttributes\(\)](#), [tickLength\(\)](#), [titleText\(\)](#), [Top](#), [KDChart::AbstractDiagram::unitPrefix\(\)](#), and [KDChart::AbstractDiagram::unitSuffix\(\)](#).

Referenced by [paint\(\)](#).

```

375 {
376
377     Q_ASSERT_X ( d->diagram(), "CartesianAxis::paint",
378                 "Function call not allowed: The axis is not assigned to any diagram." );
379
380     CartesianCoordinatePlane* plane = dynamic_cast<CartesianCoordinatePlane*>(context->coordinatePlane);
381     Q_ASSERT_X ( plane, "CartesianAxis::paint",
382                 "Bad function call: PaintContext::coordinatePlane() NOT a cartesian plane." );
383
384     // note: Not having any data model assigned is no bug
385     //         but we can not draw an axis then either.
386     if ( ! d->diagram()->model() )
387         return;
388
389
390     /*
391     * let us paint the labels at a
392     * smaller resolution
393     * Same mini pixel value as for
394     * Cartesian Grid
395     */
396     //const qreal MinimumPixelsBetweenRulers = 1.0;
397     DataDimensionsList dimensions( plane->gridDimensionsList() );
398     //qDebug("CartesianAxis::paintCtx() gets DataDimensionsList.first():  start: %f  end: %f  stepWid
399
400     // test for programming errors: critical
401     Q_ASSERT_X ( dimensions.count() == 2, "CartesianAxis::paint",
402                 "Error: plane->gridDimensionsList() did not return exactly two dimensions." );
403     DataDimension dimX =
404         AbstractGrid::adjustedLowerUpperRange( dimensions.first(), true, true );
405     const DataDimension dimY =
406         AbstractGrid::adjustedLowerUpperRange( dimensions.last(), true, true );
407     const DataDimension& dim = (isAbscissa() ? dimX : dimY);
408
409     /*
410     if(isAbscissa())
411         qDebug() << "          " << "Abscissa:" << dimX.start <<".."<<dimX.end <<"  step"<<dimX.stepWid
412     else
413         qDebug() << "          " << "Ordinate:" << dimY.start <<".."<<dimY.end <<"  step"<<dimY.stepWid
414     */
415

```

```

416
417  /*
418   * let us paint the labels at a
419   * smaller resolution
420   * Same mini pixel value as for
421   * Cartesian Grid
422   */
423  const qreal MinimumPixelsBetweenRulers = qMin( dimX.stepWidth, dimY.stepWidth );//1.0;
424
425  // preparations:
426  // - calculate the range that will be displayed:
427  const qreal absRange = qAbs( dim.distance() );
428
429  qreal numberOfUnitRulers;
430  if ( isAbscissa() ) {
431      if( dimX.isCalculated )
432          numberOfUnitRulers = absRange / qAbs( dimX.stepWidth ) + 1.0;
433      else
434          numberOfUnitRulers = d->diagram()->model()->rowCount() - 1.0;
435  }else{
436      numberOfUnitRulers = absRange / qAbs( dimY.stepWidth ) + 1.0;
437  }
438
439  //      qDebug() << "absRange" << absRange << "dimY.stepWidth:" << dimY.stepWidth << "numberOfUnitRulers" << numberOfUnitRulers;
440
441  qreal numberOfSubUnitRulers;
442  if ( isAbscissa() ){
443      if( dimX.isCalculated )
444          numberOfSubUnitRulers = absRange / qAbs( dimX.subStepWidth ) + 1.0;
445      else
446          numberOfSubUnitRulers = dimX.subStepWidth>0 ? absRange / qAbs( dimX.subStepWidth ) + 1.0 : 1.0;
447  }else{
448      numberOfSubUnitRulers = absRange / qAbs( dimY.subStepWidth ) + 1.0;
449  }
450
451  // - calculate the absolute range in screen pixels:
452  const QPointF p1 = plane->translate( QPointF(dimX.start, dimY.start) );
453  const QPointF p2 = plane->translate( QPointF(dimX.end, dimY.end) );
454
455  double screenRange;
456  if ( isAbscissa() )
457  {
458      screenRange = qAbs ( p1.x() - p2.x() );
459  } else {
460      screenRange = qAbs ( p1.y() - p2.y() );
461  }
462
463  const bool useItemCountLabels = isAbscissa() && ! dimX.isCalculated;
464
465  const bool drawUnitRulers = screenRange / ( numberOfUnitRulers / dimX.stepWidth ) > MinimumPixelsBetweenRulers;
466  const bool drawSubUnitRulers =
467      (numberOfSubUnitRulers != 0.0) &&
468      (screenRange / numberOfSubUnitRulers > MinimumPixelsBetweenRulers);
469
470  const TextAttributes labelTA = textAttributes();
471  const bool drawLabels = labelTA.isVisible();
472
473  // - find the reference point at which to start drawing and the increment (line distance);
474  QPointF rulerRef;
475  const QRect areaGeoRect( areaGeometry() );
476  const QRect geoRect( geometry() );
477  QRectF rulerRect;
478  double rulerWidth;
479  double rulerHeight;
480
481  QPainter* const ptr = context->painter();

```

```

483
484 //for debugging: if( isAbscissa() )ptr->drawRect(areaGeoRect.adjusted(0,0,-1,-1));
485 //qDebug() << "          " << (isAbscissa() ? "Abscissa":"Ordinate") << "axis painting with geometr
486
487 // FIXME references are of course different for all locations:
488 rulerWidth = areaGeoRect.width();
489 rulerHeight = areaGeoRect.height();
490 switch( position() )
491 {
492 case Top:
493     rulerRef.setX( areaGeoRect.topLeft().x() );
494     rulerRef.setY( areaGeoRect.topLeft().y() + rulerHeight );
495     break;
496 case Bottom:
497     rulerRef.setX( areaGeoRect.bottomLeft().x() );
498     rulerRef.setY( areaGeoRect.bottomLeft().y() - rulerHeight );
499     break;
500 case Right:
501     rulerRef.setX( areaGeoRect.bottomRight().x() - rulerWidth );
502     rulerRef.setY( areaGeoRect.bottomRight().y() );
503     break;
504 case Left:
505     rulerRef.setX( areaGeoRect.bottomLeft().x() + rulerWidth );
506     rulerRef.setY( areaGeoRect.bottomLeft().y() );
507     break;
508 }
509
510 // set up the lines to paint:
511
512 // set up a map of integer positions,
513
514 // - starting with the fourth
515 // - the the halves
516 // - then the tens
517 // this will override all halves and fourth that hit a higher-order ruler
518 // MAKE SURE TO START AT (0, 0)!
519
520 // set up a reference point, a step vector and a unit vector for the drawing:
521
522 const qreal minValueY = dimY.start;
523 const qreal maxValueY = dimY.end;
524 const qreal minValueX = dimX.start;
525 const qreal maxValueX = dimX.end;
526 const bool isLogarithmicX = (dimX.calcMode == AbstractCoordinatePlane::Logarithmic );
527 const bool isLogarithmicY = (dimY.calcMode == AbstractCoordinatePlane::Logarithmic );
528 // #define AXES_PAINTING_DEBUG 1
529 #ifdef AXES_PAINTING_DEBUG
530     qDebug() << "CartesianAxis::paint: reference values:" << endl
531         << "-- range x/y: " << dimX.distance() << "/" << dimY.distance() << endl
532         << "-- absRange: " << absRange << endl
533         << "-- numberOfUnitRulers: " << numberOfUnitRulers << endl
534         << "-- screenRange: " << screenRange << endl
535         << "-- drawUnitRulers: " << drawUnitRulers << endl
536         << "-- drawLabels: " << drawLabels << endl
537         << "-- ruler reference point:: " << rulerRef << endl
538         << "-- minValueX: " << minValueX << "    maxValueX: " << maxValueX << endl
539         << "-- minValueY: " << minValueY << "    maxValueY: " << maxValueY << endl
540     ;
541 #endif
542
543 // solving issue #4075 in a quick way:
544 ptr->setPen ( labelTA.pen() ); // perhaps we want to add a setter method later?
545
546 //ptr->setPen ( Qt::black );
547
548 const QObject* referenceArea = plane->parent();
549

```

```

550 // that QVector contains all drawn x-ticks (so no subticks are drawn there also)
551 QVector< int > drawnXTicks;
552 // and that does the same for the y-ticks
553 QVector< int > drawnYTicks;
554
555 /*
556  * Find out if it is a bar diagram
557  * bar diagrams display their data per column
558  * we need to handle the last label another way
559  * 1 - Last label == QString null ( Header Labels )
560  * 2 - Display labels and ticks in the middle of the column
561  */
562
563 const bool isBarDiagram = referenceDiagramIsBarDiagram(d->diagram());
564
565 // this draws the unit rulers
566 if ( drawUnitRulers ) {
567     const QStringList labelsList( labels() );
568     const QStringList shortLabelsList( shortLabels() );
569     const int hardLabelsCount = labelsList.count();
570     const int shortLabelsCount = shortLabelsList.count();
571     bool useShortLabels = false;
572
573
574     bool useConfiguredStepsLabels = false;
575     QStringList headerLabels;
576     if( useItemCountLabels ){
577         //qDebug() << (isOrdinate() ? "is Ordinate" : "is Abscissa");
578         headerLabels =
579             isOrdinate()
580             ? d->diagram()->datasetLabels()
581             : d->diagram()->itemRowLabels();
582         // check if configured stepWidth
583         useConfiguredStepsLabels = isAbscissa() &&
584             dimX.stepWidth &&
585             (( headerLabels.count() - 1) / dimX.stepWidth ) != numberOfUnitRulers);
586         if( useConfiguredStepsLabels ) {
587             numberOfUnitRulers = ( headerLabels.count() - 1) / dimX.stepWidth;
588             // we need to register data values for the steps
589             // in case it is configured by the user
590             QStringList configuredStepsLabels;
591             double value = headerLabels.isEmpty() ? 0.0 : headerLabels.first().toDouble();
592             configuredStepsLabels << QString::number( value );
593             for ( int i = 0; i < numberOfUnitRulers; i++ ) {
594                 //qDebug() << value;
595                 value += dimX.stepWidth;
596                 configuredStepsLabels.append( QString::number( value ) );
597             }
598             headerLabels = configuredStepsLabels;
599         }
600
601         if ( isBarDiagram )
602             headerLabels.append( QString::null );
603     }
604
605
606     const int headerLabelsCount = headerLabels.count();
607     //qDebug() << "headerLabelsCount" << headerLabelsCount;
608
609     TextLayoutItem* labelItem =
610         drawLabels
611         ? new TextLayoutItem( QString::number( minValueY ),
612                             labelTA,
613                             referenceArea,
614                             KDChartEnums::MeasureOrientationMinimum,
615                             Qt::AlignLeft )
616         : 0;

```

```

617     TextLayoutItem* labelItem2 =
618         drawLabels
619         ? new TextLayoutItem( QString::number( minValueY ),
620                               labelTA,
621                               referenceArea,
622                               KDChartEnums::MeasureOrientationMinimum,
623                               Qt::AlignLeft )
624         : 0;
625     const QFontMetricsF met(
626         drawLabels
627         ? labelItem->realFont()
628         : QFontMetricsF( QApplication::font(), GlobalMeasureScaling::paintDevice() ) );
629     const qreal halfFontHeight = met.height() * 0.5;
630
631     if ( isAbscissa() ) {
632
633         // If we have a labels list AND a short labels list, we first find out,
634         // if there is enough space for the labels: if not, use the short labels.
635         if( drawLabels && hardLabelsCount > 0 && shortLabelsCount > 0 ){
636             bool labelsAreOverlapping = false;
637             int iLabel = 0;
638             qreal i = minValueX;
639             while ( i < maxValueX && !labelsAreOverlapping )
640             {
641                 if ( dimX.stepWidth != 1.0 && ! dim.isCalculated )
642                 {
643                     labelItem->setText( customizedLabel(QString::number( i, 'f', 0 )) );
644                     labelItem2->setText( customizedLabel(QString::number( i + dimX.stepWidth, 'f',
645
646
647                     } else {
648
649                         int index = iLabel;
650                         labelItem->setText( customizedLabel(labelsList[ index < hardLabelsCount ? ind
651                         labelItem2->setText( customizedLabel(labelsList[ index < hardLabelsCount - 1 ?
652                     }
653                     QPointF firstPos( i, 0.0 );
654                     firstPos = plane->translate( firstPos );
655
656                     QPointF secondPos( i + dimX.stepWidth, 0.0 );
657                     secondPos = plane->translate( secondPos );
658
659                     labelsAreOverlapping = labelItem->intersects( *labelItem2, firstPos, secondPos );
660                     if ( iLabel++ > hardLabelsCount - 1 )
661                         iLabel = 0;
662                     if ( isLogarithmicX )
663                         i *= 10.0;
664                     else
665                         i += dimX.stepWidth;
666                 }
667             }
668             useShortLabels = labelsAreOverlapping;
669         }
670
671         qreal labelDiff = dimX.stepWidth;
672         // qDebug() << "initial labelDiff " << labelDiff;
673         if ( drawLabels )
674         {
675             qreal i = minValueX;
676             int iLabel = 0;
677             const int precision = ( QString::number( labelDiff ).section( QLatin1Char('.'), 1, 2
678
679             while ( i + labelDiff < maxValueX )
680             {
681
682                 //qDebug() << "drawLabels" << drawLabels << " hardLabelsCount" << hardLabelsCount
683                 // << " dimX.stepWidth" << dimX.stepWidth << " dim.isCalculated" << dim.i

```

```

684         if ( !drawLabels || hardLabelsCount < 1 || ( dimX.stepWidth != 1.0 && ! dim.isCalculated ) )
685         {
686             // Check intersects for the header label - we need to pass the full string
687             // here and not only the i value.
688             if( useConfiguredStepsLabels ){
689                 labelItem->setText( customizedLabel(headerLabels[ iLabel ] ) );
690                 labelItem2->setText( customizedLabel(headerLabels[ iLabel+1 ] ) );
691             }else{
692                 //qDebug() << "i + labelDiff " << i + labelDiff;
693                 labelItem->setText( customizedLabel(headerLabelsCount > i && i >= 0 ?
694                     headerLabels[static_cast<int>(i)] :
695                     QString::number( i, 'f', precision ) ) );
696                 //
697                 //qDebug() << "1 - labelItem->text() " << labelItem->text();
698                 //qDebug() << "labelDiff" << labelDiff
699                 //
700                 //qDebug() << " index" << i+labelDiff << " count" << headerLabelsCount;
701                 labelItem2->setText( customizedLabel(headerLabelsCount > i + labelDiff &&
702                     headerLabels[static_cast<int>(i+labelDiff)] :
703                     QString::number( i + labelDiff, 'f', precision ) ) );
704                 //qDebug() << "2 - labelItem->text() " << labelItem->text();
705                 //qDebug() << "labelItem2->text() " << labelItem2->text();
706             }
707         } else {
708             const int idx = (iLabel < hardLabelsCount ) ? iLabel : 0;
709             const int idx2 = (iLabel < hardLabelsCount - 1) ? iLabel + 1 : 0;
710             const int shortIdx = (iLabel < shortLabelsCount ) ? iLabel : 0;
711             const int shortIdx2 = (iLabel < shortLabelsCount - 1) ? iLabel + 1 : 0;
712             labelItem->setText( customizedLabel(
713                 useShortLabels ? shortLabelsList[ shortIdx ] : labelsList[ idx ] ) );
714             labelItem2->setText( customizedLabel(
715                 useShortLabels ? shortLabelsList[ shortIdx2 ] : labelsList[ idx2 ] ) );
716         }
717
718         QPointF firstPos( i, 0.0 );
719         firstPos = plane->translate( firstPos );
720
721         QPointF secondPos( i + labelDiff, 0.0 );
722         secondPos = plane->translate( secondPos );
723
724         if ( labelItem->intersects( *labelItem2, firstPos, secondPos ) )
725         {
726             i = minValueX;
727
728             // fix for issue #4179:
729             labelDiff *= 10.0;
730             // old code:
731             // labelDiff += labelDiff;
732
733             iLabel = 0;
734         }
735         else
736         {
737             i += labelDiff;
738         }
739
740         ++iLabel;
741         if ( (iLabel > hardLabelsCount - 1) && !useConfiguredStepsLabels )
742         {
743             iLabel = 0;
744         }
745     }
746
747     int idxLabel = 0;
748     qreal iLabelF = minValueX;
749     //qDebug() << iLabelF;
750     qreal i = minValueX;

```

```

751         qreal labelStep = 0.0;
752         // qDebug() << "dimX.stepWidth:" << dimX.stepWidth << "labelDiff:" << labelDiff;
753         //dimX.stepWidth = 0.5;
754         while( i <= maxValueX ) {
755             // Line charts: we want the first tick to begin at 0.0 not at 0.5 otherwise labels and
756             // values does not fit each others
757             QPointF topPoint ( i + ( isBarDiagram ? 0.5 : 0.0 ), 0.0 );
758             QPointF bottomPoint ( topPoint );
759             topPoint = plane->translate( topPoint );
760             bottomPoint = plane->translate( bottomPoint );
761             topPoint.setY( rulerRef.y() + tickLength() );
762             bottomPoint.setY( rulerRef.y() );
763
764             const qreal translatedValue = topPoint.x();
765             const bool bIsVisibleLabel =
766                 ( translatedValue >= geoRect.left() && translatedValue <= geoRect.right() );
767
768             // fix for issue #4179:
769             bool painttick = bIsVisibleLabel && labelStep <= 0;;
770             // old code:
771             // bool painttick = true;
772
773             //Dont paint more ticks than we need
774             //when diagram type is Bar
775             if ( isBarDiagram && i == maxValueX )
776                 painttick = false;
777
778             if ( bIsVisibleLabel && painttick )
779                 ptr->drawLine( topPoint, bottomPoint );
780
781             drawnXTicks.append( static_cast<int>( topPoint.x() ) );
782             if( drawLabels ) {
783                 if( bIsVisibleLabel ){
784                     if ( isLogarithmicX )
785                         labelItem->setText( customizedLabel( QString::number( i, 'f', 0 ) ) );
786                     /* We dont need that
787                     * it causes header labels to be skipped even if there is enough
788                     * space for them to displayed.
789                     * Commenting for now - I need to test more in details - Let me know if I am wr
790                     */
791                     /*
792                     else if( (dimX.stepWidth != 1.0) && ! dimX.isCalculated ) {
793                         labelItem->setText( customizedLabel( QString::number( i, 'f', 0 ) ) );
794                     }
795                     */
796                     else {
797                         const int idx = idxLabel + static_cast<int>(minValueX);
798                         labelItem->setText(
799                             customizedLabel(
800                                 hardLabelsCount
801                                 ? ( useShortLabels ? shortLabelsList[ idx ] : labelsList[ idx ]
802                                   : ( headerLabelsCount ? headerLabels[ idx ] : QString::number( iLabelF
803                                     //qDebug() << "x - labelItem->text() " << labelItem->text() << headerLabel
804                                 )
805                         // No need to call labelItem->setParentWidget(), since we are using
806                         // the layout item temporarily only.
807                         if( labelStep <= 0 ) {
808                             const PainterSaver p( ptr );
809                             const QSize size( labelItem->sizeHint() );
810                             labelItem->setGeometry(
811                                 QRect(
812                                     QPoint(
813                                         static_cast<int>( topPoint.x() - size.width() / 2 ),
814                                         static_cast<int>( topPoint.y() +
815                                             ( position() == Bottom
816                                                 ? halfFontHeight
817                                                 : ((halfFontHeight + size.height()) * -1.0

```

```

818         size ) );
819
820         QRect labelGeo = labelItem->geometry();
821         // if our item would only half fit, we disable clipping for that one
822         if( labelGeo.left() < geoRect.left() && labelGeo.right() > geoRect.left() )
823             ptr->setClipping( false );
824         if( labelGeo.left() < geoRect.right() && labelGeo.right() > geoRect.right() )
825             ptr->setClipping( false );
826
827         labelItem->setGeometry( labelGeo );
828
829         labelStep = labelDiff - dimX.stepWidth;
830         labelItem->paint( ptr );
831
832         // do not call customizedLabel() again:
833         labelItem2->setText( labelItem->text() );
834
835     } else {
836         labelStep -= dimX.stepWidth;
837     }
838 }
839
840 if( hardLabelsCount ) {
841     if( useShortLabels && idxLabel >= shortLabelsCount - 1 )
842         idxLabel = 0;
843     else if( !useShortLabels && idxLabel >= hardLabelsCount - 1 )
844         idxLabel = 0;
845     else{
846         idxLabel += static_cast<int>(dimX.stepWidth);
847         //qDebug() << "dimX.stepWidth:" << dimX.stepWidth << "   idxLabel:" << idxLabel;
848     }
849 } else if( headerLabelsCount ) {
850     if( idxLabel >= headerLabelsCount - 1 ) {
851         idxLabel = 0;
852     }else
853         ++idxLabel;
854 } else {
855     iLabelF += dimX.stepWidth;
856 }
857
858 if ( isLogarithmicX )
859 {
860     i *= 10.0;
861     if( i == 0.0 )
862         i = 1.0; //std::numeric_limits< double >::epsilon();
863 }
864 else
865 {
866     i += dimX.stepWidth;
867 }
868 }
869 } else {
870     const PainterSaver p( ptr );
871     const double maxLimit = maxValueY;
872     const double steg = dimY.stepWidth;
873     int maxLabelsWidth = 0;
874     qreal labelValue;
875     if( drawLabels && position() == Right ){
876         // Find the widest label, so we to know how much we need to right-shift
877         // our labels, to get them drawn right aligned:
878         labelValue = minValueY;
879         while ( labelValue <= maxLimit ) {
880             const QString labelText = diagram()->unitPrefix( static_cast< int >( labelValue ),
881                 QString::number( labelValue ) +
882                 diagram()->unitSuffix( static_cast< int >( labelValue ),
883             labelItem->setText( customizedLabel( labelText ) );
884             maxLabelsWidth = qMax( maxLabelsWidth, labelItem->sizeHint().width() );

```



```

885         calculateNextLabel( labelValue, steg, isLogarithmicY );
886     }
887 }
888
889 ptr->setClipping( false );
890 labelValue = minValueY;
891 qreal step = steg;
892 bool nextLabel = false;
893 //qDebug("minValueY: %f    maxLimit: %f    steg: %f", minValueY, maxLimit, steg);
894
895 if( drawLabels )
896 {
897     // first calculate the steps depending on labels colision
898     while( labelValue <= maxLimit ) {
899         QPointF leftPoint = plane->translate( QPointF( 0, labelValue ) );
900         const qreal translatedValue = leftPoint.y();
901         //qDebug() << "geoRect:" << geoRect << "    geoRect.top()" << geoRect.top()
902         //<< "geoRect.bottom()" << geoRect.bottom() << "    translatedValue:" << translatedValue;
903         if( translatedValue > geoRect.top() && translatedValue <= geoRect.bottom() ){
904             const QString labelText = diagram()->unitPrefix( static_cast< int >( labelValue ) ) +
905                                     QString::number( labelValue ) +
906                                     diagram()->unitSuffix( static_cast< int >( labelValue ) );
907             const QString label2Text = diagram()->unitPrefix( static_cast< int >( labelValue + step ) ) +
908                                     QString::number( labelValue + step ) +
909                                     diagram()->unitSuffix( static_cast< int >( labelValue + step ) );
910             labelItem->setText( customizedLabel( labelText ) );
911             labelItem2->setText( customizedLabel( QString::number( labelValue + step ) ) );
912             QPointF nextPoint = plane->translate( QPointF( 0, labelValue + step ) );
913             if ( labelItem->intersects( *labelItem2, leftPoint, nextPoint ) )
914             {
915                 step += steg;
916                 nextLabel = false;
917             }else{
918                 nextLabel = true;
919             }
920         }else{
921             nextLabel = true;
922         }
923     }
924
925     if ( nextLabel || isLogarithmicY )
926         calculateNextLabel( labelValue, step, isLogarithmicY );
927     else
928         labelValue = minValueY;
929 }
930
931 // Second - Paint the labels
932 labelValue = minValueY;
933 //qDebug() << "axis labels starting at" << labelValue << "step width" << step;
934 while( labelValue <= maxLimit ) {
935     //qDebug() << "value now" << labelValue;
936     const QString labelText = diagram()->unitPrefix( static_cast< int >( labelValue ) ) +
937                             QString::number( labelValue ) +
938                             diagram()->unitSuffix( static_cast< int >( labelValue ) );
939     labelItem->setText( customizedLabel( labelText ) );
940     QPointF leftPoint = plane->translate( QPointF( 0, labelValue ) );
941     QPointF rightPoint ( 0.0, labelValue );
942     rightPoint = plane->translate( rightPoint );
943     leftPoint.setX( rulerRef.x() + tickLength() );
944     rightPoint.setX( rulerRef.x() );
945
946     const qreal translatedValue = rightPoint.y();
947     const bool bIsVisibleLabel =
948         ( translatedValue >= geoRect.top() && translatedValue <= geoRect.bottom() );
949
950     if( bIsVisibleLabel ){
951         ptr->drawLine( leftPoint, rightPoint );
952     }
953 }

```

```

952         drawnYTicks.append( static_cast<int>( leftPoint.y() ) );
953         const QSize labelSize( labelItem->sizeHint() );
954         leftPoint.setX( leftPoint.x() );
955         const int x =
956             static_cast<int>( leftPoint.x() + met.height() * ( position() == Left ? -0.6 : 0.6 ) );
957         - ( position() == Left ? labelSize.width() : (labelSize.width() - maxLabelWidth) );
958         const int y =
959             static_cast<int>( leftPoint.y() - ( met.ascent() + met.descent() ) * 0.6 );
960         labelItem->setGeometry( QRect( QPoint( x, y ), labelSize ) );
961         labelItem->paint( ptr );
962     }
963
964     calculateNextLabel( labelValue, step, isLogarithmicY );
965 }
966 }
967 }
968 delete labelItem;
969 delete labelItem2;
970 }
971
972 // this draws the subunit rulers
973 if ( drawSubUnitRulers ) {
974     d->drawSubUnitRulers( ptr, plane, dim, rulerRef, isAbscissa() ? drawnXTicks : drawnYTicks );
975 }
976
977 if( ! titleText().isEmpty() ){
978     d->drawTitleText( ptr, plane, areaGeoRect );
979 }
980
981 //qDebug() << "KDChart::CartesianAxis::paintCtx() done.";
982 }

```

#### 9.19.4.35 void AbstractAreaBase::paintFrame (QPainter & *painter*, const QRect & *rectangle*) [virtual, inherited]

Definition at line 204 of file KDChartAbstractAreaBase.cpp.

References `d`, and `KDChart::AbstractAreaBase::paintFrameAttributes()`.

Referenced by `KDChart::TextArea::paintAll()`, `KDChart::AbstractAreaWidget::paintAll()`, and `KDChart::AbstractArea::paintAll()`.

```

205 {
206     Q_ASSERT_X ( d != 0, "AbstractAreaBase::paintFrame()",
207                 "Private class was not initialized!" );
208     paintFrameAttributes( painter, rect, d->frameAttributes );
209 }

```

#### 9.19.4.36 void AbstractAreaBase::paintFrameAttributes (QPainter & *painter*, const QRect & *rectangle*, const KDChart::FrameAttributes & *attributes*) [static, inherited]

Definition at line 177 of file KDChartAbstractAreaBase.cpp.

References `KDChart::FrameAttributes::isVisible()`, and `KDChart::FrameAttributes::pen()`.

Referenced by `KDChart::AbstractAreaBase::paintFrame()`.

```

179 {
180
181     if( !attributes.isVisible() ) return;

```

```

182
183     // Note: We set the brush to NoBrush explicitly here.
184     //         Otherwise we might get a filled rectangle, so any
185     //         previously drawn background would be overwritten by that area.
186
187     const QPen   oldPen(   painter.pen() );
188     const QBrush oldBrush( painter.brush() );
189     painter.setPen(   attributes.pen() );
190     painter.setBrush( Qt::NoBrush );
191     painter.drawRect( rect.adjusted( 0, 0, -1, -1 ) );
192     painter.setBrush( oldBrush );
193     painter.setPen(   oldPen );
194 }

```

#### 9.19.4.37 void AbstractArea::paintIntoRect (QPainter & painter, const QRect & rect)

[virtual, inherited]

Draws the background and frame, then calls [paint\(\)](#).

In most cases there is no need to overwrite this method in a derived class, but you would overwrite [AbstractLayoutItem::paint\(\)](#) instead.

Definition at line 111 of file KDChartAbstractArea.cpp.

References [KDChart::AbstractArea::paintAll\(\)](#).

```

112 {
113     const QRect oldGeometry( geometry() );
114     if( oldGeometry != rect )
115         setGeometry( rect );
116     painter.translate( rect.left(), rect.top() );
117     paintAll( painter );
118     painter.translate( -rect.left(), -rect.top() );
119     if( oldGeometry != rect )
120         setGeometry( oldGeometry );
121 }

```

#### 9.19.4.38 QLayout\* KDChart::AbstractLayoutItem::parentLayout ()

[inherited]

Definition at line 76 of file KDChartLayoutItems.h.

```

77     {
78         return mParentLayout;
79     }

```

#### 9.19.4.39 const [CartesianAxis::Position](#) CartesianAxis::position () const

[virtual]

Definition at line 145 of file KDChartCartesianAxis.cpp.

References [d](#).

Referenced by [compare\(\)](#), [expandingDirections\(\)](#), [isAbscissa\(\)](#), [isOrdinate\(\)](#), [maximumSize\(\)](#), [paintCtx\(\)](#), and [tickLength\(\)](#).

```

146 {
147     return d->position;
148 }

```

**9.19.4.40** `void KDChart::AbstractArea::positionChanged (AbstractArea \*)` [signal, inherited]

Referenced by `KDChart::AbstractArea::positionHasChanged()`.

**9.19.4.41** `void AbstractArea::positionHasChanged ()` [protected, virtual, inherited]

Reimplemented from [KDChart::AbstractAreaBase](#).

Definition at line 155 of file `KDChartAbstractArea.cpp`.

References `KDChart::AbstractArea::positionChanged()`.

```
156 {
157     emit positionChanged( this );
158 }
```

**9.19.4.42** `void KDChart::AbstractLayoutItem::removeFromParentLayout ()` [inherited]

Definition at line 80 of file `KDChartLayoutItems.h`.

Referenced by `KDChart::Chart::takeCoordinatePlane()`.

```
81     {
82         if( mParentLayout ){
83             if( widget() )
84                 mParentLayout->removeWidget( widget() );
85             else
86                 mParentLayout->removeItem( this );
87         }
88     }
```

**9.19.4.43** `void CartesianAxis::resetTitleTextAttributes ()`

Reset the title text attributes to the built-in default:.

Same font and pen as [AbstractAxis::textAttributes\(\)](#) and 1.5 times their size.

Definition at line 127 of file `KDChartCartesianAxis.cpp`.

References `d`, and `layoutPlanes()`.

```
128 {
129     d->useDefaultTextAttributes = true;
130     layoutPlanes();
131 }
```

**9.19.4.44** `int AbstractArea::rightOverlap (bool doNotRecalculate = false) const` [virtual, inherited]

This is called at layout time by [KDChart::AutoSpacerLayoutItem::sizeHint\(\)](#).

The method triggers `AbstractArea::sizeHint()` to find out the amount of overlap at the right edge of the area.

**Note:**

The default implementation is not using any caching, it might make sense to implement a more sophisticated solution for derived classes that have complex work to do in `sizeHint()`. All we have here is a primitive flag to be set by the caller if it is sure that no `sizeHint()` needs to be called.

Definition at line 85 of file `KDChartAbstractArea.cpp`.

References `d`.

Referenced by `maximumSize()`.

```
86 {
87     // Re-calculate the sizes,
88     // so we also get the amountOf..Overlap members set newly:
89     if( ! doNotRecalculate )
90         sizeHint();
91     return d->amountOfRightOverlap;
92 }
```

**9.19.4.45 void AbstractAreaBase::setBackgroundAttributes (const [BackgroundAttributes](#) & a)**  
[[inherited](#)]

Definition at line 111 of file `KDChartAbstractAreaBase.cpp`.

References `d`, and `KDChart::AbstractAreaBase::positionHasChanged()`.

```
112 {
113     if( d->backgroundAttributes == a )
114         return;
115
116     d->backgroundAttributes = a;
117     positionHasChanged();
118 }
```

**9.19.4.46 void AbstractAreaBase::setFrameAttributes (const [FrameAttributes](#) & a)**  
[[inherited](#)]

Definition at line 97 of file `KDChartAbstractAreaBase.cpp`.

References `d`, and `KDChart::AbstractAreaBase::positionHasChanged()`.

Referenced by `KDChart::Legend::clone()`.

```
98 {
99     if( d->frameAttributes == a )
100         return;
101
102     d->frameAttributes = a;
103     positionHasChanged();
104 }
```

**9.19.4.47 void CartesianAxis::setGeometry (const [QRect](#) & r)** [[virtual](#)]

pure virtual in [QLayoutItem](#)

Implements [KDChart::AbstractAxis](#).

Definition at line 1227 of file KDChartCartesianAxis.cpp.

References [d](#).

```
1228 {
1229 //      qDebug() << "KDChart::CartesianAxis::setGeometry(" << r << ") called"
1230 //              << (isAbscissa() ? "for Abscissa":"for Ordinate") << "axis";
1231      d->geometry = r;
1232 }
```

#### 9.19.4.48 void AbstractAxis::setLabels (const QStringList & *list*) [inherited]

Use this to specify your own set of strings, to be used as axis labels.

Labels specified via setLabels take precedence: If a non-empty list is passed, KD [Chart](#) will use these strings as axis labels, instead of calculating them.

If you a smaller number of strings than the number of labels drawn at this axis, KD [Chart](#) will iterate over the list, repeating the strings, until all labels are drawn. As an example you could specify the seven days of the week as abscissa labels, which would be repeatedly used then.

By passing an empty QStringList you can reset the default behaviour.

**See also:**

[labels](#), [setShortLabels](#)

Definition at line 267 of file KDChartAbstractAxis.cpp.

References [d](#), and [KDChart::AbstractAxis::update\(\)](#).

```
268 {
269     if( d->hardLabels == list )
270         return;
271
272     d->hardLabels = list;
273     update();
274 }
```

#### 9.19.4.49 void KDChart::AbstractLayoutItem::setParentLayout (QLayout \* *lay*) [inherited]

Definition at line 72 of file KDChartLayoutItems.h.

```
73     {
74         mParentLayout = lay;
75     }
```

#### 9.19.4.50 void KDChart::AbstractLayoutItem::setParentWidget (QWidget \* *widget*) [virtual, inherited]

Inform the item about its widget: This enables the item, to trigger that widget's update, whenever the size of the item's contents has changed.

Thus, you need to call setParentWidget on every item, that has a non-fixed size.

Definition at line 64 of file KDChartLayoutItems.cpp.

References KDChart::AbstractLayoutItem::mParent.

Referenced by KDChart::HeaderFooter::setParent(), and KDChart::AbstractCartesianDiagram::takeAxis().

```
65 {  
66     mParent = widget;  
67 }
```

#### 9.19.4.51 void CartesianAxis::setPosition ([Position](#) *p*) [virtual]

Definition at line 139 of file KDChartCartesianAxis.cpp.

References d, and layoutPlanes().

```
140 {  
141     d->position = p;  
142     layoutPlanes();  
143 }
```

#### 9.19.4.52 void AbstractAxis::setShortLabels (const QStringList & *list*) [inherited]

Use this to specify your own set of strings, to be used as axis labels, in case the normal labels are too long.

##### Note:

Setting done via setShortLabels will be ignored, if you did not pass a non-empty string list via setLabels too!

By passing an empty QStringList you can reset the default behaviour.

##### See also:

[shortLabels](#), [setLabels](#)

Definition at line 297 of file KDChartAbstractAxis.cpp.

References d, and KDChart::AbstractAxis::update().

```
298 {  
299     if( d->hardShortLabels == list )  
300         return;  
301  
302     d->hardShortLabels = list;  
303     update();  
304 }
```

#### 9.19.4.53 void AbstractAxis::setTextAttributes (const [TextAttributes](#) & *a*) [inherited]

Use this to specify the text attributes to be used for axis labels.

By default, the reference area will be set at painting time. It will be the then-valid coordinate plane's parent widget, so normally, it will be the [KDChart::Chart](#). Thus the labels of all of your axes in all of your diagrams within that [Chart](#) will be drawn in same font size, by default.

See also:

[textAttributes](#), [setLabels](#)

Definition at line 231 of file KDChartAbstractAxis.cpp.

References [d](#), and [KDChart::AbstractAxis::update\(\)](#).

```
232 {  
233     if( d->textAttributes == a )  
234         return;  
235  
236     d->textAttributes = a;  
237     update();  
238 }
```

#### 9.19.4.54 void CartesianAxis::setTitleText (const QString & text)

Definition at line 97 of file KDChartCartesianAxis.cpp.

References [d](#), and [layoutPlanes\(\)](#).

```
98 {  
99     d->titleText = text;  
100     layoutPlanes();  
101 }
```

#### 9.19.4.55 void CartesianAxis::setTitleTextAttributes (const [TextAttributes](#) & a)

Definition at line 108 of file KDChartCartesianAxis.cpp.

References [d](#), and [layoutPlanes\(\)](#).

```
109 {  
110     d->titleTextAttributes = a;  
111     d->useDefaultTextAttributes = false;  
112     layoutPlanes();  
113 }
```

#### 9.19.4.56 QStringList AbstractAxis::shortLabels () const [inherited]

Returns a list of strings, that are used as axis labels, as set via [setShortLabels](#).

**Note:**

Setting done via [setShortLabels](#) will be ignored, if you did not pass a non-empty string list via [setLabels](#) too!

See also:

[setShortLabels](#)

Definition at line 314 of file KDChartAbstractAxis.cpp.

References [d](#).

Referenced by [KDChart::AbstractAxis::compare\(\)](#), and [paintCtx\(\)](#).



```
315 {  
316     return d->hardShortLabels;  
317 }
```

#### 9.19.4.57 QSize CartesianAxis::sizeHint () const [virtual]

pure virtual in [QLayoutItem](#)

Definition at line 1222 of file KDChartCartesianAxis.cpp.

References [maximumSize\(\)](#).

```
1223 {  
1224     return maximumSize();  
1225 }
```

#### 9.19.4.58 void KDChart::AbstractLayoutItem::sizeHintChanged () const [virtual, inherited]

Report changed size hint: ask the parent widget to recalculate the layout.

Definition at line 86 of file KDChartLayoutItems.cpp.

References [KDChart::AbstractLayoutItem::mParent](#).

Referenced by [KDChart::TextLayoutItem::sizeHint\(\)](#).

```
87 {  
88     // This is exactly like what QWidget::updateGeometry does.  
89     qDebug("KDChart::AbstractLayoutItem::sizeHintChanged() called");  
90     if( mParent ) {  
91         if ( mParent->layout() )  
92             mParent->layout()->invalidate();  
93         else  
94             QApplication::postEvent( mParent, new QEvent( QEvent::LayoutRequest ) );  
95     }  
96 }
```

#### 9.19.4.59 [TextAttributes](#) AbstractAxis::textAttributes () const [inherited]

Returns the text attributes to be used for axis labels.

See also:

[setTextAttributes](#)

Definition at line 245 of file KDChartAbstractAxis.cpp.

References [d](#).

Referenced by [KDChart::AbstractAxis::compare\(\)](#), [maximumSize\(\)](#), [paintCtx\(\)](#), and [titleTextAttributes\(\)](#).

```
246 {  
247     return d->textAttributes;  
248 }
```

**9.19.4.60 int CartesianAxis::tickLength (bool *subUnitTicks* = false) const**

Definition at line 1239 of file KDChartCartesianAxis.cpp.

References `isAbscissa()`, `Left`, `position()`, and `Top`.

Referenced by `maximumSize()`, and `paintCtx()`.

```

1240 {
1241     int result = 0;
1242
1243     if ( isAbscissa() ) {
1244         result = position() == Top ? -4 : 3;
1245     } else {
1246         result = position() == Left ? -4 : 3;
1247     }
1248
1249     if ( subUnitTicks )
1250         result = result < 0 ? result + 1 : result - 1;
1251
1252     return result;
1253 }
```

**9.19.4.61 QString CartesianAxis::titleText () const**

Definition at line 103 of file KDChartCartesianAxis.cpp.

References `d`.

Referenced by `compare()`, `maximumSize()`, and `paintCtx()`.

```

104 {
105     return d->titleText;
106 }
```

**9.19.4.62 [TextAttributes](#) CartesianAxis::titleTextAttributes () const**

Returns the text attributes that will be used for displaying the title text.

This is either the text attributes as specified by `setTitleTextAttributes`, or (if [setTitleTextAttributes\(\)](#) was not called) the default text attributes.

**See also:**

[resetTitleTextAttributes](#), [hasDefaultTitleTextAttributes](#)

Definition at line 115 of file KDChartCartesianAxis.cpp.

References `d`, `KDChart::TextAttributes::fontSize()`, `hasDefaultTitleTextAttributes()`, `KDChart::TextAttributes::setFontSize()`, and `KDChart::AbstractAxis::textAttributes()`.

Referenced by `compare()`.

```

116 {
117     if( hasDefaultTitleTextAttributes() ){
118         TextAttributes ta( textAttributes() );
119         Measure me( ta.fontSize() );
120         me.setValue( me.value() * 1.5 );
121         ta.setFontSize( me );

```

```

122         return ta;
123     }
124     return d->titleTextAttributes;
125 }

```

**9.19.4.63** `int AbstractArea::topOverlap (bool doNotRecalculate = false) const` [virtual, inherited]

This is called at layout time by [KDCart::AutoSpacerLayoutItem::sizeHint\(\)](#).

The method triggers AbstractArea::sizeHint() to find out the amount of overlap at the top edge of the area.

**Note:**

The default implementation is not using any caching, it might make sense to implement a more sophisticated solution for derived classes that have complex work to do in sizeHint(). All we have here is a primitive flag to be set by the caller if it is sure that no sizeHint() needs to be called.

Definition at line 93 of file KDCartAbstractArea.cpp.

References [d](#).

Referenced by [maximumSize\(\)](#).

```

94 {
95     // Re-calculate the sizes,
96     // so we also get the amountOf..Overlap members set newly:
97     if( ! doNotRecalculate )
98         sizeHint();
99     return d->amountOfTopOverlap;
100 }

```

**9.19.4.64** `void KDCart::AbstractAxis::update ()` [slot, inherited]

Definition at line 341 of file KDCartAbstractAxis.cpp.

References [d](#).

Referenced by [KDCart::AbstractAxis::connectSignals\(\)](#), [KDCart::AbstractAxis::setLabels\(\)](#), [KDCart::AbstractAxis::setShortLabels\(\)](#), and [KDCart::AbstractAxis::setTextAttributes\(\)](#).

```

342 {
343     if( d->diagram() )
344         d->diagram()->update();
345 }

```

## 9.19.5 Member Data Documentation

**9.19.5.1** `QWidget* KDCart::AbstractLayoutItem::mParent` [protected, inherited]

Definition at line 90 of file KDCartLayoutItems.h.

Referenced by [KDCart::AbstractLayoutItem::setParentWidget\(\)](#), [KDCart::TextLayoutItem::setText\(\)](#), [KDCart::TextLayoutItem::setTextAttributes\(\)](#), and [KDCart::AbstractLayoutItem::sizeHintChanged\(\)](#).

#### 9.19.5.2 `QLayout*` [KDChart::AbstractLayoutItem::mParentLayout](#) [protected, inherited]

Definition at line 91 of file `KDChartLayoutItems.h`.

Referenced by `KDChart::AutoSpacerLayoutItem::paint()`.

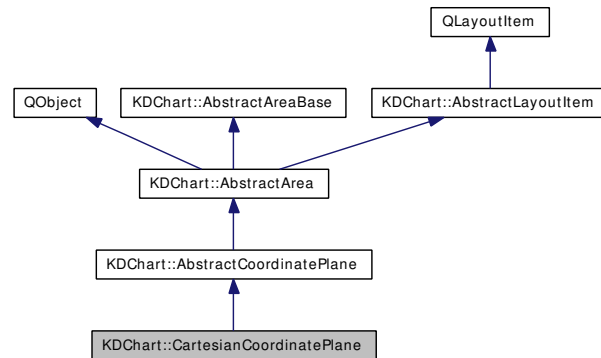
The documentation for this class was generated from the following files:

- [KDChartCartesianAxis.h](#)
- [KDChartCartesianAxis.cpp](#)

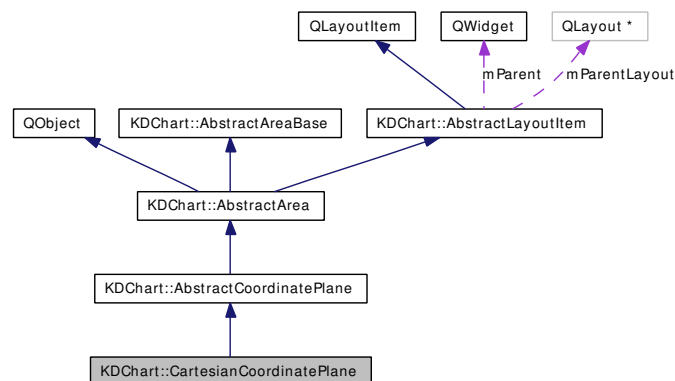
## 9.20 KDChart::CartesianCoordinatePlane Class Reference

```
#include <KDChartCartesianCoordinatePlane.h>
```

Inheritance diagram for KDChart::CartesianCoordinatePlane:



Collaboration diagram for KDChart::CartesianCoordinatePlane:



### 9.20.1 Detailed Description

Cartesian coordinate plane.

Definition at line 42 of file `KDChartCartesianCoordinatePlane.h`.

#### Public Types

- enum [AxesCalcMode](#) {  
[Linear](#),  
[Logarithmic](#) }

#### Public Slots

- void [adjustHorizontalRangeToData](#) ()

*Adjust horizontal range settings to the ranges covered by the model's data values.*

- void [adjustRangesToData](#) ()  
*Adjust both, horizontal and vertical range settings to the ranges covered by the model's data values.*
- void [adjustVerticalRangeToData](#) ()  
*Adjust vertical range settings to the ranges covered by the model's data values.*
- void [layoutPlanes](#) ()  
*Calling [layoutPlanes\(\)](#) on the plane triggers the global `KDChart::Chart::slotLayoutPlanes()`.*
- void [relayout](#) ()  
*Calling [relayout\(\)](#) on the plane triggers the global `KDChart::Chart::slotRelayout()`.*
- void [setGridNeedsRecalculate](#) ()  
*Used by the chart to clear the cached grid data.*
- void [update](#) ()  
*Calling [update\(\)](#) on the plane triggers the global `KDChart::Chart::update()`.*

## Signals

- void [destroyedCoordinatePlane](#) ([AbstractCoordinatePlane](#) \*)  
*Emitted when this coordinate plane is destroyed.*
- void [needLayoutPlanes](#) ()  
*Emitted when plane needs to trigger the Chart's layouting of the coord.*
- void [needRelayout](#) ()  
*Emitted when plane needs to trigger the Chart's layouting.*
- void [needUpdate](#) ()  
*Emitted when plane needs to update its drawings.*
- void [positionChanged](#) ([AbstractArea](#) \*)
- void [propertiesChanged](#) ()  
*Emitted upon change of a property of the Coordinate Plane or any of its components.*

## Public Member Functions

- void [addDiagram](#) ([AbstractDiagram](#) \*diagram)  
*Adds a diagram to this coordinate plane.*
- void [alignToReferencePoint](#) (const [RelativePosition](#) &position)
- const bool [autoAdjustGridToZoom](#) () const  
*Return the status of the built-in grid adjusting feature.*

- unsigned int [autoAdjustHorizontalRangeToData](#) () const  
*Returns the maximal allowed percent of the horizontal space covered by the coordinate plane that may be empty.*
- unsigned int [autoAdjustVerticalRangeToData](#) () const  
*Returns the maximal allowed percent of the vertical space covered by the coordinate plane that may be empty.*
- [AxesCalcMode](#) [axesCalcModeX](#) () const
- [AxesCalcMode](#) [axesCalcModeY](#) () const
- [BackgroundAttributes](#) [backgroundAttributes](#) () const
- virtual int [bottomOverlap](#) (bool doNotRecalculate=false) const  
*This is called at layout time by KDChart::AutoSpacerLayoutItem::sizeHint().*
- [CartesianCoordinatePlane](#) ([Chart](#) \*parent=0)
- bool [compare](#) (const [AbstractAreaBase](#) \*other) const  
*Returns true if both areas have the same settings.*
- [AbstractDiagram](#) \* [diagram](#) ()  
**Returns:**  
*The first diagram associated with this coordinate plane.*
- [ConstAbstractDiagramList](#) [diagrams](#) () const  
**Returns:**  
*The list of diagrams associated with this coordinate plane.*
- [AbstractDiagramList](#) [diagrams](#) ()  
**Returns:**  
*The list of diagrams associated with this coordinate plane.*
- bool [doesIsometricScaling](#) () const
- virtual Qt::Orientations [expandingDirections](#) () const  
*pure virtual in [QLayoutItem](#)*
- [FrameAttributes](#) [frameAttributes](#) () const
- virtual QRect [geometry](#) () const  
*pure virtual in [QLayoutItem](#)*
- void [getFrameLeadings](#) (int &left, int &top, int &right, int &bottom) const
- [GridAttributes](#) [globalGridAttributes](#) () const  
**Returns:**  
*The grid attributes used by this coordinate plane.*
- const [GridAttributes](#) [gridAttributes](#) (Qt::Orientation orientation) const  
**Returns:**  
*The attributes used for grid lines drawn in horizontal direction (or in vertical direction, resp.*
- [DataDimensionsList](#) [gridDimensionsList](#) ()  
*Returns the dimensions used for drawing the grid lines.*

- bool [hasFixedDataCoordinateSpaceRelation](#) () const
- bool [hasOwnGridAttributes](#) (Qt::Orientation orientation) const

**Returns:**

*Returns whether the grid attributes have been set for the respective direction via [setGridAttributes\(orientation\)](#).*

- QPair< qreal, qreal > [horizontalRange](#) () const

**Returns:**

*The largest and smallest visible horizontal value space value.*

- virtual bool [isEmpty](#) () const  
*pure virtual in [QLayoutItem](#)*

- bool [isHorizontalRangeReversed](#) () const

**Returns:**

*Whether the horizontal range is reversed or not*

- bool [isRubberBandZoomingEnabled](#) () const

**Returns:**

*Whether zooming with a rubber band using the mouse is enabled.*

- bool [isVerticalRangeReversed](#) () const

**Returns:**

*Whether the vertical range is reversed or not*

- const bool [isVisiblePoint](#) (const QPointF &point) const

*Tests, if a point is visible on the coordinate plane.*

- virtual int [leftOverlap](#) (bool doNotRecalculate=false) const

*This is called at layout time by [KDChart::AutoSpacerLayoutItem::sizeHint\(\)](#).*

- virtual QSize [maximumSize](#) () const  
*pure virtual in [QLayoutItem](#)*

- virtual QSize [minimumSize](#) () const  
*pure virtual in [QLayoutItem](#)*

- virtual QSize [minimumSizeHint](#) () const  
*[reimplemented]*

- void [mouseDoubleClickEvent](#) (QMouseEvent \*event)
- void [mouseMoveEvent](#) (QMouseEvent \*event)
- void [mousePressEvent](#) (QMouseEvent \*event)
- void [mouseReleaseEvent](#) (QMouseEvent \*event)
- virtual void [paint](#) (QPainter \*)  
*reimpl*

- virtual void [paintAll](#) (QPainter &painter)



Call `paintAll`, if you want the background and the frame to be drawn before the normal `paint()` is invoked automatically.

- virtual void `paintBackground` (QPainter &painter, const QRect &rectangle)
- virtual void `paintCtx` (PaintContext \*context)

Default impl: Paint the complete item using its layouted position and size.

- virtual void `paintFrame` (QPainter &painter, const QRect &rectangle)
- virtual void `paintIntoRect` (QPainter &painter, const QRect &rect)

Draws the background and frame, then calls `paint()`.

- const Chart \* `parent` () const
- Chart \* `parent` ()
- QLayout \* `parentLayout` ()
- AbstractCoordinatePlane \* `referenceCoordinatePlane` () const

There are two ways, in which planes can be caused to interact, in where they are put layouting wise: The first is the reference plane.

- void `removeFromParentLayout` ()
- virtual void `replaceDiagram` (AbstractDiagram \*diagram, AbstractDiagram \*oldDiagram=0)

Replaces the old diagram, or appends the diagram, if there is none yet.

- void `resetGridAttributes` (Qt::Orientation orientation)

Reset the attributes to be used for grid lines drawn in horizontal direction (or in vertical direction, resp.

- virtual int `rightOverlap` (bool doNotRecalculate=false) const
- This is called at layout time by `KDChart::AutoSpacerLayoutItem::sizeHint()`.

- void `setAutoAdjustGridToZoom` (bool autoAdjust)

Disable / re-enable the built-in grid adjusting feature.

- void `setAutoAdjustHorizontalRangeToData` (unsigned int percentEmpty=67)

Automatically adjust horizontal range settings to the ranges covered by the model's values, when ever the data have changed, and then emit `horizontalRangeAutomaticallyAdjusted`.

- void `setAutoAdjustVerticalRangeToData` (unsigned int percentEmpty=67)

Automatically adjust vertical range settings to the ranges covered by the model's values, when ever the data have changed, and then emit `verticalRangeAutomaticallyAdjusted`.

- void `setAxesCalcModes` (AxesCalcMode mode)

Specifies the calculation modes for all axes.

- void `setAxesCalcModeX` (AxesCalcMode mode)

Specifies the calculation mode for all Abscissa axes.

- void `setAxesCalcModeY` (AxesCalcMode mode)

Specifies the calculation mode for all Ordinate axes.

- void `setBackgroundAttributes` (const BackgroundAttributes &a)
- void `setFixedDataCoordinateSpaceRelation` (bool fixed)
- void `setFrameAttributes` (const FrameAttributes &a)

- void [setGeometry](#) (const QRect &r)  
*reimplement from [AbstractCoordinatePlane](#)*
- void [setGlobalGridAttributes](#) (const [GridAttributes](#) &)  
*Set the grid attributes to be used by this coordinate plane.*
- void [setGridAttributes](#) (Qt::Orientation orientation, const [GridAttributes](#) &)  
*Set the attributes to be used for grid lines drawn in horizontal direction (or in vertical direction, resp.*
- void [setHorizontalRange](#) (const QPair< qreal, qreal > &range)  
*Set the boundaries of the visible value space displayed in horizontal direction.*
- void [setHorizontalRangeReversed](#) (bool reverse)  
*Sets whether the horizontal range should be reversed or not, i.e.*
- void [setIsometricScaling](#) (bool onOff)
- void [setParent](#) ([Chart](#) \*parent)  
*Called internally by [KDChart::Chart](#).*
- void [setParentLayout](#) (QLayout \*lay)
- virtual void [setParentWidget](#) (QWidget \*widget)  
*Inform the item about its widget: This enables the item, to trigger that widget's update, whenever the size of the item's contents has changed.*
- void [setReferenceCoordinatePlane](#) ([AbstractCoordinatePlane](#) \*plane)  
*Set another coordinate plane to be used as the reference plane for this one.*
- void [setRubberBandZoomingEnabled](#) (bool enable)  
*Enables or disables zooming with a rubber band using the mouse.*
- void [setVerticalRange](#) (const QPair< qreal, qreal > &range)  
*Set the boundaries of the visible value space displayed in vertical direction.*
- void [setVerticalRangeReversed](#) (bool reverse)  
*Sets whether the vertical range should be reversed or not, i.e.*
- virtual void [setZoomCenter](#) (const QPointF &center)  
*See also:*  
[zoomCenter](#), [setZoomFactorX](#), [setZoomFactorY](#)
- virtual void [setZoomFactorX](#) (double factor)  
*See also:*  
[zoomFactorX](#), [setZoomCenter](#)
- virtual void [setZoomFactorY](#) (double factor)  
*See also:*  
[zoomFactorY](#), [setZoomCenter](#)
- [AbstractCoordinatePlane](#) \* [sharedAxisMasterPlane](#) (QPainter \*p=0)

*reimpl*

- virtual QSize [sizeHint](#) () const  
*pure virtual in [QLayoutItem](#)*
- virtual void [sizeHintChanged](#) () const  
*Report changed size hint: ask the parent widget to recalculate the layout.*
- virtual QSizePolicy [sizePolicy](#) () const  
*[reimplemented]*
- virtual void [takeDiagram](#) ([AbstractDiagram](#) \*diagram)  
*Removes the diagram from the plane, without deleting it.*
- virtual int [topOverlap](#) (bool doNotRecalculate=false) const  
*This is called at layout time by [KDChart::AutoSpacerLayoutItem::sizeHint\(\)](#).*
- const QPointF [translate](#) (const QPointF &diagramPoint) const  
*Translate the given point in value space coordinates to a position in pixel space.*
- QPair< qreal, qreal > [verticalRange](#) () const  
**Returns:**  
*The largest and smallest visible horizontal value space value.*
- QRectF [visibleDataRange](#) () const  
*Returns the currently visible data range.*
- virtual QPointF [zoomCenter](#) () const  
**See also:**  
[setZoomCenter](#), [setZoomFactorX](#), [setZoomFactorY](#)
- virtual double [zoomFactorX](#) () const  
**See also:**  
[setZoomFactorX](#), [setZoomCenter](#)
- virtual double [zoomFactorY](#) () const  
**See also:**  
[setZoomFactorY](#), [setZoomCenter](#)
- [~CartesianCoordinatePlane](#) ()

## Static Public Member Functions

- static void [paintBackgroundAttributes](#) (QPainter &painter, const QRect &rectangle, const [KDChart::BackgroundAttributes](#) &attributes)
- static void [paintFrameAttributes](#) (QPainter &painter, const QRect &rectangle, const [KDChart::FrameAttributes](#) &attributes)

## Protected Slots

- void [slotLayoutChanged](#) ([AbstractDiagram](#) \*)

## Protected Member Functions

- [QRectF](#) [adjustedToMaxEmptyInnerPercentage](#) (const [QRectF](#) &r, unsigned int percentX, unsigned int percentY) const
- virtual [QRect](#) [areaGeometry](#) () const
- virtual [QRectF](#) [calculateRawDataBoundingRect](#) () const
- bool [doneSetZoomCenter](#) (const [QPointF](#) &center)
- bool [doneSetZoomFactorX](#) (double factor)
- bool [doneSetZoomFactorY](#) (double factor)
- virtual [QRectF](#) [drawingArea](#) () const
- virtual [DataDimensionsList](#) [getDataDimensionsList](#) () const
- [QRectF](#) [getRawDataBoundingRectFromDiagrams](#) () const
- void [handleFixedDataCoordinateSpaceRelation](#) (const [QRectF](#) &geometry)
- [QRect](#) [innerRect](#) () const
- void [layoutDiagrams](#) ()

*Distribute the available space among the diagrams and axes.*

- void [paintEvent](#) ([QPaintEvent](#) \*)
- virtual void [positionHasChanged](#) ()
- const [QPointF](#) [translateBack](#) (const [QPointF](#) &screenPoint) const

## Protected Attributes

- [QWidget](#) \* [mParent](#)
- [QLayout](#) \* [mParentLayout](#)

## 9.20.2 Member Enumeration Documentation

**9.20.2.1** enum [KDChart::AbstractCoordinatePlane::AxesCalcMode](#) [inherited]

Enumerator:

*Linear*

*Logarithmic*

Definition at line 57 of file [KDChartAbstractCoordinatePlane.h](#).

```
57 { Linear, Logarithmic };
```

## 9.20.3 Constructor & Destructor Documentation

**9.20.3.1** [CartesianCoordinatePlane::CartesianCoordinatePlane](#) ([Chart](#) \* *parent* = 0)  
[explicit]

Definition at line 68 of file [KDChartCartesianCoordinatePlane.cpp](#).

```

69      : AbstractCoordinatePlane ( new Private(), parent )
70 {
71     // this bloc left empty intentionally
72 }

```

### 9.20.3.2 CartesianCoordinatePlane::~~CartesianCoordinatePlane ()

Definition at line 74 of file KDChartCartesianCoordinatePlane.cpp.

```

75 {
76     // this bloc left empty intentionally
77 }

```

## 9.20.4 Member Function Documentation

### 9.20.4.1 void CartesianCoordinatePlane::addDiagram ([AbstractDiagram](#) \* *diagram*) [virtual]

Adds a diagram to this coordinate plane.

#### Parameters:

*diagram* The diagram to add.

#### See also:

[replaceDiagram](#), [takeDiagram](#)

Reimplemented from [KDChart::AbstractCoordinatePlane](#).

Definition at line 85 of file KDChartCartesianCoordinatePlane.cpp.

References [KDChart::AbstractCoordinatePlane::addDiagram\(\)](#), [KDChart::AbstractCoordinatePlane::diagram\(\)](#), [KDChart::AbstractCoordinatePlane::propertiesChanged\(\)](#), and [slotLayoutChanged\(\)](#).

```

86 {
87     Q_ASSERT_X ( dynamic_cast<AbstractCartesianDiagram*> ( diagram ),
88                 "CartesianCoordinatePlane::addDiagram", "Only cartesian "
89                 "diagrams can be added to a cartesian coordinate plane!" );
90     AbstractCoordinatePlane::addDiagram ( diagram );
91     connect ( diagram, SIGNAL ( layoutChanged ( AbstractDiagram* ) ),
92             SLOT ( slotLayoutChanged ( AbstractDiagram* ) ) );
93
94     connect ( diagram, SIGNAL ( propertiesChanged() ), this, SIGNAL ( propertiesChanged() ) );
95 }

```

### 9.20.4.2 QRectF CartesianCoordinatePlane::adjustedToMaxEmptyInnerPercentage (const QRectF & *r*, unsigned int *percentX*, unsigned int *percentY*) const [protected]

Definition at line 175 of file KDChartCartesianCoordinatePlane.cpp.

Referenced by [calculateRawDataBoundingRect\(\)](#).

```

177 {
178     QRectF erg ( r );

```

```

179     if( percentX < 100 || percentX == 1000 ) {
180         const bool isPositive = (r.left() >= 0);
181         if( (r.right() >= 0) == isPositive ){
182             const qreal innerBound =
183                 isPositive ? qMin(r.left(), r.right()) : qMax(r.left(), r.right());
184             const qreal outerBound =
185                 isPositive ? qMax(r.left(), r.right()) : qMin(r.left(), r.right());
186             if( innerBound / outerBound * 100 <= percentX )
187             {
188                 if( isPositive )
189                     erg.setLeft( 0.0 );
190                 else
191                     erg.setRight( 0.0 );
192             }
193         }
194     }
195     if( percentY < 100 || percentY == 1000 ) {
196         const bool isPositive = (r.bottom() >= 0);
197         if( (r.top() >= 0) == isPositive ){
198             const qreal innerBound =
199                 isPositive ? qMin(r.top(), r.bottom()) : qMax(r.top(), r.bottom());
200             const qreal outerBound =
201                 isPositive ? qMax(r.top(), r.bottom()) : qMin(r.top(), r.bottom());
202             if( innerBound / outerBound * 100 <= percentY )
203             {
204                 if( isPositive )
205                     erg.setBottom( 0.0 );
206                 else
207                     erg.setTop( 0.0 );
208             }
209         }
210     }
211     return erg;
212 }

```

#### 9.20.4.3 void CartesianCoordinatePlane::adjustHorizontalRangeToData () [slot]

Adjust horizontal range settings to the ranges covered by the model's data values.

See also:

[adjustRangesToData](#)

Definition at line 644 of file KDChartCartesianCoordinatePlane.cpp.

References `d`, `getRawDataBoundingRectFromDiagrams()`, `layoutDiagrams()`, and `KDChart::AbstractCoordinatePlane::propertiesChanged()`.

```

645 {
646     const QRectF dataBoundingRect( getRawDataBoundingRectFromDiagrams() );
647     d->horizontalMin = dataBoundingRect.left();
648     d->horizontalMax = dataBoundingRect.right();
649     layoutDiagrams();
650     emit propertiesChanged();
651 }

```

#### 9.20.4.4 void CartesianCoordinatePlane::adjustRangesToData () [slot]

Adjust both, horizontal and vertical range settings to the ranges covered by the model's data values.

See also:

[setHorizontalRange](#), [setVerticalRange](#)  
[adjustHorizontalRangeToData](#), [adjustVerticalRangeToData](#)  
[setAutoAdjustHorizontalRangeToData](#), [setAutoAdjustVerticalRangeToData](#)

Definition at line 633 of file KDChartCartesianCoordinatePlane.cpp.

References [d](#), [getRawDataBoundingRectFromDiagrams\(\)](#), [layoutDiagrams\(\)](#), and [KDChart::AbstractCoordinatePlane::propertiesChanged\(\)](#).

```
634 {
635     const QRectF dataBoundingRect( getRawDataBoundingRectFromDiagrams() );
636     d->horizontalMin = dataBoundingRect.left();
637     d->horizontalMax = dataBoundingRect.right();
638     d->verticalMin = dataBoundingRect.top();
639     d->verticalMax = dataBoundingRect.bottom();
640     layoutDiagrams();
641     emit propertiesChanged();
642 }
```

#### 9.20.4.5 void CartesianCoordinatePlane::adjustVerticalRangeToData () [slot]

Adjust vertical range settings to the ranges covered by the model's data values.

See also:

[adjustRangesToData](#)

Definition at line 653 of file KDChartCartesianCoordinatePlane.cpp.

References [d](#), [getRawDataBoundingRectFromDiagrams\(\)](#), [layoutDiagrams\(\)](#), and [KDChart::AbstractCoordinatePlane::propertiesChanged\(\)](#).

```
654 {
655     const QRectF dataBoundingRect( getRawDataBoundingRectFromDiagrams() );
656     d->verticalMin = dataBoundingRect.bottom();
657     d->verticalMax = dataBoundingRect.top();
658     layoutDiagrams();
659     emit propertiesChanged();
660 }
```

#### 9.20.4.6 void AbstractAreaBase::alignToReferencePoint (const [RelativePosition](#) & position) [inherited]

Definition at line 90 of file KDChartAbstractAreaBase.cpp.

```
91 {
92     Q_UNUSED( position );
93     // PENDING(kalle) FIXME
94     qWarning( "Sorry, not implemented: void AbstractAreaBase::alignToReferencePoint( const RelativePosi
95 }
```

#### 9.20.4.7 **QRect AbstractArea::areaGeometry () const** [protected, virtual, inherited]

Implements [KDChart::AbstractAreaBase](#).

Definition at line 150 of file KDChartAbstractArea.cpp.

Referenced by [drawingArea\(\)](#), [KDChart::TernaryCoordinatePlane::layoutDiagrams\(\)](#), [KDChart::PolarCoordinatePlane::layoutDiagrams\(\)](#), [KDChart::TernaryCoordinatePlane::paint\(\)](#), [KDChart::CartesianAxis::paint\(\)](#), [KDChart::AbstractArea::paintAll\(\)](#), and [KDChart::CartesianAxis::paintCtx\(\)](#).

```
151 {  
152     return geometry();  
153 }
```

#### 9.20.4.8 **const bool CartesianCoordinatePlane::autoAdjustGridToZoom () const**

Return the status of the built-in grid adjusting feature.

**See also:**

[setAutoAdjustGridToZoom](#)

Definition at line 751 of file KDChartCartesianCoordinatePlane.cpp.

References [d](#).

```
752 {  
753     return d->autoAdjustGridToZoom;  
754 }
```

#### 9.20.4.9 **unsigned int CartesianCoordinatePlane::autoAdjustHorizontalRangeToData () const**

Returns the maximal allowed percent of the horizontal space covered by the coordinate plane that may be empty.

**Returns:**

A percent value indicating how much of the horizontal space may be empty. If more than this is empty, automatic range adjusting is applied. A return value of 100 indicates that no such automatic adjusting is done at all.

**See also:**

[setAutoAdjustHorizontalRangeToData](#), [adjustRangesToData](#)

Definition at line 680 of file KDChartCartesianCoordinatePlane.cpp.

References [d](#).

```
681 {  
682     return d->autoAdjustHorizontalRangeToData;  
683 }
```



**9.20.4.10 unsigned int CartesianCoordinatePlane::autoAdjustVerticalRangeToData () const**

Returns the maximal allowed percent of the vertical space covered by the coordinate plane that may be empty.

**Returns:**

A percent value indicating how much of the vertical space may be empty. If more than this is empty, automatic range adjusting is applied. A return value of 100 indicates that no such automatic adjusting is done at all.

**See also:**

[setAutoAdjustVerticalRangeToData](#), [adjustRangesToData](#)

Definition at line 685 of file KDChartCartesianCoordinatePlane.cpp.

References [d](#).

```
686 {  
687     return d->autoAdjustVerticalRangeToData;  
688 }
```

**9.20.4.11 [CartesianCoordinatePlane::AxesCalcMode](#) CartesianCoordinatePlane::axesCalcModeX () const**

Definition at line 569 of file KDChartCartesianCoordinatePlane.cpp.

References [d](#).

Referenced by [getDataDimensionsList\(\)](#).

```
570 {  
571     return d->coordinateTransformation.axesCalcModeX;  
572 }
```

**9.20.4.12 [CartesianCoordinatePlane::AxesCalcMode](#) CartesianCoordinatePlane::axesCalcModeY () const**

Definition at line 564 of file KDChartCartesianCoordinatePlane.cpp.

References [d](#).

Referenced by [getDataDimensionsList\(\)](#).

```
565 {  
566     return d->coordinateTransformation.axesCalcModeY;  
567 }
```

**9.20.4.13 [BackgroundAttributes](#) AbstractAreaBase::backgroundAttributes () const**  
[inherited]

Definition at line 120 of file KDChartAbstractAreaBase.cpp.

References [d](#).

Referenced by [KDChart::AbstractAreaBase::compare\(\)](#), and [updateCommonBrush\(\)](#).

```

121 {
122     return d->backgroundAttributes;
123 }

```

#### 9.20.4.14 `int AbstractArea::bottomOverlap (bool doNotRecalculate = false) const` [virtual, inherited]

This is called at layout time by `KDChart::AutoSpacerLayoutItem::sizeHint()`.

The method triggers `AbstractArea::sizeHint()` to find out the amount of overlap at the bottom edge of the area.

#### Note:

The default implementation is not using any caching, it might make sense to implement a more sophisticated solution for derived classes that have complex work to do in `sizeHint()`. All we have here is a primitive flag to be set by the caller if it is sure that no `sizeHint()` needs to be called.

Definition at line 101 of file `KDChartAbstractArea.cpp`.

References d.

Referenced by `KDChart::CartesianAxis::maximumSize()`.

```

102 {
103     // Re-calculate the sizes,
104     // so we also get the amountOf..Overlap members set newly:
105     if( ! doNotRecalculate )
106         sizeHint();
107     return d->amountOfBottomOverlap;
108 }

```

#### 9.20.4.15 `QRectF CartesianCoordinatePlane::calculateRawDataBoundingRect () const` [protected, virtual]

Definition at line 215 of file `KDChartCartesianCoordinatePlane.cpp`.

References `adjustedToMaxEmptyInnerPercentage()`, `d`, and `getRawDataBoundingRectFromDiagrams()`.

Referenced by `getDataDimensionsList()`.

```

216 {
217     // are manually set ranges to be applied?
218     const bool bAutoAdjustHorizontalRange = (d->autoAdjustHorizontalRangeToData < 100);
219     const bool bAutoAdjustVerticalRange   = (d->autoAdjustVerticalRangeToData   < 100);
220
221     const bool bHardHorizontalRange = (d->horizontalMin != d->horizontalMax) && ! bAutoAdjustHorizontalRange;
222     const bool bHardVerticalRange   = (d->verticalMin   != d->verticalMax)   && ! bAutoAdjustVerticalRange;
223     QRectF dataBoundingRect;
224
225     // if custom boundaries are set on the plane, use them
226     if ( bHardHorizontalRange && bHardVerticalRange ) {
227         dataBoundingRect.setLeft( d->horizontalMin );
228         dataBoundingRect.setRight( d->horizontalMax );
229         dataBoundingRect.setBottom( d->verticalMin );
230         dataBoundingRect.setTop( d->verticalMax );
231     } else {
232         // determine unit of the rectangles of all involved diagrams:
233         dataBoundingRect = getRawDataBoundingRectFromDiagrams();
234     }
235 }

```

```

234         if ( bHardHorizontalRange ) {
235             dataBoundingRect.setLeft( d->horizontalMin );
236             dataBoundingRect.setRight( d->horizontalMax );
237         }
238         if ( bHardVerticalRange ) {
239             dataBoundingRect.setBottom( d->verticalMin );
240             dataBoundingRect.setTop( d->verticalMax );
241         }
242     }
243     // recalculate the bounds, if automatic adjusting of ranges is desired AND
244     // both bounds are at the same side of the zero line
245     dataBoundingRect = adjustedToMaxEmptyInnerPercentage(
246         dataBoundingRect, d->autoAdjustHorizontalRangeToData, d->autoAdjustVerticalRangeToData );
247     if( bAutoAdjustHorizontalRange ){
248         const_cast<CartesianCoordinatePlane::Private *>(d)->horizontalMin = dataBoundingRect.left();
249         const_cast<CartesianCoordinatePlane::Private *>(d)->horizontalMax = dataBoundingRect.right();
250     }
251     if( bAutoAdjustVerticalRange ){
252         const_cast<CartesianCoordinatePlane*>(this)->d->verticalMin = dataBoundingRect.bottom();
253         const_cast<CartesianCoordinatePlane*>(this)->d->verticalMax = dataBoundingRect.top();
254     }
255     //qDebug() << "CartesianCoordinatePlane::calculateRawDataBoundingRect()\nreturns data boundaries:
256     return dataBoundingRect;
257 }

```

#### 9.20.4.16 bool AbstractAreaBase::compare (const [AbstractAreaBase](#) \* other) const

[inherited]

Returns true if both areas have the same settings.

Definition at line 75 of file KDChartAbstractAreaBase.cpp.

References [KDChart::AbstractAreaBase::backgroundAttributes\(\)](#), and [KDChart::AbstractAreaBase::frameAttributes\(\)](#).

```

76 {
77     if( other == this ) return true;
78     if( ! other ){
79         //qDebug() << "CartesianAxis::compare() cannot compare to Null pointer";
80         return false;
81     }
82     /*
83     qDebug() << "AbstractAreaBase:" << (frameAttributes() == other->frameAttributes())
84     << (backgroundAttributes() == other->backgroundAttributes()) << "\n";
85     */
86     return (frameAttributes() == other->frameAttributes()) &&
87         (backgroundAttributes() == other->backgroundAttributes());
88 }

```

#### 9.20.4.17 void KDChart::AbstractCoordinatePlane::destroyedCoordinatePlane

([AbstractCoordinatePlane](#) \*) [signal, inherited]

Emitted when this coordinate plane is destroyed.

Referenced by [KDChart::AbstractCoordinatePlane::~~AbstractCoordinatePlane\(\)](#).

#### 9.20.4.18 [AbstractDiagram](#) \* AbstractCoordinatePlane::diagram ()

[inherited]

**Returns:**

The first diagram associated with this coordinate plane.

Definition at line 117 of file KDChartAbstractCoordinatePlane.cpp.

References d.

Referenced by KDChart::TernaryCoordinatePlane::addDiagram(), KDChart::PolarCoordinatePlane::addDiagram(), addDiagram(), KDChart::AbstractCoordinatePlane::addDiagram(), KDChart::Widget::diagram(), getRawDataBoundingRectFromDiagrams(), KDChart::TernaryCoordinatePlane::layoutDiagrams(), KDChart::PolarCoordinatePlane::layoutDiagrams(), KDChart::AbstractCoordinatePlane::replaceDiagram(), setGeometry(), KDChart::PolarCoordinatePlane::setStartPosition(), sharedAxisMasterPlane(), and KDChart::AbstractCoordinatePlane::takeDiagram().

```

118 {
119     if ( d->diagrams.isEmpty() )
120     {
121         return 0;
122     } else {
123         return d->diagrams.first();
124     }
125 }
```

#### 9.20.4.19 [ConstAbstractDiagramList](#) AbstractCoordinatePlane::diagrams () const [inherited]

##### Returns:

The list of diagrams associated with this coordinate plane.

Definition at line 132 of file KDChartAbstractCoordinatePlane.cpp.

References d.

```

133 {
134     ConstAbstractDiagramList list;
135 #ifndef QT_NO_STL
136     qCopy( d->diagrams.begin(), d->diagrams.end(), std::back_inserter( list ) );
137 #else
138     Q_FOREACH( AbstractDiagram * a, d->diagrams )
139         list.push_back( a );
140 #endif
141     return list;
142 }
```

#### 9.20.4.20 [AbstractDiagramList](#) AbstractCoordinatePlane::diagrams () [inherited]

##### Returns:

The list of diagrams associated with this coordinate plane.

Definition at line 127 of file KDChartAbstractCoordinatePlane.cpp.

References d.

Referenced by getDataDimensionsList(), getRawDataBoundingRectFromDiagrams(), KDChart::TernaryCoordinatePlane::layoutDiagrams(), KDChart::PolarCoordinatePlane::layoutDiagrams(), layoutDiagrams(), KDChart::Chart::mouseDoubleClickEvent(), KDChart::Chart::mouseMoveEvent(), KDChart::Chart::mousePressEvent(), KDChart::Chart::mouseReleaseEvent(), KDChart::TernaryCoordinatePlane::paint(), KDChart::PolarCoordinatePlane::paint(), paint(), and setGeometry().

```
128 {  
129     return d->diagrams;  
130 }
```

#### 9.20.4.21 bool CartesianCoordinatePlane::doesIsometricScaling () const

Definition at line 489 of file KDChartCartesianCoordinatePlane.cpp.

References [d](#).

```
490 {  
491     return d->isometricScaling;  
492 }
```

#### 9.20.4.22 bool CartesianCoordinatePlane::doneSetZoomCenter (const QPointF & *center*) [protected]

Definition at line 516 of file KDChartCartesianCoordinatePlane.cpp.

References [d](#).

Referenced by [setZoomCenter\(\)](#).

```
517 {  
518     const bool done = ( d->coordinateTransformation.zoom.center() != point );  
519     if( done ){  
520         d->coordinateTransformation.zoom.setCenter( point );  
521         if( d->autoAdjustGridToZoom )  
522             d->grid->setNeedRecalculate();  
523     }  
524     return done;  
525 }
```

#### 9.20.4.23 bool CartesianCoordinatePlane::doneSetZoomFactorX (double *factor*) [protected]

Definition at line 494 of file KDChartCartesianCoordinatePlane.cpp.

References [d](#).

Referenced by [setZoomFactorX\(\)](#).

```
495 {  
496     const bool done = ( d->coordinateTransformation.zoom.xFactor != factor );  
497     if( done ){  
498         d->coordinateTransformation.zoom.xFactor = factor;  
499         if( d->autoAdjustGridToZoom )  
500             d->grid->setNeedRecalculate();  
501     }  
502     return done;  
503 }
```

#### 9.20.4.24 bool CartesianCoordinatePlane::doneSetZoomFactorY (double *factor*) [protected]

Definition at line 505 of file KDChartCartesianCoordinatePlane.cpp.

References d.

Referenced by `setZoomFactorY()`.

```

506 {
507     const bool done = ( d->coordinateTransformation.zoom.yFactor != factor );
508     if( done ){
509         d->coordinateTransformation.zoom.yFactor = factor;
510         if( d->autoAdjustGridToZoom )
511             d->grid->setNeedRecalculate();
512     }
513     return done;
514 }
```

#### 9.20.4.25 **QRectF CartesianCoordinatePlane::drawingArea () const** [protected, virtual]

Definition at line 310 of file `KDChartCartesianCoordinatePlane.cpp`.

References `KDChart::AbstractArea::areaGeometry()`.

Referenced by `layoutDiagrams()`, `paint()`, `setGeometry()`, and `visibleDataRange()`.

```

311 {
312     const QRect rect( areaGeometry() );
313     return QRectF ( rect.left()+1, rect.top()+1, rect.width() - 3, rect.height() - 3 );
314 }
```

#### 9.20.4.26 **Qt::Orientations KDChart::AbstractCoordinatePlane::expandingDirections () const** [virtual, inherited]

pure virtual in [QLayoutItem](#)

Definition at line 212 of file `KDChartAbstractCoordinatePlane.cpp`.

```

213 {
214     return Qt::Vertical | Qt::Horizontal;
215 }
```

#### 9.20.4.27 **FrameAttributes AbstractAreaBase::frameAttributes () const** [inherited]

Definition at line 106 of file `KDChartAbstractAreaBase.cpp`.

References d.

Referenced by `KDChart::Legend::clone()`, `KDChart::AbstractAreaBase::compare()`, and `updateCommon-Brush()`.

```

107 {
108     return d->frameAttributes;
109 }
```

#### 9.20.4.28 QRect KDChart::AbstractCoordinatePlane::geometry () const [virtual, inherited]

pure virtual in [QLayoutItem](#)

Definition at line 246 of file KDChartAbstractCoordinatePlane.cpp.

References [d](#).

Referenced by [KDChart::Chart::mouseDoubleClickEvent\(\)](#), [KDChart::Chart::mouseMoveEvent\(\)](#), [KDChart::AbstractCoordinatePlane::mouseMoveEvent\(\)](#), [KDChart::Chart::mousePressEvent\(\)](#), [KDChart::Chart::mouseReleaseEvent\(\)](#), [KDChart::AbstractCoordinatePlane::mouseReleaseEvent\(\)](#), and [KDChart::PolarCoordinatePlane::paint\(\)](#).

```
247 {
248     return d->geometry;
249 }
```

#### 9.20.4.29 DataDimensionsList CartesianCoordinatePlane::getDataDimensionsList () const [protected, virtual]

Implements [KDChart::AbstractCoordinatePlane](#).

Definition at line 260 of file KDChartCartesianCoordinatePlane.cpp.

References [axesCalcModeX\(\)](#), [axesCalcModeY\(\)](#), [calculateRawDataBoundingRect\(\)](#), [KDChart::AbstractDiagram::datasetDimension\(\)](#), [KDChart::AbstractCoordinatePlane::diagrams\(\)](#), [KDChart-Enums::GranularitySequence\\_10\\_20](#), [gridAttributes\(\)](#), [KDChart::GridAttributes::gridGranularitySequence\(\)](#), [KDChart::GridAttributes::gridStepWidth\(\)](#), [KDChart::GridAttributes::gridSubStepWidth\(\)](#), and [KDChart::AbstractDiagram::percentMode\(\)](#).

```
261 {
262
263     DataDimensionsList l;
264     const AbstractCartesianDiagram* dgr
265         = diagrams().isEmpty() ? 0 : dynamic_cast<const AbstractCartesianDiagram*> (diagrams().first())
266
267     if( dgr ){
268         const QRectF r( calculateRawDataBoundingRect() );
269         // note:
270         // We do *not* access d->gridAttributesHorizontal here, but
271         // we use the getter function, to get the global attrs, if no
272         // special ones have been set for the respective orientation.
273         const GridAttributes gaH( gridAttributes( Qt::Horizontal ) );
274         const GridAttributes gaV( gridAttributes( Qt::Vertical ) );
275         // append the first dimension: for Abscissa axes
276         l.append(
277             DataDimension(
278                 r.left(), r.right(),
279                 dgr->datasetDimension() > 1,
280                 axesCalcModeX(),
281                 gaH.gridGranularitySequence(),
282                 gaH.gridStepWidth(),
283                 gaH.gridSubStepWidth() ) );
284         // append the second dimension: for Ordinate axes
285         if( dgr->percentMode() )
286             l.append(
287                 DataDimension(
288                     // always return 0-100 when in percentMode
289                     0.0, 100.0,
290                     true,
```

```

291             axesCalcModeY(),
292             KDChartEnums::GranularitySequence_10_20,
293             10.0 ) );
294         else
295             l.append(
296                 DataDimension(
297                     r.bottom(), r.top(),
298                     true,
299                     axesCalcModeY(),
300                     gaV.gridGranularitySequence(),
301                     gaV.gridStepWidth(),
302                     gaV.gridSubStepWidth() ) );
303     }else{
304         l.append( DataDimension() ); // This gets us the default 1..0 / 1..0 grid
305         l.append( DataDimension() ); // shown, if there is no diagram on this plane.
306     }
307     return l;
308 }

```

#### 9.20.4.30 void AbstractAreaBase::getFrameLeadings (int & left, int & top, int & right, int & bottom) const [inherited]

Definition at line 212 of file KDChartAbstractAreaBase.cpp.

References d.

Referenced by KDChart::AbstractAreaBase::innerRect(), and KDChart::AbstractAreaWidget::paintAll().

```

213 {
214     if( d && d->frameAttributes.isVisible() ){
215         const int padding = qMax( d->frameAttributes.padding(), 0 );
216         left   = padding;
217         top    = padding;
218         right  = padding;
219         bottom = padding;
220     }else{
221         left   = 0;
222         top    = 0;
223         right  = 0;
224         bottom = 0;
225     }
226 }

```

#### 9.20.4.31 QRectF CartesianCoordinatePlane::getRawDataBoundingRectFromDiagrams () const [protected]

Definition at line 152 of file KDChartCartesianCoordinatePlane.cpp.

References KDChart::AbstractDiagram::dataBoundaries(), KDChart::AbstractCoordinatePlane::diagram(), and KDChart::AbstractCoordinatePlane::diagrams().

Referenced by adjustHorizontalRangeToData(), adjustRangesToData(), adjustVerticalRangeToData(), and calculateRawDataBoundingRect().

```

153 {
154     // determine unit of the rectangles of all involved diagrams:
155     qreal minX, maxX, minY, maxY;
156     bool bStarting = true;
157     Q_FOREACH( const AbstractDiagram* diagram, diagrams() )
158     {

```



```

159         QPair<QPointF, QPointF> dataBoundariesPair = diagram->dataBoundaries();
160         //qDebug() << "CartesianCoordinatePlane::getRawDataBoundingRectFromDiagrams()\ngets diagram->
161         if ( bStarting || dataBoundariesPair.first.x() < minX ) minX = dataBoundariesPair.first.x();
162         if ( bStarting || dataBoundariesPair.first.y() < minY ) minY = dataBoundariesPair.first.y();
163         if ( bStarting || dataBoundariesPair.second.x() > maxX ) maxX = dataBoundariesPair.second.x();
164         if ( bStarting || dataBoundariesPair.second.y() > maxY ) maxY = dataBoundariesPair.second.y();
165         bStarting = false;
166     }
167     //qDebug() << "CartesianCoordinatePlane::getRawDataBoundingRectFromDiagrams()\nreturns data bounda
168     QRectF dataBoundingRect;
169     dataBoundingRect.setBottomLeft( QPointF( minX, minY ) );
170     dataBoundingRect.setTopRight(    QPointF( maxX, maxY ) );
171     return dataBoundingRect;
172 }

```

#### 9.20.4.32 [GridAttributes](#) KDChart::AbstractCoordinatePlane::globalGridAttributes () const [inherited]

##### Returns:

The grid attributes used by this coordinate plane.

##### See also:

[setGlobalGridAttributes](#)  
[CartesianCoordinatePlane::gridAttributes](#)

Definition at line 161 of file KDChartAbstractCoordinatePlane.cpp.

References [d](#).

Referenced by [KDChart::PolarCoordinatePlane::gridAttributes\(\)](#), and [gridAttributes\(\)](#).

```

162 {
163     return d->gridAttributes;
164 }

```

#### 9.20.4.33 [const GridAttributes](#) CartesianCoordinatePlane::gridAttributes (Qt::Orientation *orientation*) const

##### Returns:

The attributes used for grid lines drawn in horizontal direction (or in vertical direction, resp.

).

##### Note:

This function always returns a valid set of grid attributes: If no special grid attributes were set for this orientation the global attributes are returned, as returned by [AbstractCoordinatePlane::globalGridAttributes](#).

##### See also:

[setGridAttributes](#)  
[resetGridAttributes](#)  
[AbstractCoordinatePlane::globalGridAttributes](#)  
[hasOwnGridAttributes](#)

Definition at line 710 of file KDChartCartesianCoordinatePlane.cpp.

References [d](#), [KDChart::AbstractCoordinatePlane::globalGridAttributes\(\)](#), and [hasOwnGridAttributes\(\)](#).

Referenced by [getDataDimensionsList\(\)](#).

```

712 {
713     if( hasOwnGridAttributes( orientation ) ){
714         if( orientation == Qt::Horizontal )
715             return d->gridAttributesHorizontal();
716         else
717             return d->gridAttributesVertical();
718     }else{
719         return globalGridAttributes();
720     }
721 }
```

#### 9.20.4.34 [KDChart::DataDimensionsList](#) [KDChart::AbstractCoordinatePlane::gridDimensionsList\(\)](#) [inherited]

Returns the dimensions used for drawing the grid lines.

Returned data is the result of (cached) grid calculations, so - if you need that information for your own tasks - make sure to call again this function after every data modification that has changed the data range, since grid calculation is based upon the data range, thus the grid start/end might have changed if the data was changed.

##### Note:

Returned list will contain different numbers of [DataDimension](#), depending on the kind of coordinate plane used. For [CartesianCoordinatePlane](#) two [DataDimension](#) are returned: the first representing grid lines in X direction (matching the Abscissa axes) and the second indicating vertical grid lines (or Ordinate axes, resp.).

##### Returns:

The dimensions used for drawing the grid lines.

##### See also:

[DataDimension](#)

Definition at line 166 of file KDChartAbstractCoordinatePlane.cpp.

References [d](#).

Referenced by [layoutDiagrams\(\)](#), and [KDChart::CartesianAxis::maximumSize\(\)](#).

```

167 {
168     //KDChart::DataDimensionsList l( d->grid->updateData( this ) );
169     //qDebug() << "AbstractCoordinatePlane::gridDimensionsList() Y-range:" << l.last().end - l.last().first();
170     //qDebug() << "AbstractCoordinatePlane::gridDimensionsList() X-range:" << l.first().end - l.first().first();
171     return d->grid->updateData( this );
172 }
```

#### 9.20.4.35 void CartesianCoordinatePlane::handleFixedDataCoordinateSpaceRelation (const QRectF & geometry) [protected]

Definition at line 437 of file KDChartCartesianCoordinatePlane.cpp.

References `d`, `setZoomCenter()`, `setZoomFactorX()`, `setZoomFactorY()`, `zoomCenter()`, `zoomFactorX()`, and `zoomFactorY()`.

Referenced by `layoutDiagrams()`.

```

438 {
439     // is the feature enabled?
440     if( !d->fixedDataCoordinateSpaceRelation )
441         return;
442
443     // is the new geometry ok?
444     if( geometry.height() < 1 || geometry.width() < 1 )
445         return;
446
447     // if the size was changed, we calculate new zoom settings
448     if( d->fixedDataCoordinateSpaceRelationOldSize != geometry && !d->fixedDataCoordinateSpaceRelation
449     {
450         const double newZoomX = zoomFactorX() * d->fixedDataCoordinateSpaceRelationOldSize.width() / g
451         const double newZoomY = zoomFactorY() * d->fixedDataCoordinateSpaceRelationOldSize.height() /
452
453         const QPointF oldCenter = zoomCenter();
454         const QPointF newCenter = QPointF( oldCenter.x() * geometry.width() / d->fixedDataCoordinateSp
455                                           oldCenter.y() * geometry.height() / d->fixedDataCoordinateS
456
457         setZoomCenter( newCenter );
458         setZoomFactorX( newZoomX );
459         setZoomFactorY( newZoomY );
460     }
461
462     d->fixedDataCoordinateSpaceRelationOldSize = geometry;
463 }
```

#### 9.20.4.36 bool CartesianCoordinatePlane::hasFixedDataCoordinateSpaceRelation () const

Definition at line 432 of file KDChartCartesianCoordinatePlane.cpp.

References `d`.

```

433 {
434     return d->fixedDataCoordinateSpaceRelation;
435 }
```

#### 9.20.4.37 bool CartesianCoordinatePlane::hasOwnGridAttributes (Qt::Orientation orientation) const

##### Returns:

Returns whether the grid attributes have been set for the respective direction via `setGridAttributes(orientation)`.

If false, the grid will use the global attributes set by [AbstractCoordinatePlane::globalGridAttributes](#) (or the default attributes, resp.)

See also:

[setGridAttributes](#)  
[resetGridAttributes](#)  
[AbstractCoordinatePlane::globalGridAttributes](#)

Definition at line 733 of file KDChartCartesianCoordinatePlane.cpp.

References d.

Referenced by gridAttributes().

```
735 {
736     return
737         ( orientation == Qt::Horizontal )
738         ? d->hasOwnGridAttributesHorizontal
739         : d->hasOwnGridAttributesVertical;
740 }
```

#### 9.20.4.38 QPair< qreal, qreal > CartesianCoordinatePlane::horizontalRange () const

Returns:

The largest and smallest visible horizontal value space value.

If this is not explicitly set, or if both values are the same, the plane will use the union of the dataBoundaries of all associated diagrams.

See also:

[KDChart::AbstractDiagram::dataBoundaries](#)

Definition at line 623 of file KDChartCartesianCoordinatePlane.cpp.

References d.

```
624 {
625     return QPair<qreal, qreal>( d->horizontalMin, d->horizontalMax );
626 }
```

#### 9.20.4.39 QRect AbstractAreaBase::innerRect () const [protected, inherited]

Definition at line 228 of file KDChartAbstractAreaBase.cpp.

References KDChart::AbstractAreaBase::areaGeometry(), and KDChart::AbstractAreaBase::getFrameLeadings().

Referenced by KDChart::TextArea::paintAll(), and KDChart::AbstractArea::paintAll().

```
229 {
230     int left;
231     int top;
232     int right;
233     int bottom;
234     getFrameLeadings( left, top, right, bottom );
235     return
236         QRect( QPoint(0,0), areaGeometry().size() )
237         .adjusted( left, top, -right, -bottom );
238 }
```

**9.20.4.40** `bool KDChart::AbstractCoordinatePlane::isEmpty () const` [virtual, inherited]

pure virtual in [QLayoutItem](#)

Definition at line 205 of file KDChartAbstractCoordinatePlane.cpp.

Referenced by layoutDiagrams().

```
206 {
207     return false; // never empty!
208     // coordinate planes with no associated diagrams
209     // are showing a default grid of (1..10, 1..10) stepWidth 1
210 }
```

**9.20.4.41** `bool CartesianCoordinatePlane::isHorizontalRangeReversed () const`

**Returns:**

Whether the horizontal range is reversed or not

Definition at line 813 of file KDChartCartesianCoordinatePlane.cpp.

References d.

```
814 {
815     return d->reverseHorizontalPlane;
816 }
```

**9.20.4.42** `bool KDChart::AbstractCoordinatePlane::isRubberBandZoomingEnabled () const` [inherited]

**Returns:**

Whether zooming with a rubber band using the mouse is enabled.

Definition at line 280 of file KDChartAbstractCoordinatePlane.cpp.

References d.

```
281 {
282     return d->enableRubberBandZooming;
283 }
```

**9.20.4.43** `bool CartesianCoordinatePlane::isVerticalRangeReversed () const`

**Returns:**

Whether the vertical range is reversed or not

Definition at line 828 of file KDChartCartesianCoordinatePlane.cpp.

References d.

```
829 {
830     return d->reverseVerticalPlane;
831 }
```

#### 9.20.4.44 `const bool KDChart::AbstractCoordinatePlane::isVisiblePoint (const QPointF & point)` `const` [inherited]

Tests, if a point is visible on the coordinate plane.

#### Note:

Before calling this function the point must have been translated into coordinate plane space.

Definition at line 403 of file `KDChartAbstractCoordinatePlane.cpp`.

References `d`.

```
404 {
405     return d->isVisiblePoint( this, point );
406 }
```

#### 9.20.4.45 `void CartesianCoordinatePlane::layoutDiagrams ()` [protected, virtual]

Distribute the available space among the diagrams and axes.

Implements [KDChart::AbstractCoordinatePlane](#).

Definition at line 317 of file `KDChartCartesianCoordinatePlane.cpp`.

References `d`, `KDChart::AbstractCoordinatePlane::diagrams()`, `KDChart::DataDimension::distance()`, `drawingArea()`, `KDChart::DataDimension::end`, `KDChart::AbstractCoordinatePlane::gridDimensionsList()`, `handleFixedDataCoordinateSpaceRelation()`, `KDChart::AbstractCoordinatePlane::isEmpty()`, `KDChart::DataDimension::start`, `translate()`, and `KDChart::AbstractCoordinatePlane::update()`.

Referenced by `adjustHorizontalRangeToData()`, `adjustRangesToData()`, `adjustVerticalRangeToData()`, `setAutoAdjustHorizontalRangeToData()`, `setAutoAdjustVerticalRangeToData()`, `setHorizontalRange()`, `setHorizontalRangeReversed()`, `setIsometricScaling()`, `setVerticalRange()`, `setVerticalRangeReversed()`, and `slotLayoutChanged()`.

```
318 {
319     //qDebug("CartesianCoordinatePlane::layoutDiagrams() called");
320     if ( diagrams().isEmpty() )
321     { // FIXME evaluate what can still be prepared
322         // FIXME decide default dimension if no diagrams are present (to make empty planes useable)
323     }
324     // the rectangle the diagrams cover in the *plane*:
325     // (Why -3? We save 1px on each side for the antialiased drawing, and
326     // respect the way QPainter calculates the width of a painted rect (the
327     // size is the rectangle size plus the pen width). This way, most clipping
328     // for regular pens should be avoided. When pens with a penWidth or larger
329     // than 1 are used, this may not be sufficient.
330     const QRectF drawArea( drawingArea() );
331     //qDebug() << "drawingArea() returns" << drawArea;
332
333     const DataDimensionsList dimensions( gridDimensionsList() );
334     // test for programming errors: critical
335     Q_ASSERT_X ( dimensions.count() == 2, "CartesianCoordinatePlane::layoutDiagrams",
336         "Error: gridDimensionsList() did not return exactly two dimensions." );
337     const DataDimension dimX = dimensions.first();
338     const DataDimension dimY = dimensions.last();
339     const qreal distX = dimX.distance();
340     const qreal distY = dimY.distance();
341     //qDebug() << distX << distY;
342     const QPointF pt( qMin(dimX.start, dimX.end), qMax(dimY.start, dimY.end) );
```

```

343     const QSizeF siz( qAbs(distX), -qAbs(distY) );
344     const QRectF dataBoundingRect( pt, siz );
345     //qDebug() << "dataBoundingRect" << dataBoundingRect;
346
347     // calculate the remaining rectangle, and use it as the diagram area:
348     QRectF diagramArea = drawArea;
349     diagramArea.setTopLeft ( QPointF ( drawArea.left(), drawArea.top() ) );
350     diagramArea.setBottomRight ( QPointF ( drawArea.right(), drawArea.bottom() ) );
351
352     // determine coordinate transformation:
353     QPointF diagramTopLeft;
354     if( !d->reverseVerticalPlane && !d->reverseHorizontalPlane )
355         diagramTopLeft = dataBoundingRect.topLeft();
356     else if( d->reverseVerticalPlane && !d->reverseHorizontalPlane )
357         diagramTopLeft = dataBoundingRect.bottomLeft();
358     else if( d->reverseVerticalPlane && d->reverseHorizontalPlane )
359         diagramTopLeft = dataBoundingRect.bottomRight();
360     else if( !d->reverseVerticalPlane && d->reverseHorizontalPlane )
361         diagramTopLeft = dataBoundingRect.topRight();
362
363     double diagramWidth;
364     if( !d->reverseHorizontalPlane )
365         diagramWidth = dataBoundingRect.width();
366     else
367         diagramWidth = -dataBoundingRect.width();
368
369     double diagramHeight;
370     if( !d->reverseVerticalPlane )
371         diagramHeight = dataBoundingRect.height();
372     else
373         diagramHeight = -dataBoundingRect.height();
374
375     double planeWidth = diagramArea.width();
376     double planeHeight = diagramArea.height();
377     double scaleX;
378     double scaleY;
379
380     double diagramXUnitInCoordinatePlane;
381     double diagramYUnitInCoordinatePlane;
382
383     diagramXUnitInCoordinatePlane = diagramWidth != 0 ? planeWidth / diagramWidth : 1;
384     diagramYUnitInCoordinatePlane = diagramHeight != 0 ? planeHeight / diagramHeight : 1;
385     // calculate isometric scaling factor to maxscale the diagram into
386     // the coordinate system:
387     if ( d->isometricScaling )
388     {
389         double scale = qMin ( qAbs ( diagramXUnitInCoordinatePlane ),
390                               qAbs ( diagramYUnitInCoordinatePlane ) );
391
392         scaleX = qAbs( scale / diagramXUnitInCoordinatePlane );
393         scaleY = qAbs( scale / diagramYUnitInCoordinatePlane );
394     } else {
395         scaleX = 1.0;
396         scaleY = 1.0;
397     }
398
399     // calculate diagram origin in plane coordinates:
400     QPointF coordinateOrigin = QPointF (
401         diagramTopLeft.x() * -diagramXUnitInCoordinatePlane,
402         diagramTopLeft.y() * -diagramYUnitInCoordinatePlane );
403     coordinateOrigin += diagramArea.topLeft();
404
405     d->coordinateTransformation.originTranslation = coordinateOrigin;
406
407     d->coordinateTransformation.diagramRect = dataBoundingRect;
408
409     d->coordinateTransformation.unitVectorX = diagramXUnitInCoordinatePlane;

```

```

410     d->coordinateTransformation.unitVectorY = diagramYUnitInCoordinatePlane;
411
412     d->coordinateTransformation.isoScaleX = scaleX;
413     d->coordinateTransformation.isoScaleY = scaleY;
414
415     //      adapt diagram area to effect of isometric scaling:
416     diagramArea.setTopLeft( translate ( dataBoundingRect.topLeft() ) );
417     diagramArea.setBottomRight ( translate ( dataBoundingRect.bottomRight() ) );
418
419     // the plane area might have changed, so the zoom values might also be changed
420     handleFixedDataCoordinateSpaceRelation( drawArea );
421
422     //qDebug("CartesianCoordinatePlane::layoutDiagrams() done,\ncalling update() now:");
423     update();
424 }

```

#### 9.20.4.46 void KDChart::AbstractCoordinatePlane::layoutPlanes () [slot, inherited]

Calling [layoutPlanes\(\)](#) on the plane triggers the global `KDChart::Chart::slotLayoutPlanes()`.

Definition at line 263 of file `KDChartAbstractCoordinatePlane.cpp`.

References `KDChart::AbstractCoordinatePlane::needLayoutPlanes()`.

Referenced by `KDChart::AbstractCoordinatePlane::addDiagram()`, `KDChart::CartesianAxis::layoutPlanes()`, `KDChart::AbstractCartesianDiagram::layoutPlanes()`, and `KDChart::AbstractCoordinatePlane::replaceDiagram()`.

```

264 {
265     //qDebug("KDChart::AbstractCoordinatePlane::relayout() called");
266     emit needLayoutPlanes();
267 }

```

#### 9.20.4.47 int AbstractArea::leftOverlap (bool *doNotRecalculate* = false) const [virtual, inherited]

This is called at layout time by [KDChart::AutoSpacerLayoutItem::sizeHint\(\)](#).

The method triggers `AbstractArea::sizeHint()` to find out the amount of overlap at the left edge of the area.

##### Note:

The default implementation is not using any caching, it might make sense to implement a more sophisticated solution for derived classes that have complex work to do in `sizeHint()`. All we have here is a primitive flag to be set by the caller if it is sure that no `sizeHint()` needs to be called.

Definition at line 77 of file `KDChartAbstractArea.cpp`.

References `d`.

Referenced by `KDChart::CartesianAxis::maximumSize()`.

```

78 {
79     // Re-calculate the sizes,
80     // so we also get the amountOf..Overlap members set newly:
81     if( ! doNotRecalculate )
82         sizeHint();
83     return d->amountOfLeftOverlap;
84 }

```



#### 9.20.4.48 QSize KDChart::AbstractCoordinatePlane::maximumSize () const [virtual, inherited]

pure virtual in [QLayoutItem](#)

Definition at line 217 of file KDChartAbstractCoordinatePlane.cpp.

Referenced by KDChart::AbstractCoordinatePlane::sizeHint().

```
218 {
219     // No maximum size set. Especially not parent()->size(), we are not layouting
220     // to the parent widget's size when using Chart::paint()!
221     return QSize(QLAYOUTSIZE_MAX, QLAYOUTSIZE_MAX);
222 }
```

#### 9.20.4.49 QSize KDChart::AbstractCoordinatePlane::minimumSize () const [virtual, inherited]

pure virtual in [QLayoutItem](#)

Definition at line 224 of file KDChartAbstractCoordinatePlane.cpp.

```
225 {
226     return QSize(60, 60); // this default can be overwritten by derived classes
227 }
```

#### 9.20.4.50 QSize KDChart::AbstractCoordinatePlane::minimumSizeHint () const [virtual, inherited]

[reimplemented]

Reimplemented in [KDChart::TernaryCoordinatePlane](#).

Definition at line 144 of file KDChartAbstractCoordinatePlane.cpp.

```
145 {
146     return QSize( 200, 200 );
147 }
```

#### 9.20.4.51 void KDChart::AbstractCoordinatePlane::mouseDoubleClickEvent (QMouseEvent \* event) [inherited]

Definition at line 325 of file KDChartAbstractCoordinatePlane.cpp.

References [d](#), and KDChart::AbstractCoordinatePlane::mousePressEvent().

Referenced by KDChart::Chart::mouseDoubleClickEvent().

```
326 {
327     if( event->button() == Qt::RightButton )
328     {
329         // otherwise the second click gets lost
330         // which is pretty annoying when zooming out fast
331         mousePressEvent( event );
332     }
333     KDAB_FOREACH( AbstractDiagram * a, d->diagrams )
```

```

334     {
335         a->mouseDoubleClickEvent( event );
336     }
337 }

```

#### 9.20.4.52 void KDChart::AbstractCoordinatePlane::mouseMoveEvent (QMouseEvent \* event) [inherited]

Definition at line 387 of file KDChartAbstractCoordinatePlane.cpp.

References [d](#), and [KDChart::AbstractCoordinatePlane::geometry\(\)](#).

Referenced by [KDChart::Chart::mouseMoveEvent\(\)](#).

```

388 {
389     if( d->rubberBand != 0 )
390     {
391         const QRect normalized = QRect( d->rubberBandOrigin, event->pos() ).normalized();
392         d->rubberBand->setGeometry( normalized & geometry() );
393
394         event->accept();
395     }
396
397     KDAB_FOREACH( AbstractDiagram * a, d->diagrams )
398     {
399         a->mouseMoveEvent( event );
400     }
401 }

```

#### 9.20.4.53 void KDChart::AbstractCoordinatePlane::mousePressEvent (QMouseEvent \* event) [inherited]

Definition at line 285 of file KDChartAbstractCoordinatePlane.cpp.

References [d](#), [KDChart::AbstractCoordinatePlane::parent\(\)](#), [KDChart::AbstractCoordinatePlane::setZoomCenter\(\)](#), [KDChart::AbstractCoordinatePlane::setZoomFactorX\(\)](#), and [KDChart::AbstractCoordinatePlane::setZoomFactorY\(\)](#).

Referenced by [KDChart::AbstractCoordinatePlane::mouseDoubleClickEvent\(\)](#), and [KDChart::Chart::mousePressEvent\(\)](#).

```

286 {
287     if( event->button() == Qt::LeftButton )
288     {
289         if( d->enableRubberBandZooming && d->rubberBand == 0 )
290             d->rubberBand = new QRubberBand( QRubberBand::Rectangle, qobject_cast< QWidget* >( parent() ) );
291
292         if( d->rubberBand != 0 )
293         {
294             d->rubberBandOrigin = event->pos();
295             d->rubberBand->setGeometry( QRect( event->pos(), QSize() ) );
296             d->rubberBand->show();
297
298             event->accept();
299         }
300     }
301     else if( event->button() == Qt::RightButton )
302     {
303         if( d->enableRubberBandZooming && !d->rubberBandZoomConfigHistory.isEmpty() )
304         {

```

```

305         // restore the last config from the stack
306         ZoomParameters config = d->rubberBandZoomConfigHistory.pop();
307         setZoomFactorX( config.xFactor );
308         setZoomFactorY( config.yFactor );
309         setZoomCenter( config.center() );
310
311         QWidget* const p = qobject_cast< QWidget* >( parent() );
312         if( p != 0 )
313             p->update();
314
315         event->accept();
316     }
317 }
318
319 KDAB_FOREACH( AbstractDiagram * a, d->diagrams )
320 {
321     a->mousePressEvent( event );
322 }
323 }

```

#### 9.20.4.54 void KDChart::AbstractCoordinatePlane::mouseReleaseEvent (QMouseEvent \* event) [inherited]

Definition at line 339 of file KDChartAbstractCoordinatePlane.cpp.

References `d`, `KDChart::AbstractCoordinatePlane::geometry()`, `KDChart::AbstractCoordinatePlane::setZoomCenter()`, `KDChart::AbstractCoordinatePlane::setZoomFactorX()`, `KDChart::AbstractCoordinatePlane::setZoomFactorY()`, `KDChart::AbstractCoordinatePlane::zoomCenter()`, `KDChart::AbstractCoordinatePlane::zoomFactorX()`, and `KDChart::AbstractCoordinatePlane::zoomFactorY()`.

Referenced by `KDChart::Chart::mouseReleaseEvent()`.

```

340 {
341     if( d->rubberBand != 0 )
342     {
343         // save the old config on the stack
344         d->rubberBandZoomConfigHistory.push( ZoomParameters( zoomFactorX(), zoomFactorY(), zoomCenter() ) );
345
346         // this is the height/width of the rubber band in pixel space
347         const double rubberWidth = static_cast< double >( d->rubberBand->width() );
348         const double rubberHeight = static_cast< double >( d->rubberBand->height() );
349
350         // this is the center of the rubber band in pixel space
351         const double rubberCenterX = static_cast< double >( d->rubberBand->geometry().center().x() - 0.5 );
352         const double rubberCenterY = static_cast< double >( d->rubberBand->geometry().center().y() - 0.5 );
353
354         // this is the height/width of the plane in pixel space
355         const double myWidth = static_cast< double >( geometry().width() );
356         const double myHeight = static_cast< double >( geometry().height() );
357
358         // this describes the new center of zooming, relative to the plane pixel space
359         const double newCenterX = rubberCenterX / myWidth / zoomFactorX() + zoomCenter().x() - 0.5 / zoomFactorX();
360         const double newCenterY = rubberCenterY / myHeight / zoomFactorY() + zoomCenter().y() - 0.5 / zoomFactorY();
361
362         // this will be the new zoom factor
363         const double newZoomFactorX = zoomFactorX() * myWidth / rubberWidth;
364         const double newZoomFactorY = zoomFactorY() * myHeight / rubberHeight;
365
366         // and this the new center
367         const QPointF newZoomCenter( newCenterX, newCenterY );
368
369         setZoomFactorX( newZoomFactorX );
370         setZoomFactorY( newZoomFactorY );

```

```

371         setZoomCenter( newZoomCenter );
372
373
374         d->rubberBand->parentWidget()->update();
375         delete d->rubberBand;
376         d->rubberBand = 0;
377
378         event->accept();
379     }
380
381     KDAB_FOREACH( AbstractDiagram * a, d->diagrams )
382     {
383         a->mouseReleaseEvent( event );
384     }
385 }

```

#### 9.20.4.55 void KDChart::AbstractCoordinatePlane::needLayoutPlanes () [signal, inherited]

Emitted when plane needs to trigger the Chart's layouting of the coord. planes.

Referenced by KDChart::AbstractCoordinatePlane::layoutPlanes().

#### 9.20.4.56 void KDChart::AbstractCoordinatePlane::needRelayout () [signal, inherited]

Emitted when plane needs to trigger the Chart's layouting.

Referenced by KDChart::AbstractCoordinatePlane::relayout().

#### 9.20.4.57 void KDChart::AbstractCoordinatePlane::needUpdate () [signal, inherited]

Emitted when plane needs to update its drawings.

Referenced by KDChart::AbstractCoordinatePlane::update().

#### 9.20.4.58 void CartesianCoordinatePlane::paint (QPainter \*) [virtual]

reimpl

Implements [KDChart::AbstractLayoutItem](#).

Definition at line 98 of file KDChartCartesianCoordinatePlane.cpp.

References [d](#), [KDChart::AbstractCoordinatePlane::diagrams\(\)](#), and [drawingArea\(\)](#).

```

99 {
100     // prevent recursive call:
101     //qDebug("attempt plane::paint()");
102     if( d->bPaintIsRunning ){
103         return;
104     }
105     d->bPaintIsRunning = true;
106
107     //qDebug() << "start plane::paint()";
108
109     AbstractDiagramList diags = diagrams();
110     if ( !diags.isEmpty() )

```

```

111     {
112         PaintContext ctx;
113         ctx.setPainter ( painter );
114         ctx.setCoordinatePlane ( this );
115         const QRectF drawArea( drawingArea() );
116         ctx.setRectangle ( drawArea );
117
118         // enabling clipping so that we're not drawing outside
119         PainterSaver painterSaver( painter );
120         QRect clipRect = drawArea.toRect().adjusted( -1, -1, 1, 1 );
121         QRegion clipRegion( clipRect );
122         painter->setClipRegion( clipRegion );
123
124         // paint the coordinate system rulers:
125         d->grid->drawGrid( &ctx );
126
127         // paint the diagrams:
128         for ( int i = 0; i < diags.size(); i++ )
129         {
130             //qDebug(" start diags[i]->paint ( &ctx );");
131             PainterSaver diagramPainterSaver( painter );
132             diags[i]->paint ( &ctx );
133             //qDebug(" done: diags[i]->paint ( &ctx );");
134         }
135
136         //for debugging:
137         // painter->drawRect( drawArea.adjusted(4,4,-4,-4) );
138         // painter->drawRect( drawArea.adjusted(2,2,-2,-2) );
139         // painter->drawRect( drawArea );
140     }
141     d->bPaintIsRunning = false;
142     //qDebug("done: plane::paint()");
143 }

```

#### 9.20.4.59 void AbstractArea::paintAll (QPainter & painter) [virtual, inherited]

Call `paintAll`, if you want the background and the frame to be drawn before the normal `paint()` is invoked automatically.

Reimplemented from [KDChart::AbstractLayoutItem](#).

Reimplemented in [KDChart::TernaryAxis](#).

Definition at line 123 of file `KDChartAbstractArea.cpp`.

References `KDChart::AbstractArea::areaGeometry()`, `d`, `KDChart::AbstractAreaBase::innerRect()`, `KDChart::AbstractLayoutItem::paint()`, `KDChart::AbstractAreaBase::paintBackground()`, and `KDChart::AbstractAreaBase::paintFrame()`.

Referenced by `KDChart::AbstractArea::paintIntoRect()`.

```

124 {
125     // Paint the background and frame
126     const QRect overlappingArea( geometry().adjusted(
127         -d->amountOfLeftOverlap,
128         -d->amountOfTopOverlap,
129         d->amountOfRightOverlap,
130         d->amountOfBottomOverlap ) );
131     paintBackground( painter, overlappingArea );
132     paintFrame( painter, overlappingArea );
133
134     // temporarily adjust the widget size, to be sure all content gets calculated
135     // to fit into the inner rectangle
136     const QRect oldGeometry( areaGeometry() );

```

```

137     QRect inner( innerRect() );
138     inner.moveTo(
139         oldGeometry.left() + inner.left(),
140         oldGeometry.top() + inner.top() );
141     const bool needAdjustGeometry = oldGeometry != inner;
142     if( needAdjustGeometry )
143         setGeometry( inner );
144     paint( &painter );
145     if( needAdjustGeometry )
146         setGeometry( oldGeometry );
147     //qDebug() << "AbstractAreaWidget::paintAll() done.";
148 }

```

#### 9.20.4.60 void AbstractAreaBase::paintBackground (QPainter & *painter*, const QRect & *rectangle*) [virtual, inherited]

Definition at line 196 of file KDChartAbstractAreaBase.cpp.

References `d`, and `KDChart::AbstractAreaBase::paintBackgroundAttributes()`.

Referenced by `KDChart::TextArea::paintAll()`, `KDChart::AbstractAreaWidget::paintAll()`, and `KDChart::AbstractArea::paintAll()`.

```

197 {
198     Q_ASSERT_X ( d != 0, "AbstractAreaBase::paintBackground()",
199                 "Private class was not initialized!" );
200     paintBackgroundAttributes( painter, rect, d->backgroundAttributes );
201 }

```

#### 9.20.4.61 void AbstractAreaBase::paintBackgroundAttributes (QPainter & *painter*, const QRect & *rectangle*, const [KDChart::BackgroundAttributes](#) & *attributes*) [static, inherited]

Definition at line 127 of file KDChartAbstractAreaBase.cpp.

References `KDChart::BackgroundAttributes::BackgroundPixmapModeCentered`, `KDChart::BackgroundAttributes::BackgroundPixmapModeNone`, `KDChart::BackgroundAttributes::BackgroundPixmapModeScaled`, `KDChart::BackgroundAttributes::BackgroundPixmapModeStretched`, `KDChart::BackgroundAttributes::brush()`, `KDChart::BackgroundAttributes::isVisible()`, `KDChart::BackgroundAttributes::pixmap()`, and `KDChart::BackgroundAttributes::pixmapMode()`.

Referenced by `KDChart::AbstractAreaBase::paintBackground()`.

```

129 {
130     if( !attributes.isVisible() ) return;
131
132     /* first draw the brush (may contain a pixmap) */
133     if( Qt::NoBrush != attributes.brush().style() ) {
134         KDChart::PainterSaver painterSaver( &painter );
135         painter.setPen( Qt::NoPen );
136         const QPointF newTopLeft( painter.deviceMatrix().map( rect.topLeft() ) );
137         painter.setBrushOrigin( newTopLeft );
138         painter.setBrush( attributes.brush() );
139         painter.drawRect( rect.adjusted( 0, 0, -1, -1 ) );
140     }
141     /* next draw the backPixmap over the brush */
142     if( !attributes.pixmap().isNull() &&
143         attributes.pixmapMode() != BackgroundAttributes::BackgroundPixmapModeNone ) {
144         QPointF ol = rect.topLeft();

```

```

145         if( BackgroundAttributes::BackgroundPixmapModeCentered == attributes.pixmapMode() )
146         {
147             ol.setX( rect.center().x() - attributes.pixmap().width() / 2 );
148             ol.setY( rect.center().y() - attributes.pixmap().height() / 2 );
149             painter.drawPixmap( ol, attributes.pixmap() );
150         } else {
151             QMatrix m;
152             double zW = (double)rect.width() / (double)attributes.pixmap().width();
153             double zH = (double)rect.height() / (double)attributes.pixmap().height();
154             switch( attributes.pixmapMode() ) {
155                 case BackgroundAttributes::BackgroundPixmapModeScaled:
156                 {
157                     double z;
158                     z = qMin( zW, zH );
159                     m.scale( z, z );
160                 }
161                 break;
162                 case BackgroundAttributes::BackgroundPixmapModeStretched:
163                     m.scale( zW, zH );
164                     break;
165                 default:
166                     ; // Cannot happen, previously checked
167             }
168             QPixmap pm = attributes.pixmap().transformed( m );
169             ol.setX( rect.center().x() - pm.width() / 2 );
170             ol.setY( rect.center().y() - pm.height() / 2 );
171             painter.drawPixmap( ol, pm );
172         }
173     }
174 }

```

#### 9.20.4.62 void KDChart::AbstractLayoutItem::paintCtx ([PaintContext](#) \* *context*) [virtual, inherited]

Default impl: Paint the complete item using its layouted position and size.

Reimplemented in [KDChart::CartesianAxis](#), and [KDChart::TernaryAxis](#).

Definition at line 77 of file `KDChartLayoutItems.cpp`.

References `KDChart::AbstractLayoutItem::paint()`, and `KDChart::PaintContext::painter()`.

```

78 {
79     if( context )
80         paint( context->painter() );
81 }

```

#### 9.20.4.63 void KDChart::CartesianCoordinatePlane::paintEvent (QPaintEvent \*) [protected]

#### 9.20.4.64 void AbstractAreaBase::paintFrame (QPainter & *painter*, const QRect & *rectangle*) [virtual, inherited]

Definition at line 204 of file `KDChartAbstractAreaBase.cpp`.

References `d`, and `KDChart::AbstractAreaBase::paintFrameAttributes()`.

Referenced by `KDChart::TextArea::paintAll()`, `KDChart::AbstractAreaWidget::paintAll()`, and `KDChart::AbstractArea::paintAll()`.

```

205 {
206     Q_ASSERT_X ( d != 0, "AbstractAreaBase::paintFrame()",
207                 "Private class was not initialized!" );
208     paintFrameAttributes( painter, rect, d->frameAttributes );
209 }

```

#### 9.20.4.65 void AbstractAreaBase::paintFrameAttributes (QPainter & *painter*, const QRect & *rectangle*, const [KDChart::FrameAttributes](#) & *attributes*) [static, inherited]

Definition at line 177 of file KDChartAbstractAreaBase.cpp.

References [KDChart::FrameAttributes::isVisible\(\)](#), and [KDChart::FrameAttributes::pen\(\)](#).

Referenced by [KDChart::AbstractAreaBase::paintFrame\(\)](#).

```

179 {
180     if( !attributes.isVisible() ) return;
181
182     // Note: We set the brush to NoBrush explicitly here.
183     //       Otherwise we might get a filled rectangle, so any
184     //       previously drawn background would be overwritten by that area.
185
186     const QPen    oldPen(    painter.pen() );
187     const QBrush  oldBrush( painter.brush() );
188     painter.setPen(    attributes.pen() );
189     painter.setBrush(  Qt::NoBrush );
190     painter.drawRect( rect.adjusted( 0, 0, -1, -1 ) );
191     painter.setBrush( oldBrush );
192     painter.setPen(    oldPen );
193 }
194 }

```

#### 9.20.4.66 void AbstractArea::paintIntoRect (QPainter & *painter*, const QRect & *rect*) [virtual, inherited]

Draws the background and frame, then calls [paint\(\)](#).

In most cases there is no need to overwrite this method in a derived class, but you would overwrite [AbstractLayoutItem::paint\(\)](#) instead.

Definition at line 111 of file KDChartAbstractArea.cpp.

References [KDChart::AbstractArea::paintAll\(\)](#).

```

112 {
113     const QRect oldGeometry( geometry() );
114     if( oldGeometry != rect )
115         setGeometry( rect );
116     painter.translate( rect.left(), rect.top() );
117     paintAll( painter );
118     painter.translate( -rect.left(), -rect.top() );
119     if( oldGeometry != rect )
120         setGeometry( oldGeometry );
121 }

```

#### 9.20.4.67 const [KDChart::Chart](#) \* KDChart::AbstractCoordinatePlane::parent () const [inherited]

Definition at line 194 of file KDChartAbstractCoordinatePlane.cpp.



References d.

```
195 {
196     return d->parent;
197 }
```

#### 9.20.4.68 [KDChart::Chart](#) \* KDChart::AbstractCoordinatePlane::parent () [inherited]

Definition at line 199 of file KDChartAbstractCoordinatePlane.cpp.

References d.

Referenced by KDChart::AbstractCoordinatePlane::AbstractCoordinatePlane(), KDChart::CartesianAxis::maximumSize(), KDChart::AbstractCoordinatePlane::mousePressEvent(), and KDChart::AbstractCoordinatePlane::setParent().

```
200 {
201     return d->parent;
202 }
```

#### 9.20.4.69 QLayout\* KDChart::AbstractLayoutItem::parentLayout () [inherited]

Definition at line 76 of file KDChartLayoutItems.h.

```
77     {
78         return mParentLayout;
79     }
```

#### 9.20.4.70 void KDChart::AbstractArea::positionChanged ([AbstractArea](#) \*) [signal, inherited]

Referenced by KDChart::AbstractArea::positionHasChanged().

#### 9.20.4.71 void AbstractArea::positionHasChanged () [protected, virtual, inherited]

Reimplemented from [KDChart::AbstractAreaBase](#).

Definition at line 155 of file KDChartAbstractArea.cpp.

References KDChart::AbstractArea::positionChanged().

```
156 {
157     emit positionChanged( this );
158 }
```

#### 9.20.4.72 void KDChart::AbstractCoordinatePlane::propertiesChanged () [signal, inherited]

Emitted upon change of a property of the Coordinate Plane or any of its components.

Referenced by `addDiagram()`, `adjustHorizontalRangeToData()`, `adjustRangesToData()`, `adjustVerticalRangeToData()`, `setAutoAdjustGridToZoom()`, `setAutoAdjustHorizontalRangeToData()`, `setAutoAdjustVerticalRangeToData()`, `setAxesCalcModes()`, `setAxesCalcModeX()`, `setAxesCalcModeY()`, `KDChart::PolarCoordinatePlane::setGridAttributes()`, `setGridAttributes()`, `setHorizontalRange()`, `setHorizontalRangeReversed()`, `setIsometricScaling()`, `setVerticalRange()`, `setVerticalRangeReversed()`, `setZoomCenter()`, `setZoomFactorX()`, and `setZoomFactorY()`.

#### 9.20.4.73 **AbstractCoordinatePlane** \* **KDChart::AbstractCoordinatePlane::referenceCoordinatePlane () const** [inherited]

There are two ways, in which planes can be caused to interact, in where they are put layouting wise: The first is the reference plane.

If such a reference plane is set, on a plane, it will use the same cell in the layout as that one. In addition to this, planes can share an axis. In that case they will be layed out in relation to each other as suggested by the position of the axis. If, for example Plane1 and Plane2 share an axis at position Left, that will result in the layout: Axis Plane1 Plane 2, vertically. If Plane1 also happens to be Plane2's reference plane, both planes are drawn over each other. The reference plane concept allows two planes to share the same space even if neither has any axis, and in case there are shared axis, it is used to decided, whether the planes should be painted on top of each other or layed out vertically or horizontally next to each other.

#### Returns:

The reference coordinate plane associated with this one.

Definition at line 184 of file `KDChartAbstractCoordinatePlane.cpp`.

References `d`.

```
185 {
186     return d->referenceCoordinatePlane;
187 }
```

#### 9.20.4.74 **void KDChart::AbstractCoordinatePlane::relayout ()** [slot, inherited]

Calling `relayout()` on the plane triggers the global `KDChart::Chart::slotRelayout()`.

Definition at line 257 of file `KDChartAbstractCoordinatePlane.cpp`.

References `KDChart::AbstractCoordinatePlane::needRelayout()`.

```
258 {
259     //QDebug("KDChart::AbstractCoordinatePlane::relayout() called");
260     emit needRelayout();
261 }
```

#### 9.20.4.75 **void KDChart::AbstractLayoutItem::removeFromParentLayout ()** [inherited]

Definition at line 80 of file `KDChartLayoutItems.h`.

Referenced by `KDChart::Chart::takeCoordinatePlane()`.

```
81     {
82         if( mParentLayout ){
```

```

83         if( widget() )
84             mParentLayout->removeWidget( widget() );
85         else
86             mParentLayout->removeItem( this );
87     }
88 }

```

#### 9.20.4.76 void AbstractCoordinatePlane::replaceDiagram ([AbstractDiagram](#) \* *diagram*, [AbstractDiagram](#) \* *oldDiagram* = 0) [virtual, inherited]

Replaces the old diagram, or appends the diagram, if there is none yet.

##### Parameters:

***diagram*** The diagram to be used instead of the old diagram. This parameter must not be zero, or the method will do nothing.

***oldDiagram*** The diagram to be removed by the new diagram. This diagram will be deleted automatically. If the parameter is omitted, the very first diagram will be replaced. In case, there was no diagram yet, the new diagram will just be added.

##### Note:

If you want to re-use the old diagram, call `takeDiagram` and `addDiagram`, instead of using `replaceDiagram`.

##### See also:

[addDiagram](#), [takeDiagram](#)

Definition at line 86 of file `KDChartAbstractCoordinatePlane.cpp`.

References `KDChart::AbstractCoordinatePlane::addDiagram()`, `d`, `KDChart::AbstractCoordinatePlane::diagram()`, `KDChart::AbstractCoordinatePlane::layoutDiagrams()`, `KDChart::AbstractCoordinatePlane::layoutPlanes()`, `KDChart::AbstractCoordinatePlane::takeDiagram()`, and `KDChart::AbstractCoordinatePlane::update()`.

Referenced by `KDChart::Widget::setType()`.

```

87 {
88     if( diagram && oldDiagram_ != diagram ){
89         AbstractDiagram* oldDiagram = oldDiagram_;
90         if( d->diagrams.count() ){
91             if( ! oldDiagram )
92                 oldDiagram = d->diagrams.first();
93             takeDiagram( oldDiagram );
94         }
95         delete oldDiagram;
96         addDiagram( diagram );
97         layoutDiagrams();
98         layoutPlanes(); // there might be new axes, etc
99         update();
100     }
101 }

```

#### 9.20.4.77 void CartesianCoordinatePlane::resetGridAttributes ([Qt::Orientation](#) *orientation*)

Reset the attributes to be used for grid lines drawn in horizontal direction (or in vertical direction, resp.

). By calling this method you specify that the global attributes set by [AbstractCoordinatePlane::setGlobalGridAttributes](#) be used.

**See also:**

[setGridAttributes](#), [gridAttributes](#)  
[setAutoAdjustGridToZoom](#)  
[AbstractCoordinatePlane::globalGridAttributes](#)  
[hasOwnGridAttributes](#)

Definition at line 703 of file `KDChartCartesianCoordinatePlane.cpp`.

References `KDChart::AbstractCoordinatePlane::update()`.

```
705 {
706     setHasOwnGridAttributes( orientation, false );
707     update();
708 }
```

#### 9.20.4.78 `int AbstractArea::rightOverlap (bool doNotRecalculate = false) const` [virtual, inherited]

This is called at layout time by [KDChart::AutoSpacerLayoutItem::sizeHint\(\)](#).

The method triggers `AbstractArea::sizeHint()` to find out the amount of overlap at the right edge of the area.

**Note:**

The default implementation is not using any caching, it might make sense to implement a more sophisticated solution for derived classes that have complex work to do in `sizeHint()`. All we have here is a primitive flag to be set by the caller if it is sure that no `sizeHint()` needs to be called.

Definition at line 85 of file `KDChartAbstractArea.cpp`.

References `d`.

Referenced by `KDChart::CartesianAxis::maximumSize()`.

```
86 {
87     // Re-calculate the sizes,
88     // so we also get the amountOf..Overlap members set newly:
89     if( ! doNotRecalculate )
90         sizeHint();
91     return d->amountOfRightOverlap;
92 }
```

#### 9.20.4.79 `void CartesianCoordinatePlane::setAutoAdjustGridToZoom (bool autoAdjust)`

Disable / re-enable the built-in grid adjusting feature.

By default additional lines will be drawn in a Linear grid when zooming in.

**See also:**

[autoAdjustGridToZoom](#), [setGridAttributes](#)

Definition at line 742 of file KDChartCartesianCoordinatePlane.cpp.

References `d`, and `KDChart::AbstractCoordinatePlane::propertiesChanged()`.

```
743 {  
744     if ( d->autoAdjustGridToZoom != autoAdjust ){  
745         d->autoAdjustGridToZoom = autoAdjust;  
746         d->grid->setNeedRecalculate();  
747         emit propertiesChanged();  
748     }  
749 }
```

#### 9.20.4.80 void CartesianCoordinatePlane::setAutoAdjustHorizontalRangeToData (unsigned int *percentEmpty* = 67)

Automatically adjust horizontal range settings to the ranges covered by the model's values, when ever the data have changed, and then emit `horizontalRangeAutomaticallyAdjusted`.

By default the horizontal range is adjusted automatically, if more than 67 percent of the available horizontal space would be empty otherwise.

Range setting is adjusted if more than `percentEmpty` percent of the horizontal space covered by the coordinate plane would otherwise be empty. Automatic range adjusting can happen, when either all of the data are positive or all are negative.

Set `percentEmpty` to 100 to disable automatic range adjusting.

##### Parameters:

*percentEmpty* The maximal percentage of horizontal space that may be empty.

##### See also:

`horizontalRangeAutomaticallyAdjusted`  
[autoAdjustHorizontalRangeToData](#), [adjustRangesToData](#)  
[setHorizontalRange](#), [setVerticalRange](#)  
[setAutoAdjustVerticalRangeToData](#)

Definition at line 662 of file KDChartCartesianCoordinatePlane.cpp.

References `d`, `layoutDiagrams()`, and `KDChart::AbstractCoordinatePlane::propertiesChanged()`.

```
663 {  
664     d->autoAdjustHorizontalRangeToData = percentEmpty;  
665     d->horizontalMin = 0.0;  
666     d->horizontalMax = 0.0;  
667     layoutDiagrams();  
668     emit propertiesChanged();  
669 }
```

#### 9.20.4.81 void CartesianCoordinatePlane::setAutoAdjustVerticalRangeToData (unsigned int *percentEmpty* = 67)

Automatically adjust vertical range settings to the ranges covered by the model's values, when ever the data have changed, and then emit `verticalRangeAutomaticallyAdjusted`.

By default the vertical range is adjusted automatically, if more than 67 percent of the available vertical space would be empty otherwise.

Range setting is adjusted if more than `percentEmpty` percent of the horizontal space covered by the coordinate plane would otherwise be empty. Automatic range adjusting can happen, when either all of the data are positive or all are negative.

Set `percentEmpty` to 100 to disable automatic range adjusting.

#### Parameters:

***percentEmpty*** The maximal percentage of horizontal space that may be empty.

#### See also:

`verticalRangeAutomaticallyAdjusted`  
[autoAdjustVerticalRangeToData](#), [adjustRangesToData](#)  
[setHorizontalRange](#), [setVerticalRange](#)  
[setAutoAdjustHorizontalRangeToData](#)

Definition at line 671 of file `KDChartCartesianCoordinatePlane.cpp`.

References `d`, `layoutDiagrams()`, and `KDChart::AbstractCoordinatePlane::propertiesChanged()`.

```
672 {
673     d->autoAdjustVerticalRangeToData = percentEmpty;
674     d->verticalMin = 0.0;
675     d->verticalMax = 0.0;
676     layoutDiagrams();
677     emit propertiesChanged();
678 }
```

#### 9.20.4.82 void CartesianCoordinatePlane::setAxesCalcModes ([AxesCalcMode mode](#))

Specifies the calculation modes for all axes.

Definition at line 574 of file `KDChartCartesianCoordinatePlane.cpp`.

References `d`, and `KDChart::AbstractCoordinatePlane::propertiesChanged()`.

```
575 {
576     if( d->coordinateTransformation.axesCalcModeY != mode ||
577         d->coordinateTransformation.axesCalcModeX != mode ){
578         d->coordinateTransformation.axesCalcModeY = mode;
579         d->coordinateTransformation.axesCalcModeX = mode;
580         emit propertiesChanged();
581     }
582 }
```

#### 9.20.4.83 void CartesianCoordinatePlane::setAxesCalcModeX ([AxesCalcMode mode](#))

Specifies the calculation mode for all Abscissa axes.

Definition at line 592 of file `KDChartCartesianCoordinatePlane.cpp`.

References `d`, and `KDChart::AbstractCoordinatePlane::propertiesChanged()`.

```
593 {
594     if( d->coordinateTransformation.axesCalcModeX != mode ){
595         d->coordinateTransformation.axesCalcModeX = mode;
596         emit propertiesChanged();
597     }
598 }
```

**9.20.4.84 void CartesianCoordinatePlane::setAxesCalcModeY (AxesCalcMode *mode*)**

Specifies the calculation mode for all Ordinate axes.

Definition at line 584 of file KDChartCartesianCoordinatePlane.cpp.

References `d`, and `KDChart::AbstractCoordinatePlane::propertiesChanged()`.

```
585 {
586     if( d->coordinateTransformation.axesCalcModeY != mode ){
587         d->coordinateTransformation.axesCalcModeY = mode;
588         emit propertiesChanged();
589     }
590 }
```

**9.20.4.85 void AbstractAreaBase::setBackgroundAttributes (const BackgroundAttributes & *a*)**  
[inherited]

Definition at line 111 of file KDChartAbstractAreaBase.cpp.

References `d`, and `KDChart::AbstractAreaBase::positionHasChanged()`.

```
112 {
113     if( d->backgroundAttributes == a )
114         return;
115
116     d->backgroundAttributes = a;
117     positionHasChanged();
118 }
```

**9.20.4.86 void CartesianCoordinatePlane::setFixedDataCoordinateSpaceRelation (bool *fixed*)**

Definition at line 426 of file KDChartCartesianCoordinatePlane.cpp.

References `d`.

```
427 {
428     d->fixedDataCoordinateSpaceRelation = fixed;
429     d->fixedDataCoordinateSpaceRelationOldSize = QRectF();
430 }
```

**9.20.4.87 void AbstractAreaBase::setFrameAttributes (const FrameAttributes & *a*)**  
[inherited]

Definition at line 97 of file KDChartAbstractAreaBase.cpp.

References `d`, and `KDChart::AbstractAreaBase::positionHasChanged()`.

Referenced by `KDChart::Legend::clone()`.

```
98 {
99     if( d->frameAttributes == a )
100         return;
101
102     d->frameAttributes = a;
103     positionHasChanged();
104 }
```

**9.20.4.88 void CartesianCoordinatePlane::setGeometry (const QRect & r) [virtual]**

reimplement from [AbstractCoordinatePlane](#)

Reimplemented from [KDChart::AbstractCoordinatePlane](#).

Definition at line 845 of file KDChartCartesianCoordinatePlane.cpp.

References [KDChart::AbstractCoordinatePlane::diagram\(\)](#), [KDChart::AbstractCoordinatePlane::diagrams\(\)](#), [drawingArea\(\)](#), [KDChart::AbstractDiagram::resize\(\)](#), and [KDChart::AbstractCoordinatePlane::setGeometry\(\)](#).

```
846 {
847     AbstractCoordinatePlane::setGeometry( rectangle );
848     Q_FOREACH( AbstractDiagram* diagram, diagrams() ) {
849         diagram->resize( drawingArea().size() );
850     }
851 }
```

**9.20.4.89 void KDChart::AbstractCoordinatePlane::setGlobalGridAttributes (const GridAttributes &) [inherited]**

Set the grid attributes to be used by this coordinate plane.

To disable grid painting, for example, your code should like this:

```
GridAttributes ga = plane->globalGridAttributes();
ga.setGlobalGridVisible( false );
plane->setGlobalGridAttributes( ga );
```

See also:

[globalGridAttributes](#)  
[CartesianCoordinatePlane::setGridAttributes](#)

Definition at line 155 of file KDChartAbstractCoordinatePlane.cpp.

References [d](#), and [KDChart::AbstractCoordinatePlane::update\(\)](#).

```
156 {
157     d->gridAttributes = a;
158     update();
159 }
```

**9.20.4.90 void CartesianCoordinatePlane::setGridAttributes (Qt::Orientation orientation, const GridAttributes &)**

Set the attributes to be used for grid lines drawn in horizontal direction (or in vertical direction, resp.).

To disable horizontal grid painting, for example, your code should like this:

```
GridAttributes ga = plane->gridAttributes( Qt::Horizontal );
ga.setGridVisible( false );
plane->setGridAttributes( Qt::Horizontal, ga );
```



**Note:**

setGridAttributes overwrites the global attributes that were set by [AbstractCoordinatePlane::setGlobalGridAttributes](#). To re-activate these global attributes you can call resetGridAttributes.

**See also:**

[resetGridAttributes](#), [gridAttributes](#)  
[setAutoAdjustGridToZoom](#)  
[AbstractCoordinatePlane::setGlobalGridAttributes](#)  
[hasOwnGridAttributes](#)

Definition at line 690 of file KDChartCartesianCoordinatePlane.cpp.

References [d](#), [KDChart::AbstractCoordinatePlane::propertiesChanged\(\)](#), and [KDChart::AbstractCoordinatePlane::update\(\)](#).

```
693 {
694     if( orientation == Qt::Horizontal )
695         d->gridAttributesHorizontal = a;
696     else
697         d->gridAttributesVertical = a;
698     setHasOwnGridAttributes( orientation, true );
699     update();
700     emit propertiesChanged();
701 }
```

#### 9.20.4.91 void KDChart::AbstractCoordinatePlane::setGridNeedsRecalculate () [slot, inherited]

Used by the chart to clear the cached grid data.

Definition at line 174 of file KDChartAbstractCoordinatePlane.cpp.

References [d](#).

Referenced by [KDChart::Chart::resizeEvent\(\)](#).

```
175 {
176     d->grid->setNeedRecalculate();
177 }
```

#### 9.20.4.92 void CartesianCoordinatePlane::setHorizontalRange (const QPair< qreal, qreal > &range)

Set the boundaries of the visible value space displayed in horizontal direction.

This is also known as the horizontal viewport.

By default the horizontal range is adjusted to the range covered by the model's data, see [setAutoAdjustHorizontalRangeToData](#) for details. Calling setHorizontalRange with a valid range disables this default automatic adjusting, while on the other hand automatic adjusting will set these ranges.

To disable use of this range you can either pass an empty pair by using the default constructor [QPair\(\)](#) or you can set both values to the same which constitutes a null range.

**Note:**

By default the visible data range often is larger than the range calculated from the data model (or set by `setHoriz.|Vert.Range()`, resp.). This is due to the built-in grid calculation feature: The visible start/end values get adjusted so that they match a main-grid line. You can turn this feature off for any of the four bounds by calling `GridAttributes::setAdjustBoundsToGrid()` for either the global grid-attributes or for the horizontal/vertical attrs separately.

**Parameters:**

**range** a pair of values representing the smallest and the largest horizontal value space coordinate displayed.

**See also:**

[setAutoAdjustHorizontalRangeToData](#), [setVerticalRange](#)  
[GridAttributes::setAdjustBoundsToGrid\(\)](#)

Definition at line 600 of file `KDChartCartesianCoordinatePlane.cpp`.

References `d`, `layoutDiagrams()`, and `KDChart::AbstractCoordinatePlane::propertiesChanged()`.

```

601 {
602     if ( d->horizontalMin != range.first || d->horizontalMax != range.second ) {
603         d->autoAdjustHorizontalRangeToData = 100;
604         d->horizontalMin = range.first;
605         d->horizontalMax = range.second;
606         layoutDiagrams();
607         emit propertiesChanged();
608     }
609 }
```

**9.20.4.93 void CartesianCoordinatePlane::setHorizontalRangeReversed (bool reverse)**

Sets whether the horizontal range should be reversed or not, i.e.

small values to the left and large values to the right (the default) or vice versa.

**Parameters:**

**reverse** Whether the horizontal range should be reversed or not

Definition at line 803 of file `KDChartCartesianCoordinatePlane.cpp`.

References `d`, `layoutDiagrams()`, and `KDChart::AbstractCoordinatePlane::propertiesChanged()`.

```

804 {
805     if( d->reverseHorizontalPlane == reverse )
806         return;
807
808     d->reverseHorizontalPlane = reverse;
809     layoutDiagrams();
810     emit propertiesChanged();
811 }
```

**9.20.4.94 void CartesianCoordinatePlane::setIsometricScaling (bool *onOff*)**

Definition at line 479 of file KDChartCartesianCoordinatePlane.cpp.

References `d`, `layoutDiagrams()`, and `KDChart::AbstractCoordinatePlane::propertiesChanged()`.

```

480 {
481     if ( d->isometricScaling != onOff )
482     {
483         d->isometricScaling = onOff;
484         layoutDiagrams();
485         emit propertiesChanged();
486     }
487 }
```

**9.20.4.95 void KDChart::AbstractCoordinatePlane::setParent (Chart \* *parent*)** [inherited]

Called internally by [KDChart::Chart](#).

Definition at line 189 of file KDChartAbstractCoordinatePlane.cpp.

References `d`, and `KDChart::AbstractCoordinatePlane::parent()`.

Referenced by `KDChart::Chart::addCoordinatePlane()`, and `KDChart::Chart::takeCoordinatePlane()`.

```

190 {
191     d->parent = parent;
192 }
```

**9.20.4.96 void KDChart::AbstractLayoutItem::setParentLayout (QLayout \* *lay*)** [inherited]

Definition at line 72 of file KDChartLayoutItems.h.

```

73     {
74         mParentLayout = lay;
75     }
```

**9.20.4.97 void KDChart::AbstractLayoutItem::setParentWidget (QWidget \* *widget*)**  
[virtual, inherited]

Inform the item about its widget: This enables the item, to trigger that widget's update, whenever the size of the item's contents has changed.

Thus, you need to call `setParentWidget` on every item, that has a non-fixed size.

Definition at line 64 of file KDChartLayoutItems.cpp.

References `KDChart::AbstractLayoutItem::mParent`.

Referenced by `KDChart::HeaderFooter::setParent()`, and `KDChart::AbstractCartesianDiagram::takeAxis()`.

```

65 {
66     mParent = widget;
67 }
```

#### 9.20.4.98 void KDChart::AbstractCoordinatePlane::setReferenceCoordinatePlane ([AbstractCoordinatePlane](#) \* *plane*) [inherited]

Set another coordinate plane to be used as the reference plane for this one.

##### Parameters:

*plane* The coordinate plane to be used the reference plane for this one.

##### See also:

[referenceCoordinatePlane](#)

Definition at line 179 of file KDChartAbstractCoordinatePlane.cpp.

References d.

```
180 {
181     d->referenceCoordinatePlane = plane;
182 }
```

#### 9.20.4.99 void KDChart::AbstractCoordinatePlane::setRubberBandZoomingEnabled (bool *enable*) [inherited]

Enables or disables zooming with a rubber band using the mouse.

Definition at line 269 of file KDChartAbstractCoordinatePlane.cpp.

References d.

```
270 {
271     d->enableRubberBandZooming = enable;
272
273     if( !enable && d->rubberBand != 0 )
274     {
275         delete d->rubberBand;
276         d->rubberBand = 0;
277     }
278 }
```

#### 9.20.4.100 void CartesianCoordinatePlane::setVerticalRange (const QPair< qreal, qreal > & *range*)

Set the boundaries of the visible value space displayed in vertical direction.

This is also known as the vertical viewport.

By default the vertical range is adjusted to the range covered by the model's data, see setAutoAdjustVerticalRangeToData for details. Calling setVerticalRange with a valid range disables this default automatic adjusting, while on the other hand automatic adjusting will set these ranges.

To disable use of this range you can either pass an empty pair by using the default constructor QPair() or you can set setting both values to the same which constitutes a null range.

##### Note:

By default the visible data range often is larger than the range calculated from the data model (or set by setHoriz.|Vert.Range(), resp.). This is due to the built-in grid calculation feature: The visible start/end

values get adjusted so that they match a main-grid line. You can turn this feature off for any of the four bounds by calling [GridAttributes::setAdjustBoundsToGrid\(\)](#) for either the global grid-attributes or for the horizontal/vertical attrs separately.

#### Parameters:

*range* a pair of values representing the smallest and the largest vertical value space coordinate displayed.

#### See also:

[setAutoAdjustVerticalRangeToData](#), [setHorizontalRange](#)  
[GridAttributes::setAdjustBoundsToGrid\(\)](#)

Definition at line 611 of file KDChartCartesianCoordinatePlane.cpp.

References [d](#), [layoutDiagrams\(\)](#), and [KDChart::AbstractCoordinatePlane::propertiesChanged\(\)](#).

```

612 {
613
614     if ( d->verticalMin != range.first || d->verticalMax != range.second ) {
615         d->autoAdjustVerticalRangeToData = 100;
616         d->verticalMin = range.first;
617         d->verticalMax = range.second;
618         layoutDiagrams();
619         emit propertiesChanged();
620     }
621 }
```

#### 9.20.4.101 void CartesianCoordinatePlane::setVerticalRangeReversed (bool *reverse*)

Sets whether the vertical range should be reversed or not, i.e.

small values at the bottom and large values at the top (the default) or vice versa.

#### Parameters:

*reverse* Whether the vertical range should be reversed or not

Definition at line 818 of file KDChartCartesianCoordinatePlane.cpp.

References [d](#), [layoutDiagrams\(\)](#), and [KDChart::AbstractCoordinatePlane::propertiesChanged\(\)](#).

```

819 {
820     if ( d->reverseVerticalPlane == reverse )
821         return;
822
823     d->reverseVerticalPlane = reverse;
824     layoutDiagrams();
825     emit propertiesChanged();
826 }
```

#### 9.20.4.102 void CartesianCoordinatePlane::setZoomCenter (const QPointF & *center*) [virtual]

#### See also:

[zoomCenter](#), [setZoomFactorX](#), [setZoomFactorY](#)

Reimplemented from [KDChart::AbstractCoordinatePlane](#).

Definition at line 541 of file KDChartCartesianCoordinatePlane.cpp.

References `doneSetZoomCenter()`, and `KDChart::AbstractCoordinatePlane::propertiesChanged()`.

Referenced by `handleFixedDataCoordinateSpaceRelation()`.

```
542 {
543     if( doneSetZoomCenter( point ) ){
544         emit propertiesChanged();
545     }
546 }
```

#### 9.20.4.103 void CartesianCoordinatePlane::setZoomFactorX (double *factor*) [virtual]

See also:

[zoomFactorX](#), [setZoomCenter](#)

Reimplemented from [KDChart::AbstractCoordinatePlane](#).

Definition at line 527 of file KDChartCartesianCoordinatePlane.cpp.

References `doneSetZoomFactorX()`, and `KDChart::AbstractCoordinatePlane::propertiesChanged()`.

Referenced by `handleFixedDataCoordinateSpaceRelation()`.

```
528 {
529     if( doneSetZoomFactorX( factor ) ){
530         emit propertiesChanged();
531     }
532 }
```

#### 9.20.4.104 void CartesianCoordinatePlane::setZoomFactorY (double *factor*) [virtual]

See also:

[zoomFactorY](#), [setZoomCenter](#)

Reimplemented from [KDChart::AbstractCoordinatePlane](#).

Definition at line 534 of file KDChartCartesianCoordinatePlane.cpp.

References `doneSetZoomFactorY()`, and `KDChart::AbstractCoordinatePlane::propertiesChanged()`.

Referenced by `handleFixedDataCoordinateSpaceRelation()`.

```
535 {
536     if( doneSetZoomFactorY( factor ) ){
537         emit propertiesChanged();
538     }
539 }
```

#### 9.20.4.105 [AbstractCoordinatePlane](#) \* CartesianCoordinatePlane::sharedAxisMasterPlane (QPainter \* *p* = 0) [virtual]

reimpl

Reimplemented from [KDChart::AbstractCoordinatePlane](#).

Definition at line 756 of file KDChartCartesianCoordinatePlane.cpp.

References [KDChart::AbstractCartesianDiagram::axes\(\)](#), [KDChart::AbstractAxis::coordinatePlane\(\)](#), [KDChart::AbstractCoordinatePlane::diagram\(\)](#), and [translate\(\)](#).

```

757 {
758     CartesianCoordinatePlane* plane = this;
759     AbstractCartesianDiagram* diag = dynamic_cast< AbstractCartesianDiagram* >( plane->diagram() );
760     const CartesianAxis* sharedAxis = 0;
761     if( diag != 0 )
762     {
763         const CartesianAxisList axes = diag->axes();
764         KDAB_FOREACH( const CartesianAxis* a, axes )
765         {
766             CartesianCoordinatePlane* p = const_cast< CartesianCoordinatePlane* >(
767                                     dynamic_cast< const CartesianCoordinatePlane* >( a->coordinatePlane() ));
768             if( p != 0 && p != this )
769             {
770                 plane = p;
771                 sharedAxis = a;
772             }
773         }
774     }
775     if( plane == this || painter == 0 )
776         return plane;
777
778     const QPointF zero = QPointF( 0, 0 );
779     const QPointF tenX = QPointF( 10, 0 );
780     const QPointF tenY = QPointF( 0, 10 );
781
782     if( sharedAxis->isOrdinate() )
783     {
784         painter->translate( translate( zero ).x(), 0.0 );
785         const qreal factor = (translate( tenX ) - translate( zero ) ).x() / ( plane->translate( tenX ).x() );
786         painter->scale( factor, 1.0 );
787         painter->translate( -plane->translate( zero ).x(), 0.0 );
788     }
789     if( sharedAxis->isAbscissa() )
790     {
791         painter->translate( 0.0, translate( zero ).y() );
792         const qreal factor = (translate( tenY ) - translate( zero ) ).y() / ( plane->translate( tenY ).y() );
793         painter->scale( 1.0, factor );
794         painter->translate( 0.0, -plane->translate( zero ).y() );
795     }
796     return plane;
797 }
798
799
800
801 }
```

#### 9.20.4.106 QSize KDChart::AbstractCoordinatePlane::sizeHint () const [virtual, inherited]

pure virtual in [QLayoutItem](#)

Definition at line 229 of file KDChartAbstractCoordinatePlane.cpp.

References [KDChart::AbstractCoordinatePlane::maximumSize\(\)](#).

```

230 {
231     // we return our maximum (which is the full size of the Chart)
```

```

232     // even if we know the plane will be smaller
233     return maximumSize();
234 }

```

#### 9.20.4.107 void KDChart::AbstractLayoutItem::sizeHintChanged () const [virtual, inherited]

Report changed size hint: ask the parent widget to recalculate the layout.

Definition at line 86 of file KDChartLayoutItems.cpp.

References KDChart::AbstractLayoutItem::mParent.

Referenced by KDChart::TextLayoutItem::sizeHint().

```

87 {
88     // This is exactly like what QWidget::updateGeometry does.
89     qDebug("KDChart::AbstractLayoutItem::sizeHintChanged() called");
90     if( mParent ) {
91         if ( mParent->layout() )
92             mParent->layout()->invalidate();
93         else
94             QApplication::postEvent( mParent, new QEvent( QEvent::LayoutRequest ) );
95     }
96 }

```

#### 9.20.4.108 QSizePolicy KDChart::AbstractCoordinatePlane::sizePolicy () const [virtual, inherited]

[reimplemented]

Reimplemented in [KDChart::TernaryCoordinatePlane](#).

Definition at line 150 of file KDChartAbstractCoordinatePlane.cpp.

```

151 {
152     return QSizePolicy( QSizePolicy::MinimumExpanding, QSizePolicy::MinimumExpanding );
153 }

```

#### 9.20.4.109 void CartesianCoordinatePlane::slotLayoutChanged ([AbstractDiagram](#) \*) [protected, slot]

Definition at line 146 of file KDChartCartesianCoordinatePlane.cpp.

References layoutDiagrams().

Referenced by addDiagram().

```

147 {
148     // old: if ( d->initialResizeEventReceived )
149     layoutDiagrams();
150 }

```



#### 9.20.4.110 void AbstractCoordinatePlane::takeDiagram ([AbstractDiagram](#) \* *diagram*) [virtual, inherited]

Removes the diagram from the plane, without deleting it.

The plane no longer owns the diagram, so it is the caller's responsibility to delete the diagram.

See also:

[addDiagram](#), [replaceDiagram](#)

Definition at line 104 of file KDChartAbstractCoordinatePlane.cpp.

References [d](#), [KDChart::AbstractCoordinatePlane::diagram\(\)](#), [KDChart::AbstractCoordinatePlane::layoutDiagrams\(\)](#), [KDChart::AbstractDiagram::setCoordinatePlane\(\)](#), and [KDChart::AbstractCoordinatePlane::update\(\)](#).

Referenced by [KDChart::AbstractCoordinatePlane::replaceDiagram\(\)](#).

```

105 {
106     const int idx = d->diagrams.indexOf( diagram );
107     if( idx != -1 ){
108         d->diagrams.removeAt( idx );
109         diagram->setParent( 0 );
110         diagram->setCoordinatePlane( 0 );
111         layoutDiagrams();
112         update();
113     }
114 }
```

#### 9.20.4.111 int AbstractArea::topOverlap (bool *doNotRecalculate* = false) const [virtual, inherited]

This is called at layout time by [KDChart::AutoSpacerLayoutItem::sizeHint\(\)](#).

The method triggers [AbstractArea::sizeHint\(\)](#) to find out the amount of overlap at the top edge of the area.

**Note:**

The default implementation is not using any caching, it might make sense to implement a more sophisticated solution for derived classes that have complex work to do in [sizeHint\(\)](#). All we have here is a primitive flag to be set by the caller if it is sure that no [sizeHint\(\)](#) needs to be called.

Definition at line 93 of file KDChartAbstractArea.cpp.

References [d](#).

Referenced by [KDChart::CartesianAxis::maximumSize\(\)](#).

```

94 {
95     // Re-calculate the sizes,
96     // so we also get the amountOf..Overlap members set newly:
97     if( ! doNotRecalculate )
98         sizeHint();
99     return d->amountOfTopOverlap;
100 }
```

#### 9.20.4.112 **const QPointF CartesianCoordinatePlane::translate (const QPointF & *diagramPoint*) const** [virtual]

Translate the given point in value space coordinates to a position in pixel space.

##### Parameters:

***diagramPoint*** The point in value coordinates.

##### Returns:

The translated point.

Implements [KDChart::AbstractCoordinatePlane](#).

Definition at line 465 of file KDChartCartesianCoordinatePlane.cpp.

References d.

Referenced by [layoutDiagrams\(\)](#), and [sharedAxisMasterPlane\(\)](#).

```

466 {
467     // Note: We do not test if the point lays inside of the data area,
468     //       but we just apply the transformation calculations to the point.
469     //       This allows for basic calculations done by the user, see e.g.
470     //       the file examples/Lines/BubbleChart/mainwindow.cpp
471     return d->coordinateTransformation.translate ( diagramPoint );
472 }
```

#### 9.20.4.113 **const QPointF CartesianCoordinatePlane::translateBack (const QPointF & *screenPoint*) const** [protected]

Definition at line 474 of file KDChartCartesianCoordinatePlane.cpp.

References d.

Referenced by [visibleDataRange\(\)](#).

```

475 {
476     return d->coordinateTransformation.translateBack ( screenPoint );
477 }
```

#### 9.20.4.114 **void KDChart::AbstractCoordinatePlane::update ()** [slot, inherited]

Calling [update\(\)](#) on the plane triggers the global [KDChart::Chart::update\(\)](#).

Definition at line 251 of file KDChartAbstractCoordinatePlane.cpp.

References [KDChart::AbstractCoordinatePlane::needUpdate\(\)](#).

Referenced by [KDChart::AbstractCoordinatePlane::addDiagram\(\)](#), [layoutDiagrams\(\)](#), [KDChart::AbstractCoordinatePlane::replaceDiagram\(\)](#), [KDChart::PolarCoordinatePlane::resetGridAttributes\(\)](#), [resetGridAttributes\(\)](#), [KDChart::AbstractCoordinatePlane::setGlobalGridAttributes\(\)](#), [KDChart::PolarCoordinatePlane::setGridAttributes\(\)](#), [setGridAttributes\(\)](#), and [KDChart::AbstractCoordinatePlane::takeDiagram\(\)](#).

```

252 {
253     //qDebug("KDChart::AbstractCoordinatePlane::update() called");
254     emit needUpdate();
255 }
```

**9.20.4.115 QPair< qreal, qreal > CartesianCoordinatePlane::verticalRange () const****Returns:**

The largest and smallest visible horizontal value space value.

If this is not explicitly set, or if both values are the same, the plane will use the union of the dataBoundaries of all associated diagrams.

**See also:**

[KDChart::AbstractDiagram::dataBoundaries](#)

Definition at line 628 of file KDChartCartesianCoordinatePlane.cpp.

References [d](#).

```
629 {
630     return QPair<qreal, qreal>( d->verticalMin, d->verticalMax );
631 }
```

**9.20.4.116 QRectF CartesianCoordinatePlane::visibleDataRange () const**

Returns the currently visible data range.

Might be greater than the range of the grid.

Definition at line 833 of file KDChartCartesianCoordinatePlane.cpp.

References [drawingArea\(\)](#), and [translateBack\(\)](#).

```
834 {
835     QRectF result;
836
837     const QRectF drawArea = drawingArea();
838
839     result.setTopLeft( translateBack( drawArea.topLeft() ) );
840     result.setBottomRight( translateBack( drawArea.bottomRight() ) );
841
842     return result.normalized();
843 }
```

**9.20.4.117 QPointF CartesianCoordinatePlane::zoomCenter () const** [virtual]**See also:**

[setZoomCenter](#), [setZoomFactorX](#), [setZoomFactorY](#)

Reimplemented from [KDChart::AbstractCoordinatePlane](#).

Definition at line 548 of file KDChartCartesianCoordinatePlane.cpp.

References [d](#).

Referenced by [handleFixedDataCoordinateSpaceRelation\(\)](#).

```
549 {
550     return d->coordinateTransformation.zoom.center();
551 }
```

**9.20.4.118** `double CartesianCoordinatePlane::zoomFactorX () const` [virtual]

See also:

[setZoomFactorX](#), [setZoomCenter](#)

Reimplemented from [KDChart::AbstractCoordinatePlane](#).

Definition at line 553 of file `KDChartCartesianCoordinatePlane.cpp`.

References d.

Referenced by `handleFixedDataCoordinateSpaceRelation()`.

```
554 {  
555     return d->coordinateTransformation.zoom.xFactor;  
556 }
```

**9.20.4.119** `double CartesianCoordinatePlane::zoomFactorY () const` [virtual]

See also:

[setZoomFactorY](#), [setZoomCenter](#)

Reimplemented from [KDChart::AbstractCoordinatePlane](#).

Definition at line 558 of file `KDChartCartesianCoordinatePlane.cpp`.

References d.

Referenced by `handleFixedDataCoordinateSpaceRelation()`.

```
559 {  
560     return d->coordinateTransformation.zoom.yFactor;  
561 }
```

**9.20.5 Member Data Documentation****9.20.5.1** `QWidget* KDChart::AbstractLayoutItem::mParent` [protected, inherited]

Definition at line 90 of file `KDChartLayoutItems.h`.

Referenced by `KDChart::AbstractLayoutItem::setParentWidget()`, `KDChart::TextLayoutItem::setText()`, `KDChart::TextLayoutItem::setTextAttributes()`, and `KDChart::AbstractLayoutItem::sizeHintChanged()`.

**9.20.5.2** `QLayout* KDChart::AbstractLayoutItem::mParentLayout` [protected, inherited]

Definition at line 91 of file `KDChartLayoutItems.h`.

Referenced by `KDChart::AutoSpacerLayoutItem::paint()`.

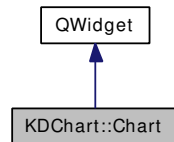
The documentation for this class was generated from the following files:

- [KDChartCartesianCoordinatePlane.h](#)
- [KDChartCartesianCoordinatePlane.cpp](#)

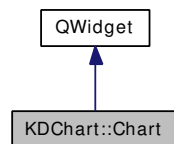
## 9.21 KDChart::Chart Class Reference

```
#include <KDChartChart>
```

Inheritance diagram for KDChart::Chart:



Collaboration diagram for KDChart::Chart:



### 9.21.1 Detailed Description

A chart with one or more diagrams.

The [Chart](#) class represents a drawing consisting of one or more diagrams and various optional elements such as legends, axes, text boxes, headers or footers. It takes ownership of all these elements when they are assigned to it. Each diagram is associated with a coordinate plane, of which the chart can have more than one. The coordinate planes (and thus the associated diagrams) can be layed out in various ways.

The [Chart](#) class makes heavy use of the Qt Interview framework for model/view programming, and thus requires data to be presented to it in a QAbstractItemModel compatible way. For many simple charts, especially if the visualized data is static, [KDChart::Widget](#) provides an abstracted interface, that hides the complexity of Interview to a large extent.

Definition at line 72 of file KDChartChart.h.

### Signals

- void [propertiesChanged](#) ()  
*Emitted upon change of a property of the [Chart](#) or any of its components.*

### Public Member Functions

- void [addCoordinatePlane](#) ([AbstractCoordinatePlane](#) \*plane)  
*Adds a coordinate plane to the chart.*
- void [addHeaderFooter](#) ([HeaderFooter](#) \*headerFooter)  
*Adds a header or a footer to the chart.*

- void [addLegend](#) ([Legend](#) \*legend)  
*Add the given legend to the chart.*
- [BackgroundAttributes](#) [backgroundAttributes](#) () const
- [Chart](#) ([QWidget](#) \*parent=0)
- [AbstractCoordinatePlane](#) \* [coordinatePlane](#) ()  
*Each chart must have at least one coordinate plane.*
- [QLayout](#) \* [coordinatePlaneLayout](#) ()
- [CoordinatePlaneList](#) [coordinatePlanes](#) ()  
*The list of coordinate planes.*
- [FrameAttributes](#) [frameAttributes](#) () const
- int [globalLeadingBottom](#) () const  
*The padding between the start of the widget and the start of the area that is used for drawing at the bottom.*
- int [globalLeadingLeft](#) () const  
*The padding between the start of the widget and the start of the area that is used for drawing on the left.*
- int [globalLeadingRight](#) () const  
*The padding between the start of the widget and the start of the area that is used for drawing on the right.*
- int [globalLeadingTop](#) () const  
*The padding between the start of the widget and the start of the area that is used for drawing at the top.*
- [HeaderFooter](#) \* [headerFooter](#) ()  
*The first header or footer of the chart.*
- [HeaderFooterList](#) [headerFooters](#) ()  
*The list of headers and footers associated with the chart.*
- [Legend](#) \* [legend](#) ()  
*The first legend of the chart or 0 if there was none added to the chart.*
- [LegendList](#) [legends](#) ()  
*The list of all legends associated with the chart.*
- void [paint](#) ([QPainter](#) \*painter, const [QRect](#) &target)  
*Paints all the contents of the chart.*
- void [reLayoutFloatingLegends](#) ()
- void [replaceCoordinatePlane](#) ([AbstractCoordinatePlane](#) \*plane, [AbstractCoordinatePlane](#) \*oldPlane=0)  
*Replaces the old coordinate plane, or appends the plane, if there is none yet.*
- void [replaceHeaderFooter](#) ([HeaderFooter](#) \*headerFooter, [HeaderFooter](#) \*oldHeaderFooter=0)  
*Replaces the old header (or footer, resp.*
- void [replaceLegend](#) ([Legend](#) \*legend, [Legend](#) \*oldLegend=0)  
*Replaces the old legend, or appends the new legend, if there is none yet.*

- void [setBackgroundAttributes](#) (const [BackgroundAttributes](#) &a)  
*Specify the background attributes to be used, by default there is no background.*
- void [setCoordinatePlaneLayout](#) (QLayout \*layout)
- void [setFrameAttributes](#) (const [FrameAttributes](#) &a)  
*Specify the frame attributes to be used, by default is it a thin black line.*
- void [setGlobalLeading](#) (int left, int top, int right, int bottom)  
*Set the padding between the margin of the widget and the area that the contents are drawn into.*
- void [setGlobalLeadingBottom](#) (int leading)  
*Set the padding between the start of the widget and the start of the area that is used for drawing on the bottom.*
- void [setGlobalLeadingLeft](#) (int leading)  
*Set the padding between the start of the widget and the start of the area that is used for drawing on the left.*
- void [setGlobalLeadingRight](#) (int leading)  
*Set the padding between the start of the widget and the start of the area that is used for drawing on the right.*
- void [setGlobalLeadingTop](#) (int leading)  
*Set the padding between the start of the widget and the start of the area that is used for drawing at the top.*
- void [takeCoordinatePlane](#) ([AbstractCoordinatePlane](#) \*plane)  
*Removes the coordinate plane from the chart, without deleting it.*
- void [takeHeaderFooter](#) ([HeaderFooter](#) \*headerFooter)  
*Removes the header (or footer, resp).*
- void [takeLegend](#) ([Legend](#) \*legend)  
*Removes the legend from the chart, without deleting it.*
- [~Chart](#) ()

## Protected Member Functions

- void [mouseDoubleClickEvent](#) (QMouseEvent \*event)  
*reimp*
- void [mouseMoveEvent](#) (QMouseEvent \*event)  
*reimp*
- void [mousePressEvent](#) (QMouseEvent \*event)  
*reimp*
- void [mouseReleaseEvent](#) (QMouseEvent \*event)  
*reimp*
- void [paintEvent](#) (QPaintEvent \*event)

*Draws the background and frame, then calls [paint\(\)](#).*

- void [resizeEvent](#) (QResizeEvent \*event)

*Adjusts the internal layout when the chart is resized.*

## Properties

- int [globalLeadingBottom](#) []
- int [globalLeadingLeft](#) []
- int [globalLeadingRight](#) []
- int [globalLeadingTop](#) []

## 9.21.2 Constructor & Destructor Documentation

### 9.21.2.1 Chart::Chart (QWidget \*parent = 0) [explicit]

Definition at line 783 of file KDChartChart.cpp.

References [addCoordinatePlane\(\)](#), [setFrameAttributes\(\)](#), [KDChart::FrameAttributes::setPadding\(\)](#), [KDChart::FrameAttributes::setPen\(\)](#), and [KDChart::FrameAttributes::setVisible\(\)](#).

```

784     : QWidget ( parent )
785     , _d( new Private( this ) )
786 {
787     #if defined KDAB_EVAL
788         EvalDialog::checkEvalLicense( "KD Chart" );
789     #endif
790
791     FrameAttributes frameAttrs;
792     frameAttrs.setVisible( true );
793     frameAttrs.setPen( QPen( Qt::black ) );
794     frameAttrs.setPadding( 1 );
795     setFrameAttributes( frameAttrs );
796
797     addCoordinatePlane( new CartesianCoordinatePlane ( this ) );
798 }
```

### 9.21.2.2 Chart::~~Chart ()

Definition at line 800 of file KDChartChart.cpp.

```

801 {
802     delete _d;
803 }
```

## 9.21.3 Member Function Documentation

### 9.21.3.1 void Chart::addCoordinatePlane (AbstractCoordinatePlane \*plane)

Adds a coordinate plane to the chart.

The chart takes ownership.



**Parameters:**

*plane* The coordinate plane to add.

**See also:**

[replaceCoordinatePlane](#), [takeCoordinatePlane](#)

Definition at line 848 of file KDChartChart.cpp.

References [d](#), [propertiesChanged\(\)](#), and [KDChart::AbstractCoordinatePlane::setParent\(\)](#).

Referenced by [Chart\(\)](#), and [replaceCoordinatePlane\(\)](#).

```

849 {
850     connect( plane, SIGNAL( destroyedCoordinatePlane( AbstractCoordinatePlane* ) ),
851             d,     SLOT( slotUnregisterDestroyedPlane( AbstractCoordinatePlane* ) ) );
852     connect( plane, SIGNAL( needUpdate() ),      this,     SLOT( update() ) );
853     connect( plane, SIGNAL( needRelayout() ),    d,         SLOT( slotRelayout() ) );
854     connect( plane, SIGNAL( needLayoutPlanes() ), d,         SLOT( slotLayoutPlanes() ) );
855     connect( plane, SIGNAL( propertiesChanged() ),this,     SIGNAL( propertiesChanged() ) );
856     d->coordinatePlanes.append( plane );
857     plane->setParent( this );
858     d->slotLayoutPlanes();
859 }
```

**9.21.3.2 void Chart::addHeaderFooter ([HeaderFooter](#) \* *headerFooter*)**

Adds a header or a footer to the chart.

The chart takes ownership.

**Parameters:**

*headerFooter* The header (or footer, resp.) to add.

**See also:**

[replaceHeaderFooter](#), [takeHeaderFooter](#)

Definition at line 1057 of file KDChartChart.cpp.

References [d](#), [headerFooter\(\)](#), and [KDChart::HeaderFooter::setParent\(\)](#).

Referenced by [replaceHeaderFooter\(\)](#).

```

1058 {
1059     d->headerFooters.append( headerFooter );
1060     headerFooter->setParent( this );
1061     connect( headerFooter, SIGNAL( destroyedHeaderFooter( HeaderFooter* ) ),
1062             d,     SLOT( slotUnregisterDestroyedHeaderFooter( HeaderFooter* ) ) );
1063     connect( headerFooter, SIGNAL( positionChanged( HeaderFooter* ) ),
1064             d,     SLOT( slotRelayout() ) );
1065     d->slotRelayout();
1066 }
```

**9.21.3.3 void Chart::addLegend ([Legend](#) \* *legend*)**

Add the given legend to the chart.

The chart takes ownership.

**Parameters:**

*legend* The legend to add.

**See also:**

[replaceLegend](#), [takeLegend](#)

Definition at line 1109 of file KDChartChart.cpp.

References [d](#), [legend\(\)](#), [KDChartEnums::MeasureOrientationMinimum](#), [propertiesChanged\(\)](#), [KDChart::TextAttributes::setFontSize\(\)](#), [KDChart::Legend::setReferenceArea\(\)](#), [KDChart::Legend::setTextAttributes\(\)](#), [KDChart::Legend::setTitleTextAttributes\(\)](#), [KDChart::Legend::setVisible\(\)](#), [KDChart::Legend::textAttributes\(\)](#), and [KDChart::Legend::titleTextAttributes\(\)](#).

Referenced by [replaceLegend\(\)](#).

```

1110 {
1111     if( ! legend ) return;
1112
1113     //qDebug() << "adding the legend";
1114     d->legends.append( legend );
1115     legend->setParent( this );
1116
1117     TextAttributes textAttrs( legend->textAttributes() );
1118
1119     KDChart::Measure measure( textAttrs.fontSize() );
1120     measure.setRelativeMode( this, KDChartEnums::MeasureOrientationMinimum );
1121     measure.setValue( 20 );
1122     textAttrs.setFontSize( measure );
1123     legend->setTextAttributes( textAttrs );
1124
1125     textAttrs = legend->titleTextAttributes();
1126     measure.setRelativeMode( this, KDChartEnums::MeasureOrientationMinimum );
1127     measure.setValue( 24 );
1128     textAttrs.setFontSize( measure );
1129
1130     legend->setTitleTextAttributes( textAttrs );
1131
1132     legend->setReferenceArea( this );
1133
1134     /*
1135     future: Use relative sizes for the markers too!
1136
1137     const uint nMA = Legend::datasetCount();
1138     for( uint iMA = 0; iMA < nMA; ++iMA ){
1139         MarkerAttributes ma( legend->markerAttributes( iMA ) );
1140         ma.setMarkerSize( ... )
1141         legend->setMarkerAttributes( iMA, ma )
1142     }
1143     */
1144
1145     connect( legend, SIGNAL( destroyedLegend( Legend* ) ),
1146             d, SLOT( slotUnregisterDestroyedLegend( Legend* ) ) );
1147     connect( legend, SIGNAL( positionChanged( AbstractAreaWidget* ) ),
1148             d, SLOT( slotLayoutPlanes() ) ); //slotRelayout() );
1149     connect( legend, SIGNAL( propertiesChanged() ),this, SIGNAL( propertiesChanged() ) );
1150     legend->setVisible( true );
1151     d->slotRelayout();
1152 }
```

#### 9.21.3.4 [BackgroundAttributes](#) Chart::backgroundAttributes () const

Definition at line 822 of file KDChartChart.cpp.

References d.

```
823 {  
824     return d->backgroundAttributes;  
825 }
```

#### 9.21.3.5 [AbstractCoordinatePlane](#) \* Chart::coordinatePlane ()

Each chart must have at least one coordinate plane.

Initially a default [CartesianCoordinatePlane](#) is created. Use [replaceCoordinatePlane\(\)](#) to replace it with a different one, such as a [PolarCoordinatePlane](#).

##### Returns:

The first coordinate plane of the chart.

Definition at line 832 of file KDChartChart.cpp.

References d.

```
833 {  
834     if ( d->coordinatePlanes.isEmpty() )  
835     {  
836         qWarning() << "Chart::coordinatePlane: warning: no coordinate plane defined."  
837         return 0;  
838     } else {  
839         return d->coordinatePlanes.first();  
840     }  
841 }
```

#### 9.21.3.6 [QLayout](#) \* Chart::coordinatePlaneLayout ()

Definition at line 827 of file KDChartChart.cpp.

References d.

```
828 {  
829     return d->planesLayout;  
830 }
```

#### 9.21.3.7 [CoordinatePlaneList](#) Chart::coordinatePlanes ()

The list of coordinate planes.

##### Returns:

The list of coordinate planes.

Definition at line 843 of file KDChartChart.cpp.

References d.

```
844 {  
845     return d->coordinatePlanes;  
846 }
```

### 9.21.3.8 [FrameAttributes](#) Chart::frameAttributes () const

Definition at line 812 of file KDChartChart.cpp.

References [d](#).

```
813 {  
814     return d->frameAttributes;  
815 }
```

### 9.21.3.9 int KDChart::Chart::globalLeadingBottom () const

The padding between the start of the widget and the start of the area that is used for drawing at the bottom.

#### Returns:

The padding between the start of the widget and the start of the area that is used for drawing at the bottom.

#### See also:

[setGlobalLeading](#)

### 9.21.3.10 int KDChart::Chart::globalLeadingLeft () const

The padding between the start of the widget and the start of the area that is used for drawing on the left.

#### Returns:

The padding between the start of the widget and the start of the area that is used for drawing on the left.

#### See also:

[setGlobalLeading](#)

### 9.21.3.11 int KDChart::Chart::globalLeadingRight () const

The padding between the start of the widget and the start of the area that is used for drawing on the right.

#### Returns:

The padding between the start of the widget and the start of the area that is used for drawing on the right.

#### See also:

[setGlobalLeading](#)

### 9.21.3.12 `int KDChart::Chart::globalLeadingTop () const`

The padding between the start of the widget and the start of the area that is used for drawing at the top.

#### Returns:

The padding between the start of the widget and the start of the area that is used for drawing at the top.

#### See also:

[setGlobalLeading](#)

### 9.21.3.13 `HeaderFooter * Chart::headerFooter ()`

The first header or footer of the chart.

By default there is none.

#### Returns:

The first header or footer of the chart or 0 if there was none added to the chart.

Definition at line 1095 of file KDChartChart.cpp.

References `d`.

Referenced by `addHeaderFooter()`, `replaceHeaderFooter()`, and `takeHeaderFooter()`.

```
1096 {  
1097     if( d->headerFooters.isEmpty() ) {  
1098         return 0;  
1099     } else {  
1100         return d->headerFooters.first();  
1101     }  
1102 }
```

### 9.21.3.14 `HeaderFooterList Chart::headerFooters ()`

The list of headers and footers associated with the chart.

#### Returns:

The list of headers and footers associated with the chart.

Definition at line 1104 of file KDChartChart.cpp.

References `d`.

```
1105 {  
1106     return d->headerFooters;  
1107 }
```

**9.21.3.15** [Legend](#) \* **Chart::legend ()**

The first legend of the chart or 0 if there was none added to the chart.

**Returns:**

The first legend of the chart or 0 if none exists.

Definition at line 1181 of file KDChartChart.cpp.

References [d](#).

Referenced by [addLegend\(\)](#), [paint\(\)](#), [reLayoutFloatingLegends\(\)](#), [replaceLegend\(\)](#), and [takeLegend\(\)](#).

```

1182 {
1183     if ( d->legends.isEmpty() )
1184     {
1185         return 0;
1186     } else {
1187         return d->legends.first();
1188     }
1189 }
```

**9.21.3.16** [LegendList](#) **Chart::legends ()**

The list of all legends associated with the chart.

**Returns:**

The list of all legends associated with the chart.

Definition at line 1191 of file KDChartChart.cpp.

References [d](#).

```

1192 {
1193     return d->legends;
1194 }
```

**9.21.3.17** **void Chart::mouseDoubleClickEvent (QMouseEvent \* *event*)** [protected]

reimp

Definition at line 1388 of file KDChartChart.cpp.

References [d](#), [KDChart::AbstractCoordinatePlane::diagrams\(\)](#), [KDChart::AbstractCoordinatePlane::geometry\(\)](#), and [KDChart::AbstractCoordinatePlane::mouseDoubleClickEvent\(\)](#).

```

1389 {
1390     const QPoint pos = mapFromGlobal( event->globalPos() );
1391
1392     KDAB_FOREACH( AbstractCoordinatePlane* plane, d->coordinatePlanes )
1393     {
1394         if ( plane->geometry().contains( event->pos() ) )
1395         {
1396             if ( plane->diagrams().size() > 0 )
1397             {
```

```

1398             QMouseEvent ev( QEvent::MouseButtonPress, pos, event->globalPos(),
1399                             event->button(), event->buttons(),
1400                             event->modifiers() );
1401             plane->mouseDoubleClickEvent( &ev );
1402         }
1403     }
1404 }
1405 }

```

### 9.21.3.18 void Chart::mouseMoveEvent (QMouseEvent \* event) [protected]

reimp

Definition at line 1407 of file KDChartChart.cpp.

References `d`, `KDChart::AbstractCoordinatePlane::diagrams()`, `KDChart::AbstractCoordinatePlane::geometry()`, and `KDChart::AbstractCoordinatePlane::mouseMoveEvent()`.

```

1408 {
1409     QSet< AbstractCoordinatePlane* > eventReceivers = QSet< AbstractCoordinatePlane* >::fromList( d->
1410
1411     KDAB_FOREACH( AbstractCoordinatePlane* plane, d->coordinatePlanes )
1412     {
1413         if( plane->geometry().contains( event->pos() ) )
1414         {
1415             if( plane->diagrams().size() > 0 )
1416             {
1417                 eventReceivers.insert( plane );
1418             }
1419         }
1420     }
1421
1422     const QPoint pos = mapFromGlobal( event->globalPos() );
1423
1424     KDAB_FOREACH( AbstractCoordinatePlane* plane, eventReceivers )
1425     {
1426         QMouseEvent ev( QEvent::MouseMove, pos, event->globalPos(),
1427                         event->button(), event->buttons(),
1428                         event->modifiers() );
1429         plane->mouseMoveEvent( &ev );
1430     }
1431 }

```

### 9.21.3.19 void Chart::mousePressEvent (QMouseEvent \* event) [protected]

reimp

Definition at line 1197 of file KDChartChart.cpp.

References `d`, `KDChart::AbstractCoordinatePlane::diagrams()`, `KDChart::AbstractCoordinatePlane::geometry()`, and `KDChart::AbstractCoordinatePlane::mousePressEvent()`.

```

1198 {
1199     const QPoint pos = mapFromGlobal( event->globalPos() );
1200
1201     KDAB_FOREACH( AbstractCoordinatePlane* plane, d->coordinatePlanes )
1202     {
1203         if ( plane->geometry().contains( event->pos() ) )
1204         {
1205             if ( plane->diagrams().size() > 0 )
1206             {

```

```

1207             QMouseEvent ev( QEvent::MouseButtonPress, pos, event->globalPos(),
1208                             event->button(), event->buttons(),
1209                             event->modifiers() );
1210
1211             plane->mousePressEvent( &ev );
1212             d->mouseClickedPlanes.append( plane );
1213         }
1214     }
1215 }
1216 }

```

### 9.21.3.20 void Chart::mouseReleaseEvent( QMouseEvent \* event) [protected]

reimp

Definition at line 1433 of file KDChartChart.cpp.

References [d](#), [KDChart::AbstractCoordinatePlane::diagrams\(\)](#), [KDChart::AbstractCoordinatePlane::geometry\(\)](#), and [KDChart::AbstractCoordinatePlane::mouseReleaseEvent\(\)](#).

```

1434 {
1435     QSet< AbstractCoordinatePlane* > eventReceivers = QSet< AbstractCoordinatePlane* >::fromList( d->
1436
1437     KDAB_FOREACH( AbstractCoordinatePlane* plane, d->coordinatePlanes )
1438     {
1439         if ( plane->geometry().contains( event->pos() ) )
1440         {
1441             if( plane->diagrams().size() > 0 )
1442             {
1443                 eventReceivers.insert( plane );
1444             }
1445         }
1446     }
1447
1448     const QPoint pos = mapFromGlobal( event->globalPos() );
1449
1450     KDAB_FOREACH( AbstractCoordinatePlane* plane, eventReceivers )
1451     {
1452         QMouseEvent ev( QEvent::MouseButtonRelease, pos, event->globalPos(),
1453                         event->button(), event->buttons(),
1454                         event->modifiers() );
1455         plane->mouseReleaseEvent( &ev );
1456     }
1457
1458     d->mouseClickedPlanes.clear();
1459 }

```

### 9.21.3.21 void Chart::paint( QPainter \* painter, const QRect & target)

Paints all the contents of the chart.

Use this method, to make [KDChart](#) draw into your QPainter.

#### Note:

Any global leading settings will be used by the paint method too, so make sure to set them to zero, if you want the drawing to have the exact size of the target rectangle.

#### Parameters:

**painter** The painter to be drawn into.



*target* The rectangle to be filled by the Chart's drawing.

See also:

[setGlobalLeading](#)

Definition at line 942 of file KDCartChart.cpp.

References `d`, `KDCart::GlobalMeasureScaling::instance()`, `legend()`, `KDCart::AbstractAreaWidget::paintIntoRect()`, `KDCart::GlobalMeasureScaling::resetFactors()`, `KDCart::GlobalMeasureScaling::setFactors()`, and `KDCart::GlobalMeasureScaling::setPaintDevice()`.

```

943 {
944     if( target.isEmpty() || !painter ) return;
945     //qDebug() << "Chart::paint( .., " << target << " )";
946
947     GlobalMeasureScaling::setPaintDevice( painter->device() );
948
949     // Output on a widget
950     if( dynamic_cast< QWidget* >( painter->device() ) != 0 )
951     {
952         GlobalMeasureScaling::setFactors(
953             static_cast< qreal >( target.width() ) /
954             static_cast< qreal >( geometry().size().width() ),
955             static_cast< qreal >( target.height() ) /
956             static_cast< qreal >( geometry().size().height() ) );
957     }
958     // Output onto a QPixmap
959     else
960     {
961         const qreal resX = static_cast< qreal >( logicalDpiX() ) / static_cast< qreal >( painter->device()->logicalDpiX() );
962         const qreal resY = static_cast< qreal >( logicalDpiY() ) / static_cast< qreal >( painter->device()->logicalDpiY() );
963
964         GlobalMeasureScaling::setFactors(
965             static_cast< qreal >( target.width() ) /
966             static_cast< qreal >( geometry().size().width() ) * resX,
967             static_cast< qreal >( target.height() ) /
968             static_cast< qreal >( geometry().size().height() ) * resY );
969     }
970
971
972     if( target.size() != d->currentLayoutSize ){
973         d->resizeLayout( target.size() );
974     }
975     const QPoint translation = target.topLeft();
976     painter->translate( translation );
977
978     d->paintAll( painter );
979
980     // for debugging:
981     //painter->setPen( QPen( Qt::blue, 8) );
982     //painter->drawRect( target.adjusted(12,12,-12,-12) );
983
984     KDAB_FOREACH( Legend *legend, d->legends ) {
985         const bool hidden = legend->isHidden() && legend->testAttribute( Qt::WA_WState_ExplicitShowHide );
986         if ( !hidden ) {
987             //qDebug() << "painting legend at " << legend->geometry();
988             legend->paintIntoRect( *painter, legend->geometry() );
989             //testing:
990             //legend->paintIntoRect( *painter, legend->geometry().adjusted(-100,0,-100,0) );
991         }
992     }
993
994     painter->translate( -translation.x(), -translation.y() );
995
996     GlobalMeasureScaling::instance()->resetFactors();

```

```

997
998      //qDebug() << "KDChart::Chart::paint() done.\n";
999  }

```

### 9.21.3.22 void Chart::paintEvent (QPaintEvent \* event) [protected]

Draws the background and frame, then calls [paint\(\)](#).

In most cases there is no need to override this method in a derived class, but if you do, do not forget to call [paint\(\)](#).

See also:

[paint](#)

Definition at line 1043 of file KDChartChart.cpp.

References [d](#), and [reLayoutFloatingLegends\(\)](#).

```

1044 {
1045     QPainter painter( this );
1046
1047     if( size() != d->currentLayoutSize ){
1048         d->resizeLayout( size() );
1049         reLayoutFloatingLegends();
1050     }
1051
1052     //FIXME(khz): Paint the background/frame too!
1053     //             (can we derive Chart from AreaWidget ??)
1054     d->paintAll( &painter );
1055 }

```

### 9.21.3.23 void KDChart::Chart::propertiesChanged () [signal]

Emitted upon change of a property of the [Chart](#) or any of its components.

Referenced by [addCoordinatePlane\(\)](#), and [addLegend\(\)](#).

### 9.21.3.24 void Chart::reLayoutFloatingLegends ()

Definition at line 1011 of file KDChartChart.cpp.

References [d](#), [KDChart::Legend::floatingPosition\(\)](#), [KDChart::Position::isFloating\(\)](#), [legend\(\)](#), [KDChart::Legend::position\(\)](#), and [KDChart::Legend::sizeHint\(\)](#).

Referenced by [paintEvent\(\)](#), and [resizeEvent\(\)](#).

```

1012 {
1013     KDAB_FOREACH( Legend *legend, d->legends ) {
1014         const bool hidden = legend->isHidden() && legend->testAttribute(Qt::WA_WState_ExplicitShowHidden);
1015         if ( legend->position().isFloating() && !hidden ){
1016             // resize the legend
1017             const QSize legendSize( legend->sizeHint() );
1018             legend->setGeometry( QRect( legend->geometry().topLeft(), legendSize ) );
1019             // find the legends corner point (reference point plus any paddings)
1020             const RelativePosition relPos( legend->floatingPosition() );
1021             QPointF pt( relPos.calculatedPoint( size() ) );
1022             qDebug() << pt;

```

```

1023         // calculate the legend's top left point
1024         const Qt::Alignment alignTopLeft = Qt::AlignBottom | Qt::AlignLeft;
1025         if( (relPos.alignment() & alignTopLeft) != alignTopLeft ){
1026             if( relPos.alignment() & Qt::AlignRight )
1027                 pt.rx() -= legendSize.width();
1028             else if( relPos.alignment() & Qt::AlignHCenter )
1029                 pt.rx() -= 0.5 * legendSize.width();
1030
1031             if( relPos.alignment() & Qt::AlignBottom )
1032                 pt.ry() -= legendSize.height();
1033             else if( relPos.alignment() & Qt::AlignVCenter )
1034                 pt.ry() -= 0.5 * legendSize.height();
1035         }
1036         qDebug() << pt << endl;
1037         legend->move( static_cast<int>(pt.x()), static_cast<int>(pt.y()) );
1038     }
1039 }
1040 }

```

### 9.21.3.25 void Chart::replaceCoordinatePlane ([AbstractCoordinatePlane](#) \* *plane*, [AbstractCoordinatePlane](#) \* *oldPlane* = 0)

Replaces the old coordinate plane, or appends the plane, if there is none yet.

#### Parameters:

***plane*** The coordinate plane to be used instead of the old plane. This parameter must not be zero, or the method will do nothing.

***oldPlane*** The coordinate plane to be removed by the new plane. This plane will be deleted automatically. If the parameter is omitted, the very first coordinate plane will be replaced. In case, there was no plane yet, the new plane will just be added.

#### Note:

If you want to re-use the old coordinate plane, call `takeCoordinatePlane` and `addCoordinatePlane`, instead of using `replaceCoordinatePlane`.

#### See also:

[addCoordinatePlane](#), [takeCoordinatePlane](#)

Definition at line 861 of file `KDChartChart.cpp`.

References `addCoordinatePlane()`, `d`, and `takeCoordinatePlane()`.

```

863 {
864     if( plane && oldPlane_ != plane ){
865         AbstractCoordinatePlane* oldPlane = oldPlane_;
866         if( d->coordinatePlanes.count() ){
867             if( ! oldPlane )
868                 oldPlane = d->coordinatePlanes.first();
869             takeCoordinatePlane( oldPlane );
870         }
871         delete oldPlane;
872         addCoordinatePlane( plane );
873     }
874 }

```

### 9.21.3.26 void Chart::replaceHeaderFooter ([HeaderFooter](#) \* *headerFooter*, [HeaderFooter](#) \* *oldHeaderFooter* = 0)

Replaces the old header (or footer, resp.  
) , or appends the new header or footer, if there is none yet.

#### Parameters:

***headerFooter*** The header or footer to be used instead of the old one. This parameter must not be zero, or the method will do nothing.

***oldHeaderFooter*** The header or footer to be removed by the new one. This header or footer will be deleted automatically. If the parameter is omitted, the very first header or footer will be replaced. In case, there was no header and no footer yet, the new header or footer will just be added.

#### Note:

If you want to re-use the old header or footer, call takeHeaderFooter and addHeaderFooter, instead of using replaceHeaderFooter.

#### See also:

[addHeaderFooter](#), [takeHeaderFooter](#)

Definition at line 1068 of file KDChartChart.cpp.

References [addHeaderFooter\(\)](#), [d](#), [headerFooter\(\)](#), and [takeHeaderFooter\(\)](#).

```

1070 {
1071     if( headerFooter && oldHeaderFooter_ != headerFooter ){
1072         HeaderFooter* oldHeaderFooter = oldHeaderFooter_;
1073         if( d->headerFooters.count() ){
1074             if( ! oldHeaderFooter )
1075                 oldHeaderFooter = d->headerFooters.first();
1076             takeHeaderFooter( oldHeaderFooter );
1077         }
1078         delete oldHeaderFooter;
1079         addHeaderFooter( headerFooter );
1080     }
1081 }
```

### 9.21.3.27 void Chart::replaceLegend ([Legend](#) \* *legend*, [Legend](#) \* *oldLegend* = 0)

Replaces the old legend, or appends the new legend, if there is none yet.

#### Parameters:

***legend*** The legend to be used instead of the old one. This parameter must not be zero, or the method will do nothing.

***oldLegend*** The legend to be removed by the new one. This legend will be deleted automatically. If the parameter is omitted, the very first legend will be replaced. In case, there was no legend yet, the new legend will just be added.

If you want to re-use the old legend, call takeLegend and addLegend, instead of using replaceLegend.

**Note:**

Whenever addLegend is called the font sizes used by the [Legend](#) are set to relative and they get coupled to the Chart's size, with their relative values being 20 for the item texts and 24 to the title text. So if you want to use custom font sizes for the [Legend](#) make sure to set them after calling addLegend.

**See also:**

[addLegend](#), [takeLegend](#)

Definition at line 1155 of file KDChartChart.cpp.

References [addLegend\(\)](#), [d](#), [legend\(\)](#), and [takeLegend\(\)](#).

```

1156 {
1157     if( legend && oldLegend_ != legend ){
1158         Legend* oldLegend = oldLegend_;
1159         if( d->legends.count() ){
1160             if( ! oldLegend )
1161                 oldLegend = d->legends.first();
1162             takeLegend( oldLegend );
1163         }
1164         delete oldLegend;
1165         addLegend( legend );
1166     }
1167 }
```

**9.21.3.28 void Chart::resizeEvent (QResizeEvent \* event) [protected]**

Adjusts the internal layout when the chart is resized.

Definition at line 1001 of file KDChartChart.cpp.

References [d](#), [reLayoutFloatingLegends\(\)](#), and [KDChart::AbstractCoordinatePlane::setGridNeedsRecalculate\(\)](#).

```

1002 {
1003     d->resizeLayout( size() );
1004     KDAB_FOREACH( AbstractCoordinatePlane* plane, d->coordinatePlanes ){
1005         plane->setGridNeedsRecalculate();
1006     }
1007     reLayoutFloatingLegends();
1008 }
```

**9.21.3.29 void Chart::setBackgroundAttributes (const [BackgroundAttributes](#) & a)**

Specify the background attributes to be used, by default there is no background.

To set a light blue background, you could do something like this:

```

KDChart::BackgroundAttributes backgroundAttrs( my_chart->backgroundAttributes() );
backgroundAttrs.setVisible( true );
backgroundAttrs.setBrush( QColor(0xd0,0xd0,0xff) );
my_chart->setBackgroundAttributes( backgroundAttrs );
```

**See also:**

[setFrameAttributes](#)

Definition at line 817 of file KDChartChart.cpp.

References d.

```
818 {  
819     d->backgroundAttributes = a;  
820 }
```

#### 9.21.3.30 void KDChart::Chart::setCoordinatePlaneLayout (QLayout \* *layout*)

#### 9.21.3.31 void Chart::setFrameAttributes (const [FrameAttributes](#) & *a*)

Specify the frame attributes to be used, by default is it a thin black line.

To hide the frame line, you could do something like this:

```
KDChart::FrameAttributes frameAttrs( my_chart->frameAttributes() );  
frameAttrs.setVisible( false );  
my_chart->setFrameAttributes( frameAttrs );
```

See also:

[setBackgroundAttributes](#)

Definition at line 807 of file KDChartChart.cpp.

References d.

Referenced by Chart().

```
808 {  
809     d->frameAttributes = a;  
810 }
```

#### 9.21.3.32 void Chart::setGlobalLeading (int *left*, int *top*, int *right*, int *bottom*)

Set the padding between the margin of the widget and the area that the contents are drawn into.

**Parameters:**

*left* The padding on the left side.

*top* The padding at the top.

*right* The padding on the left hand side.

*bottom* The padding on the bottom.

**Note:**

Using previous versions of KD [Chart](#) you might have called [setGlobalLeading\(\)](#) to make room for long Abscissa labels (or for an overlapping top label of an Ordinate axis, resp.) that would not fit into the normal axis area. This is *no longer needed* because KD [Chart](#) now is using hidden auto-spacer items reserving as much free space as is needed for axes with overlapping content at the respective sides.

**See also:**

[setGlobalLeadingTop](#), [setGlobalLeadingBottom](#), [setGlobalLeadingLeft](#), [setGlobalLeadingRight](#)  
[globalLeadingTop](#), [globalLeadingBottom](#), [globalLeadingLeft](#), [globalLeadingRight](#)

Definition at line 889 of file KDChartChart.cpp.

References [d](#), [setGlobalLeadingBottom\(\)](#), [setGlobalLeadingLeft\(\)](#), [setGlobalLeadingRight\(\)](#), and [setGlobalLeadingTop\(\)](#).

```
890 {  
891     setGlobalLeadingLeft( left );  
892     setGlobalLeadingTop( top );  
893     setGlobalLeadingRight( right );  
894     setGlobalLeadingBottom( bottom );  
895     d->slotRelayout();  
896 }
```

**9.21.3.33 void Chart::setGlobalLeadingBottom (int *leading*)**

Set the padding between the start of the widget and the start of the area that is used for drawing on the bottom.

**Parameters:**

*leading* The padding value.

**See also:**

[setGlobalLeading](#)

Definition at line 931 of file KDChartChart.cpp.

References [d](#).

Referenced by [setGlobalLeading\(\)](#).

```
932 {  
933     d->globalLeadingBottom = leading;  
934     d->slotRelayout();  
935 }
```

**9.21.3.34 void Chart::setGlobalLeadingLeft (int *leading*)**

Set the padding between the start of the widget and the start of the area that is used for drawing on the left.

**Parameters:**

*leading* The padding value.

**See also:**

[setGlobalLeading](#)

Definition at line 898 of file KDChartChart.cpp.

References [d](#).

Referenced by [setGlobalLeading\(\)](#).

```
899 {  
900     d->globalLeadingLeft = leading;  
901     d->slotRelayout();  
902 }
```

#### 9.21.3.35 void Chart::setGlobalLeadingRight (int *leading*)

Set the padding between the start of the widget and the start of the area that is used for drawing on the right.

##### Parameters:

*leading* The padding value.

##### See also:

[setGlobalLeading](#)

Definition at line 920 of file KDChartChart.cpp.

References [d](#).

Referenced by [setGlobalLeading\(\)](#).

```
921 {  
922     d->globalLeadingRight = leading;  
923     d->slotRelayout();  
924 }
```

#### 9.21.3.36 void Chart::setGlobalLeadingTop (int *leading*)

Set the padding between the start of the widget and the start of the area that is used for drawing at the top.

##### Parameters:

*leading* The padding value.

##### See also:

[setGlobalLeading](#)

Definition at line 909 of file KDChartChart.cpp.

References [d](#).

Referenced by [setGlobalLeading\(\)](#).

```
910 {  
911     d->globalLeadingTop = leading;  
912     d->slotRelayout();  
913 }
```



**9.21.3.37 void Chart::takeCoordinatePlane ([AbstractCoordinatePlane](#) \* *plane*)**

Removes the coordinate plane from the chart, without deleting it.

The chart no longer owns the plane, so it is the caller's responsibility to delete the plane.

**See also:**

[addCoordinatePlane](#), [takeCoordinatePlane](#)

Definition at line 876 of file KDChartChart.cpp.

References [d](#), [KDChart::AbstractLayoutItem::removeFromParentLayout\(\)](#), and [KDChart::AbstractCoordinatePlane::setParent\(\)](#).

Referenced by [replaceCoordinatePlane\(\)](#).

```
877 {
878     const int idx = d->coordinatePlanes.indexOf( plane );
879     if( idx != -1 ){
880         d->coordinatePlanes.takeAt( idx );
881         disconnect( plane, SIGNAL( destroyedCoordinatePlane( AbstractCoordinatePlane* ) ),
882                    d, SLOT( slotUnregisterDestroyedPlane( AbstractCoordinatePlane* ) ) );
883         plane->removeFromParentLayout();
884         plane->setParent( 0 );
885     }
886     d->slotLayoutPlanes();
887 }
```

**9.21.3.38 void Chart::takeHeaderFooter ([HeaderFooter](#) \* *headerFooter*)**

Removes the header (or footer, resp.

) from the chart, without deleting it.

The chart no longer owns the header or footer, so it is the caller's responsibility to delete the header or footer.

**See also:**

[addHeaderFooter](#), [replaceHeaderFooter](#)

Definition at line 1083 of file KDChartChart.cpp.

References [d](#), [headerFooter\(\)](#), and [KDChart::HeaderFooter::setParent\(\)](#).

Referenced by [replaceHeaderFooter\(\)](#).

```
1084 {
1085     const int idx = d->headerFooters.indexOf( headerFooter );
1086     if( idx != -1 ){
1087         d->headerFooters.takeAt( idx );
1088         disconnect( headerFooter, SIGNAL( destroyedHeaderFooter( HeaderFooter* ) ),
1089                    d, SLOT( slotUnregisterDestroyedHeaderFooter( HeaderFooter* ) ) );
1090         headerFooter->setParent( 0 );
1091     }
1092     d->slotRelayout();
1093 }
```

### 9.21.3.39 void Chart::takeLegend ([Legend](#) \* *legend*)

Removes the legend from the chart, without deleting it.

The chart no longer owns the legend, so it is the caller's responsibility to delete the legend.

See also:

[addLegend](#), [takeLegend](#)

Definition at line 1169 of file KDChartChart.cpp.

References [d](#), and [legend\(\)](#).

Referenced by [replaceLegend\(\)](#).

```
1170 {
1171     const int idx = d->legends.indexOf( legend );
1172     if( idx != -1 ){
1173         d->legends.takeAt( idx );
1174         disconnect( legend, SIGNAL( destroyedLegend( Legend* ) ),
1175                     d, SLOT( slotUnregisterDestroyedLegend( Legend* ) ) );
1176         legend->setParent( 0 );
1177     }
1178     d->slotRelayout();
1179 }
```

## 9.21.4 Property Documentation

### 9.21.4.1 int Chart::globalLeadingBottom [read, write]

Definition at line 76 of file KDChartChart.h.

### 9.21.4.2 int Chart::globalLeadingLeft [read, write]

Definition at line 77 of file KDChartChart.h.

### 9.21.4.3 int Chart::globalLeadingRight [read, write]

Definition at line 78 of file KDChartChart.h.

### 9.21.4.4 int Chart::globalLeadingTop [read, write]

Definition at line 75 of file KDChartChart.h.

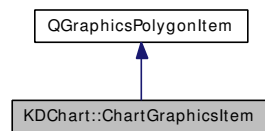
The documentation for this class was generated from the following files:

- [KDChartChart.h](#)
- [KDChartChart.cpp](#)

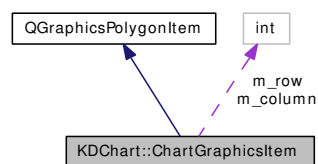
## 9.22 KDChart::ChartGraphicsItem Class Reference

```
#include <ChartGraphicsItem.h>
```

Inheritance diagram for KDChart::ChartGraphicsItem:



Collaboration diagram for KDChart::ChartGraphicsItem:



### 9.22.1 Detailed Description

Graphics item used inside of the [ReverseMapper](#).

Definition at line 41 of file ChartGraphicsItem.h.

### Public Types

- enum { [Type](#) = UserType + 1 }

### Public Member Functions

- [ChartGraphicsItem](#) (int row, int column)
- [ChartGraphicsItem](#) ()
- int [column](#) () const
- int [row](#) () const
- int [type](#) () const

### 9.22.2 Member Enumeration Documentation

#### 9.22.2.1 anonymous enum

Enumerator:

*Type*

Definition at line 44 of file ChartGraphicsItem.h.

```
44 { Type = UserType + 1 };
```

### 9.22.3 Constructor & Destructor Documentation

#### 9.22.3.1 ChartGraphicsItem::ChartGraphicsItem ()

Definition at line 34 of file ChartGraphicsItem.cpp.

```
35      : QGraphicsPolygonItem()  
36      , m_row( -1 )  
37      , m_column( -1 )  
38 {  
39 }
```

#### 9.22.3.2 ChartGraphicsItem::ChartGraphicsItem (int row, int column)

Definition at line 41 of file ChartGraphicsItem.cpp.

```
42      : QGraphicsPolygonItem()  
43      , m_row( row )  
44      , m_column( column )  
45 {  
46 }
```

### 9.22.4 Member Function Documentation

#### 9.22.4.1 int KDChart::ChartGraphicsItem::column () const

Definition at line 51 of file ChartGraphicsItem.h.

Referenced by KDChart::ReverseMapper::indexesAt(), and KDChart::ReverseMapper::indexesIn().

```
51 { return m_column; }
```

#### 9.22.4.2 int KDChart::ChartGraphicsItem::row () const

Definition at line 50 of file ChartGraphicsItem.h.

Referenced by KDChart::ReverseMapper::indexesAt(), and KDChart::ReverseMapper::indexesIn().

```
50 { return m_row; }
```

#### 9.22.4.3 int KDChart::ChartGraphicsItem::type () const

Definition at line 52 of file ChartGraphicsItem.h.

References Type.

```
52 { return Type; }
```

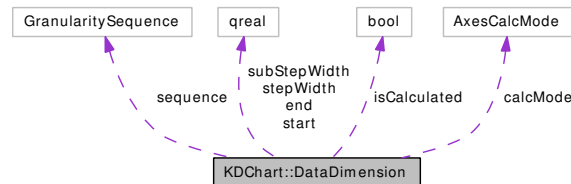
The documentation for this class was generated from the following files:

- [ChartGraphicsItem.h](#)
- [ChartGraphicsItem.cpp](#)

## 9.23 KDChart::DataDimension Class Reference

```
#include <KDChartAbstractCoordinatePlane.h>
```

Collaboration diagram for KDChart::DataDimension:



### 9.23.1 Detailed Description

Helper class for one dimension of data, e.g.

for the rows in a data model, or for the labels of an axis, or for the vertical lines in a grid.

isCalculated specifies whether this dimension's values are calculated or counted. (counted == "Item 1", "Item 2", "Item 3" ...)

sequence is the GranularitySequence, as specified at for the respective coordinate plane.

Step width is an optional parameter, to be omitted (or set to Zero, resp.) if the step width is unknown.

The default c'tor just gets you counted values from 1..10, using step width 1, used by the CartesianGrid, when showing an empty plane without any diagrams.

Definition at line 348 of file KDChartAbstractCoordinatePlane.h.

### Public Member Functions

- [DataDimension](#) (qreal start\_, qreal end\_, bool isCalculated\_, [AbstractCoordinatePlane::AxesCalcMode](#) calcMode\_, [KDChartEnums::GranularitySequence](#) sequence\_, qreal stepWidth\_=0.0, qreal subStepWidth\_=0.0)
- [DataDimension](#) ()
- qreal [distance](#) () const

*Returns the size of the distance, equivalent to the width() (or height()), resp.*

- bool [operator!=](#) (const [DataDimension](#) &other) const
- bool [operator==](#) (const [DataDimension](#) &r) const

### Public Attributes

- [AbstractCoordinatePlane::AxesCalcMode](#) calcMode
- qreal end
- bool isCalculated
- [KDChartEnums::GranularitySequence](#) sequence
- qreal start
- qreal stepWidth
- qreal subStepWidth

## 9.23.2 Constructor & Destructor Documentation

### 9.23.2.1 KDChart::DataDimension::DataDimension ()

Definition at line 350 of file KDChartAbstractCoordinatePlane.h.

```

351         : start(          1.0 )
352         , end(          10.0 )
353         , isCalculated( false )
354         , calcMode( AbstractCoordinatePlane::Linear )
355         , sequence( KDChartEnums::GranularitySequence_10_20 )
356         , stepWidth(    1.0 )
357         , subStepWidth( 0.0 )
358     {}

```

### 9.23.2.2 KDChart::DataDimension::DataDimension (qreal *start\_*, qreal *end\_*, bool *isCalculated\_*, [AbstractCoordinatePlane::AxesCalcMode](#) *calcMode\_*, [KDChartEnums::GranularitySequence](#) *sequence\_*, qreal *stepWidth\_* = 0.0, qreal *subStepWidth\_* = 0.0)

Definition at line 359 of file KDChartAbstractCoordinatePlane.h.

```

366         : start(          start_ )
367         , end(          end_ )
368         , isCalculated( isCalculated_ )
369         , calcMode(      calcMode_ )
370         , sequence(      sequence_ )
371         , stepWidth(      stepWidth_ )
372         , subStepWidth( subStepWidth_ )
373     {}

```

## 9.23.3 Member Function Documentation

### 9.23.3.1 qreal KDChart::DataDimension::distance () const

Returns the size of the distance, equivalent to the width() (or height(), resp.) of a QRectF.

Note that this value can be negative, e.g. indicating axis labels going in reversed direction.

Definition at line 381 of file KDChartAbstractCoordinatePlane.h.

References end, and start.

Referenced by KDChart::CartesianCoordinatePlane::layoutDiagrams().

```

382     {
383         return end-start;
384     }

```

### 9.23.3.2 bool KDChart::DataDimension::operator!=(const [DataDimension](#) & *other*) const

Definition at line 398 of file KDChartAbstractCoordinatePlane.h.

References operator==().

```

399     { return !operator==( other ); }

```

**9.23.3.3 bool KDChart::DataDimension::operator==(const DataDimension & r) const**

Definition at line 386 of file KDChartAbstractCoordinatePlane.h.

References calcMode, end, isCalculated, sequence, start, stepWidth, and subStepWidth.

Referenced by operator!=(()).

```

387     {
388         return
389             (start      == r.start) &&
390             (end        == r.end) &&
391             (sequence    == r.sequence) &&
392             (isCalculated == r.isCalculated) &&
393             (calcMode    == r.calcMode) &&
394             (stepWidth   == r.stepWidth) &&
395             (subStepWidth == r.subStepWidth);
396     }
```

**9.23.4 Member Data Documentation****9.23.4.1 AbstractCoordinatePlane::AxesCalcMode KDChart::DataDimension::calcMode**

Definition at line 405 of file KDChartAbstractCoordinatePlane.h.

Referenced by KDChart::operator<<(), and operator==(()).

**9.23.4.2 qreal KDChart::DataDimension::end**

Definition at line 403 of file KDChartAbstractCoordinatePlane.h.

Referenced by distance(), KDChart::CartesianCoordinatePlane::layoutDiagrams(), KDChart::operator<<(), and operator==(()).

**9.23.4.3 bool KDChart::DataDimension::isCalculated**

Definition at line 404 of file KDChartAbstractCoordinatePlane.h.

Referenced by KDChart::operator<<(), and operator==(()).

**9.23.4.4 KDChartEnums::GranularitySequence KDChart::DataDimension::sequence**

Definition at line 406 of file KDChartAbstractCoordinatePlane.h.

Referenced by KDChart::operator<<(), and operator==(()).

**9.23.4.5 qreal KDChart::DataDimension::start**

Definition at line 402 of file KDChartAbstractCoordinatePlane.h.

Referenced by distance(), KDChart::CartesianCoordinatePlane::layoutDiagrams(), KDChart::operator<<(), and operator==(()).

#### 9.23.4.6 `qreal KDChart::DataDimension::stepWidth`

Definition at line 407 of file `KDChartAbstractCoordinatePlane.h`.

Referenced by `KDChart::operator<<()`, and `operator==()`.

#### 9.23.4.7 `qreal KDChart::DataDimension::subStepWidth`

Definition at line 408 of file `KDChartAbstractCoordinatePlane.h`.

Referenced by `KDChart::operator<<()`, and `operator==()`.

The documentation for this class was generated from the following file:

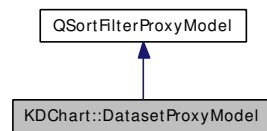
- [KDChartAbstractCoordinatePlane.h](#)



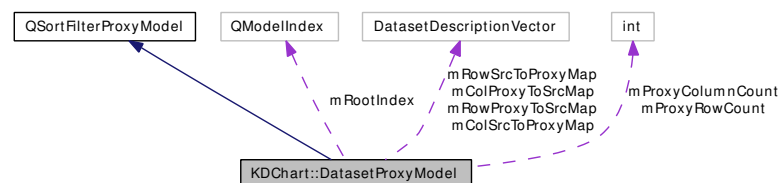
## 9.24 KDChart::DatasetProxyModel Class Reference

```
#include <KDChartDatasetProxyModel.h>
```

Inheritance diagram for KDChart::DatasetProxyModel:



Collaboration diagram for KDChart::DatasetProxyModel:



### 9.24.1 Detailed Description

[DatasetProxyModel](#) takes a [KDChart](#) dataset configuration and translates it into a filtering proxy model.

The resulting model will only contain the part of the model that is selected by the dataset, and the according row and column header data.

Currently, this model is implemented for table models only. The way it would work with models representing a tree is to be decided.

The column selection is configured by passing a dataset description vector to the model. This vector (of integers) is supposed to have one value for each column of the original model. If the value at position  $x$  is  $-1$ , column  $x$  of the original model is not included in the dataset. If it is between  $0$  and  $(\text{columnCount}() - 1)$ , it is the column the source column is mapped to in the resulting model. Any other value is an error.

Definition at line 58 of file `KDChartDatasetProxyModel.h`.

### Public Slots

- void [resetDatasetDescriptions](#) ()  
*Reset all dataset description.*
- void [setDatasetColumnDescriptionVector](#) (const [DatasetDescriptionVector](#) &columnConfig)  
*Configure the dataset selection for the columns.*
- void [setDatasetDescriptionVectors](#) (const [DatasetDescriptionVector](#) &rowConfig, const [DatasetDescriptionVector](#) &columnConfig)  
*Convenience method to configure rows and columns in one step.*
- void [setDatasetRowDescriptionVector](#) (const [DatasetDescriptionVector](#) &rowConfig)  
*Configure the dataset selection for the rows.*

## Public Member Functions

- QModelIndex [buddy](#) (const QModelIndex &index) const
- QVariant [data](#) (const QModelIndex &index, int role) const  
*Overloaded from base class.*
- [DatasetProxyModel](#) (QObject \*parent=0)  
*Create a [DatasetProxyModel](#).*
- Qt::ItemFlags [flags](#) (const QModelIndex &index) const
- QVariant [headerData](#) (int section, Qt::Orientation orientation, int role=Qt::DisplayRole) const  
*Overloaded from base class.*
- QModelIndex [index](#) (int row, int column, const QModelIndex &parent=QModelIndex()) const
- QModelIndex [mapFromSource](#) (const QModelIndex &sourceIndex) const  
*Implements the mapping from the source to the proxy indexes.*
- QModelIndex [mapToSource](#) (const QModelIndex &proxyIndex) const  
*Implements the mapping from the proxy to the source indexes.*
- QModelIndex [parent](#) (const QModelIndex &child) const
- bool [setData](#) (const QModelIndex &index, const QVariant &value, int role)  
*Overloaded from base class.*
- void [setSourceModel](#) (QAbstractItemModel \*sourceModel)  
*Overloaded from base class.*
- void [setSourceRootIndex](#) (const QModelIndex &rootIdx)  
*Set the root index of the table in the source model.*

## Protected Member Functions

- bool [filterAcceptsColumn](#) (int sourceColumn, const QModelIndex &) const  
*Decide whether the column is accepted.*
- bool [filterAcceptsRow](#) (int source\_row, const QModelIndex &source\_parent) const  
*Decide whether the row is accepted.*

## 9.24.2 Constructor & Destructor Documentation

### 9.24.2.1 DatasetProxyModel::DatasetProxyModel (QObject \*parent = 0) [explicit]

Create a [DatasetProxyModel](#).

Without further configuration, this model is invalid.

**See also:**

[setDatasetDescriptionVector](#)

Definition at line 35 of file KDChartDatasetProxyModel.cpp.

```
36      : QSortFilterProxyModel ( parent )
37  {
38  }
```

### 9.24.3 Member Function Documentation

#### 9.24.3.1 QModelIndex DatasetProxyModel::buddy (const QModelIndex & *index*) const

Definition at line 40 of file KDChartDatasetProxyModel.cpp.

```
41 {
42     return index;
43 }
```

#### 9.24.3.2 QVariant DatasetProxyModel::data (const QModelIndex & *index*, int *role*) const

Overloaded from base class.

Definition at line 219 of file KDChartDatasetProxyModel.cpp.

References `mapToSource()`.

```
220 {
221     return sourceModel()->data( mapToSource ( index ), role );
222 }
```

#### 9.24.3.3 bool DatasetProxyModel::filterAcceptsColumn (int *sourceColumn*, const QModelIndex &) const [protected]

Decide whether the column is accepted.

Definition at line 146 of file KDChartDatasetProxyModel.cpp.

```
148 {
149     if ( mColSrcToProxyMap.isEmpty() )
150     { // no column mapping set up yet, all columns are passed down:
151         return true;
152     } else {
153         Q_ASSERT ( sourceModel() );
154         Q_ASSERT ( mColSrcToProxyMap.size() == sourceModel()->columnCount(mRootIndex) );
155         if ( mColSrcToProxyMap[sourceColumn] == -1 )
156         { // this column is explicitly not accepted:
157             return false;
158         } else {
159             Q_ASSERT ( mColSrcToProxyMap[sourceColumn] >= 0
160                     && mColSrcToProxyMap[sourceColumn] < mColSrcToProxyMap.size() );
161             return true;
162         }
163     }
164 }
```

#### 9.24.3.4 **bool DatasetProxyModel::filterAcceptsRow (int *source\_row*, const QModelIndex & *source\_parent*) const** [protected]

Decide whether the row is accepted.

Definition at line 126 of file KDChartDatasetProxyModel.cpp.

```

128 {
129     if ( mRowSrcToProxyMap.isEmpty() )
130     { // no row mapping set, all rows are passed down:
131         return true;
132     } else {
133         Q_ASSERT ( sourceModel() );
134         Q_ASSERT ( mRowSrcToProxyMap.size() == sourceModel()->rowCount (mRootIndex) );
135         if ( mRowSrcToProxyMap[sourceRow] == -1 )
136         { // this row is explicitly not accepted:
137             return false;
138         } else {
139             Q_ASSERT ( mRowSrcToProxyMap[sourceRow] >= 0
140                     && mRowSrcToProxyMap[sourceRow] < mRowSrcToProxyMap.size() );
141             return true;
142         }
143     }
144 }
```

#### 9.24.3.5 **Qt::ItemFlags DatasetProxyModel::flags (const QModelIndex & *index*) const**

Definition at line 45 of file KDChartDatasetProxyModel.cpp.

References mapToSource().

```

46 {
47     return sourceModel()->flags ( mapToSource ( index ) );
48 }
```

#### 9.24.3.6 **QVariant DatasetProxyModel::headerData (int *section*, Qt::Orientation *orientation*, int *role* = Qt::DisplayRole) const**

Overloaded from base class.

Definition at line 229 of file KDChartDatasetProxyModel.cpp.

```

230 {
231     if ( orientation == Qt::Horizontal )
232     {
233         if ( mapProxyColumnToSource ( section ) == -1 )
234         {
235             return QVariant();
236         } else {
237             return sourceModel()->headerData ( mapProxyColumnToSource ( section ),
238                                             orientation, role );
239         }
240     } else {
241         if ( mapProxyRowToSource ( section ) == -1 )
242         {
243             return QVariant();
244         } else {
245             return sourceModel()->headerData ( mapProxyRowToSource ( section ),
246                                             orientation, role );
247         }
248     }
249 }
```

```

247     }
248 }
249 }

```

#### 9.24.3.7 QModelIndex DatasetProxyModel::index (int row, int column, const QModelIndex & parent = QModelIndex()) const

Definition at line 78 of file KDChartDatasetProxyModel.cpp.

References `mapFromSource()`.

```

80 {
81     return mapFromSource( sourceModel()->index( mapProxyRowToSource(row),
82                                                mapProxyColumnToSource(column),
83                                                parent ) );
84 }

```

#### 9.24.3.8 QModelIndex DatasetProxyModel::mapFromSource (const QModelIndex & sourceIndex) const

Implements the mapping from the source to the proxy indexes.

Definition at line 92 of file KDChartDatasetProxyModel.cpp.

Referenced by `index()`, and `parent()`.

```

93 {
94     Q_ASSERT_X ( sourceModel(), "DatasetProxyModel::mapFromSource", "A source "
95                "model must be set before the selection can be configured." );
96
97     if ( !sourceIndex.isValid() ) return sourceIndex;
98
99     if ( mRowSrcToProxyMap.isEmpty() && mColSrcToProxyMap.isEmpty() )
100     {
101         return createIndex ( sourceIndex.row(), sourceIndex.column(),
102                             sourceIndex.internalPointer() );
103     } else {
104         int row = mapSourceRowToProxy ( sourceIndex.row() );
105         int column = mapSourceColumnToProxy ( sourceIndex.column() );
106         return createIndex ( row, column, sourceIndex.internalPointer() );
107     }
108 }

```

#### 9.24.3.9 QModelIndex DatasetProxyModel::mapToSource (const QModelIndex & proxyIndex) const

Implements the mapping from the proxy to the source indexes.

Definition at line 110 of file KDChartDatasetProxyModel.cpp.

Referenced by `data()`, `flags()`, `parent()`, and `setData()`.

```

111 {
112     Q_ASSERT_X ( sourceModel(), "DatasetProxyModel::mapToSource", "A source "
113                "model must be set before the selection can be configured." );
114
115     if ( !proxyIndex.isValid() ) return proxyIndex;

```

```

116     if ( mRowSrcToProxyMap.isEmpty() && mColSrcToProxyMap.isEmpty() )
117     {
118         return sourceModel()->index( proxyIndex.row(), proxyIndex.column(), mRootIndex );
119     } else {
120         int row = mapProxyRowToSource ( proxyIndex.row() );
121         int column = mapProxyColumnToSource ( proxyIndex.column() );
122         return sourceModel()->index( row, column, mRootIndex );
123     }
124 }

```

#### 9.24.3.10 QModelIndex DatasetProxyModel::parent (const QModelIndex & *child*) const

Definition at line 86 of file KDChartDatasetProxyModel.cpp.

References `mapFromSource()`, and `mapToSource()`.

```

87 {
88 //     return mapFromSource( sourceModel()->parent( child ) );
89     return mapFromSource( sourceModel()->parent( mapToSource( child ) ) );
90 }

```

#### 9.24.3.11 void DatasetProxyModel::resetDatasetDescriptions () [slot]

Reset all dataset description.

After that, the result of the proxying is an empty model (a new dataset description needs to be set to achieve a non-empty result).

Definition at line 210 of file KDChartDatasetProxyModel.cpp.

Referenced by `setSourceModel()`, and `setSourceRootIndex()`.

```

211 {
212     mRowSrcToProxyMap.clear();
213     mRowProxyToSrcMap.clear();
214     mColSrcToProxyMap.clear();
215     mColProxyToSrcMap.clear();
216     clear();
217 }

```

#### 9.24.3.12 bool DatasetProxyModel::setData (const QModelIndex & *index*, const QVariant & *value*, int *role*)

Overloaded from base class.

Definition at line 224 of file KDChartDatasetProxyModel.cpp.

References `mapToSource()`.

```

225 {
226     return sourceModel()->setData( mapToSource( index ), value, role );
227 }

```

### 9.24.3.13 void DatasetProxyModel::setDatasetColumnDescriptionVector (const DatasetDescriptionVector & columnConfig) [slot]

Configure the dataset selection for the columns.

Every call to this method resets the previous dataset description.

Definition at line 60 of file KDChartDatasetProxyModel.cpp.

Referenced by setDatasetDescriptionVectors().

```
62 {
63     Q_ASSERT_X ( sourceModel(), "DatasetProxyModel::setDatasetColumnDescriptionVector",
64                 "A source model must be set before the selection can be configured." );
65     initializeDatasetDecriptors ( configuration, sourceModel()->columnCount(mRootIndex),
66                                 mColSrcToProxyMap, mColProxyToSrcMap );
67     clear(); // clear emits layoutChanged()
68 }
```

### 9.24.3.14 void DatasetProxyModel::setDatasetDescriptionVectors (const DatasetDescriptionVector & rowConfig, const DatasetDescriptionVector & columnConfig) [slot]

Convenience method to configure rows and columns in one step.

Definition at line 70 of file KDChartDatasetProxyModel.cpp.

References setDatasetColumnDescriptionVector(), and setDatasetRowDescriptionVector().

```
73 {
74     setDatasetRowDescriptionVector( rowConfig );
75     setDatasetColumnDescriptionVector ( columnConfig );
76 }
```

### 9.24.3.15 void DatasetProxyModel::setDatasetRowDescriptionVector (const DatasetDescriptionVector & rowConfig) [slot]

Configure the dataset selection for the rows.

Every call to this method resets the previous dataset description.

Definition at line 50 of file KDChartDatasetProxyModel.cpp.

Referenced by setDatasetDescriptionVectors().

```
52 {
53     Q_ASSERT_X ( sourceModel(), "DatasetProxyModel::setDatasetRowDescriptionVector",
54                 "A source model must be set before the selection can be configured." );
55     initializeDatasetDecriptors ( configuration, sourceModel()->rowCount(mRootIndex),
56                                 mRowSrcToProxyMap, mRowProxyToSrcMap );
57     clear(); // clear emits layoutChanged()
58 }
```

### 9.24.3.16 void DatasetProxyModel::setSourceModel (QAbstractItemModel \* sourceModel)

Overloaded from base class.

Definition at line 276 of file KDChartDatasetProxyModel.cpp.

References resetDatasetDescriptions().

```
277 {  
278     QSortFilterProxyModel::setSourceModel ( sourceModel );  
279     mRootIndex = QModelIndex();  
280     connect ( sourceModel, SIGNAL ( layoutChanged() ),  
281             SLOT( resetDatasetDescriptions() ) );  
282  
283     resetDatasetDescriptions();  
284 }
```

#### 9.24.3.17 void DatasetProxyModel::setSourceRootIndex (const QModelIndex & rootIdx)

Set the root index of the table in the source model.

Definition at line 286 of file KDChartDatasetProxyModel.cpp.

References `resetDatasetDescriptions()`.

```
287 {  
288     mRootIndex = rootIdx;  
289     resetDatasetDescriptions();  
290 }
```

The documentation for this class was generated from the following files:

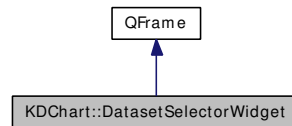
- [KDChartDatasetProxyModel.h](#)
- [KDChartDatasetProxyModel.cpp](#)



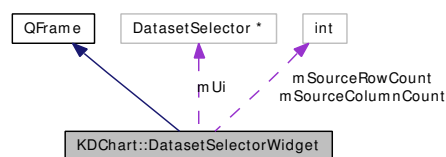
## 9.25 KDChart::DatasetSelectorWidget Class Reference

```
#include <KDChartDatasetSelector.h>
```

Inheritance diagram for KDChart::DatasetSelectorWidget:



Collaboration diagram for KDChart::DatasetSelectorWidget:



### 9.25.1 Detailed Description

Definition at line 48 of file KDChartDatasetSelector.h.

#### Public Slots

- void [setSourceColumnCount](#) (const int &columnCount)
- void [setSourceRowCount](#) (const int &rowCount)

#### Signals

- void [configureDatasetProxyModel](#) (const [DatasetDescriptionVector](#) &rowConfig, const [DatasetDescriptionVector](#) &columnConfig)
- void [mappingDisabled](#) ()

#### Public Member Functions

- [DatasetSelectorWidget](#) ([QWidget](#) \*parent=0)

### 9.25.2 Constructor & Destructor Documentation

#### 9.25.2.1 DatasetSelectorWidget::DatasetSelectorWidget ([QWidget](#) \*parent = 0) [explicit]

Definition at line 36 of file KDChartDatasetSelector.cpp.

```

37     : QFrame ( parent )
38     , mUi ( new Ui::DatasetSelector () )
39     , mSourceRowCount ( 0 )

```

```
40     , mSourceColumnCount ( 0 )
41 {
42     qWarning("For DatasetSelectorWidget to become useful, it has to be connected to the proxy model it
43
44     mUi->setupUi ( this );
45     setMinimumSize ( minimumSizeHint() );
46 }
```

### 9.25.3 Member Function Documentation

**9.25.3.1 void KDChart::DatasetSelectorWidget::configureDatasetProxyModel (const [DatasetDescriptionVector](#) & *rowConfig*, const [DatasetDescriptionVector](#) & *columnConfig*)** [signal]

**9.25.3.2 void KDChart::DatasetSelectorWidget::mappingDisabled ()** [signal]

**9.25.3.3 void DatasetSelectorWidget::setSourceColumnCount (const int & *columnCount*)** [slot]

Definition at line 98 of file KDChartDatasetSelector.cpp.

```
99 {
100     if ( columnCount != mSourceColumnCount )
101     {
102         mSourceColumnCount = columnCount;
103         resetDisplayValues();
104     }
105 }
```

**9.25.3.4 void DatasetSelectorWidget::setSourceRowCount (const int & *rowCount*)** [slot]

Definition at line 89 of file KDChartDatasetSelector.cpp.

```
90 {
91     if ( rowCount != mSourceRowCount )
92     {
93         mSourceRowCount = rowCount;
94         resetDisplayValues();
95     }
96 }
```

The documentation for this class was generated from the following files:

- [KDChartDatasetSelector.h](#)
- [KDChartDatasetSelector.cpp](#)

## 9.26 KDChart::DataValueAttributes Class Reference

```
#include <KDChartDataValueAttributes>
```

### 9.26.1 Detailed Description

Diagram attributes dealing with data value labels.

The [DataValueAttributes](#) group all properties that can be set wrt data value labels and if and how they are displayed. This includes things like the text attributes (font, color), what markers are used, howmany decimal digits are displayed, etc.

Definition at line 58 of file KDChartDataValueAttributes.h.

### Public Member Functions

- [BackgroundAttributes](#) [backgroundAttributes](#) () const  
*Returns:*  
*The background attributes used for painting the data value labels area.*
- [QString](#) [dataLabel](#) () const  
*Returns the string displayed instead of the data value label.*
- [DataValueAttributes](#) (const [DataValueAttributes](#) &)
- [DataValueAttributes](#) ()
- [int](#) [decimalDigits](#) () const  
*Returns:*  
*The number of decimal digits displayed.*
- [FrameAttributes](#) [frameAttributes](#) () const  
*Returns:*  
*The frame attributes used for painting the data value labels area.*
- [bool](#) [isVisible](#) () const  
*Returns:*  
*Whether data value labels should be displayed.*
- [MarkerAttributes](#) [markerAttributes](#) () const  
*Returns:*  
*The marker attributes used for decorating the data values.*
- [const](#) [RelativePosition](#) [negativePosition](#) () const  
*Return the relative positioning of the data value labels.*
- [bool](#) [operator!=](#) (const [DataValueAttributes](#) &other) const
- [DataValueAttributes](#) & [operator=](#) (const [DataValueAttributes](#) &)
- [bool](#) [operator==](#) (const [DataValueAttributes](#) &) const
- [const](#) [RelativePosition](#) [position](#) (bool positive) const
- [const](#) [RelativePosition](#) [positivePosition](#) () const

*Return the relative positioning of the data value labels.*

- `QString prefix () const`  
*Returns the string used as a prefix to the data value text.*
- `void setBackgroundAttributes (const BackgroundAttributes &a)`  
*Set the background attributes to use for the data value labels area.*
- `void setDataLabel (const QString label)`  
*display a string label instead of the original data value label Supports HTML code.*
- `void setDecimalDigits (int digits)`  
*Set how many decimal digits to display when rendering the data value labels.*
- `void setFrameAttributes (const FrameAttributes &a)`  
*Set the frame attributes to use for the data value labels area.*
- `void setMarkerAttributes (const MarkerAttributes &a)`  
*Set the marker attributes to use for the data values.*
- `void setNegativePosition (const RelativePosition &relPosition)`  
*Defines the relative positioning of the data value labels for negative values.*
- `void setPositivePosition (const RelativePosition &relPosition)`  
*Defines the relative position of the data value labels for positive values.*
- `void setPrefix (const QString prefix)`  
*Prepend a prefix string to the data value label.*
- `void setSuffix (const QString suffix)`  
*Append a suffix string to the data value label.*
- `void setTextAttributes (const TextAttributes &a)`  
*Set the text attributes to use for the data value labels.*
- `void setVisible (bool visible)`  
*Set whether data value labels should be displayed.*
- `QString suffix () const`  
*Returns the string used as a suffix to the data value text.*
- `TextAttributes textAttributes () const`  
*Returns:*  
*The text attributes used for painting data value labels.*
- `~DataValueAttributes ()`

## Static Public Member Functions

- `static const DataValueAttributes & defaultAttributes ()`
- `static const QVariant & defaultAttributesAsVariant ()`

## 9.26.2 Constructor & Destructor Documentation

### 9.26.2.1 DataValueAttributes::DataValueAttributes ()

Definition at line 106 of file KDChartDataValueAttributes.cpp.

```
107      : _d( new Private() )
108  {
109  }
```

### 9.26.2.2 DataValueAttributes::DataValueAttributes (const [DataValueAttributes](#) &)

Definition at line 111 of file KDChartDataValueAttributes.cpp.

```
112      : _d( new Private( *r.d ) )
113  {
114  }
```

### 9.26.2.3 DataValueAttributes::~DataValueAttributes ()

Definition at line 126 of file KDChartDataValueAttributes.cpp.

```
127  {
128      delete _d; _d = 0;
129  }
```

## 9.26.3 Member Function Documentation

### 9.26.3.1 [BackgroundAttributes](#) DataValueAttributes::backgroundAttributes () const

#### Returns:

The background attributes used for painting the data value labels area.

#### See also:

[BackgroundAttributes](#)

Definition at line 217 of file KDChartDataValueAttributes.cpp.

References [d](#).

Referenced by [operator<<\(\)](#), and [operator==\(\)](#).

```
218  {
219      return d->backgroundAttributes;
220  }
```

### 9.26.3.2 QString DataValueAttributes::dataLabel () const

Returns the string displayed instead of the data value label.

**See also:**

[setDataLabel](#)

Definition at line 268 of file KDChartDataValueAttributes.cpp.

References [d](#).

Referenced by [operator==\(\(\)\)](#), and [KDChart::AbstractDiagram::paintDataValueText\(\)](#).

```
269 {
270     return d->dataLabel;
271 }
```

### 9.26.3.3 int DataValueAttributes::decimalDigits () const

**Returns:**

The number of decimal digits displayed.

Definition at line 238 of file KDChartDataValueAttributes.cpp.

References [d](#).

Referenced by [operator<<\(\)](#), [operator==\(\(\)\)](#), and [KDChart::AbstractDiagram::paintDataValueText\(\)](#).

```
239 {
240     return d->decimalDigits;
241 }
```

### 9.26.3.4 const [DataValueAttributes](#) & DataValueAttributes::defaultAttributes () [static]

Definition at line 168 of file KDChartDataValueAttributes.cpp.

Referenced by [defaultAttributesAsVariant\(\)](#).

```
169 {
170     static const DataValueAttributes theDefaultDataValueAttributes;
171     return theDefaultDataValueAttributes;
172 }
```

### 9.26.3.5 const QVariant & DataValueAttributes::defaultAttributesAsVariant () [static]

Definition at line 175 of file KDChartDataValueAttributes.cpp.

References [defaultAttributes\(\)](#).

Referenced by [KDChart::AttributesModel::AttributesModel\(\)](#).

```
176 {
177     static const QVariant theDefaultDataValueAttributesVariant = qVariantFromValue(defaultAttributes());
178     return theDefaultDataValueAttributesVariant;
179 }
```

### 9.26.3.6 [FrameAttributes](#) DataValueAttributes::frameAttributes () const

**Returns:**

The frame attributes used for painting the data value labels area.

**See also:**

[FrameAttributes](#)

Definition at line 207 of file KDChartDataValueAttributes.cpp.

References [d](#).

Referenced by `operator<<()`, and `operator==()`.

```
208 {  
209     return d->frameAttributes;  
210 }
```

### 9.26.3.7 `bool` DataValueAttributes::isVisible () const

**Returns:**

Whether data value labels should be displayed.

Definition at line 187 of file KDChartDataValueAttributes.cpp.

References [d](#).

Referenced by `operator<<()`, `operator==()`, `KDChart::AbstractDiagram::paintDataValueText()`, and `KDChart::AbstractDiagram::paintMarker()`.

```
188 {  
189     return d->visible;  
190 }
```

### 9.26.3.8 [MarkerAttributes](#) DataValueAttributes::markerAttributes () const

**Returns:**

The marker attributes used for decorating the data values.

**See also:**

[MarkerAttributes](#)

Definition at line 227 of file KDChartDataValueAttributes.cpp.

References [d](#).

Referenced by `operator==()`, and `KDChart::AbstractDiagram::paintMarker()`.

```
228 {  
229     return d->markerAttributes;  
230 }
```

### 9.26.3.9 const [RelativePosition](#) DataValueAttributes::negativePosition () const

Return the relative positioning of the data value labels.

See also:

[setNegativePosition](#)

Definition at line 308 of file KDChartDataValueAttributes.cpp.

References [d](#).

Referenced by [operator<<\(\)](#), and [operator==\(\)](#).

```
309 {
310     return d->negativeRelPos;
311 }
```

### 9.26.3.10 bool KDChart::DataValueAttributes::operator!= (const [DataValueAttributes](#) & *other*) const

Definition at line 65 of file KDChartDataValueAttributes.h.

```
65 { return !operator==(other); }
```

### 9.26.3.11 [DataValueAttributes](#) & DataValueAttributes::operator= (const [DataValueAttributes](#) &)

Definition at line 116 of file KDChartDataValueAttributes.cpp.

References [d](#).

```
117 {
118     if( this == &r )
119         return *this;
120
121     *d = *r.d;
122
123     return *this;
124 }
```

### 9.26.3.12 bool DataValueAttributes::operator== (const [DataValueAttributes](#) &) const

Definition at line 132 of file KDChartDataValueAttributes.cpp.

References [backgroundAttributes\(\)](#), [dataLabel\(\)](#), [decimalDigits\(\)](#), [frameAttributes\(\)](#), [isVisible\(\)](#), [markerAttributes\(\)](#), [negativePosition\(\)](#), [positivePosition\(\)](#), [prefix\(\)](#), [suffix\(\)](#), and [textAttributes\(\)](#).

```
133 {
134     /*
135     qDebug() << "DataValueAttributes::operator== finds"
136         << "b" << (isVisible() == r.isVisible())
137         << "c" << (textAttributes() == r.textAttributes())
138         << "d" << (frameAttributes() == r.frameAttributes())
139         << "e" << (backgroundAttributes() == r.backgroundAttributes())
140         << "f" << (markerAttributes() == r.markerAttributes())
```



```

141         << "g" << (decimalDigits() == r.decimalDigits())
142         << "h" << (prefix() == r.prefix())
143         << "i" << (suffix() == r.suffix())
144         << "j" << (dataLabel() == r.dataLabel())
145         << "k" << (powerOfTenDivisor() == r.powerOfTenDivisor())
146         << "l" << (showInfinite() == r.showInfinite())
147         << "m" << (negativePosition() == r.negativePosition())
148         << "n" << (positivePosition() == r.positivePosition())
149         << "o" << (showRepetitiveDataLabels() == r.showRepetitiveDataLabels());
150     */
151     return ( isVisible() == r.isVisible() &&
152             textAttributes() == r.textAttributes() &&
153             frameAttributes() == r.frameAttributes() &&
154             backgroundAttributes() == r.backgroundAttributes() &&
155             markerAttributes() == r.markerAttributes() &&
156             decimalDigits() == r.decimalDigits() &&
157             prefix() == r.prefix() &&
158             suffix() == r.suffix() &&
159             dataLabel() == r.dataLabel() &&
160             powerOfTenDivisor() == r.powerOfTenDivisor() &&
161             showInfinite() == r.showInfinite() &&
162             negativePosition() == r.negativePosition() &&
163             positivePosition() == r.positivePosition() &&
164             showRepetitiveDataLabels() == r.showRepetitiveDataLabels() );
165 }

```

### 9.26.3.13 const [RelativePosition](#) KDChart::DataValueAttributes::position (bool *positive*) const

Definition at line 259 of file KDChartDataValueAttributes.h.

Referenced by KDChart::AbstractDiagram::paintDataValueText().

```

260 {
261     return positive ? positivePosition() : negativePosition();
262 }

```

### 9.26.3.14 const [RelativePosition](#) DataValueAttributes::positivePosition () const

Return the relative positioning of the data value labels.

See also:

[setPositivePosition](#)

Definition at line 318 of file KDChartDataValueAttributes.cpp.

References d.

Referenced by operator<<(), and operator==().

```

319 {
320     return d->positiveRelPos;
321 }

```

### 9.26.3.15 QString DataValueAttributes::prefix () const

Returns the string used as a prefix to the data value text.

**See also:**

[setPrefix](#)

Definition at line 248 of file KDChartDataValueAttributes.cpp.

References d.

Referenced by operator==( ), and KDChart::AbstractDiagram::paintDataValueText( ).

```
249 {  
250     return d->prefix;  
251 }
```

#### 9.26.3.16 void DataValueAttributes::setBackgroundAttributes (const [BackgroundAttributes](#) & a)

Set the background attributes to use for the data value labels area.

**Parameters:**

*a* The background attributes to set.

**See also:**

[BackgroundAttributes](#)

Definition at line 212 of file KDChartDataValueAttributes.cpp.

References d.

```
213 {  
214     d->backgroundAttributes = a;  
215 }
```

#### 9.26.3.17 void DataValueAttributes::setDataLabel (const QString *label*)

display a string label instead of the original data value label Supports HTML code.

**See also:**

[dataLabel](#)

Definition at line 263 of file KDChartDataValueAttributes.cpp.

References d.

```
264 {  
265     d->dataLabel = label;  
266 }
```

### 9.26.3.18 void DataValueAttributes::setDecimalDigits (int *digits*)

Set how many decimal digits to display when rendering the data value labels.

If there are no decimal digits it will not be displayed.

#### Parameters:

*digits* The number of decimal digits to use.

Definition at line 233 of file KDChartDataValueAttributes.cpp.

References [d](#).

```
234 {  
235     d->decimalDigits = digits;  
236 }
```

### 9.26.3.19 void DataValueAttributes::setFrameAttributes (const [FrameAttributes](#) & *a*)

Set the frame attributes to use for the data value labels area.

#### Parameters:

*a* The frame attributes to set.

#### See also:

[FrameAttributes](#)

Definition at line 202 of file KDChartDataValueAttributes.cpp.

References [d](#).

```
203 {  
204     d->frameAttributes = a;  
205 }
```

### 9.26.3.20 void DataValueAttributes::setMarkerAttributes (const [MarkerAttributes](#) & *a*)

Set the marker attributes to use for the data values.

This includes the marker type.

#### Parameters:

*a* The marker attributes to set.

#### See also:

[MarkerAttributes](#)

Definition at line 222 of file KDChartDataValueAttributes.cpp.

References [d](#).

Referenced by [KDChart::TernaryLineDiagram::TernaryLineDiagram\(\)](#).

```
223 {  
224     d->markerAttributes = a;  
225 }
```

**9.26.3.21 void DataValueAttributes::setNegativePosition (const [RelativePosition](#) & *relPosition*)**

Defines the relative positioning of the data value labels for negative values.

The position is specified in relation to the respective data value point, or in relation to the respective data representation area, that's one area segment in a [LineDiagram](#) showing areas, or one bar in a [BarDiagram](#), one pie slice ...

**See also:**

[negativePosition](#)

Definition at line 303 of file KDChartDataValueAttributes.cpp.

References d.

```
304 {  
305     d->negativeRelPos = relPosition;  
306 }
```

**9.26.3.22 void DataValueAttributes::setPositivePosition (const [RelativePosition](#) & *relPosition*)**

Defines the relative position of the data value labels for positive values.

The position is specified in relation to the respective data value point, or in relation to the respective data representation area, that's one area segment in a [LineDiagram](#) showing areas, or one bar in a [BarDiagram](#), one pie slice ...

**See also:**

[positivePosition](#)

Definition at line 313 of file KDChartDataValueAttributes.cpp.

References d.

```
314 {  
315     d->positiveRelPos = relPosition;  
316 }
```

**9.26.3.23 void DataValueAttributes::setPrefix (const QString *prefix*)**

Prepend a prefix string to the data value label.

**See also:**

[prefix](#)

Definition at line 243 of file KDChartDataValueAttributes.cpp.

References d.

```
244 {  
245     d->prefix = prefixString;  
246 }
```

### 9.26.3.24 void DataValueAttributes::setSuffix (const QString *suffix*)

Append a suffix string to the data value label.

**See also:**

[suffix](#)

Definition at line 253 of file KDChartDataValueAttributes.cpp.

References [d](#).

```
254 {  
255     d->suffix = suffixString;  
256 }
```

### 9.26.3.25 void DataValueAttributes::setTextAttributes (const [TextAttributes](#) & *a*)

Set the text attributes to use for the data value labels.

**Parameters:**

*a* The text attributes to set.

**See also:**

[TextAttributes](#)

Definition at line 192 of file KDChartDataValueAttributes.cpp.

References [d](#).

```
193 {  
194     d->textAttributes = a;  
195 }
```

### 9.26.3.26 void DataValueAttributes::setVisible (bool *visible*)

Set whether data value labels should be displayed.

**Parameters:**

*visible* Whether data value labels should be displayed.

Definition at line 182 of file KDChartDataValueAttributes.cpp.

References [d](#).

Referenced by [KDChart::TernaryLineDiagram::TernaryLineDiagram\(\)](#).

```
183 {  
184     d->visible = visible;  
185 }
```

### 9.26.3.27 `QString DataValueAttributes::suffix () const`

Returns the string used as a suffix to the data value text.

**See also:**

[setSuffix](#)

Definition at line 258 of file `KDChartDataValueAttributes.cpp`.

References `d`.

Referenced by `operator==()`, and `KDChart::AbstractDiagram::paintDataValueText()`.

```
259 {  
260     return d->suffix;  
261 }
```

### 9.26.3.28 `TextAttributes DataValueAttributes::textAttributes () const`

**Returns:**

The text attributes used for painting data value labels.

Definition at line 197 of file `KDChartDataValueAttributes.cpp`.

References `d`.

Referenced by `operator<<()`, `operator==()`, and `KDChart::AbstractDiagram::paintDataValueText()`.

```
198 {  
199     return d->textAttributes;  
200 }
```

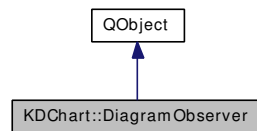
The documentation for this class was generated from the following files:

- [KDChartDataValueAttributes.h](#)
- [KDChartDataValueAttributes.cpp](#)

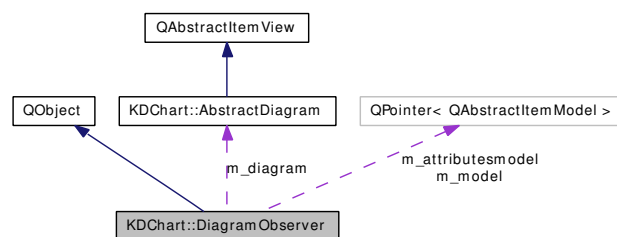
## 9.27 KDCart::DiagramObserver Class Reference

```
#include <KDCartDiagramObserver.h>
```

Inheritance diagram for KDCart::DiagramObserver:



Collaboration diagram for KDCart::DiagramObserver:



### 9.27.1 Detailed Description

A [DiagramObserver](#) watches the associated diagram for changes and deletion and emits corresponding signals.

Definition at line 44 of file KDCartDiagramObserver.h.

### Signals

- void [diagramAttributesChanged](#) ([AbstractDiagram](#) \*diagram)  
*This signal is emitted whenever the attributes of the diagram change.*
- void [diagramDataChanged](#) ([AbstractDiagram](#) \*diagram)  
*This signal is emitted whenever the data of the diagram changes.*
- void [diagramDataHidden](#) ([AbstractDiagram](#) \*diagram)  
*This signal is emitted whenever any of the data of the diagram was set (un)hidden.*
- void [diagramDestroyed](#) ([AbstractDiagram](#) \*diagram)  
*This signal is emitted immediately before the diagram is being destroyed.*

### Public Member Functions

- [AbstractDiagram](#) \* [diagram](#) ()
- const [AbstractDiagram](#) \* [diagram](#) () const
- [DiagramObserver](#) ([AbstractDiagram](#) \*diagram, [QObject](#) \*parent=0)

*Constructs a new observer observing the given diagram.*

- [~DiagramObserver\(\)](#)

## 9.27.2 Constructor & Destructor Documentation

### 9.27.2.1 [DiagramObserver::DiagramObserver](#) ([AbstractDiagram](#) \* *diagram*, [QObject](#) \* *parent* = 0) [explicit]

Constructs a new observer observing the given diagram.

Definition at line 40 of file `KDChartDiagramObserver.cpp`.

```
41      : QObject( parent ), m_diagram( diagram )
42  {
43      if ( m_diagram ) {
44          connect( m_diagram, SIGNAL(destroyed(QObject*)), SLOT(slotDestroyed(QObject*)));
45          connect( m_diagram, SIGNAL(modelsChanged()), SLOT(slotModelsChanged()));
46      }
47      init();
48  }
```

### 9.27.2.2 [DiagramObserver::~~DiagramObserver\(\)](#)

Definition at line 50 of file `KDChartDiagramObserver.cpp`.

```
51 {
52 }
```

## 9.27.3 Member Function Documentation

### 9.27.3.1 [AbstractDiagram](#) \* [DiagramObserver::diagram\(\)](#)

Definition at line 59 of file `KDChartDiagramObserver.cpp`.

```
60 {
61     return m_diagram;
62 }
```

### 9.27.3.2 `const` [AbstractDiagram](#) \* [DiagramObserver::diagram\(\)](#) `const`

Definition at line 54 of file `KDChartDiagramObserver.cpp`.

```
55 {
56     return m_diagram;
57 }
```

### 9.27.3.3 `void` [KDChart::DiagramObserver::diagramAttributesChanged](#) ([AbstractDiagram](#) \* *diagram*) [signal]

This signal is emitted whenever the attributes of the diagram change.



**9.27.3.4 void KDCart::DiagramObserver::diagramDataChanged ([AbstractDiagram](#) \* *diagram*)**  
[signal]

This signal is emitted whenever the data of the diagram changes.

**9.27.3.5 void KDCart::DiagramObserver::diagramDataHidden ([AbstractDiagram](#) \* *diagram*)**  
[signal]

This signal is emitted whenever any of the data of the diagram was set (un)hidden.

**9.27.3.6 void KDCart::DiagramObserver::diagramDestroyed ([AbstractDiagram](#) \* *diagram*)**  
[signal]

This signal is emitted immediately before the diagram is being destroyed.

The documentation for this class was generated from the following files:

- [KDCartDiagramObserver.h](#)
- [KDCartDiagramObserver.cpp](#)

## 9.28 KDCart::FrameAttributes Class Reference

```
#include <KDCartFrameAttributes.h>
```

### 9.28.1 Detailed Description

A set of attributes for frames around items.

Definition at line 43 of file KDCartFrameAttributes.h.

### Public Member Functions

- [FrameAttributes](#) (const [FrameAttributes](#) &)
- [FrameAttributes](#) ()
- bool [isVisible](#) () const
- bool [operator!=](#) (const [FrameAttributes](#) &other) const
- [FrameAttributes](#) & [operator=](#) (const [FrameAttributes](#) &)
- bool [operator==](#) (const [FrameAttributes](#) &) const
- int [padding](#) () const
- QPen [pen](#) () const
- void [setPadding](#) (int padding)
- void [setPen](#) (const QPen &pen)
- void [setVisible](#) (bool visible)
- [~FrameAttributes](#) ()

### 9.28.2 Constructor & Destructor Documentation

#### 9.28.2.1 FrameAttributes::FrameAttributes ()

Definition at line 52 of file KDCartFrameAttributes.cpp.

```
53      : _d( new Private() )
54  {
55  }
```

#### 9.28.2.2 FrameAttributes::FrameAttributes (const [FrameAttributes](#) &)

Definition at line 57 of file KDCartFrameAttributes.cpp.

```
58      : _d( new Private( *r.d ) )
59  {
60  }
```

#### 9.28.2.3 FrameAttributes::~~FrameAttributes ()

Definition at line 72 of file KDCartFrameAttributes.cpp.

```
73  {
74      delete _d; _d = 0;
75  }
```

### 9.28.3 Member Function Documentation

#### 9.28.3.1 bool FrameAttributes::isVisible () const

Definition at line 98 of file KDCartFrameAttributes.cpp.

References `d`.

Referenced by `operator<<()`, `operator==()`, `KDCart::AbstractAreaBase::paintFrameAttributes()`, and `updateCommonBrush()`.

```
99 {  
100     return d->visible;  
101 }
```

#### 9.28.3.2 bool KDCart::FrameAttributes::operator!=(const FrameAttributes & other) const

Definition at line 62 of file KDCartFrameAttributes.h.

```
62 { return !operator==(other); }
```

#### 9.28.3.3 FrameAttributes & FrameAttributes::operator= (const FrameAttributes &)

Definition at line 62 of file KDCartFrameAttributes.cpp.

References `d`.

```
63 {  
64     if( this == &r )  
65         return *this;  
66  
67     *d = *r.d;  
68  
69     return *this;  
70 }
```

#### 9.28.3.4 bool FrameAttributes::operator==(const FrameAttributes &) const

Definition at line 78 of file KDCartFrameAttributes.cpp.

References `isVisible()`, `padding()`, and `pen()`.

```
79 {  
80     /*  
81     qDebug() << "FrameAttributes:" << (isVisible() == r.isVisible())  
82         << (pen() == r.pen())  
83         << (padding() == r.padding()) << "\n";  
84     */  
85     return ( isVisible() == r.isVisible() &&  
86         pen() == r.pen() &&  
87         padding() == r.padding() );  
88 }
```

### 9.28.3.5 int FrameAttributes::padding () const

Definition at line 118 of file KDChartFrameAttributes.cpp.

References d.

Referenced by operator<<(), and operator==().

```
119 {  
120     return d->padding;  
121 }
```

### 9.28.3.6 QPen FrameAttributes::pen () const

Definition at line 108 of file KDChartFrameAttributes.cpp.

References d.

Referenced by operator<<(), operator==(), and KDChart::AbstractAreaBase::paintFrameAttributes().

```
109 {  
110     return d->pen;  
111 }
```

### 9.28.3.7 void FrameAttributes::setPadding (int *padding*)

Definition at line 113 of file KDChartFrameAttributes.cpp.

References d.

Referenced by KDChart::Chart::Chart().

```
114 {  
115     d->padding = padding;  
116 }
```

### 9.28.3.8 void FrameAttributes::setPen (const QPen & *pen*)

Definition at line 103 of file KDChartFrameAttributes.cpp.

References d.

Referenced by KDChart::Chart::Chart().

```
104 {  
105     d->pen = pen;  
106 }
```

### 9.28.3.9 void FrameAttributes::setVisible (bool *visible*)

Definition at line 93 of file KDChartFrameAttributes.cpp.

References d.

Referenced by KDChart::Chart::Chart().

```
94 {  
95     d->visible = visible;  
96 }
```

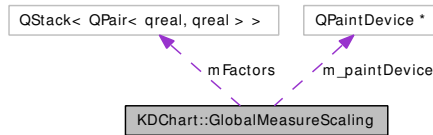
The documentation for this class was generated from the following files:

- [KDChartFrameAttributes.h](#)
- [KDChartFrameAttributes.cpp](#)

## 9.29 KDChart::GlobalMeasureScaling Class Reference

```
#include <KDChartMeasure.h>
```

Collaboration diagram for KDChart::GlobalMeasureScaling:



### 9.29.1 Detailed Description

Auxiliary class used by the [KDChart::Measure](#) and [KDChart::Chart](#) class.

Normally there should be no need to call any of these methods yourself.

They are used by [KDChart::Chart::paint\( QPainter\\*, const QRect& \)](#) to adjust all of the relative Measures according to the target rectangle's size.

Default factors are (1.0, 1.0)

Definition at line 149 of file [KDChartMeasure.h](#).

### Public Member Functions

- [GlobalMeasureScaling](#) ()
- virtual [~GlobalMeasureScaling](#) ()

### Static Public Member Functions

- static const [QPair< qreal, qreal >](#) [currentFactors](#) ()  
*Returns the currently active factors.*
- static [GlobalMeasureScaling \\*](#) [instance](#) ()
- static [QPaintDevice \\*](#) [paintDevice](#) ()  
*Returns the paint device usable for calculating fort metrics.*
- static void [resetFactors](#) ()  
*Reset factors to the values active before the previous call of setFactors.*
- static void [setFactors](#) (qreal factorX, qreal factorY)  
*Set new factors to be used by all [Measure](#) objects from now on.*
- static void [setPaintDevice](#) ([QPaintDevice \\*](#)paintDevice)  
*Sets the paint device usable for calculating fort metrics.*

## 9.29.2 Constructor & Destructor Documentation

### 9.29.2.1 KDChart::GlobalMeasureScaling::GlobalMeasureScaling ()

Definition at line 187 of file KDChartMeasure.cpp.

```
188 {  
189     mFactors.push( qMakePair(1.0, 1.0) );  
190 }
```

### 9.29.2.2 KDChart::GlobalMeasureScaling::~~GlobalMeasureScaling () [virtual]

Definition at line 192 of file KDChartMeasure.cpp.

```
193 {  
194     // this space left empty intentionally  
195 }
```

## 9.29.3 Member Function Documentation

### 9.29.3.1 const QPair< qreal, qreal > KDChart::GlobalMeasureScaling::currentFactors () [static]

Returns the currently active factors.

Definition at line 215 of file KDChartMeasure.cpp.

References instance(), and mFactors.

Referenced by KDChart::Measure::sizeOfArea().

```
216 {  
217     return instance()->mFactors.top();  
218 }
```

### 9.29.3.2 GlobalMeasureScaling \* KDChart::GlobalMeasureScaling::instance () [static]

Definition at line 197 of file KDChartMeasure.cpp.

Referenced by currentFactors(), KDChart::Chart::paint(), paintDevice(), resetFactors(), setFactors(), setPaintDevice(), and KDChart::Measure::sizeOfArea().

```
198 {  
199     static GlobalMeasureScaling instance;  
200     return &instance;  
201 }
```

### 9.29.3.3 QPaintDevice \* KDChart::GlobalMeasureScaling::paintDevice () [static]

Returns the paint device usable for calculating fort metrics.

Definition at line 225 of file KDChartMeasure.cpp.

References instance(), and m\_paintDevice.

Referenced by KDChart::CartesianAxis::maximumSize(), and KDChart::CartesianAxis::paintCtx().

```
226 {  
227     return instance()->m_paintDevice;  
228 }
```

#### 9.29.3.4 void KDChart::GlobalMeasureScaling::resetFactors () [static]

Reset factors to the values active before the previous call of setFactors.

This works on a stack, so recursive calls works fine, like: setFactors, setFactors, unseFactors, unsetFactors

Definition at line 208 of file KDChartMeasure.cpp.

References instance(), and mFactors.

Referenced by KDChart::Chart::paint().

```
209 {  
210     // never remove the initial (1.0, 1.0) setting  
211     if( instance()->mFactors.count() > 1 )  
212         instance()->mFactors.pop();  
213 }
```

#### 9.29.3.5 void KDChart::GlobalMeasureScaling::setFactors (qreal factorX, qreal factorY) [static]

Set new factors to be used by all [Measure](#) objects from now on.

Previous values will be stored.

Definition at line 203 of file KDChartMeasure.cpp.

References instance(), and mFactors.

Referenced by KDChart::Chart::paint().

```
204 {  
205     instance()->mFactors.push( qMakePair(factorX, factorY) );  
206 }
```

#### 9.29.3.6 void KDChart::GlobalMeasureScaling::setPaintDevice (QPaintDevice \* paintDevice) [static]

Sets the paint device usable for calculating fort metrics.

Definition at line 220 of file KDChartMeasure.cpp.

References instance(), and m\_paintDevice.

Referenced by KDChart::Chart::paint().

```
221 {  
222     instance()->m_paintDevice = paintDevice;  
223 }
```

The documentation for this class was generated from the following files:

- [KDChartMeasure.h](#)
- [KDChartMeasure.cpp](#)



## 9.30 KDCart::GridAttributes Class Reference

```
#include <KDCartGridAttributes.h>
```

### 9.30.1 Detailed Description

A set of attributes controlling the appearance of grids.

Definition at line 44 of file KDCartGridAttributes.h.

### Public Member Functions

- bool [adjustLowerBoundToGrid](#) () const
- bool [adjustUpperBoundToGrid](#) () const
- [GridAttributes](#) (const [GridAttributes](#) &)
- [GridAttributes](#) ()
- [KDCartEnums::GranularitySequence](#) [gridGranularitySequence](#) () const  
*Returns the granularity sequence to be used for calculating the grid lines.*
- [QPen](#) [gridPen](#) () const
- [qreal](#) [gridStepWidth](#) () const  
*Returns the step width to be used for calculating the grid lines.*
- [qreal](#) [gridSubStepWidth](#) () const  
*Returns the sub-step width to be used for calculating the sub-grid lines.*
- bool [isGridVisible](#) () const
- bool [isSubGridVisible](#) () const
- bool [operator!=](#) (const [GridAttributes](#) &other) const
- [GridAttributes](#) & [operator=](#) (const [GridAttributes](#) &)
- bool [operator==](#) (const [GridAttributes](#) &) const
- void [setAdjustBoundsToGrid](#) (bool adjustLower, bool adjustUpper)  
*By default visible bounds of the data area are adjusted to match a main grid line.*
- void [setGridGranularitySequence](#) ([KDCartEnums::GranularitySequence](#) sequence)  
*Specify which granularity sequence is to be used to find a matching grid granularity.*
- void [setGridPen](#) (const [QPen](#) &pen)
- void [setGridStepWidth](#) ([qreal](#) stepWidth=0.0)  
*Specifies the step width to be used for calculating the grid lines.*
- void [setGridSubStepWidth](#) ([qreal](#) subStepWidth=0.0)  
*Specifies the sub-step width to be used for calculating the grid sub-lines.*
- void [setGridVisible](#) (bool visible)
- void [setSubGridPen](#) (const [QPen](#) &pen)
- void [setSubGridVisible](#) (bool visible)
- void [setZeroLinePen](#) (const [QPen](#) &pen)
- [QPen](#) [subGridPen](#) () const
- [QPen](#) [zeroLinePen](#) () const
- [~GridAttributes](#) ()

## 9.30.2 Constructor & Destructor Documentation

### 9.30.2.1 GridAttributes::GridAttributes ()

Definition at line 73 of file KDChartGridAttributes.cpp.

```
74      : _d( new Private() )
75  {
76      // this bloc left empty intentionally
77  }
```

### 9.30.2.2 GridAttributes::GridAttributes (const [GridAttributes](#) &)

Definition at line 79 of file KDChartGridAttributes.cpp.

```
80      : _d( new Private( *r.d ) )
81  {
82  }
```

### 9.30.2.3 GridAttributes::~~GridAttributes ()

Definition at line 94 of file KDChartGridAttributes.cpp.

```
95  {
96      delete _d; _d = 0;
97  }
```

## 9.30.3 Member Function Documentation

### 9.30.3.1 bool GridAttributes::adjustLowerBoundToGrid () const

Definition at line 226 of file KDChartGridAttributes.cpp.

References [d](#).

Referenced by [operator==\(.\)](#).

```
227  {
228      return d->adjustLower;
229  }
```

### 9.30.3.2 bool GridAttributes::adjustUpperBoundToGrid () const

Definition at line 230 of file KDChartGridAttributes.cpp.

References [d](#).

Referenced by [operator==\(.\)](#).

```
231  {
232      return d->adjustUpper;
233  }
```

### 9.30.3.3 [KDChartEnums::GranularitySequence](#) GridAttributes::gridGranularitySequence () const

Returns the granularity sequence to be used for calculating the grid lines.

**See also:**

[setGridGranularitySequence](#)

Definition at line 216 of file KDChartGridAttributes.cpp.

References [d](#).

Referenced by [KDChart::CartesianCoordinatePlane::getDataDimensionsList\(\)](#), and [operator==\(\(\)\)](#).

```
217 {  
218     return d->sequence;  
219 }
```

### 9.30.3.4 [QPen](#) GridAttributes::gridPen () const

Definition at line 241 of file KDChartGridAttributes.cpp.

References [d](#).

Referenced by [operator<<\(\)](#), and [operator==\(\(\)\)](#).

```
242 {  
243     return d->pen;  
244 }
```

### 9.30.3.5 [qreal](#) GridAttributes::gridStepWidth () const

Returns the step width to be used for calculating the grid lines.

**See also:**

[setGridStepWidth](#)

Definition at line 152 of file KDChartGridAttributes.cpp.

References [d](#).

Referenced by [KDChart::CartesianCoordinatePlane::getDataDimensionsList\(\)](#), and [operator<<\(\)](#).

```
153 {  
154     return d->stepWidth;  
155 }
```

### 9.30.3.6 [qreal](#) GridAttributes::gridSubStepWidth () const

Returns the sub-step width to be used for calculating the sub-grid lines.

See also:

[setGridStepWidth](#)

Definition at line 183 of file KDChartGridAttributes.cpp.

References d.

Referenced by KDChart::CartesianCoordinatePlane::getDataDimensionsList(), and operator<<().

```
184 {
185     return d->subStepWidth;
186 }
```

### 9.30.3.7 bool GridAttributes::isGridVisible () const

Definition at line 118 of file KDChartGridAttributes.cpp.

References d.

Referenced by operator<<(), and operator==( ).

```
119 {
120     return d->visible;
121 }
```

### 9.30.3.8 bool GridAttributes::isSubGridVisible () const

Definition at line 251 of file KDChartGridAttributes.cpp.

References d.

Referenced by operator<<(), and operator==( ).

```
252 {
253     return d->subVisible;
254 }
```

### 9.30.3.9 bool KDChart::GridAttributes::operator!= (const [GridAttributes](#) & other) const

Definition at line 107 of file KDChartGridAttributes.h.

```
107 { return !operator==(other); }
```

### 9.30.3.10 [GridAttributes](#) & GridAttributes::operator= (const [GridAttributes](#) &)

Definition at line 84 of file KDChartGridAttributes.cpp.

References d.

```
85 {
86     if( this == &r )
87         return *this;
88
89     *d = *r.d;
90
91     return *this;
92 }
```

**9.30.3.11 bool GridAttributes::operator==(const [GridAttributes](#) &) const**

Definition at line 100 of file KDChartGridAttributes.cpp.

References [adjustLowerBoundToGrid\(\)](#), [adjustUpperBoundToGrid\(\)](#), [gridGranularitySequence\(\)](#), [gridPen\(\)](#), [isGridVisible\(\)](#), [isSubGridVisible\(\)](#), [subGridPen\(\)](#), and [zeroLinePen\(\)](#).

```

101 {
102     return  isGridVisible() == r.isGridVisible() &&
103            gridGranularitySequence() == r.gridGranularitySequence() &&
104            adjustLowerBoundToGrid() == r.adjustLowerBoundToGrid() &&
105            adjustUpperBoundToGrid() == r.adjustUpperBoundToGrid() &&
106            gridPen() == r.gridPen() &&
107            isSubGridVisible() == r.isSubGridVisible() &&
108            subGridPen() == r.subGridPen() &&
109            zeroLinePen() == r.zeroLinePen();
110 }
```

**9.30.3.12 void GridAttributes::setAdjustBoundsToGrid (bool *adjustLower*, bool *adjustUpper*)**

By default visible bounds of the data area are adjusted to match a main grid line.

If you set the respective adjust flag to false the bound will not start at a grid line's value but it will be the exact value of the data range set.

**See also:**

[CartesianCoordinatePlane::setHorizontalRange](#)  
[CartesianCoordinatePlane::setVerticalRange](#)

Definition at line 221 of file KDChartGridAttributes.cpp.

References [d](#).

```

222 {
223     d->adjustLower = adjustLower;
224     d->adjustUpper = adjustUpper;
225 }
```

**9.30.3.13 void GridAttributes::setGridGranularitySequence ([KDChartEnums::GranularitySequence](#) *sequence*)**

Specify which granularity sequence is to be used to find a matching grid granularity.

See details explained at [KDChartEnums::GranularitySequence](#).

You might also want to use [setAdjustBoundsToGrid](#) for fine-tuning the start/end value.

**See also:**

[setAdjustBoundsToGrid](#), [GranularitySequence](#)

Definition at line 205 of file KDChartGridAttributes.cpp.

References [d](#).

```

206 {
207     d->sequence = sequence;
208 }
```

### 9.30.3.14 void GridAttributes::setGridPen (const QPen & *pen*)

Definition at line 235 of file KDChartGridAttributes.cpp.

References [d](#).

```
236 {  
237     d->pen = pen;  
238     d->pen.setCapStyle( Qt::FlatCap );  
239 }
```

### 9.30.3.15 void GridAttributes::setGridStepWidth (qreal *stepWidth* = 0 . 0)

Specifies the step width to be used for calculating the grid lines.

#### Note:

Step width can be set for Linear axis calculation mode only, there is no way to specify a step width for Logarithmic axes.

By default the [GridAttributes](#) class does not use a fixed step width, but it uses [KDChartEnums::GranularitySequence\\_10\\_20](#).

#### Parameters:

*stepWidth* the step width to be used. If this parameter is omitted (or set to Zero, resp.) the automatic step width calculation will be done, using the granularity sequence specified. This is the default.

#### See also:

[gridStepWidth](#), [setGranularitySequence](#)

Definition at line 141 of file KDChartGridAttributes.cpp.

References [d](#).

```
142 {  
143     d->stepWidth = stepWidth;  
144 }
```

### 9.30.3.16 void GridAttributes::setGridSubStepWidth (qreal *subStepWidth* = 0 . 0)

Specifies the sub-step width to be used for calculating the grid sub-lines.

#### Parameters:

*subStepWidth* the sub-step width to be used. If this parameter is omitted (or set to Zero, resp.) the automatic calculation will be done, using the granularity sequence specified. This is the default.

#### See also:

[gridSubStepWidth](#)

Definition at line 172 of file KDChartGridAttributes.cpp.

References [d](#).

```
173 {  
174     d->subStepWidth = subStepWidth;  
175 }
```

#### 9.30.3.17 void GridAttributes::setGridVisible (bool *visible*)

Definition at line 113 of file KDChartGridAttributes.cpp.

References [d](#).

```
114 {  
115     d->visible = visible;  
116 }
```

#### 9.30.3.18 void GridAttributes::setSubGridPen (const QPen & *pen*)

Definition at line 256 of file KDChartGridAttributes.cpp.

References [d](#).

```
257 {  
258     d->subPen = pen;  
259     d->subPen.setCapStyle( Qt::FlatCap );  
260 }
```

#### 9.30.3.19 void GridAttributes::setSubGridVisible (bool *visible*)

Definition at line 246 of file KDChartGridAttributes.cpp.

References [d](#).

```
247 {  
248     d->subVisible = visible;  
249 }
```

#### 9.30.3.20 void GridAttributes::setZeroLinePen (const QPen & *pen*)

Definition at line 267 of file KDChartGridAttributes.cpp.

References [d](#).

```
268 {  
269     d->zeroPen = pen;  
270     d->zeroPen.setCapStyle( Qt::FlatCap );  
271 }
```

#### 9.30.3.21 QPen GridAttributes::subGridPen () const

Definition at line 262 of file KDChartGridAttributes.cpp.

References [d](#).

Referenced by [operator<<\(\)](#), and [operator==\(\)](#).

```
263 {  
264     return d->subPen;  
265 }
```

### 9.30.3.22 QPen GridAttributes::zeroLinePen () const

Definition at line 273 of file KDChartGridAttributes.cpp.

References [d](#).

Referenced by [operator<<\(\)](#), and [operator==\(\)](#).

```
274 {  
275     return d->zeroPen;  
276 }
```

The documentation for this class was generated from the following files:

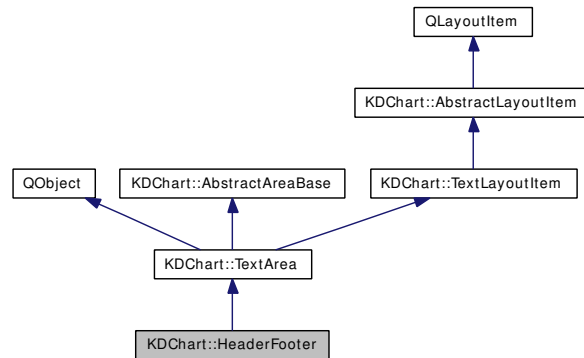
- [KDChartGridAttributes.h](#)
- [KDChartGridAttributes.cpp](#)



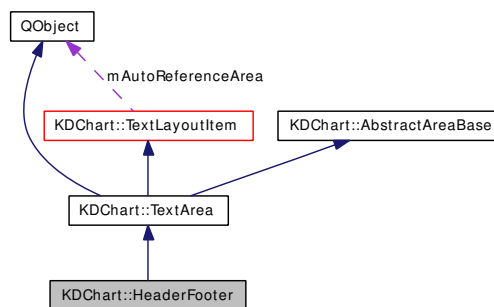
## 9.31 KDChart::HeaderFooter Class Reference

```
#include <KDChartHeaderFooter.h>
```

Inheritance diagram for KDChart::HeaderFooter:



Collaboration diagram for KDChart::HeaderFooter:



### 9.31.1 Detailed Description

A header or even footer displaying text above or below charts.

Definition at line 44 of file `KDChartHeaderFooter.h`.

### Public Types

- enum `HeaderFooterType` {  
     `Header`,  
     `Footer` }

### Signals

- void `destroyedHeaderFooter` (`HeaderFooter *`)
- void `positionChanged` (`TextArea *`)
- void `positionChanged` (`HeaderFooter *`)

## Public Member Functions

- void [alignToReferencePoint](#) (const [RelativePosition](#) &position)
- const [QObject](#) \* [autoReferenceArea](#) () const
- [BackgroundAttributes](#) [backgroundAttributes](#) () const
- virtual [HeaderFooter](#) \* [clone](#) () const  
*Creates an exact copy of this header/footer.*
- bool [compare](#) (const [AbstractAreaBase](#) \*other) const  
*Returns true if both areas have the same settings.*
- bool [compare](#) (const [HeaderFooter](#) &other) const
- virtual Qt::Orientations [expandingDirections](#) () const  
*pure virtual in [QLayoutItem](#)*
- [FrameAttributes](#) [frameAttributes](#) () const
- virtual [QRect](#) [geometry](#) () const  
*pure virtual in [QLayoutItem](#)*
- void [getFrameLeadings](#) (int &left, int &top, int &right, int &bottom) const
- [HeaderFooter](#) ([Chart](#) \*parent=0)
- virtual bool [intersects](#) (const [TextLayoutItem](#) &other, const [QPoint](#) &myPos, const [QPoint](#) &otherPos) const
- virtual bool [intersects](#) (const [TextLayoutItem](#) &other, const [QPointF](#) &myPos, const [QPointF](#) &otherPos) const
- virtual bool [isEmpty](#) () const  
*pure virtual in [QLayoutItem](#)*
- virtual [QSize](#) [maximumSize](#) () const  
*pure virtual in [QLayoutItem](#)*
- virtual [QSize](#) [minimumSize](#) () const  
*pure virtual in [QLayoutItem](#)*
- virtual void [paint](#) ([QPainter](#) \*)
- void [paintAll](#) ([QPainter](#) &painter)  
*Call paintAll, if you want the background and the frame to be drawn before the normal [paint\(\)](#) is invoked automatically.*
- virtual void [paintBackground](#) ([QPainter](#) &painter, const [QRect](#) &rectangle)
- virtual void [paintCtx](#) ([PaintContext](#) \*context)  
*Default impl: Paint the complete item using its layouted position and size.*
- virtual void [paintFrame](#) ([QPainter](#) &painter, const [QRect](#) &rectangle)
- virtual void [paintIntoRect](#) ([QPainter](#) &painter, const [QRect](#) &rect)  
*Draws the background and frame, then calls [paint\(\)](#).*
- [QLayout](#) \* [parentLayout](#) ()
- [Position](#) [position](#) () const
- virtual [QFont](#) [realFont](#) () const

- virtual qreal [realFontSize](#) () const
- void [removeFromParentLayout](#) ()
- void [setAutoReferenceArea](#) (const [QObject](#) \*area)
- void [setBackgroundAttributes](#) (const [BackgroundAttributes](#) &a)
- void [setFrameAttributes](#) (const [FrameAttributes](#) &a)
- virtual void [setGeometry](#) (const [QRect](#) &r)  
pure virtual in [QLayoutItem](#)
- void [setParent](#) ([QObject](#) \*parent)
- void [setParentLayout](#) ([QLayout](#) \*lay)
- virtual void [setParentWidget](#) ([QWidget](#) \*widget)  
*Inform the item about its widget: This enables the item, to trigger that widget's update, whenever the size of the item's contents has changed.*
- void [setPosition](#) ([Position](#) position)
- void [setText](#) (const [QString](#) &text)
- void [setTextAttributes](#) (const [TextAttributes](#) &a)  
*Use this to specify the text attributes to be used for this item.*
- void [setType](#) ([HeaderFooterType](#) type)
- virtual [QSize](#) [sizeHint](#) () const  
pure virtual in [QLayoutItem](#)
- virtual void [sizeHintChanged](#) () const  
*Report changed size hint: ask the parent widget to recalculate the layout.*
- [QString](#) [text](#) () const
- [TextAttributes](#) [textAttributes](#) () const  
*Returns the text attributes to be used for this item.*
- [HeaderFooterType](#) [type](#) () const
- virtual [~HeaderFooter](#) ()

## Static Public Member Functions

- static void [paintBackgroundAttributes](#) ([QPainter](#) &painter, const [QRect](#) &rectangle, const [KDChart::BackgroundAttributes](#) &attributes)
- static void [paintFrameAttributes](#) ([QPainter](#) &painter, const [QRect](#) &rectangle, const [KDChart::FrameAttributes](#) &attributes)

## Protected Member Functions

- virtual [QRect](#) [areaGeometry](#) () const
- [QRect](#) [innerRect](#) () const
- virtual void [positionHasChanged](#) ()

## Protected Attributes

- [QWidget](#) \* [mParent](#)
- [QLayout](#) \* [mParentLayout](#)

### 9.31.2 Member Enumeration Documentation

#### 9.31.2.1 enum [KDChart::HeaderFooter::HeaderFooterType](#)

Enumerator:

*Header*

*Footer*

Definition at line 59 of file KDChartHeaderFooter.h.

```
59             { Header,
60             Footer };
```

### 9.31.3 Constructor & Destructor Documentation

#### 9.31.3.1 [HeaderFooter::HeaderFooter](#) ([Chart](#) \* *parent* = 0)

Definition at line 55 of file KDChartHeaderFooter.cpp.

References [setParent\(\)](#).

Referenced by [clone\(\)](#).

```
55                                     :
56     TextArea( new Private() )
57 {
58     setParent( parent );
59     init();
60 }
```

#### 9.31.3.2 [HeaderFooter::~HeaderFooter](#) () [virtual]

Definition at line 62 of file KDChartHeaderFooter.cpp.

References [destroyedHeaderFooter\(\)](#).

```
63 {
64     emit destroyedHeaderFooter( this );
65 }
```

### 9.31.4 Member Function Documentation

#### 9.31.4.1 void [AbstractAreaBase::alignToReferencePoint](#) (const [RelativePosition](#) & *position*) [inherited]

Definition at line 90 of file KDChartAbstractAreaBase.cpp.

```
91 {
92     Q_UNUSED( position );
93     // PENDING(kalle) FIXME
94     qWarning( "Sorry, not implemented: void AbstractAreaBase::alignToReferencePoint( const RelativePosi
95 }
```

**9.31.4.2 QRect TextArea::areaGeometry () const** [protected, virtual, inherited]

Implements [KDChart::AbstractAreaBase](#).

Definition at line 105 of file KDChartTextArea.cpp.

References [KDChart::TextLayoutItem::geometry\(\)](#).

Referenced by [KDChart::TextArea::paintAll\(\)](#).

```
106 {  
107     return geometry();  
108 }
```

**9.31.4.3 const QObject \* KDChart::TextLayoutItem::autoReferenceArea () const**  
[inherited]

Definition at line 135 of file KDChartLayoutItems.cpp.

Referenced by [compare\(\)](#), and [setParent\(\)](#).

```
136 {  
137     return mAutoReferenceArea;  
138 }
```

**9.31.4.4 BackgroundAttributes AbstractAreaBase::backgroundAttributes () const**  
[inherited]

Definition at line 120 of file KDChartAbstractAreaBase.cpp.

References [d](#).

Referenced by [KDChart::AbstractAreaBase::compare\(\)](#), and [updateCommonBrush\(\)](#).

```
121 {  
122     return d->backgroundAttributes;  
123 }
```

**9.31.4.5 HeaderFooter \* HeaderFooter::clone () const** [virtual]

Creates an exact copy of this header/footer.

Definition at line 95 of file KDChartHeaderFooter.cpp.

References [d](#), [HeaderFooter\(\)](#), [position\(\)](#), [setPosition\(\)](#), [KDChart::TextLayoutItem::setTextAttributes\(\)](#), [setType\(\)](#), [KDChart::TextLayoutItem::textAttributes\(\)](#), and [type\(\)](#).

```
96 {  
97     HeaderFooter* headerFooter = new HeaderFooter( new Private( *d ), 0 );  
98     headerFooter->setType( type() );  
99     headerFooter->setPosition( position() );  
100     headerFooter->setTextAttributes( textAttributes() );  
101     return headerFooter;  
102 }
```

#### 9.31.4.6 bool AbstractAreaBase::compare (const [AbstractAreaBase](#) \* *other*) const

[inherited]

Returns true if both areas have the same settings.

Definition at line 75 of file KDChartAbstractAreaBase.cpp.

References [KDChart::AbstractAreaBase::backgroundAttributes\(\)](#), and [KDChart::AbstractAreaBase::frameAttributes\(\)](#).

```

76 {
77     if( other == this ) return true;
78     if( ! other ){
79         //qDebug() << "CartesianAxis::compare() cannot compare to Null pointer";
80         return false;
81     }
82     /*
83     qDebug() << "AbstractAreaBase:" << (frameAttributes() == other->frameAttributes())
84         << (backgroundAttributes() == other->backgroundAttributes()) << "\n";
85     */
86     return (frameAttributes() == other->frameAttributes()) &&
87         (backgroundAttributes() == other->backgroundAttributes());
88 }
```

#### 9.31.4.7 bool HeaderFooter::compare (const [HeaderFooter](#) & *other*) const

Definition at line 104 of file KDChartHeaderFooter.cpp.

References [KDChart::TextLayoutItem::autoReferenceArea\(\)](#), [position\(\)](#), [KDChart::TextLayoutItem::text\(\)](#), [KDChart::TextLayoutItem::textAttributes\(\)](#), and [type\(\)](#).

```

105 {
106     return (type() == other.type()) &&
107         (position() == other.position()) &&
108         // also compare members inherited from the base class:
109         (autoReferenceArea() == other.autoReferenceArea()) &&
110         (text() == other.text()) &&
111         (textAttributes() == other.textAttributes());
112 }
```

#### 9.31.4.8 void KDChart::HeaderFooter::destroyedHeaderFooter ([HeaderFooter](#) \*) [signal]

Referenced by [~HeaderFooter\(\)](#).

#### 9.31.4.9 Qt::Orientations KDChart::TextLayoutItem::expandingDirections () const [virtual, inherited]

pure virtual in [QLayoutItem](#)

Definition at line 179 of file KDChartLayoutItems.cpp.

```

180 {
181     return 0; // Grow neither vertically nor horizontally
182 }
```

**9.31.4.10** [FrameAttributes](#) **AbstractAreaBase::frameAttributes () const** [inherited]

Definition at line 106 of file KDChartAbstractAreaBase.cpp.

References [d](#).

Referenced by [KDChart::Legend::clone\(\)](#), [KDChart::AbstractAreaBase::compare\(\)](#), and [updateCommonBrush\(\)](#).

```
107 {
108     return d->frameAttributes;
109 }
```

**9.31.4.11** **QRect KDChart::TextLayoutItem::geometry () const** [virtual, inherited]

pure virtual in [QLayoutItem](#)

Definition at line 184 of file KDChartLayoutItems.cpp.

Referenced by [KDChart::TextArea::areaGeometry\(\)](#), [KDChart::TextLayoutItem::paint\(\)](#), [KDChart::TextArea::paintAll\(\)](#), [KDChart::CartesianAxis::paintCtx\(\)](#), and [KDChart::TextArea::paintIntoRect\(\)](#).

```
185 {
186     return mRect;
187 }
```

**9.31.4.12** **void AbstractAreaBase::getFrameLeadings (int & left, int & top, int & right, int & bottom) const** [inherited]

Definition at line 212 of file KDChartAbstractAreaBase.cpp.

References [d](#).

Referenced by [KDChart::AbstractAreaBase::innerRect\(\)](#), and [KDChart::AbstractAreaWidget::paintAll\(\)](#).

```
213 {
214     if( d && d->frameAttributes.isVisible() ){
215         const int padding = qMax( d->frameAttributes.padding(), 0 );
216         left   = padding;
217         top    = padding;
218         right  = padding;
219         bottom = padding;
220     }else{
221         left   = 0;
222         top    = 0;
223         right  = 0;
224         bottom = 0;
225     }
226 }
```

**9.31.4.13** **QRect AbstractAreaBase::innerRect () const** [protected, inherited]

Definition at line 228 of file KDChartAbstractAreaBase.cpp.

References [KDChart::AbstractAreaBase::areaGeometry\(\)](#), and [KDChart::AbstractAreaBase::getFrameLeadings\(\)](#).

Referenced by [KDChart::TextArea::paintAll\(\)](#), and [KDChart::AbstractArea::paintAll\(\)](#).

```

229 {
230     int left;
231     int top;
232     int right;
233     int bottom;
234     getFrameLeadings( left, top, right, bottom );
235     return
236         QRect( QPoint(0,0), areaGeometry().size() )
237             .adjusted( left, top, -right, -bottom );
238 }

```

#### 9.31.4.14 bool KDChart::TextLayoutItem::intersects (const [TextLayoutItem](#) & other, const QPoint & myPos, const QPoint & otherPos) const [virtual, inherited]

Definition at line 258 of file KDChartLayoutItems.cpp.

References KDChart::TextLayoutItem::mAttributes, PI, KDChart::TextLayoutItem::rotatedCorners(), KDChart::TextAttributes::rotation(), and KDChart::TextLayoutItem::unrotatedSizeHint().

```

259 {
260     if ( mAttributes.rotation() != other.mAttributes.rotation() )
261     {
262         // that's the code for the common case: the rotation angles don't need to match here
263         QPolygon myPolygon( rotatedCorners() );
264         QPolygon otherPolygon( other.rotatedCorners() );
265
266         // move the polygons to their positions
267         myPolygon.translate( myPos );
268         otherPolygon.translate( otherPos );
269
270         // create regions out of it
271         QRegion myRegion( myPolygon );
272         QRegion otherRegion( otherPolygon );
273
274         // now the question - do they intersect or not?
275         return ! myRegion.intersect( otherRegion ).isEmpty();
276     }
277     else {
278         // and that's the code for the special case: the rotation angles match, which is less time consuming
279         const qreal angle = mAttributes.rotation() * PI / 180.0;
280         // both sizes
281         const QSizeF mySize( unrotatedSizeHint() );
282         const QSizeF otherSize( other.unrotatedSizeHint() );
283
284         // that's myP1 relative to myPos
285         QPointF myP1( mySize.height() * sin( angle ), 0.0 );
286         // that's otherP1 to myPos
287         QPointF otherP1 = QPointF( otherSize.height() * sin( angle ), 0.0 ) + otherPos - myPos;
288
289         // now rotate both points the negative angle around myPos
290         myP1 = QPointF( myP1.x() * cos( -angle ), myP1.x() * sin( -angle ) );
291         qreal r = sqrt( otherP1.x() * otherP1.x() + otherP1.y() * otherP1.y() );
292         otherP1 = QPointF( r * cos( -angle ), r * sin( -angle ) );
293
294         // finally we look, whether both rectangles intersect or even not
295         return QRectF( myP1, mySize ).intersects( QRectF( otherP1, otherSize ) );
296     }
297 }

```



**9.31.4.15** `bool KDChart::TextLayoutItem::intersects (const TextLayoutItem & other, const QPointF & myPos, const QPointF & otherPos) const` [virtual, inherited]

Definition at line 253 of file KDChartLayoutItems.cpp.

Referenced by KDChart::CartesianAxis::paintCtx().

```
254 {  
255     return intersects( other, myPos.toPoint(), otherPos.toPoint() );  
256 }
```

**9.31.4.16** `bool KDChart::TextLayoutItem::isEmpty () const` [virtual, inherited]

pure virtual in [QLayoutItem](#)

Definition at line 189 of file KDChartLayoutItems.cpp.

```
190 {  
191     return false; // never empty, otherwise the layout item would not exist  
192 }
```

**9.31.4.17** `QSize KDChart::TextLayoutItem::maximumSize () const` [virtual, inherited]

pure virtual in [QLayoutItem](#)

Definition at line 194 of file KDChartLayoutItems.cpp.

References KDChart::TextLayoutItem::sizeHint().

```
195 {  
196     return sizeHint(); // PENDING(kalle) Review, quite inflexible  
197 }
```

**9.31.4.18** `QSize KDChart::TextLayoutItem::minimumSize () const` [virtual, inherited]

pure virtual in [QLayoutItem](#)

Definition at line 199 of file KDChartLayoutItems.cpp.

References KDChart::TextLayoutItem::sizeHint().

```
200 {  
201     return sizeHint(); // PENDING(kalle) Review, quite inflexible  
202 }
```

**9.31.4.19** `void KDChart::TextLayoutItem::paint (QPainter *)` [virtual, inherited]

Implements [KDChart::AbstractLayoutItem](#).

Definition at line 386 of file KDChartLayoutItems.cpp.

References KDChart::TextLayoutItem::geometry(), KDChart::TextAttributes::pen(), rotatedRect(), and KDChart::TextAttributes::rotation().

Referenced by KDChart::TextArea::paintAll(), and KDChart::CartesianAxis::paintCtx().

```

387 {
388     // make sure, cached font is updated, if needed:
389     // sizeHint();
390
391     if( !mRect.isValid() )
392         return;
393
394     PainterSaver painterSaver( painter );
395     painter->setFont( cachedFont );
396     QRectF rect( geometry() );
397
398     // #ifdef DEBUG_ITEMS_PAINT
399     //     painter->setPen( Qt::black );
400     //     painter->drawRect( rect );
401     // #endif
402     painter->translate( rect.center() );
403     rect.moveTopLeft( QPointF( - rect.width() / 2, - rect.height() / 2 ) );
404     // #ifdef DEBUG_ITEMS_PAINT
405     //     painter->setPen( Qt::blue );
406     //     painter->drawRect( rect );
407     // #endif
408     painter->rotate( mAttributes.rotation() );
409     rect = rotatedRect( rect, mAttributes.rotation() );
410     // #ifdef DEBUG_ITEMS_PAINT
411     //     painter->setPen( Qt::red );
412     //     painter->drawRect( rect );
413     // #endif
414     painter->setPen( mAttributes.pen() );
415     painter->drawText( rect, Qt::AlignHCenter | Qt::AlignVCenter, mText );
416     // if ( calcSizeHint( cachedFont ).width() > rect.width() )
417     //     qDebug() << "rect.width()" << rect.width() << "text.width()" << calcSizeHint( cachedFont ).width();
418     //
419     // //painter->drawText( rect, Qt::AlignHCenter | Qt::AlignVCenter, mText );
420 }

```

#### 9.31.4.20 void TextArea::paintAll (QPainter & painter) [virtual, inherited]

Call `paintAll`, if you want the background and the frame to be drawn before the normal `paint()` is invoked automatically.

Reimplemented from [KDChart::AbstractLayoutItem](#).

Definition at line 83 of file `KDChartTextArea.cpp`.

References `KDChart::TextArea::areaGeometry()`, `KDChart::TextLayoutItem::geometry()`, `KDChart::AbstractAreaBase::innerRect()`, `KDChart::TextLayoutItem::paint()`, `KDChart::AbstractAreaBase::paintBackground()`, `KDChart::AbstractAreaBase::paintFrame()`, and `KDChart::TextLayoutItem::setGeometry()`.

Referenced by `KDChart::TextArea::paintIntoRect()`.

```

84 {
85     // Paint the background and frame
86     paintBackground( painter, geometry() );
87     paintFrame( painter, geometry() );
88
89     // temporarily adjust the widget size, to be sure all content gets calculated
90     // to fit into the inner rectangle
91     const QRect oldGeometry( areaGeometry() );
92     QRect inner( innerRect() );
93     inner.moveTo(
94         oldGeometry.left() + inner.left(),
95         oldGeometry.top() + inner.top() );
96     const bool needAdjustGeometry = oldGeometry != inner;

```

```

97     if( needAdjustGeometry )
98         setGeometry( inner );
99     paint( &painter );
100     if( needAdjustGeometry )
101         setGeometry( oldGeometry );
102     //qDebug() << "TextAreaWidget::paintAll() done.";
103 }

```

#### 9.31.4.21 void AbstractAreaBase::paintBackground (QPainter & *painter*, const QRect & *rectangle*) [virtual, inherited]

Definition at line 196 of file KDChartAbstractAreaBase.cpp.

References `d`, and `KDChart::AbstractAreaBase::paintBackgroundAttributes()`.

Referenced by `KDChart::TextArea::paintAll()`, `KDChart::AbstractAreaWidget::paintAll()`, and `KDChart::AbstractArea::paintAll()`.

```

197 {
198     Q_ASSERT_X ( d != 0, "AbstractAreaBase::paintBackground()",
199                 "Private class was not initialized!" );
200     paintBackgroundAttributes( painter, rect, d->backgroundAttributes );
201 }

```

#### 9.31.4.22 void AbstractAreaBase::paintBackgroundAttributes (QPainter & *painter*, const QRect & *rectangle*, const [KDChart::BackgroundAttributes](#) & *attributes*) [static, inherited]

Definition at line 127 of file KDChartAbstractAreaBase.cpp.

References `KDChart::BackgroundAttributes::BackgroundPixmapModeCentered`, `KDChart::BackgroundAttributes::BackgroundPixmapModeNone`, `KDChart::BackgroundAttributes::BackgroundPixmapModeScaled`, `KDChart::BackgroundAttributes::BackgroundPixmapModeStretched`, `KDChart::BackgroundAttributes::brush()`, `KDChart::BackgroundAttributes::isVisible()`, `KDChart::BackgroundAttributes::pixmap()`, and `KDChart::BackgroundAttributes::pixmapMode()`.

Referenced by `KDChart::AbstractAreaBase::paintBackground()`.

```

129 {
130     if( !attributes.isVisible() ) return;
131
132     /* first draw the brush (may contain a pixmap)*/
133     if( Qt::NoBrush != attributes.brush().style() ) {
134         KDChart::PainterSaver painterSaver( &painter );
135         painter.setPen( Qt::NoPen );
136         const QPointF newTopLeft( painter.deviceMatrix().map( rect.topLeft() ) );
137         painter.setBrushOrigin( newTopLeft );
138         painter.setBrush( attributes.brush() );
139         painter.drawRect( rect.adjusted( 0, 0, -1, -1 ) );
140     }
141     /* next draw the backPixmap over the brush */
142     if( !attributes.pixmap().isNull() &&
143         attributes.pixmapMode() != BackgroundAttributes::BackgroundPixmapModeNone ) {
144         QPointF ol = rect.topLeft();
145         if( BackgroundAttributes::BackgroundPixmapModeCentered == attributes.pixmapMode() )
146         {
147             ol.setX( rect.center().x() - attributes.pixmap().width() / 2 );
148             ol.setY( rect.center().y() - attributes.pixmap().height() / 2 );
149             painter.drawPixmap( ol, attributes.pixmap() );

```

```

150         } else {
151             QMatrix m;
152             double zW = (double)rect.width() / (double)attributes.pixmap().width();
153             double zH = (double)rect.height() / (double)attributes.pixmap().height();
154             switch( attributes.pixmapMode() ) {
155                 case BackgroundAttributes::BackgroundPixmapModeScaled:
156                     {
157                         double z;
158                         z = qMin( zW, zH );
159                         m.scale( z, z );
160                     }
161                 break;
162                 case BackgroundAttributes::BackgroundPixmapModeStretched:
163                     m.scale( zW, zH );
164                     break;
165                 default:
166                     ; // Cannot happen, previously checked
167             }
168             QPixmap pm = attributes.pixmap().transformed( m );
169             ol.setX( rect.center().x() - pm.width() / 2 );
170             ol.setY( rect.center().y() - pm.height() / 2 );
171             painter.drawPixmap( ol, pm );
172         }
173     }
174 }

```

**9.31.4.23** `void KDChart::AbstractLayoutItem::paintCtx (PaintContext * context)` [virtual, inherited]

Default impl: Paint the complete item using its layouted position and size.

Reimplemented in [KDChart::CartesianAxis](#), and [KDChart::TernaryAxis](#).

Definition at line 77 of file `KDChartLayoutItems.cpp`.

References `KDChart::AbstractLayoutItem::paint()`, and `KDChart::PaintContext::painter()`.

```

78 {
79     if( context )
80         paint( context->painter() );
81 }

```

**9.31.4.24** `void AbstractAreaBase::paintFrame (QPainter & painter, const QRect & rectangle)` [virtual, inherited]

Definition at line 204 of file `KDChartAbstractAreaBase.cpp`.

References `d`, and `KDChart::AbstractAreaBase::paintFrameAttributes()`.

Referenced by `KDChart::TextArea::paintAll()`, `KDChart::AbstractAreaWidget::paintAll()`, and `KDChart::AbstractArea::paintAll()`.

```

205 {
206     Q_ASSERT_X ( d != 0, "AbstractAreaBase::paintFrame()",
207                 "Private class was not initialized!" );
208     paintFrameAttributes( painter, rect, d->frameAttributes );
209 }

```

#### 9.31.4.25 void AbstractAreaBase::paintFrameAttributes (QPainter & *painter*, const QRect & *rectangle*, const [KDChart::FrameAttributes](#) & *attributes*) [static, inherited]

Definition at line 177 of file KDChartAbstractAreaBase.cpp.

References [KDChart::FrameAttributes::isVisible\(\)](#), and [KDChart::FrameAttributes::pen\(\)](#).

Referenced by [KDChart::AbstractAreaBase::paintFrame\(\)](#).

```

179 {
180
181     if( !attributes.isVisible() ) return;
182
183     // Note: We set the brush to NoBrush explicitly here.
184     //       Otherwise we might get a filled rectangle, so any
185     //       previously drawn background would be overwritten by that area.
186
187     const QPen    oldPen(    painter.pen() );
188     const QBrush  oldBrush( painter.brush() );
189     painter.setPen(    attributes.pen() );
190     painter.setBrush( Qt::NoBrush );
191     painter.drawRect( rect.adjusted( 0, 0, -1, -1 ) );
192     painter.setBrush( oldBrush );
193     painter.setPen(    oldPen );
194 }
```

#### 9.31.4.26 void TextArea::paintIntoRect (QPainter & *painter*, const QRect & *rect*) [virtual, inherited]

Draws the background and frame, then calls [paint\(\)](#).

In most cases there is no need to overwrite this method in a derived class, but you would overwrite [TextLayoutItem::paint\(\)](#) instead.

Definition at line 71 of file KDChartTextArea.cpp.

References [KDChart::TextLayoutItem::geometry\(\)](#), [KDChart::TextArea::paintAll\(\)](#), and [KDChart::TextLayoutItem::setGeometry\(\)](#).

```

72 {
73     const QRect oldGeometry( geometry() );
74     if( oldGeometry != rect )
75         setGeometry( rect );
76     painter.translate( rect.left(), rect.top() );
77     paintAll( painter );
78     painter.translate( -rect.left(), -rect.top() );
79     if( oldGeometry != rect )
80         setGeometry( oldGeometry );
81 }
```

#### 9.31.4.27 QLayout\* KDChart::AbstractLayoutItem::parentLayout () [inherited]

Definition at line 76 of file KDChartLayoutItems.h.

```

77     {
78         return mParentLayout;
79     }
```

**9.31.4.28 Position** `HeaderFooter::position () const`

Definition at line 131 of file `KDChartHeaderFooter.cpp`.

References `d`.

Referenced by `clone()`, `compare()`, and `setPosition()`.

```
132 {
133     return d->position;
134 }
```

**9.31.4.29 void** `KDChart::TextArea::positionChanged (TextArea *)` `[signal, inherited]`

Referenced by `KDChart::TextArea::positionHasChanged()`.

**9.31.4.30 void** `KDChart::HeaderFooter::positionChanged (HeaderFooter *)` `[signal]`

Referenced by `setPosition()`, and `setType()`.

**9.31.4.31 void** `TextArea::positionHasChanged ()` `[protected, virtual, inherited]`

Reimplemented from `KDChart::AbstractAreaBase`.

Definition at line 110 of file `KDChartTextArea.cpp`.

References `KDChart::TextArea::positionChanged()`.

```
111 {
112     emit positionChanged( this );
113 }
```

**9.31.4.32 QFont** `KDChart::TextLayoutItem::realFont () const` `[virtual, inherited]`

Definition at line 230 of file `KDChartLayoutItems.cpp`.

Referenced by `KDChart::CartesianAxis::maximumSize()`, and `KDChart::CartesianAxis::paintCtx()`.

```
231 {
232     realFontWasRecalculated(); // we can safely ignore the boolean return value
233     return cachedFont;
234 }
```

**9.31.4.33 qreal** `KDChart::TextLayoutItem::realFontSize () const` `[virtual, inherited]`

Definition at line 210 of file `KDChartLayoutItems.cpp`.

References `KDChart::TextAttributes::calculatedFontSize()`.

```
211 {
212     return mAttributes.calculatedFontSize( mAutoReferenceArea, mAutoReferenceOrientation );
213 }
```

**9.31.4.34 void KDChart::AbstractLayoutItem::removeFromParentLayout ()** [inherited]

Definition at line 80 of file KDChartLayoutItems.h.

Referenced by KDChart::Chart::takeCoordinatePlane().

```
81     {
82         if( mParentLayout ){
83             if( widget() )
84                 mParentLayout->removeWidget( widget() );
85             else
86                 mParentLayout->removeItem( this );
87         }
88     }
```

**9.31.4.35 void KDChart::TextLayoutItem::setAutoReferenceArea (const [QObject](#) \* area)**  
[inherited]

Definition at line 128 of file KDChartLayoutItems.cpp.

References KDChart::TextLayoutItem::sizeHint().

Referenced by setParent().

```
129 {
130     mAutoReferenceArea = area;
131     cachedSizeHint = QSize();
132     sizeHint();
133 }
```

**9.31.4.36 void AbstractAreaBase::setBackgroundAttributes (const [BackgroundAttributes](#) & a)**  
[inherited]

Definition at line 111 of file KDChartAbstractAreaBase.cpp.

References d, and KDChart::AbstractAreaBase::positionHasChanged().

```
112 {
113     if( d->backgroundAttributes == a )
114         return;
115     d->backgroundAttributes = a;
116     positionHasChanged();
117 }
118 }
```

**9.31.4.37 void AbstractAreaBase::setFrameAttributes (const [FrameAttributes](#) & a)**  
[inherited]

Definition at line 97 of file KDChartAbstractAreaBase.cpp.

References d, and KDChart::AbstractAreaBase::positionHasChanged().

Referenced by KDChart::Legend::clone().

```
98 {
99     if( d->frameAttributes == a )
```

```

100         return;
101
102     d->frameAttributes = a;
103     positionHasChanged();
104 }

```

#### 9.31.4.38 void KDChart::TextLayoutItem::setGeometry (const QRect & r) [virtual, inherited]

pure virtual in [QLayoutItem](#)

Definition at line 204 of file KDChartLayoutItems.cpp.

Referenced by KDChart::TextArea::paintAll(), KDChart::CartesianAxis::paintCtx(), and KDChart::TextArea::paintIntoRect().

```

205 {
206     mRect = r;
207 }

```

#### 9.31.4.39 void HeaderFooter::setParent (QObject \* parent)

Definition at line 67 of file KDChartHeaderFooter.cpp.

References KDChart::TextLayoutItem::autoReferenceArea(), KDChart::TextLayoutItem::setAutoReferenceArea(), and KDChart::AbstractLayoutItem::setParentWidget().

Referenced by KDChart::Widget::addHeaderFooter(), KDChart::Chart::addHeaderFooter(), HeaderFooter(), KDChart::Widget::replaceHeaderFooter(), and KDChart::Chart::takeHeaderFooter().

```

68 {
69     QObject::setParent( parent );
70     setParentWidget( qobject_cast<QWidget*>( parent ) );
71     if( parent && ! autoReferenceArea() )
72         setAutoReferenceArea( parent );
73 }

```

#### 9.31.4.40 void KDChart::AbstractLayoutItem::setParentLayout (QLayout \* lay) [inherited]

Definition at line 72 of file KDChartLayoutItems.h.

```

73     {
74         mParentLayout = lay;
75     }

```

#### 9.31.4.41 void KDChart::AbstractLayoutItem::setParentWidget (QWidget \* widget) [virtual, inherited]

Inform the item about its widget: This enables the item, to trigger that widget's update, whenever the size of the item's contents has changed.

Thus, you need to call setParentWidget on every item, that has a non-fixed size.

Definition at line 64 of file KDChartLayoutItems.cpp.



References KDChart::AbstractLayoutItem::mParent.

Referenced by setParent(), and KDChart::AbstractCartesianDiagram::takeAxis().

```
65 {
66     mParent = widget;
67 }
```

#### 9.31.4.42 void HeaderFooter::setPosition ([Position](#) position)

Definition at line 125 of file KDChartHeaderFooter.cpp.

References d, position(), and positionChanged().

Referenced by KDChart::Widget::addHeaderFooter(), and clone().

```
126 {
127     d->position = position;
128     emit positionChanged( this );
129 }
```

#### 9.31.4.43 void KDChart::TextLayoutItem::setText (const QString & text) [inherited]

Definition at line 140 of file KDChartLayoutItems.cpp.

References KDChart::AbstractLayoutItem::mParent, and KDChart::TextLayoutItem::sizeHint().

Referenced by KDChart::Widget::addHeaderFooter(), KDChart::CartesianAxis::maximumSize(), and KDChart::CartesianAxis::paintCtx().

```
141 {
142     mText = text;
143     cachedSizeHint = QSize();
144     sizeHint();
145     if( mParent )
146         mParent->update();
147 }
```

#### 9.31.4.44 void KDChart::TextLayoutItem::setTextAttributes (const [TextAttributes](#) & a) [inherited]

Use this to specify the text attributes to be used for this item.

**See also:**

[textAttributes](#)

Definition at line 159 of file KDChartLayoutItems.cpp.

References KDChart::AbstractLayoutItem::mParent, and KDChart::TextLayoutItem::sizeHint().

Referenced by clone().

```
160 {
161     mAttributes = a;
162     cachedSizeHint = QSize(); // invalidate size hint
```

```

163     sizeHint();
164     if( mParent )
165         mParent->update();
166 }

```

#### 9.31.4.45 void HeaderComponent::setType (HeaderFooterType type)

Definition at line 114 of file KDChartHeaderFooter.cpp.

References d, and positionChanged().

Referenced by KDChart::Widget::addHeaderFooter(), and clone().

```

115 {
116     d->type = type;
117     emit positionChanged( this );
118 }

```

#### 9.31.4.46 QSize KDChart::TextLayoutItem::sizeHint () const [virtual, inherited]

pure virtual in [QLayoutItem](#)

Definition at line 299 of file KDChartLayoutItems.cpp.

References KDChart::AbstractLayoutItem::sizeHintChanged().

Referenced by KDChart::TextLayoutItem::maximumSize(), KDChart::CartesianAxis::maximumSize(), KDChart::TextLayoutItem::minimumSize(), KDChart::CartesianAxis::paintCtx(), KDChart::TextLayoutItem::setAutoReferenceArea(), KDChart::TextLayoutItem::setText(), and KDChart::TextLayoutItem::setTextAttributes().

```

300 {
301     if( realFontWasRecalculated() )
302     {
303         const QSize newSizeHint( calcSizeHint( cachedFont ) );
304         if( newSizeHint != cachedSizeHint ){
305             cachedSizeHint = newSizeHint;
306             sizeHintChanged();
307         }
308     }
309     //qDebug() << "----- KDChart::TextLayoutItem::sizeHint() returns:<<cachedSizeHint<< " -----
310     return cachedSizeHint;
311 }

```

#### 9.31.4.47 void KDChart::AbstractLayoutItem::sizeHintChanged () const [virtual, inherited]

Report changed size hint: ask the parent widget to recalculate the layout.

Definition at line 86 of file KDChartLayoutItems.cpp.

References KDChart::AbstractLayoutItem::mParent.

Referenced by KDChart::TextLayoutItem::sizeHint().

```

87 {
88     // This is exactly like what QWidget::updateGeometry does.

```

```

89 // qDebug("KDChart::AbstractLayoutItem::sizeHintChanged() called");
90 if( mParent ) {
91     if ( mParent->layout() )
92         mParent->layout()->invalidate();
93     else
94         QApplication::postEvent( mParent, new QEvent( QEvent::LayoutRequest ) );
95 }
96 }

```

#### 9.31.4.48 QString KDChart::TextLayoutItem::text () const [inherited]

Definition at line 149 of file KDChartLayoutItems.cpp.

Referenced by compare(), and KDChart::CartesianAxis::paintCtx().

```

150 {
151     return mText;
152 }

```

#### 9.31.4.49 KDChart::TextAttributes KDChart::TextLayoutItem::textAttributes () const [inherited]

Returns the text attributes to be used for this item.

See also:

[setTextAttributes](#)

Definition at line 173 of file KDChartLayoutItems.cpp.

Referenced by clone(), and compare().

```

174 {
175     return mAttributes;
176 }

```

#### 9.31.4.50 HeaderFooter::HeaderFooterType HeaderFooter::type () const

Definition at line 120 of file KDChartHeaderFooter.cpp.

References d.

Referenced by clone(), and compare().

```

121 {
122     return d->type;
123 }

```

### 9.31.5 Member Data Documentation

#### 9.31.5.1 QWidget\* KDChart::AbstractLayoutItem::mParent [protected, inherited]

Definition at line 90 of file KDChartLayoutItems.h.

Referenced by KDChart::AbstractLayoutItem::setParentWidget(), KDChart::TextLayoutItem::setText(), KDChart::TextLayoutItem::setTextAttributes(), and KDChart::AbstractLayoutItem::sizeHintChanged().

**9.31.5.2** `QLayout*` [KDChart::AbstractLayoutItem::mParentLayout](#) [protected,  
inherited]

Definition at line 91 of file `KDChartLayoutItems.h`.

Referenced by `KDChart::AutoSpacerLayoutItem::paint()`.

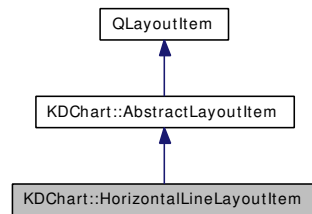
The documentation for this class was generated from the following files:

- [KDChartHeaderFooter.h](#)
- [KDChartHeaderFooter.cpp](#)

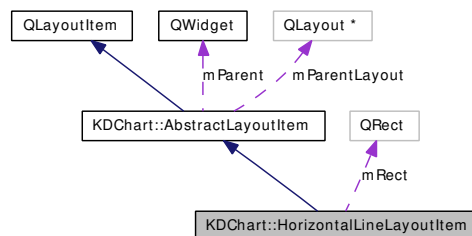
## 9.32 KDChart::HorizontalLineLayoutItem Class Reference

```
#include <KDChartLayoutItems.h>
```

Inheritance diagram for KDChart::HorizontalLineLayoutItem:



Collaboration diagram for KDChart::HorizontalLineLayoutItem:



### 9.32.1 Detailed Description

Layout item showing a horizontal line.

Definition at line 271 of file KDChartLayoutItems.h.

#### Public Member Functions

- virtual Qt::Orientations [expandingDirections](#) () const
- virtual QRect [geometry](#) () const
- [HorizontalLineLayoutItem](#) ()
- virtual bool [isEmpty](#) () const
- virtual QSize [maximumSize](#) () const
- virtual QSize [minimumSize](#) () const
- virtual void [paint](#) (QPainter \*)
- virtual void [paintAll](#) (QPainter &painter)
 

*Default impl: just call paint.*
- virtual void [paintCtx](#) ([PaintContext](#) \*context)
 

*Default impl: Paint the complete item using its layouted position and size.*
- QLayout \* [parentLayout](#) ()
- void [removeFromParentLayout](#) ()
- virtual void [setGeometry](#) (const QRect &r)
- void [setParentLayout](#) (QLayout \*lay)

- virtual void [setParentWidget](#) ([QWidget](#) \*widget)  
*Inform the item about its widget: This enables the item, to trigger that widget's update, whenever the size of the item's contents has changed.*
- virtual QSize [sizeHint](#) () const
- virtual void [sizeHintChanged](#) () const  
*Report changed size hint: ask the parent widget to recalculate the layout.*

## Protected Attributes

- [QWidget](#) \* [mParent](#)
- [QLayout](#) \* [mParentLayout](#)

## 9.32.2 Constructor & Destructor Documentation

### 9.32.2.1 [KDChart::HorizontalLineLayoutItem::HorizontalLineLayoutItem](#) ()

Definition at line 422 of file [KDChartLayoutItems.cpp](#).

```
423      : AbstractLayoutItem( Qt::AlignCenter )
424  {
425  }
```

## 9.32.3 Member Function Documentation

### 9.32.3.1 [Qt::Orientations](#) [KDChart::HorizontalLineLayoutItem::expandingDirections](#) () const [virtual]

Definition at line 427 of file [KDChartLayoutItems.cpp](#).

```
428  {
429      return Qt::Vertical|Qt::Horizontal; // Grow both vertically, and horizontally
430  }
```

### 9.32.3.2 [QRect](#) [KDChart::HorizontalLineLayoutItem::geometry](#) () const [virtual]

Definition at line 432 of file [KDChartLayoutItems.cpp](#).

```
433  {
434      return mRect;
435  }
```

### 9.32.3.3 [bool](#) [KDChart::HorizontalLineLayoutItem::isEmpty](#) () const [virtual]

Definition at line 437 of file [KDChartLayoutItems.cpp](#).

```
438  {
439      return false; // never empty, otherwise the layout item would not exist
440  }
```

#### 9.32.3.4 QSize KDCart::HorizontalLayoutItem::maximumSize () const [virtual]

Definition at line 442 of file KDCartLayoutItems.cpp.

```
443 {  
444     return QSize( QWIDGETSIZE_MAX, QWIDGETSIZE_MAX );  
445 }
```

#### 9.32.3.5 QSize KDCart::HorizontalLayoutItem::minimumSize () const [virtual]

Definition at line 447 of file KDCartLayoutItems.cpp.

```
448 {  
449     return QSize( 0, 0 );  
450 }
```

#### 9.32.3.6 void KDCart::HorizontalLayoutItem::paint (QPainter \*) [virtual]

Implements [KDCart::AbstractLayoutItem](#).

Definition at line 463 of file KDCartLayoutItems.cpp.

```
464 {  
465     if( !mRect.isValid() )  
466         return;  
467  
468     painter->drawLine( QPointF( mRect.left(), mRect.center().y() ),  
469                       QPointF( mRect.right(), mRect.center().y() ) );  
470 }
```

#### 9.32.3.7 void KDCart::AbstractLayoutItem::paintAll (QPainter & painter) [virtual, inherited]

Default impl: just call paint.

Derived classes like [KDCart::AbstractArea](#) are providing additional action here.

Reimplemented in [KDCart::AbstractArea](#), [KDCart::TextArea](#), and [KDCart::TernaryAxis](#).

Definition at line 69 of file KDCartLayoutItems.cpp.

References [KDCart::AbstractLayoutItem::paint\(\)](#).

```
70 {  
71     paint( &painter );  
72 }
```

#### 9.32.3.8 void KDCart::AbstractLayoutItem::paintCtx (PaintContext \* context) [virtual, inherited]

Default impl: Paint the complete item using its layouted position and size.

Reimplemented in [KDCart::CartesianAxis](#), and [KDCart::TernaryAxis](#).

Definition at line 77 of file KDChartLayoutItems.cpp.

References KDChart::AbstractLayoutItem::paint(), and KDChart::PaintContext::painter().

```

78 {
79     if( context )
80         paint( context->painter() );
81 }
```

### 9.32.3.9 QLayout\* KDChart::AbstractLayoutItem::parentLayout () [inherited]

Definition at line 76 of file KDChartLayoutItems.h.

```

77     {
78         return mParentLayout;
79     }
```

### 9.32.3.10 void KDChart::AbstractLayoutItem::removeFromParentLayout () [inherited]

Definition at line 80 of file KDChartLayoutItems.h.

Referenced by KDChart::Chart::takeCoordinatePlane().

```

81     {
82         if( mParentLayout ){
83             if( widget() )
84                 mParentLayout->removeWidget( widget() );
85             else
86                 mParentLayout->removeItem( this );
87         }
88     }
```

### 9.32.3.11 void KDChart::HorizontalLineLayoutItem::setGeometry (const QRect & r) [virtual]

Definition at line 452 of file KDChartLayoutItems.cpp.

```

453 {
454     mRect = r;
455 }
```

### 9.32.3.12 void KDChart::AbstractLayoutItem::setParentLayout (QLayout \* lay) [inherited]

Definition at line 72 of file KDChartLayoutItems.h.

```

73     {
74         mParentLayout = lay;
75     }
```



### 9.32.3.13 void KDChart::AbstractLayoutItem::setParentWidget (QWidget \* widget) [virtual, inherited]

Inform the item about its widget: This enables the item, to trigger that widget's update, whenever the size of the item's contents has changed.

Thus, you need to call setParentWidget on every item, that has a non-fixed size.

Definition at line 64 of file KDChartLayoutItems.cpp.

References KDChart::AbstractLayoutItem::mParent.

Referenced by KDChart::HeaderFooter::setParent(), and KDChart::AbstractCartesianDiagram::takeAxis().

```
65 {
66     mParent = widget;
67 }
```

### 9.32.3.14 QSize KDChart::HorizontalLayoutItem::sizeHint () const [virtual]

Definition at line 457 of file KDChartLayoutItems.cpp.

```
458 {
459     return QSize( -1, 3 ); // see qframe.cpp
460 }
```

### 9.32.3.15 void KDChart::AbstractLayoutItem::sizeHintChanged () const [virtual, inherited]

Report changed size hint: ask the parent widget to recalculate the layout.

Definition at line 86 of file KDChartLayoutItems.cpp.

References KDChart::AbstractLayoutItem::mParent.

Referenced by KDChart::TextLayoutItem::sizeHint().

```
87 {
88     // This is exactly like what QWidget::updateGeometry does.
89     qDebug("KDChart::AbstractLayoutItem::sizeHintChanged() called");
90     if( mParent ) {
91         if ( mParent->layout() )
92             mParent->layout()->invalidate();
93         else
94             QApplication::postEvent( mParent, new QEvent( QEvent::LayoutRequest ) );
95     }
96 }
```

## 9.32.4 Member Data Documentation

### 9.32.4.1 QWidget\* KDChart::AbstractLayoutItem::mParent [protected, inherited]

Definition at line 90 of file KDChartLayoutItems.h.

Referenced by KDChart::AbstractLayoutItem::setParentWidget(), KDChart::TextLayoutItem::setText(), KDChart::TextLayoutItem::setTextAttributes(), and KDChart::AbstractLayoutItem::sizeHintChanged().

**9.32.4.2** `QLayout*` [KDChart::AbstractLayoutItem::mParentLayout](#) [protected,  
inherited]

Definition at line 91 of file [KDChartLayoutItems.h](#).

Referenced by [KDChart::AutoSpacerLayoutItem::paint\(\)](#).

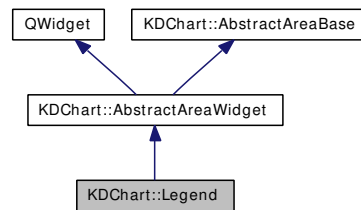
The documentation for this class was generated from the following files:

- [KDChartLayoutItems.h](#)
- [KDChartLayoutItems.cpp](#)

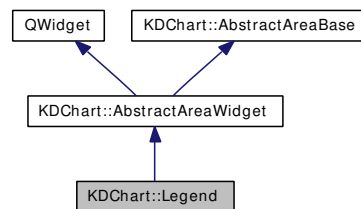
## 9.33 KDChart::Legend Class Reference

```
#include <KDChartLegend.h>
```

Inheritance diagram for KDChart::Legend:



Collaboration diagram for KDChart::Legend:



### 9.33.1 Detailed Description

[Legend](#) defines the interface for the legend drawing class.

[Legend](#) is the class for drawing legends for all kinds of diagrams ("chart types").

[Legend](#) is drawn on chart level, not per diagram, but you can have more than one legend per chart, using [KDChart::Chart::addLegend\(\)](#).

#### Note:

[Legend](#) is different from all other classes of KD [Chart](#), since it can be displayed outside of the Chart's area. If you want to, you can embed the legend into your own widget, or into another part of a bigger grid, into which you might have inserted the [Chart](#).

On the other hand, please note that you MUST call [Chart::addLegend](#) to get your legend positioned into the correct place of your chart - if you want to have the legend shown inside of the chart (that's probably true for most cases).

Definition at line 62 of file KDChartLegend.h.

### Public Types

- enum [LegendStyle](#) {  
[MarkersOnly](#) = 0,  
[LinesOnly](#) = 1,  
[MarkersAndLines](#) = 2 }

## Signals

- void [destroyedLegend](#) ([Legend](#) \*)
- void [positionChanged](#) ([AbstractAreaWidget](#) \*)
- void [propertiesChanged](#) ()

*Emitted upon change of a property of the [Legend](#) or any of its components.*

## Public Member Functions

- void [addDiagram](#) ([KDChart::AbstractDiagram](#) \*newDiagram)

*Add the given diagram to the legend.*

- Qt::Alignment [alignment](#) () const

*Returns the alignment of a non-floating legend.*

- void [alignToReferencePoint](#) (const [RelativePosition](#) &position)
- [BackgroundAttributes](#) [backgroundAttributes](#) () const
- QBrush [brush](#) (uint dataset) const
- const QMap< uint, QBrush > [brushes](#) () const
- virtual [Legend](#) \* [clone](#) () const

*Creates an exact copy of this legend.*

- bool [compare](#) (const [AbstractAreaBase](#) \*other) const

*Returns true if both areas have the same settings.*

- bool [compare](#) (const [Legend](#) \*other) const

*Returns true if both legends have the same settings.*

- [ConstDiagramList](#) [constDiagrams](#) () const

**Returns:**

*The list of diagrams associated with this coordinate plane.*

- uint [datasetCount](#) () const
- [KDChart::AbstractDiagram](#) \* [diagram](#) () const

*The first diagram of the legend or 0 if there was none added to the legend.*

- [DiagramList](#) [diagrams](#) () const

*The list of all diagrams associated with the legend.*

- const [RelativePosition](#) [floatingPosition](#) () const

*Returns the position of a floating legend.*

- virtual void [forceRebuild](#) ()

*Call this to trigger an unconditional re-building of the widget's internals.*

- [FrameAttributes](#) [frameAttributes](#) () const
- void [getFrameLeadings](#) (int &left, int &top, int &right, int &bottom) const
- [Legend](#) ([KDChart::AbstractDiagram](#) \*diagram, [QWidget](#) \*parent)
- [Legend](#) ([QWidget](#) \*parent=0)

- [LegendStyle legendStyle](#) () const
- const QMap< uint, [MarkerAttributes](#) > [markerAttributes](#) () const
- [MarkerAttributes markerAttributes](#) (uint dataset) const
- virtual QSize [minimumSizeHint](#) () const
- virtual void [needSizeHint](#) ()  
*Call this to trigger an conditional re-building of the widget's internals.*
- Qt::Orientation [orientation](#) () const
- virtual void [paint](#) (QPainter \*painter)  
*Overwrite this to paint the inner contents of your widget.*
- void [paintAll](#) (QPainter &painter)  
*Call paintAll, if you want the background and the frame to be drawn before the normal [paint\(\)](#) is invoked automatically.*
- virtual void [paintBackground](#) (QPainter &painter, const QRect &rectangle)
- virtual void [paintEvent](#) (QPaintEvent \*event)  
*Draws the background and frame, then calls [paint\(\)](#).*
- virtual void [paintFrame](#) (QPainter &painter, const QRect &rectangle)
- virtual void [paintIntoRect](#) (QPainter &painter, const QRect &rect)  
*Draws the background and frame, then calls [paint\(\)](#).*
- QPen [pen](#) (uint dataset) const
- const QMap< uint, QPen > [pens](#) () const
- [Position position](#) () const  
*Returns the position of a non-floating legend.*
- const [QWidget \\* referenceArea](#) () const  
*Returns the reference area, that is used for font size of title text, and for font size of the item texts, IF automatic area detection is set.*
- void [removeDiagram](#) ([KDChart::AbstractDiagram](#) \*oldDiagram)  
*Removes the diagram from the legend's list of diagrams.*
- void [removeDiagrams](#) ()  
*Removes all of the diagram from the legend's list of diagrams.*
- void [replaceDiagram](#) ([KDChart::AbstractDiagram](#) \*newDiagram, [KDChart::AbstractDiagram](#) \*old-Diagram=0)  
*Replaces the old diagram, or appends the new diagram, if there is none yet.*
- void [resetTexts](#) ()  
*Removes all legend texts that might have been set by [setText](#).*
- virtual void [resizeEvent](#) (QResizeEvent \*event)
- virtual void [resizeLayout](#) (const QSize &size)
- void [setAlignment](#) (Qt::Alignment)  
*Specify the alignment of a non-floating legend.*

- void [setBackgroundAttributes](#) (const [BackgroundAttributes](#) &a)
- void [setBrush](#) (uint dataset, const [QBrush](#) &brush)
- void [setBrushesFromDiagram](#) ([KDChart::AbstractDiagram](#) \*diagram)
- void [setColor](#) (uint dataset, const [QColor](#) &color)

*Note: there is no color() getter method, since setColor just sets a [QBrush](#) with the respective color, so the [brush\(\)](#) getter method is sufficient.*

- void [setDefaultColors](#) ()
- void [setDiagram](#) ([KDChart::AbstractDiagram](#) \*newDiagram)

*A convenience method doing the same as [replaceDiagram](#)( newDiagram, 0 );.*

- void [setFloatingPosition](#) (const [RelativePosition](#) &relativePosition)

*Specify the position and alignment of a floating legend.*

- void [setFrameAttributes](#) (const [FrameAttributes](#) &a)
- void [setLegendStyle](#) ([LegendStyle](#) style)
- void [setMarkerAttributes](#) (uint dataset, const [MarkerAttributes](#) &)

*Note that any sizes specified via [setMarkerAttributes](#) are ignored, unless you disable the automatic size calculation, by saying [setUseAutomaticMarkerSize](#)( false ).*

- void [setOrientation](#) ([Qt::Orientation](#) orientation)
- void [setPen](#) (uint dataset, const [QPen](#) &pen)
- void [setPosition](#) ([Position](#) position)

*Specify the position of a non-floating legend.*

- void [setRainbowColors](#) ()
- void [setReferenceArea](#) (const [QWidget](#) \*area)

*Specifies the reference area for font size of title text, and for font size of the item texts, IF automatic area detection is set.*

- void [setShowLines](#) (bool legendShowLines)
- void [setSpacing](#) (uint space)
- void [setSubduedColors](#) (bool ordered=false)
- void [setText](#) (uint dataset, const [QString](#) &text)
- void [setTextAttributes](#) (const [TextAttributes](#) &a)
- void [setTitleText](#) (const [QString](#) &text)
- void [setTitleTextAttributes](#) (const [TextAttributes](#) &a)
- void [setUseAutomaticMarkerSize](#) (bool useAutomaticMarkerSize)

*This option is on by default, it means that Marker sizes in the [Legend](#) will be the same as the font height used for their respective label texts.*

- virtual void [setVisible](#) (bool visible)
- bool [showLines](#) () const
- virtual [QSize](#) [sizeHint](#) () const
- uint [spacing](#) () const
- [QString](#) [text](#) (uint dataset) const
- [TextAttributes](#) [textAttributes](#) () const
- const [QMap](#)< uint, [QString](#) > [texts](#) () const
- [QString](#) [titleText](#) () const
- [TextAttributes](#) [titleTextAttributes](#) () const
- bool [useAutomaticMarkerSize](#) () const
- virtual [~Legend](#) ()

## Static Public Member Functions

- static void [paintBackgroundAttributes](#) (QPainter &painter, const QRect &rectangle, const [KDChart::BackgroundAttributes](#) &attributes)
- static void [paintFrameAttributes](#) (QPainter &painter, const QRect &rectangle, const [KDChart::FrameAttributes](#) &attributes)

## Protected Member Functions

- virtual QRect [areaGeometry](#) () const
- QRect [innerRect](#) () const
- virtual void [positionHasChanged](#) ()

## 9.33.2 Member Enumeration Documentation

### 9.33.2.1 enum [KDChart::Legend::LegendStyle](#)

Enumerator:

*MarkersOnly*

*LinesOnly*

*MarkersAndLines*

Definition at line 75 of file [KDChartLegend.h](#).

```

75             { MarkersOnly      = 0,
76               LinesOnly       = 1,
77               MarkersAndLines = 2 };

```

## 9.33.3 Constructor & Destructor Documentation

### 9.33.3.1 [Legend::Legend](#) ([QWidget](#) \* *parent* = 0) [explicit]

Definition at line 85 of file [KDChartLegend.cpp](#).

References [d](#).

Referenced by [clone\(\)](#).

```

85             :
86       AbstractAreaWidget( new Private(), parent )
87   {
88       d->referenceArea = parent;
89       init();
90   }

```

### 9.33.3.2 [Legend::Legend](#) ([KDChart::AbstractDiagram](#) \* *diagram*, [QWidget](#) \* *parent*) [explicit]

Definition at line 92 of file [KDChartLegend.cpp](#).

References [d](#), [diagram\(\)](#), and [setDiagram\(\)](#).

```

92                                     :
93     AbstractAreaWidget( new Private(), parent )
94 {
95     d->referenceArea = parent;
96     init();
97     setDiagram( diagram );
98 }

```

### 9.33.3.3 Legend::~~Legend () [virtual]

Definition at line 100 of file KDChartLegend.cpp.

References destroyedLegend().

```

101 {
102     emit destroyedLegend( this );
103 }

```

## 9.33.4 Member Function Documentation

### 9.33.4.1 void Legend::addDiagram (KDChart::AbstractDiagram \* newDiagram)

Add the given diagram to the legend.

#### Parameters:

*newDiagram* The diagram to add.

#### See also:

[diagram](#), [diagrams](#), [removeDiagram](#), [removeDiagrams](#), [replaceDiagram](#), [setDiagram](#)

Definition at line 337 of file KDChartLegend.cpp.

References d.

Referenced by replaceDiagram().

```

338 {
339     if ( newDiagram )
340     {
341         DiagramObserver* observer = new DiagramObserver( newDiagram, this );
342
343         DiagramObserver* oldObs = d->findObserverForDiagram( newDiagram );
344         if( oldObs ){
345             delete oldObs;
346             d->observers[ d->observers.indexOf( oldObs ) ] = observer;
347         }else{
348             d->observers.append( observer );
349         }
350         connect( observer, SIGNAL( diagramDestroyed(AbstractDiagram*) ),
351                 SLOT( resetDiagram(AbstractDiagram*) ) );
352         connect( observer, SIGNAL( diagramDataChanged(AbstractDiagram*) ),
353                 SLOT( setNeedRebuild() ) );
354         connect( observer, SIGNAL( diagramDataHidden(AbstractDiagram*) ),
355                 SLOT( setNeedRebuild() ) );
356         connect( observer, SIGNAL( diagramAttributesChanged(AbstractDiagram*) ),
357                 SLOT( setNeedRebuild() ) );

```



```

358         setNeedRebuild();
359     }
360 }

```

#### 9.33.4.2 Qt::Alignment Legend::alignment () const

Returns the alignment of a non-floating legend.

**See also:**

[setAlignment](#)

Definition at line 452 of file KDChartLegend.cpp.

References [d](#).

Referenced by [clone\(\)](#), and [compare\(\)](#).

```

453 {
454     return d->alignment;
455 }

```

#### 9.33.4.3 void AbstractAreaBase::alignToReferencePoint (const [RelativePosition](#) & position) [inherited]

Definition at line 90 of file KDChartAbstractAreaBase.cpp.

```

91 {
92     Q_UNUSED( position );
93     // PENDING(kalle) FIXME
94     qWarning( "Sorry, not implemented: void AbstractAreaBase::alignToReferencePoint( const RelativePosi
95 }

```

#### 9.33.4.4 QRect AbstractAreaWidget::areaGeometry () const [protected, virtual, inherited]

Implements [KDChart::AbstractAreaBase](#).

Definition at line 186 of file KDChartAbstractAreaWidget.cpp.

```

187 {
188     return geometry();
189 }

```

#### 9.33.4.5 [BackgroundAttributes](#) AbstractAreaBase::backgroundAttributes () const [inherited]

Definition at line 120 of file KDChartAbstractAreaBase.cpp.

References [d](#).

Referenced by [KDChart::AbstractAreaBase::compare\(\)](#), and [updateCommonBrush\(\)](#).

```

121 {
122     return d->backgroundAttributes;
123 }

```

#### 9.33.4.6 QBrush Legend::brush (uint *dataset*) const

Definition at line 558 of file KDChartLegend.cpp.

References d.

Referenced by setRainbowColors().

```

559 {
560     if( d->brushes.find( dataset ) != d->brushes.end() )
561         return d->brushes[ dataset ];
562     else
563         return d->modelBrushes[ dataset ];
564 }
```

#### 9.33.4.7 const QMap< uint, QBrush > Legend::brushes () const

Definition at line 566 of file KDChartLegend.cpp.

References d.

Referenced by compare().

```

567 {
568     return d->brushes;
569 }
```

#### 9.33.4.8 Legend \* Legend::clone () const [virtual]

Creates an exact copy of this legend.

Definition at line 204 of file KDChartLegend.cpp.

References alignment(), d, KDChart::AbstractAreaBase::frameAttributes(), Legend(), legendStyle(), position(), setAlignment(), KDChart::AbstractAreaBase::setFrameAttributes(), setLegendStyle(), setPosition(), setTextAttributes(), setTitleTextAttributes(), setUseAutomaticMarkerSize(), textAttributes(), titleTextAttributes(), and useAutomaticMarkerSize().

```

205 {
206     Legend* legend = new Legend( new Private( *d ), 0 );
207     legend->setTextAttributes( textAttributes() );
208     legend->setTitleTextAttributes( titleTextAttributes() );
209     legend->setFrameAttributes( frameAttributes() );
210     legend->setUseAutomaticMarkerSize( useAutomaticMarkerSize() );
211     legend->setPosition( position() );
212     legend->setAlignment( alignment() );
213     legend->setLegendStyle( legendStyle() );
214     return legend;
215 }
```

#### 9.33.4.9 bool AbstractAreaBase::compare (const AbstractAreaBase \* *other*) const [inherited]

Returns true if both areas have the same settings.

Definition at line 75 of file KDChartAbstractAreaBase.cpp.

References KDChart::AbstractAreaBase::backgroundAttributes(), and KDChart::AbstractAreaBase::frameAttributes().

```

76 {
77     if( other == this ) return true;
78     if( ! other ){
79         //qDebug() << "CartesianAxis::compare() cannot compare to Null pointer";
80         return false;
81     }
82     /*
83     qDebug() << "AbstractAreaBase:" << (frameAttributes() == other->frameAttributes())
84         << (backgroundAttributes() == other->backgroundAttributes()) << "\n";
85     */
86     return (frameAttributes() == other->frameAttributes()) &&
87         (backgroundAttributes() == other->backgroundAttributes());
88 }

```

#### 9.33.4.10 bool Legend::compare (const Legend \* other) const

Returns true if both legends have the same settings.

Definition at line 218 of file KDChartLegend.cpp.

References alignment(), brushes(), floatingPosition(), legendStyle(), markerAttributes(), orientation(), pens(), position(), showLines(), spacing(), textAttributes(), texts(), titleText(), titleTextAttributes(), and useAutomaticMarkerSize().

```

219 {
220     if( other == this ) return true;
221     if( ! other ){
222         //qDebug() << "Legend::compare() cannot compare to Null pointer";
223         return false;
224     }
225     /*
226     qDebug() << ( static_cast<const AbstractAreaBase*>(this)->compare( other ) );
227     qDebug() << (isVisible() == other->isVisible());
228     qDebug() << (position() == other->position());
229     qDebug() << (alignment() == other->alignment());
230     qDebug() << (floatingPosition() == other->floatingPosition());
231     qDebug() << (orientation() == other->orientation());
232     qDebug() << (showLines() == other->showLines());
233     qDebug() << (texts() == other->texts());
234     qDebug() << (brushes() == other->brushes());
235     qDebug() << (pens() == other->pens());
236     qDebug() << (markerAttributes() == other->markerAttributes());
237     qDebug() << (useAutomaticMarkerSize() == other->useAutomaticMarkerSize());
238     qDebug() << (textAttributes() == other->textAttributes());
239     qDebug() << (titleText() == other->titleText());
240     qDebug() << (titleTextAttributes() == other->titleTextAttributes());
241     qDebug() << (spacing() == other->spacing());
242     qDebug() << (legendStyle() == other->legendStyle());
243     */
244     return ( static_cast<const AbstractAreaBase*>(this)->compare( other ) ) &&
245         (isVisible() == other->isVisible()) &&
246         (position() == other->position()) &&
247         (alignment() == other->alignment()) &&
248         (floatingPosition() == other->floatingPosition()) &&
249         (orientation() == other->orientation()) &&
250         (showLines() == other->showLines()) &&
251         (texts() == other->texts()) &&
252         (brushes() == other->brushes()) &&
253         (pens() == other->pens()) &&
254         (markerAttributes() == other->markerAttributes()) &&

```

```

255         (useAutomaticMarkerSize() == other->useAutomaticMarkerSize()) &&
256         (textAttributes() == other->textAttributes()) &&
257         (titleText() == other->titleText()) &&
258         (titleTextAttributes() == other->titleTextAttributes()) &&
259         (spacing() == other->spacing()) &&
260         (legendStyle() == other->legendStyle());
261     }

```

#### 9.33.4.11 [ConstDiagramList](#) Legend::constDiagrams () const

##### Returns:

The list of diagrams associated with this coordinate plane.

Definition at line 329 of file KDChartLegend.cpp.

References [d](#).

```

330 {
331     ConstDiagramList list;
332     for (int i = 0; i < d->observers.size(); ++i)
333         list << d->observers.at(i)->diagram();
334     return list;
335 }

```

#### 9.33.4.12 [uint](#) Legend::datasetCount () const

Definition at line 286 of file KDChartLegend.cpp.

References [d](#).

```

287 {
288     int modelLabelsCount = 0;
289     int modelBrushesCount = 0;
290     for (int i = 0; i < d->observers.size(); ++i) {
291         DiagramObserver * obs = d->observers.at(i);
292         modelLabelsCount += obs->diagram()->datasetLabels().count();
293         modelBrushesCount += obs->diagram()->datasetBrushes().count();
294     }
295     Q_ASSERT( modelLabelsCount == modelBrushesCount );
296     return modelLabelsCount;
297 }

```

#### 9.33.4.13 [void](#) KDChart::Legend::destroyedLegend ([Legend](#) \*) [signal]

Referenced by [~Legend\(\)](#).

#### 9.33.4.14 [AbstractDiagram](#) \* Legend::diagram () const

The first diagram of the legend or 0 if there was none added to the legend.

##### Returns:

The first diagram of the legend or 0.

**See also:**

[diagrams](#), [addDiagram](#), [removeDiagram](#), [removeDiagrams](#), [replaceDiagram](#), [setDiagram](#)

Definition at line 314 of file KDChartLegend.cpp.

References [d](#).

Referenced by [Legend\(\)](#), [paint\(\)](#), and [setBrushesFromDiagram\(\)](#).

```
315 {  
316     if( d->observers.isEmpty() )  
317         return 0;  
318     return d->observers.first()->diagram();  
319 }
```

**9.33.4.15 [DiagramList](#) Legend::diagrams () const**

The list of all diagrams associated with the legend.

**Returns:**

The list of all diagrams associated with the legend.

**See also:**

[diagram](#), [addDiagram](#), [removeDiagram](#), [removeDiagrams](#), [replaceDiagram](#), [setDiagram](#)

Definition at line 321 of file KDChartLegend.cpp.

References [d](#).

```
322 {  
323     DiagramList list;  
324     for (int i = 0; i < d->observers.size(); ++i)  
325         list << d->observers.at(i)->diagram();  
326     return list;  
327 }
```

**9.33.4.16 [const RelativePosition](#) Legend::floatingPosition () const**

Returns the position of a floating legend.

**See also:**

[setFloatingPosition](#)

Definition at line 466 of file KDChartLegend.cpp.

References [d](#).

Referenced by [compare\(\)](#), and [KDChart::Chart::reLayoutFloatingLegends\(\)](#).

```
467 {  
468     return d->relativePosition;  
469 }
```

**9.33.4.17 void Legend::forceRebuild () [virtual]**

Call this to trigger an unconditional re-building of the widget's internals.

Reimplemented from [KDChart::AbstractAreaWidget](#).

Definition at line 670 of file KDChartLegend.cpp.

Referenced by [resizeEvent\(\)](#).

```

671 {
672 #ifdef DEBUG_LEGEND_PAINT
673     qDebug() << "entering Legend::forceRebuild()";
674 #endif
675     //setSpacing(d->layout->spacing());
676     buildLegend();
677 #ifdef DEBUG_LEGEND_PAINT
678     qDebug() << "leaving Legend::forceRebuild()";
679 #endif
680 }
```

**9.33.4.18 [FrameAttributes](#) AbstractAreaBase::frameAttributes () const [inherited]**

Definition at line 106 of file KDChartAbstractAreaBase.cpp.

References [d](#).

Referenced by [clone\(\)](#), [KDChart::AbstractAreaBase::compare\(\)](#), and [updateCommonBrush\(\)](#).

```

107 {
108     return d->frameAttributes;
109 }
```

**9.33.4.19 void AbstractAreaBase::getFrameLeadings (int & *left*, int & *top*, int & *right*, int & *bottom*) const [inherited]**

Definition at line 212 of file KDChartAbstractAreaBase.cpp.

References [d](#).

Referenced by [KDChart::AbstractAreaBase::innerRect\(\)](#), and [KDChart::AbstractAreaWidget::paintAll\(\)](#).

```

213 {
214     if( d && d->frameAttributes.isVisible() ){
215         const int padding = qMax( d->frameAttributes.padding(), 0 );
216         left   = padding;
217         top    = padding;
218         right  = padding;
219         bottom = padding;
220     }else{
221         left   = 0;
222         top    = 0;
223         right  = 0;
224         bottom = 0;
225     }
226 }
```

**9.33.4.20 QRect AbstractAreaBase::innerRect () const** [protected, inherited]

Definition at line 228 of file KDChartAbstractAreaBase.cpp.

References KDChart::AbstractAreaBase::areaGeometry(), and KDChart::AbstractAreaBase::getFrameLeadings().

Referenced by KDChart::TextArea::paintAll(), and KDChart::AbstractArea::paintAll().

```

229 {
230     int left;
231     int top;
232     int right;
233     int bottom;
234     getFrameLeadings( left, top, right, bottom );
235     return
236         QRect( QPoint(0,0), areaGeometry().size() )
237             .adjusted( left, top, -right, -bottom );
238 }
```

**9.33.4.21 Legend::LegendStyle Legend::legendStyle () const**

Definition at line 196 of file KDChartLegend.cpp.

References d.

Referenced by clone(), and compare().

```

197 {
198     return d->legendStyle;
199 }
```

**9.33.4.22 const QMap< uint, MarkerAttributes > Legend::markerAttributes () const**

Definition at line 628 of file KDChartLegend.cpp.

References d.

Referenced by compare(), and setMarkerAttributes().

```

629 {
630     return d->markerAttributes;
631 }
```

**9.33.4.23 MarkerAttributes Legend::markerAttributes (uint *dataset*) const**

Definition at line 619 of file KDChartLegend.cpp.

References d.

Referenced by compare().

```

620 {
621     if( d->markerAttributes.find( dataset ) != d->markerAttributes.end() )
622         return d->markerAttributes[ dataset ];
623     else if ( static_cast<uint>( d->modelMarkers.count() ) > dataset )
624         return d->modelMarkers[ dataset ];
625     return MarkerAttributes();
626 }
```

#### 9.33.4.24 QSize Legend::minimumSizeHint () const [virtual]

Definition at line 143 of file KDChartLegend.cpp.

References [sizeHint\(\)](#).

```
144 {  
145     return sizeHint();  
146 }
```

#### 9.33.4.25 void Legend::needSizeHint () [virtual]

Call this to trigger an conditional re-building of the widget's internals.

e.g. [AbstractAreaWidget](#) call this, before calling [layout\(\)](#)->[setGeometry\(\)](#)

Reimplemented from [KDChart::AbstractAreaWidget](#).

Definition at line 161 of file KDChartLegend.cpp.

```
162 {  
163     // Re-build the Legend's content, if it has not been build yet,  
164     // or if the Legend's geometry has changed, resp.  
165     buildLegend();  
166 }
```

#### 9.33.4.26 Qt::Orientation Legend::orientation () const

Definition at line 479 of file KDChartLegend.cpp.

References [d](#).

Referenced by [compare\(\)](#).

```
480 {  
481     return d->orientation;  
482 }
```

#### 9.33.4.27 void Legend::paint (QPainter \*painter) [virtual]

Overwrite this to paint the inner contents of your widget.

##### Note:

When overriding this method, please let your widget draw itself at the top/left corner of the painter. You should call [rect\(\)](#) (or [width\(\)](#), [height\(\)](#), resp.) to find the drawable area's size: While the [paint\(\)](#) method is being executed the frame of the widget is outside of its [rect\(\)](#), so you can use all of [rect\(\)](#) for your custom drawing!

##### See also:

[paint](#), [paintIntoRect](#)

Implements [KDChart::AbstractAreaWidget](#).

Definition at line 264 of file KDChartLegend.cpp.

References [d](#), [diagram\(\)](#), and [KDChart::AbstractLayoutItem::paint\(\)](#).



```

265 {
266     #ifdef DEBUG_LEGEND_PAINT
267         qDebug() << "entering Legend::paint( QPainter* painter )";
268     #endif
269     // rule: We do not show a legend, if there is no diagram.
270     if( ! diagram() ) return;
271
272     // re-calculate/adjust the Legend's internal layout and contents, if needed:
273     //buildLegend();
274
275     // PENDING(kalle) Support palette
276
277     Q_FOREACH( KDCart::AbstractLayoutItem* layoutItem, d->layoutItems ) {
278         layoutItem->paint( painter );
279     }
280 #ifdef DEBUG_LEGEND_PAINT
281     qDebug() << "leaving Legend::paint( QPainter* painter )";
282 #endif
283 }

```

#### 9.33.4.28 void AbstractAreaWidget::paintAll (QPainter & painter) [inherited]

Call `paintAll`, if you want the background and the frame to be drawn before the normal `paint()` is invoked automatically.

Definition at line 145 of file `KDCartAbstractAreaWidget.cpp`.

References `KDCart::AbstractAreaBase::getFrameLeadings()`, `KDCart::AbstractAreaWidget::paint()`, `KDCart::AbstractAreaBase::paintBackground()`, and `KDCart::AbstractAreaBase::paintFrame()`.

Referenced by `KDCart::AbstractAreaWidget::paintEvent()`, and `KDCart::AbstractAreaWidget::paint-IntoRect()`.

```

146 {
147     //qDebug() << "AbstractAreaWidget::paintAll() called";
148
149     // Paint the background and frame
150     paintBackground( painter, QRect(QPoint(0, 0), size() ) );
151     paintFrame( painter, QRect(QPoint(0, 0), size() ) );
152
153     /*
154     we do not call setContentsMargins() now,
155     but we call resizeLayout() whenever the size or the frame has changed
156
157     // adjust the widget's content margins,
158     // to be sure all content gets calculated
159     // to fit into the inner rectangle
160     const QRect oldGeometry( areaGeometry() );
161     const QRect inner( innerRect() );
162     //qDebug() << "areaGeometry():" << oldGeometry
163     // << " contentsRect():" << contentsRect() << " inner:" << inner;
164     if( contentsRect() != inner ){
165         //qDebug() << "old contentsRect():" << contentsRect() << " new innerRect:" << inner;
166         setContentsMargins(
167             inner.left(),
168             inner.top(),
169             oldGeometry.width() -inner.width()-1,
170             oldGeometry.height()-inner.height()-1 );
171         //forceRebuild();
172     }
173     */
174     int left;
175     int top;
176     int right;

```

```

177     int bottom;
178     setFrameLeadings( left, top, right, bottom );
179     const QPoint translation( left, top );
180     painter.translate( translation );
181     paint( &painter );
182     painter.translate( -translation.x(), -translation.y() );
183     //qDebug() << "AbstractAreaWidget::paintAll() done.";
184 }

```

#### 9.33.4.29 void AbstractAreaBase::paintBackground (QPainter & *painter*, const QRect & *rectangle*) [virtual, inherited]

Definition at line 196 of file KDChartAbstractAreaBase.cpp.

References `d`, and `KDChart::AbstractAreaBase::paintBackgroundAttributes()`.

Referenced by `KDChart::TextArea::paintAll()`, `KDChart::AbstractAreaWidget::paintAll()`, and `KDChart::AbstractArea::paintAll()`.

```

197 {
198     Q_ASSERT_X ( d != 0, "AbstractAreaBase::paintBackground()",
199                 "Private class was not initialized!" );
200     paintBackgroundAttributes( painter, rect, d->backgroundAttributes );
201 }

```

#### 9.33.4.30 void AbstractAreaBase::paintBackgroundAttributes (QPainter & *painter*, const QRect & *rectangle*, const [KDChart::BackgroundAttributes](#) & *attributes*) [static, inherited]

Definition at line 127 of file KDChartAbstractAreaBase.cpp.

References `KDChart::BackgroundAttributes::BackgroundPixmapModeCentered`, `KDChart::BackgroundAttributes::BackgroundPixmapModeNone`, `KDChart::BackgroundAttributes::BackgroundPixmapModeScaled`, `KDChart::BackgroundAttributes::BackgroundPixmapModeStretched`, `KDChart::BackgroundAttributes::brush()`, `KDChart::BackgroundAttributes::isVisible()`, `KDChart::BackgroundAttributes::pixmap()`, and `KDChart::BackgroundAttributes::pixmapMode()`.

Referenced by `KDChart::AbstractAreaBase::paintBackground()`.

```

129 {
130     if( !attributes.isVisible() ) return;
131
132     /* first draw the brush (may contain a pixmap) */
133     if( Qt::NoBrush != attributes.brush().style() ) {
134         KDChart::PainterSaver painterSaver( &painter );
135         painter.setPen( Qt::NoPen );
136         const QPointF newTopLeft( painter.deviceMatrix().map( rect.topLeft() ) );
137         painter.setBrushOrigin( newTopLeft );
138         painter.setBrush( attributes.brush() );
139         painter.drawRect( rect.adjusted( 0, 0, -1, -1 ) );
140     }
141     /* next draw the backPixmap over the brush */
142     if( !attributes.pixmap().isNull() &&
143         attributes.pixmapMode() != BackgroundAttributes::BackgroundPixmapModeNone ) {
144         QPointF ol = rect.topLeft();
145         if( BackgroundAttributes::BackgroundPixmapModeCentered == attributes.pixmapMode() )
146         {
147             ol.setX( rect.center().x() - attributes.pixmap().width() / 2 );
148             ol.setY( rect.center().y() - attributes.pixmap().height() / 2 );

```

```

149         painter.drawPixmap( ol, attributes.pixmap() );
150     } else {
151         QMatrix m;
152         double zW = (double)rect.width() / (double)attributes.pixmap().width();
153         double zH = (double)rect.height() / (double)attributes.pixmap().height();
154         switch( attributes.pixmapMode() ) {
155             case BackgroundAttributes::BackgroundPixmapModeScaled:
156                 {
157                     double z;
158                     z = qMin( zW, zH );
159                     m.scale( z, z );
160                 }
161                 break;
162             case BackgroundAttributes::BackgroundPixmapModeStretched:
163                 m.scale( zW, zH );
164                 break;
165             default:
166                 ; // Cannot happen, previously checked
167         }
168         QPixmap pm = attributes.pixmap().transformed( m );
169         ol.setX( rect.center().x() - pm.width() / 2 );
170         ol.setY( rect.center().y() - pm.height() / 2 );
171         painter.drawPixmap( ol, pm );
172     }
173 }
174 }

```

#### 9.33.4.31 void AbstractAreaWidget::paintEvent (QPaintEvent \* *event*) [virtual, inherited]

Draws the background and frame, then calls [paint\(\)](#).

In most cases there is no need to overwrite this method in a derived class, but you would overwrite [paint\(\)](#) instead.

**See also:**

[paint](#)

Definition at line 99 of file KDChartAbstractAreaWidget.cpp.

References [d](#), and [KDChart::AbstractAreaWidget::paintAll\(\)](#).

```

100 {
101     Q_UNUSED( event );
102     QPainter painter( this );
103     if( size() != d->currentLayoutSize ){
104         d->resizeLayout( this, size() );
105     }
106     paintAll( painter );
107 }

```

#### 9.33.4.32 void AbstractAreaBase::paintFrame (QPainter & *painter*, const QRect & *rectangle*) [virtual, inherited]

Definition at line 204 of file KDChartAbstractAreaBase.cpp.

References [d](#), and [KDChart::AbstractAreaBase::paintFrameAttributes\(\)](#).

Referenced by [KDChart::TextArea::paintAll\(\)](#), [KDChart::AbstractAreaWidget::paintAll\(\)](#), and [KDChart::AbstractArea::paintAll\(\)](#).

```

205 {
206     Q_ASSERT_X ( d != 0, "AbstractAreaBase::paintFrame()",
207                 "Private class was not initialized!" );
208     paintFrameAttributes( painter, rect, d->frameAttributes );
209 }

```

#### 9.33.4.33 void AbstractAreaBase::paintFrameAttributes (QPainter & *painter*, const QRect & *rectangle*, const [KDChart::FrameAttributes](#) & *attributes*) [static, inherited]

Definition at line 177 of file `KDChartAbstractAreaBase.cpp`.

References `KDChart::FrameAttributes::isVisible()`, and `KDChart::FrameAttributes::pen()`.

Referenced by `KDChart::AbstractAreaBase::paintFrame()`.

```

179 {
180
181     if( !attributes.isVisible() ) return;
182
183     // Note: We set the brush to NoBrush explicitly here.
184     //       Otherwise we might get a filled rectangle, so any
185     //       previously drawn background would be overwritten by that area.
186
187     const QPen    oldPen(    painter.pen() );
188     const QBrush  oldBrush( painter.brush() );
189     painter.setPen(    attributes.pen() );
190     painter.setBrush( Qt::NoBrush );
191     painter.drawRect( rect.adjusted( 0, 0, -1, -1 ) );
192     painter.setBrush( oldBrush );
193     painter.setPen(    oldPen );
194 }

```

#### 9.33.4.34 void AbstractAreaWidget::paintIntoRect (QPainter & *painter*, const QRect & *rect*) [virtual, inherited]

Draws the background and frame, then calls [paint\(\)](#).

In most cases there is no need to overwrite this method in a derived class, but you would overwrite [paint\(\)](#) instead.

Definition at line 109 of file `KDChartAbstractAreaWidget.cpp`.

References `d`, and `KDChart::AbstractAreaWidget::paintAll()`.

Referenced by `KDChart::Chart::paint()`.

```

110 {
111     //qDebug() << "AbstractAreaWidget::paintIntoRect() called rect=" << rect;
112
113     if( rect.isEmpty() ) return;
114
115     d->resizeLayout( this, rect.size() );
116
117     const QPoint translation( rect.topLeft() );
118     painter.translate( translation );
119     paintAll( painter );
120     painter.translate( -translation.x(), -translation.y() );
121
122     /*
123     // make sure, the contents of the widget have been set up,
124     // so we get a usefull geometry:

```

```

125     needSizeHint();
126
127     const QRect oldGeometry( layout()->geometry() );
128     const QRect newGeo( QPoint(0,0), rect.size() );
129     const bool mustChangeGeo = layout() && oldGeometry != newGeo;
130     if( mustChangeGeo )
131         layout()->setGeometry( newGeo );
132     painter.translate( rect.left(), rect.top() );
133     paintAll( painter );
134     painter.translate( -rect.left(), -rect.top() );
135     if( mustChangeGeo )
136         layout()->setGeometry( oldGeometry );
137 */
138 }

```

#### 9.33.4.35 QPen Legend::pen (uint *dataset*) const

Definition at line 597 of file KDChartLegend.cpp.

References d.

```

598 {
599     if( d->pens.find( dataset ) != d->pens.end() )
600         return d->pens[dataset];
601     else
602         return d->modelPens[ dataset ];
603 }

```

#### 9.33.4.36 const QMap< uint, QPen > Legend::pens () const

Definition at line 605 of file KDChartLegend.cpp.

References d.

Referenced by compare().

```

606 {
607     return d->pens;
608 }

```

#### 9.33.4.37 [Position](#) Legend::position () const

Returns the position of a non-floating legend.

See also:

[setPosition](#)

Definition at line 439 of file KDChartLegend.cpp.

References d.

Referenced by clone(), compare(), KDChart::Chart::reLayoutFloatingLegends(), and setPosition().

```

440 {
441     return d->position;
442 }

```

**9.33.4.38** void **KDChart::AbstractAreaWidget::positionChanged** ([AbstractAreaWidget \\*](#))  
[signal, inherited]

Referenced by `KDChart::AbstractAreaWidget::positionHasChanged()`.

**9.33.4.39** void **AbstractAreaWidget::positionHasChanged** () [protected, virtual, inherited]

Reimplemented from [KDChart::AbstractAreaBase](#).

Definition at line 191 of file `KDChartAbstractAreaWidget.cpp`.

References `KDChart::AbstractAreaWidget::positionChanged()`.

```
192 {
193     emit positionChanged( this );
194 }
```

**9.33.4.40** void **KDChart::Legend::propertiesChanged** () [signal]

Emitted upon change of a property of the [Legend](#) or any of its components.

**9.33.4.41** const [QWidget \\*](#) **Legend::referenceArea** () const

Returns the reference area, that is used for font size of title text, and for font size of the item texts, IF automatic area detection is set.

**See also:**

[setReferenceArea](#)

Definition at line 307 of file `KDChartLegend.cpp`.

References `d`.

```
308 {
309     //qDebug() << d->referenceArea;
310     return (d->referenceArea ? d->referenceArea : static_cast<const QWidget*>(parent()));
311 }
```

**9.33.4.42** void **Legend::removeDiagram** ([KDChart::AbstractDiagram \\*](#) *oldDiagram*)

Removes the diagram from the legend's list of diagrams.

**See also:**

[diagram](#), [diagrams](#), [addDiagram](#), [removeDiagrams](#), [replaceDiagram](#), [setDiagram](#)

Definition at line 362 of file `KDChartLegend.cpp`.

References `d`.

Referenced by `removeDiagrams()`, and `replaceDiagram()`.

```

363 {
364     if( oldDiagram ){
365         DiagramObserver* oldObs = d->findObserverForDiagram( oldDiagram );
366         if( oldObs ){
367             qDebug() << "before delete oldObs;";
368             delete oldObs;
369             qDebug() << "after delete oldObs;";
370             d->observers.removeAt( d->observers.indexOf( oldObs ) );
371             qDebug() << "after d->observers.removeAt() ";
372         }
373         setNeedRebuild();
374     }
375 }

```

#### 9.33.4.43 void Legend::removeDiagrams ()

Removes all of the diagram from the legend's list of diagrams.

See also:

[diagram](#), [diagrams](#), [addDiagram](#), [removeDiagram](#), [replaceDiagram](#), [setDiagram](#)

Definition at line 377 of file KDChartLegend.cpp.

References [d](#), and [removeDiagram\(\)](#).

```

378 {
379     for (int i = 0; i < d->observers.size(); ++i)
380         removeDiagram( d->observers.at(i)->diagram() );
381 }

```

#### 9.33.4.44 void Legend::replaceDiagram (KDChart::AbstractDiagram \* newDiagram, KDChart::AbstractDiagram \* oldDiagram = 0)

Replaces the old diagram, or appends the new diagram, if there is none yet.

Parameters:

***newDiagram*** The diagram to be used instead of the old one. If this parameter is zero, the first diagram will just be removed.

***oldDiagram*** The diagram to be removed by the new one. This diagram will be deleted automatically. If the parameter is omitted, the very first diagram will be replaced. In case, there was no diagram yet, the new diagram will just be added.

See also:

[diagram](#), [diagrams](#), [addDiagram](#), [removeDiagram](#), [removeDiagrams](#), [setDiagram](#)

Definition at line 383 of file KDChartLegend.cpp.

References [addDiagram\(\)](#), [d](#), and [removeDiagram\(\)](#).

Referenced by [setDiagram\(\)](#).

```

385 {
386     KDChart::AbstractDiagram* old = oldDiagram;

```

```

387     if( ! d->observers.isEmpty() && ! old ){
388         old = d->observers.first()->diagram();
389         if( ! old )
390             d->observers.removeFirst(); // first entry had a 0 diagram
391     }
392     if( old )
393         removeDiagram( old );
394     if( newDiagram )
395         addDiagram( newDiagram );
396 }

```

#### 9.33.4.45 void Legend::resetTexts ()

Removes all legend texts that might have been set by setText.

This resets the [Legend](#) to default behaviour: Texts are created automatically.

Definition at line 514 of file KDChartLegend.cpp.

References [d](#).

```

515 {
516     if( ! d->texts.count() ) return;
517     d->texts.clear();
518     setNeedRebuild();
519 }

```

#### 9.33.4.46 void Legend::resizeEvent (QResizeEvent \* event) [virtual]

Definition at line 773 of file KDChartLegend.cpp.

References [forceRebuild\(\)](#), and [sizeHint\(\)](#).

```

774 {
775     Q_UNUSED( event );
776 #ifdef DEBUG_LEGEND_PAINT
777     qDebug() << "Legend::resizeEvent() called";
778 #endif
779     forceRebuild();
780     sizeHint();
781     QTimer::singleShot(0, this, SLOT(emitPositionChanged()));
782 }

```

#### 9.33.4.47 void Legend::resizeLayout (const QSize & size) [virtual]

Reimplemented from [KDChart::AbstractAreaWidget](#).

Definition at line 168 of file KDChartLegend.cpp.

References [d](#).

```

169 {
170 #ifdef DEBUG_LEGEND_PAINT
171     qDebug() << "Legend::resizeLayout started";
172 #endif
173     if( d->layout ){
174         d->layout->setGeometry( QRect(QPoint(0,0), size) );
175         activateTheLayout();

```



```
176     }
177 #ifdef DEBUG_LEGEND_PAINT
178     qDebug() << "Legend::resizeLayout done";
179 #endif
180 }
```

#### 9.33.4.48 void Legend::setAlignment (Qt::Alignment)

Specify the alignment of a non-floating legend.

Use setFloatingPosition to set position and alignment if your legend is floating.

See also:

[alignment](#), [setPosition](#), [setFloatingPosition](#)

Definition at line 444 of file KDChartLegend.cpp.

References [d](#).

Referenced by [clone\(\)](#).

```
445 {
446     if( d->alignment == alignment )
447         return;
448     d->alignment = alignment;
449     emitPositionChanged();
450 }
```

#### 9.33.4.49 void AbstractAreaBase::setBackgroundAttributes (const [BackgroundAttributes](#) & a) [inherited]

Definition at line 111 of file KDChartAbstractAreaBase.cpp.

References [d](#), and [KDChart::AbstractAreaBase::positionHasChanged\(\)](#).

```
112 {
113     if( d->backgroundAttributes == a )
114         return;
115
116     d->backgroundAttributes = a;
117     positionHasChanged();
118 }
```

#### 9.33.4.50 void Legend::setBrush (uint *dataset*, const QBrush & *brush*)

Definition at line 550 of file KDChartLegend.cpp.

References [d](#).

```
551 {
552     if( d->brushes[ dataset ] == brush ) return;
553     d->brushes[ dataset ] = brush;
554     setNeedRebuild();
555     update();
556 }
```

### 9.33.4.51 void Legend::setBrushesFromDiagram (KDChart::AbstractDiagram \* *diagram*)

Definition at line 572 of file KDChartLegend.cpp.

References [d](#), [KDChart::AbstractDiagram::datasetBrushes\(\)](#), and [diagram\(\)](#).

```

573 {
574     bool bChangesDone = false;
575     QList<QBrush> datasetBrushes = diagram->datasetBrushes();
576     for( int i = 0; i < datasetBrushes.count(); i++ ){
577         if( d->brushes[ i ] != datasetBrushes[ i ] ){
578             d->brushes[ i ] = datasetBrushes[ i ];
579             bChangesDone = true;
580         }
581     }
582     if( bChangesDone ) {
583         setNeedRebuild();
584         update();
585     }
586 }
```

### 9.33.4.52 void Legend::setColor (uint *dataset*, const QColor & *color*)

Note: there is no color() getter method, since setColor just sets a QBrush with the respective color, so the [brush\(\)](#) getter method is sufficient.

Definition at line 542 of file KDChartLegend.cpp.

References [d](#).

Referenced by [setDefaultColors\(\)](#), [setRainbowColors\(\)](#), and [setSubduedColors\(\)](#).

```

543 {
544     if( d->brushes[ dataset ] == color ) return;
545     d->brushes[ dataset ] = color;
546     setNeedRebuild();
547     update();
548 }
```

### 9.33.4.53 void Legend::setDefaultColors ()

Definition at line 695 of file KDChartLegend.cpp.

References [setColor\(\)](#).

```

696 {
697     setColor( 0, Qt::red );
698     setColor( 1, Qt::green );
699     setColor( 2, Qt::blue );
700     setColor( 3, Qt::cyan );
701     setColor( 4, Qt::magenta );
702     setColor( 5, Qt::yellow );
703     setColor( 6, Qt::darkRed );
704     setColor( 7, Qt::darkGreen );
705     setColor( 8, Qt::darkBlue );
706     setColor( 9, Qt::darkCyan );
707     setColor( 10, Qt::darkMagenta );
708     setColor( 11, Qt::darkYellow );
709 }
```

**9.33.4.54 void Legend::setDiagram (KDChart::AbstractDiagram \* newDiagram)**

A convenience method doing the same as `replaceDiagram( newDiagram, 0 );`.

Replaces the first diagram by the given diagram. If the legend's list of diagram is empty the given diagram is added to the list.

**See also:**

[diagram](#), [diagrams](#), [addDiagram](#), [removeDiagram](#), [removeDiagrams](#), [replaceDiagram](#)

Definition at line 398 of file `KDChartLegend.cpp`.

References `replaceDiagram()`.

Referenced by `KDChart::Widget::addLegend()`, `Legend()`, `KDChart::Widget::replaceLegend()`, and `KDChart::Widget::setType()`.

```
399 {
400     replaceDiagram( newDiagram );
401 }
```

**9.33.4.55 void Legend::setFloatingPosition (const RelativePosition & relativePosition)**

Specify the position and alignment of a floating legend.

Use `setPosition` and `setAlignment` to set position and alignment if your legend is non-floating.

**Note:**

When `setFloatingPosition` is called, the Legend's position value is set to `KDChart::Position::Floating` automatically.

If your [Chart](#) is pointed to by `m_chart`, you could have the floating legend aligned exactly to the chart's coordinate plane's top-right corner with the following commands:

```
KDChart::RelativePosition relativePosition;
relativePosition.setReferenceArea( m_chart->coordinatePlane() );
relativePosition.setReferencePosition( Position::NorthEast );
relativePosition.setAlignment( Qt::AlignTop | Qt::AlignRight );
relativePosition.setHorizontalPadding(
    KDChart::Measure( -1.0, KDChartEnums::MeasureCalculationModeAbsolute ) );
relativePosition.setVerticalPadding(
    KDChart::Measure( 0.0, KDChartEnums::MeasureCalculationModeAbsolute ) );
m_legend->setFloatingPosition( relativePosition );
```

To have the legend positioned at a fixed point, measured from the `QPainter`'s top left corner, you could use the following code code:

```
KDChart::RelativePosition relativePosition;
relativePosition.setReferencePoints( PositionPoints( QPointF( 0.0, 0.0 ) ) );
relativePosition.setReferencePosition( Position::NorthWest );
relativePosition.setAlignment( Qt::AlignTop | Qt::AlignLeft );
relativePosition.setHorizontalPadding(
    KDChart::Measure( 4.0, KDChartEnums::MeasureCalculationModeAbsolute ) );
relativePosition.setVerticalPadding(
    KDChart::Measure( 4.0, KDChartEnums::MeasureCalculationModeAbsolute ) );
m_legend->setFloatingPosition( relativePosition );
```

Actually that's exactly the code KD [Chart](#) is using as default position for any floating legends, so if you just say `setPosition( KDChart::Position::Floating )` without calling `setFloatingPosition` your legend will be positioned at point 4/4.

See also:

[setPosition](#), [setAlignment](#)

Definition at line 457 of file `KDChartLegend.cpp`.

References `d`, and `KDChart::Position::Floating`.

```

458 {
459     d->position = Position::Floating;
460     if( d->relativePosition != relativePosition ){
461         d->relativePosition = relativePosition;
462         emitPositionChanged();
463     }
464 }
```

#### 9.33.4.56 void AbstractAreaBase::setFrameAttributes (const [FrameAttributes](#) & a) [inherited]

Definition at line 97 of file `KDChartAbstractAreaBase.cpp`.

References `d`, and `KDChart::AbstractAreaBase::positionHasChanged()`.

Referenced by `clone()`.

```

98 {
99     if( d->frameAttributes == a )
100         return;
101
102     d->frameAttributes = a;
103     positionHasChanged();
104 }
```

#### 9.33.4.57 void Legend::setLegendStyle ([LegendStyle](#) style)

Definition at line 189 of file `KDChartLegend.cpp`.

References `d`.

Referenced by `clone()`.

```

190 {
191     if( d->legendStyle == style ) return;
192     d->legendStyle = style;
193     setNeedRebuild();
194 }
```

#### 9.33.4.58 void Legend::setMarkerAttributes (uint *dataset*, const [MarkerAttributes](#) &)

Note that any sizes specified via `setMarkerAttributes` are ignored, unless you disable the automatic size calculation, by saying `setUseAutomaticMarkerSize( false )`.

Definition at line 611 of file KDChartLegend.cpp.

References `d`, and `markerAttributes()`.

```
612 {  
613     if( d->markerAttributes[dataset] == markerAttributes ) return;  
614     d->markerAttributes[ dataset ] = markerAttributes;  
615     setNeedRebuild();  
616     update();  
617 }
```

#### 9.33.4.59 void Legend::setOrientation (Qt::Orientation *orientation*)

Definition at line 471 of file KDChartLegend.cpp.

References `d`.

```
472 {  
473     if( d->orientation == orientation ) return;  
474     d->orientation = orientation;  
475     setNeedRebuild();  
476     emitPositionChanged();  
477 }
```

#### 9.33.4.60 void Legend::setPen (uint *dataset*, const QPen & *pen*)

Definition at line 589 of file KDChartLegend.cpp.

References `d`.

```
590 {  
591     if( d->pens[dataset] == pen ) return;  
592     d->pens[dataset] = pen;  
593     setNeedRebuild();  
594     update();  
595 }
```

#### 9.33.4.61 void Legend::setPosition ([Position](#) *position*)

Specify the position of a non-floating legend.

Use `setFloatingPosition` to set position and alignment if your legend is floating.

See also:

[setAlignment](#), [setFloatingPosition](#)

Definition at line 424 of file KDChartLegend.cpp.

References `d`, and `position()`.

Referenced by `KDChart::Widget::addLegend()`, and `clone()`.

```
425 {  
426     if( d->position == position )  
427         return;  
428     d->position = position;  
429     emitPositionChanged();  
430 }
```

#### 9.33.4.62 void Legend::setRainbowColors ()

Definition at line 711 of file KDChartLegend.cpp.

References [brush\(\)](#), and [setColor\(\)](#).

```
712 {  
713     setColor( 0, QColor(255, 0,196) );  
714     setColor( 1, QColor(255, 0, 96) );  
715     setColor( 2, QColor(255, 128,64) );  
716     setColor( 3, Qt::yellow );  
717     setColor( 4, Qt::green );  
718     setColor( 5, Qt::cyan );  
719     setColor( 6, QColor( 96, 96,255) );  
720     setColor( 7, QColor(160, 0,255) );  
721     for( int i = 8; i < 16; ++i )  
722         setColor( i, brush( i - 8 ).color().light() );  
723 }
```

#### 9.33.4.63 void Legend::setReferenceArea (const [QWidget](#) \* area)

Specifies the reference area for font size of title text, and for font size of the item texts, IF automatic area detection is set.

##### Note:

This parameter is ignored, if the [Measure](#) given for [setTitleTextAttributes](#) (or [setTextAttributes](#), resp.) is not specifying automatic area detection.

If no reference area is specified, but automatic area detection is set, then the size of the legend's parent widget will be used.

##### See also:

[KDChart::Measure](#), [KDChartEnums::MeasureCalculationMode](#)

Definition at line 300 of file KDChartLegend.cpp.

References [d](#).

Referenced by [KDChart::Chart::addLegend\(\)](#).

```
301 {  
302     if( area == d->referenceArea ) return;  
303     d->referenceArea = area;  
304     setNeedRebuild();  
305 }
```

#### 9.33.4.64 void Legend::setShowLines (bool *legendShowLines*)

Definition at line 484 of file KDChartLegend.cpp.

References [d](#).

```
485 {  
486     if( d->showLines == legendShowLines ) return;  
487     d->showLines = legendShowLines;  
488     setNeedRebuild();  
489     emitPositionChanged();  
490 }
```

**9.33.4.65 void Legend::setSpacing (uint *space*)**

Definition at line 682 of file KDChartLegend.cpp.

References [d](#).

```

683 {
684     if( d->spacing == space && d->layout->spacing() == static_cast<int>(space) ) return;
685     d->spacing = space;
686     d->layout->setSpacing( space );
687     setNeedRebuild();
688 }
```

**9.33.4.66 void Legend::setSubduedColors (bool *ordered* = false)**

Definition at line 725 of file KDChartLegend.cpp.

References [setColor\(\)](#).

```

726 {
727     static const int NUM_SUBDUEDCOLORS = 18;
728     static const QColor SUBDUEDCOLORS[ NUM_SUBDUEDCOLORS ] = {
729         QColor( 0xe0,0xf,0x70 ),
730         QColor( 0xe2,0xa5,0x6f ),
731         QColor( 0xe0,0xc9,0x70 ),
732         QColor( 0xd1,0xe0,0x70 ),
733         QColor( 0xac,0xe0,0x70 ),
734         QColor( 0x86,0xe0,0x70 ),
735         QColor( 0x70,0xe0,0x7f ),
736         QColor( 0x70,0xe0,0xa4 ),
737         QColor( 0x70,0xe0,0xc9 ),
738         QColor( 0x70,0xd1,0xe0 ),
739         QColor( 0x70,0xac,0xe0 ),
740         QColor( 0x70,0x86,0xe0 ),
741         QColor( 0x7f,0x70,0xe0 ),
742         QColor( 0xa4,0x70,0xe0 ),
743         QColor( 0xc9,0x70,0xe0 ),
744         QColor( 0xe0,0x70,0xd1 ),
745         QColor( 0xe0,0x70,0xac ),
746         QColor( 0xe0,0x70,0x86 ),
747     };
748     if( ordered )
749         for(int i=0; i<NUM_SUBDUEDCOLORS; ++i)
750             setColor( i, SUBDUEDCOLORS[i] );
751     else{
752         setColor( 0, SUBDUEDCOLORS[ 0 ] );
753         setColor( 1, SUBDUEDCOLORS[ 5 ] );
754         setColor( 2, SUBDUEDCOLORS[10] );
755         setColor( 3, SUBDUEDCOLORS[15] );
756         setColor( 4, SUBDUEDCOLORS[ 2 ] );
757         setColor( 5, SUBDUEDCOLORS[ 7 ] );
758         setColor( 6, SUBDUEDCOLORS[12] );
759         setColor( 7, SUBDUEDCOLORS[17] );
760         setColor( 8, SUBDUEDCOLORS[ 4 ] );
761         setColor( 9, SUBDUEDCOLORS[ 9 ] );
762         setColor(10, SUBDUEDCOLORS[14] );
763         setColor(11, SUBDUEDCOLORS[ 1 ] );
764         setColor(12, SUBDUEDCOLORS[ 6 ] );
765         setColor(13, SUBDUEDCOLORS[11] );
766         setColor(14, SUBDUEDCOLORS[16] );
767         setColor(15, SUBDUEDCOLORS[ 3 ] );
768         setColor(16, SUBDUEDCOLORS[ 8 ] );
769         setColor(17, SUBDUEDCOLORS[13] );

```

```
770     }  
771 }
```

#### 9.33.4.67 void Legend::setText (uint *dataset*, const QString & *text*)

Definition at line 521 of file KDChartLegend.cpp.

References d.

```
522 {  
523     if( d->texts[ dataset ] == text ) return;  
524     d->texts[ dataset ] = text;  
525     setNeedRebuild();  
526 }
```

#### 9.33.4.68 void Legend::setTextAttributes (const [TextAttributes](#) & *a*)

Definition at line 634 of file KDChartLegend.cpp.

References d.

Referenced by KDChart::Chart::addLegend(), and clone().

```
635 {  
636     if( d->textAttributes == a ) return;  
637     d->textAttributes = a;  
638     setNeedRebuild();  
639 }
```

#### 9.33.4.69 void Legend::setTitleText (const QString & *text*)

Definition at line 646 of file KDChartLegend.cpp.

References d.

```
647 {  
648     if( d->titleText == text ) return;  
649     d->titleText = text;  
650     setNeedRebuild();  
651 }
```

#### 9.33.4.70 void Legend::setTitleTextAttributes (const [TextAttributes](#) & *a*)

Definition at line 658 of file KDChartLegend.cpp.

References d.

Referenced by KDChart::Chart::addLegend(), and clone().

```
659 {  
660     if( d->titleTextAttributes == a ) return;  
661     d->titleTextAttributes = a;  
662     setNeedRebuild();  
663 }
```



**9.33.4.71 void Legend::setUseAutomaticMarkerSize (bool *useAutomaticMarkerSize*)**

This option is on by default, it means that Marker sizes in the [Legend](#) will be the same as the font height used for their respective label texts.

Set this to false, if you want to specify the marker sizes via `setMarkerAttributes` or if you want the [Legend](#) to use the same marker sizes as they are used in the Diagrams.

Definition at line 497 of file `KDChartLegend.cpp`.

References `d`.

Referenced by `clone()`.

```
498 {
499     d->useAutomaticMarkerSize = useAutomaticMarkerSize;
500     setNeedRebuild();
501     emitPositionChanged();
502 }
```

**9.33.4.72 void Legend::setVisible (bool *visible*) [virtual]**

Definition at line 409 of file `KDChartLegend.cpp`.

Referenced by `KDChart::Chart::addLegend()`.

```
410 {
411     if( isVisible() == visible )
412         return;
413     QWidget::setVisible( visible );
414     emitPositionChanged();
415 }
```

**9.33.4.73 bool Legend::showLines () const**

Definition at line 492 of file `KDChartLegend.cpp`.

References `d`.

Referenced by `compare()`.

```
493 {
494     return d->showLines;
495 }
```

**9.33.4.74 QSize Legend::sizeHint () const [virtual]**

Definition at line 150 of file `KDChartLegend.cpp`.

References `d`.

Referenced by `minimumSizeHint()`, `KDChart::Chart::reLayoutFloatingLegends()`, and `resizeEvent()`.

```
151 {
152 #ifdef DEBUG_LEGEND_PAINT
153     qDebug() << "Legend::sizeHint() started";
154 #endif
```

```

155     Q_FOREACH( KDChart::AbstractLayoutItem* layoutItem, d->layoutItems ) {
156         layoutItem->sizeHint();
157     }
158     return AbstractAreaWidget::sizeHint();
159 }

```

#### 9.33.4.75 `uint Legend::spacing () const`

Definition at line 690 of file KDChartLegend.cpp.

References `d`.

Referenced by `compare()`.

```

691 {
692     return d->spacing;
693 }

```

#### 9.33.4.76 `QString Legend::text (uint dataset) const`

Definition at line 528 of file KDChartLegend.cpp.

References `d`.

```

529 {
530     if( d->texts.find( dataset ) != d->texts.end() ){
531         return d->texts[ dataset ];
532     }else{
533         return d->modelLabels[ dataset ];
534     }
535 }

```

#### 9.33.4.77 `TextAttributes Legend::textAttributes () const`

Definition at line 641 of file KDChartLegend.cpp.

References `d`.

Referenced by `KDChart::Chart::addLegend()`, `clone()`, and `compare()`.

```

642 {
643     return d->textAttributes;
644 }

```

#### 9.33.4.78 `const QMap< uint, QString > Legend::texts () const`

Definition at line 537 of file KDChartLegend.cpp.

References `d`.

Referenced by `compare()`.

```

538 {
539     return d->texts;
540 }

```

#### 9.33.4.79 QString Legend::titleText () const

Definition at line 653 of file KDChartLegend.cpp.

References [d](#).

Referenced by [compare\(\)](#).

```
654 {  
655     return d->titleText;  
656 }
```

#### 9.33.4.80 TextAttributes Legend::titleTextAttributes () const

Definition at line 665 of file KDChartLegend.cpp.

References [d](#).

Referenced by [KDChart::Chart::addLegend\(\)](#), [clone\(\)](#), and [compare\(\)](#).

```
666 {  
667     return d->titleTextAttributes;  
668 }
```

#### 9.33.4.81 bool Legend::useAutomaticMarkerSize () const

Definition at line 504 of file KDChartLegend.cpp.

References [d](#).

Referenced by [clone\(\)](#), and [compare\(\)](#).

```
505 {  
506     return d->useAutomaticMarkerSize;  
507 }
```

The documentation for this class was generated from the following files:

- [KDChartLegend.h](#)
- [KDChartLegend.cpp](#)

## 9.34 KDChart::LineAttributes Class Reference

```
#include <KDChartLineAttributes.h>
```

### 9.34.1 Detailed Description

Set of attributes for changing the appearance of line charts.

Definition at line 37 of file KDChartLineAttributes.h.

### Public Types

- enum [MissingValuesPolicy](#) {  
    [MissingValuesAreBridged](#),  
    [MissingValuesHideSegments](#),  
    [MissingValuesShownAsZero](#),  
    [MissingValuesPolicyIgnored](#) }

*MissingValuesPolicy specifies how a missing value will be shown in a line diagram.*

### Public Member Functions

- bool [displayArea](#) () const
- [LineAttributes](#) (const [LineAttributes](#) &)
- [LineAttributes](#) ()
- [MissingValuesPolicy](#) [missingValuesPolicy](#) () const
- bool [operator!=](#) (const [LineAttributes](#) &other) const
- [LineAttributes](#) & [operator=](#) (const [LineAttributes](#) &)
- bool [operator==](#) (const [LineAttributes](#) &) const
- void [setDisplayArea](#) (bool display)
- void [setMissingValuesPolicy](#) ([MissingValuesPolicy](#) policy)
- void [setTransparency](#) (uint alpha)
- uint [transparency](#) () const
- [~LineAttributes](#) ()

### 9.34.2 Member Enumeration Documentation

#### 9.34.2.1 enum [KDChart::LineAttributes::MissingValuesPolicy](#)

[MissingValuesPolicy](#) specifies how a missing value will be shown in a line diagram.

Missing value is assumed if the data cell contains a QVariant that can not be interpreted as a double, or if the data cell is hidden while its dataset is not hidden.

- [MissingValuesAreBridged](#) the default: No markers will be shown for missing values but the line will be bridged if there is at least one valid cell before and after the missing value(s), otherwise the segment will be hidden.

- `MissingValuesHideSegments` Line segments starting with a missing value will not be shown, and no markers will be shown for missing values, so this will look like a piece of the line is missing.
- `MissingValuesShownAsZero` Missing value(s) will be treated like normal zero values, and markers will shown for them too, so there will be no visible difference between a zero value and a missing value.
- `MissingValuesPolicyIgnored` (internal value, do not use)

**Enumerator:*****MissingValuesAreBridged******MissingValuesHideSegments******MissingValuesShownAsZero******MissingValuesPolicyIgnored***

Definition at line 58 of file `KDChartLineAttributes.h`.

```
58          {
59      MissingValuesAreBridged,
60      MissingValuesHideSegments,
61      MissingValuesShownAsZero,
62      MissingValuesPolicyIgnored };
```

### 9.34.3 Constructor & Destructor Documentation

#### 9.34.3.1 `LineAttributes::LineAttributes ()`

Definition at line 57 of file `KDChartLineAttributes.cpp`.

```
58      : _d( new Private() )
59  {
60  }
```

#### 9.34.3.2 `LineAttributes::LineAttributes (const LineAttributes &)`

Definition at line 62 of file `KDChartLineAttributes.cpp`.

```
63      : _d( new Private( *r.d ) )
64  {
65  }
```

#### 9.34.3.3 `LineAttributes::~~LineAttributes ()`

Definition at line 77 of file `KDChartLineAttributes.cpp`.

```
78  {
79      delete _d; _d = 0;
80  }
```

### 9.34.4 Member Function Documentation

#### 9.34.4.1 `bool LineAttributes::displayArea () const`

Definition at line 105 of file `KDChartLineAttributes.cpp`.

References `d`.

Referenced by `operator<<()`, and `operator==()`.

```
106 {  
107     return d->displayArea;  
108 }
```

#### 9.34.4.2 `LineAttributes::MissingValuesPolicy` `LineAttributes::missingValuesPolicy () const`

Definition at line 95 of file `KDChartLineAttributes.cpp`.

References `d`.

Referenced by `KDChart::LineDiagram::getCellValues()`, and `operator==()`.

```
96 {  
97     return d->missingValuesPolicy;  
98 }
```

#### 9.34.4.3 `bool KDChart::LineAttributes::operator!= (const LineAttributes & other) const`

Definition at line 82 of file `KDChartLineAttributes.h`.

```
82 { return !operator==(other); }
```

#### 9.34.4.4 `LineAttributes` & `LineAttributes::operator= (const LineAttributes &)`

Definition at line 67 of file `KDChartLineAttributes.cpp`.

References `d`.

```
68 {  
69     if( this == &r )  
70         return *this;  
71  
72     *d = *r.d;  
73  
74     return *this;  
75 }
```

#### 9.34.4.5 `bool LineAttributes::operator== (const LineAttributes &) const`

Definition at line 82 of file `KDChartLineAttributes.cpp`.

References `displayArea()`, `missingValuesPolicy()`, and `transparency()`.

```
83 {  
84     return  
85         missingValuesPolicy() == r.missingValuesPolicy() &&  
86         displayArea() == r.displayArea() &&  
87         transparency() == r.transparency();  
88 }
```

#### 9.34.4.6 void LineAttributes::setDisplayArea (bool *display*)

Definition at line 100 of file KDChartLineAttributes.cpp.

References [d](#).

```
101 {  
102     d->displayArea = display;  
103 }
```

#### 9.34.4.7 void LineAttributes::setMissingValuesPolicy ([MissingValuesPolicy](#) *policy*)

Definition at line 90 of file KDChartLineAttributes.cpp.

References [d](#).

```
91 {  
92     d->missingValuesPolicy = policy;  
93 }
```

#### 9.34.4.8 void LineAttributes::setTransparency (uint *alpha*)

Definition at line 110 of file KDChartLineAttributes.cpp.

References [d](#).

```
111 {  
112     if ( alpha > 255 )  
113         alpha = 255;  
114     d->transparency = alpha;  
115 }
```

#### 9.34.4.9 uint LineAttributes::transparency () const

Definition at line 117 of file KDChartLineAttributes.cpp.

References [d](#).

Referenced by [operator<<\(\)](#), and [operator==\(\)](#).

```
118 {  
119     return d->transparency;  
120 }
```

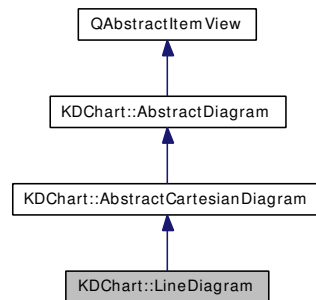
The documentation for this class was generated from the following files:

- [KDChartLineAttributes.h](#)
- [KDChartLineAttributes.cpp](#)

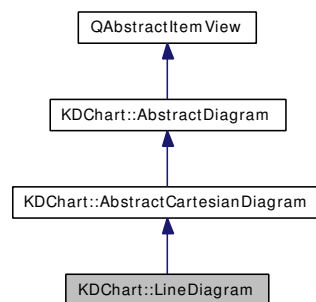
## 9.35 KDChart::LineDiagram Class Reference

```
#include <KDChartLineDiagram.h>
```

Inheritance diagram for KDChart::LineDiagram:



Collaboration diagram for KDChart::LineDiagram:



### 9.35.1 Detailed Description

[LineDiagram](#) defines a common line diagram.

It provides different subtypes which are set using *setType*.

Definition at line 49 of file KDChartLineDiagram.h.

### Public Types

- enum [LineType](#) {  
     [Normal](#) = 0,  
     [Stacked](#) = 1,  
     [Percent](#) = 2 }

### Signals

- void [dataHidden](#) ()

*This signal is emitted, when the hidden status of at least one data cell was (un)set.*



- void [layoutChanged](#) ([AbstractDiagram](#) \*)  
*Diagrams are supposed to emit this signal, when the layout of one of their element changes.*
- void [modelsChanged](#) ()  
*This signal is emitted, when either the model or the [AttributesModel](#) is replaced.*
- void [propertiesChanged](#) ()  
*Emitted upon change of a property of the Diagram.*

## Public Member Functions

- virtual void [addAxis](#) ([CartesianAxis](#) \*axis)  
*Add the axis to the diagram.*
- bool [allowOverlappingDataValueTexts](#) () const  
**Returns:**  
*Whether data value labels are allowed to overlap.*
- bool [antiAliasing](#) () const  
**Returns:**  
*Whether anti-aliasing is to be used for rendering this diagram.*
- virtual [AttributesModel](#) \* [attributesModel](#) () const  
*Returns the [AttributesModel](#), that is used by this diagram.*
- virtual [KDChart::CartesianAxisList](#) [axes](#) () const  
**Returns:**  
*a list of all axes added to the diagram*
- [QBrush](#) [brush](#) (const [QModelIndex](#) &index) const  
*Retrieve the brush to be used, for painting the datapoint at the given index in the model.*
- [QBrush](#) [brush](#) (int dataset) const  
*Retrieve the brush to be used for the given dataset.*
- [QBrush](#) [brush](#) () const  
*Retrieve the brush to be used for painting datapoints globally.*
- virtual [LineDiagram](#) \* [clone](#) () const  
*Creates an exact copy of this diagram.*
- bool [compare](#) (const [AbstractDiagram](#) \*other) const  
*Returns true if both diagrams have the same settings.*
- bool [compare](#) (const [AbstractCartesianDiagram](#) \*other) const  
*Returns true if both diagrams have the same settings.*
- bool [compare](#) (const [LineDiagram](#) \*other) const

*Returns true if both diagrams have the same settings.*

- [AbstractCoordinatePlane](#) \* [coordinatePlane](#) () const  
*The coordinate plane associated with the diagram.*
- const QPair< QPointF, QPointF > [dataBoundaries](#) () const  
*Return the bottom left and top right data point, that the diagram will display (unless the grid adjusts these values).*
- virtual void [dataChanged](#) (const QModelIndex &topLeft, const QModelIndex &bottomRight)  
*[reimplemented]*
- QList< QBrush > [datasetBrushes](#) () const  
*The set of dataset brushes currently used, for use in legends, etc.*
- int [datasetDimension](#) () const  
*The dataset dimension of a diagram determines, how many value dimensions it expects each datapoint to have.*
- QStringList [datasetLabels](#) () const  
*The set of dataset labels currently displayed, for use in legends, etc.*
- QList< MarkerAttributes > [datasetMarkers](#) () const  
*The set of dataset markers currently used, for use in legends, etc.*
- QList< QPen > [datasetPens](#) () const  
*The set of dataset pens currently used, for use in legends, etc.*
- [DataValueAttributes](#) [dataValueAttributes](#) (const QModelIndex &index) const  
*Retrieve the [DataValueAttributes](#) for the given index.*
- [DataValueAttributes](#) [dataValueAttributes](#) (int column) const  
*Retrieve the [DataValueAttributes](#) for the given dataset.*
- [DataValueAttributes](#) [dataValueAttributes](#) () const  
*Retrieve the [DataValueAttributes](#) specified globally.*
- virtual void [doItemsLayout](#) ()  
*[reimplemented]*
- virtual int [horizontalOffset](#) () const  
*[reimplemented]*
- virtual QModelIndex [indexAt](#) (const QPoint &point) const  
*[reimplemented]*
- QModelIndexList [indexesAt](#) (const QPoint &point) const  
*This method is added alongside with [indexAt](#) from QAIM, since in kdchart multiple indexes can be displayed at the same spot.*

- bool [isHidden](#) (const QModelIndex &index) const  
*Retrieve the hidden status for the given index.*
- bool [isHidden](#) (int column) const  
*Retrieve the hidden status for the given dataset.*
- bool [isHidden](#) () const  
*Retrieve the hidden status specified globally.*
- virtual bool [isIndexHidden](#) (const QModelIndex &index) const  
*[reimplemented]*
- QStringList [itemRowLabels](#) () const  
*The set of item row labels currently displayed, for use in Abscissa axes, etc.*
- virtual void [layoutPlanes](#) ()  
*Triggers layouting of all coordinate planes on the current chart.*
- [LineAttributes](#) [lineAttributes](#) (const QModelIndex &index) const  
**Returns:**  
*the line attribute set of the model index index*
- [LineAttributes](#) [lineAttributes](#) (int column) const  
**Returns:**  
*the line attribute set of data set column*
- [LineAttributes](#) [lineAttributes](#) () const  
**Returns:**  
*the global line attribute set*
- [LineDiagram](#) (QWidget \*parent=0, [CartesianCoordinatePlane](#) \*plane=0)
- virtual QModelIndex [moveCursor](#) (CursorAction cursorAction, Qt::KeyboardModifiers modifiers)  
*[reimplemented]*
- const int [numberOfAbscissaSegments](#) () const
- const int [numberOfOrdinateSegments](#) () const
- void [paintDataValueText](#) (QPainter \*painter, const QModelIndex &index, const QPointF &pos, double value)
- void [paintMarker](#) (QPainter \*painter, const QModelIndex &index, const QPointF &pos)
- virtual void [paintMarker](#) (QPainter \*painter, const [MarkerAttributes](#) &markerAttributes, const QBrush &brush, const QPen &, const QPointF &point, const QSizeF &size)
- QPen [pen](#) (const QModelIndex &index) const  
*Retrieve the pen to be used, for painting the datapoint at the given index in the model.*
- QPen [pen](#) (int dataset) const  
*Retrieve the pen to be used for the given dataset.*
- QPen [pen](#) () const  
*Retrieve the pen to be used for painting datapoints globally.*

- bool [percentMode](#) () const
- virtual [AbstractCartesianDiagram](#) \* [referenceDiagram](#) () const
  - Returns:**
    - this diagram's reference diagram*
- virtual QPointF [referenceDiagramOffset](#) () const
  - Returns:**
    - the relative offset of this diagram's reference diagram*
- void [resetLineAttributes](#) (const QModelIndex &index)
  - Remove any explicit line attributes settings that might have been specified before.*
- void [resetLineAttributes](#) (int column)
  - Resets the line attributes of data set column.*
- void [resize](#) (const QSizeF &area)
  - Called by the widget's sizeEvent.*
- virtual void [scrollTo](#) (const QModelIndex &index, ScrollHint hint=EnsureVisible)
  - [reimplemented]*
- void [setAllowOverlappingDataValueTexts](#) (bool allow)
  - Set whether data value labels are allowed to overlap.*
- void [setAntiAliasing](#) (bool enabled)
  - Set whether anti-aliasing is to be used while rendering this diagram.*
- void [setAttributesModel](#) ([AttributesModel](#) \*model)
  - Associate an [AttributesModel](#) with this diagram.*
- void [setBrush](#) (const QBrush &brush)
  - Set the brush to be used, for painting all datasets in the model.*
- void [setBrush](#) (int dataset, const QBrush &brush)
  - Set the brush to be used, for painting the given dataset.*
- void [setBrush](#) (const QModelIndex &index, const QBrush &brush)
  - Set the brush to be used, for painting the datapoint at the given index.*
- virtual void [setCoordinatePlane](#) ([AbstractCoordinatePlane](#) \*plane)
  - [reimplemented]*
- void [setDatasetDimension](#) (int dimension)
  - Sets the dataset dimension of the diagram.*
- void [setDataValueAttributes](#) (const [DataValueAttributes](#) &a)
  - Set the [DataValueAttributes](#) for all datapoints in the model.*
- void [setDataValueAttributes](#) (int dataset, const [DataValueAttributes](#) &a)

Set the [DataValueAttributes](#) for the given dataset.

- void [setDataValueAttributes](#) (const QModelIndex &index, const [DataValueAttributes](#) &a)  
Set the [DataValueAttributes](#) for the given index.
- void [setHidden](#) (bool hidden)  
Hide (or unhide, resp.
- void [setHidden](#) (int column, bool hidden)  
Hide (or unhide, resp.
- void [setHidden](#) (const QModelIndex &index, bool hidden)  
Hide (or unhide, resp.
- void [setLineAttributes](#) (const QModelIndex &index, const [LineAttributes](#) &a)  
Sets the line attributes for the model index index to la.
- void [setLineAttributes](#) (int column, const [LineAttributes](#) &a)  
Sets the line attributes of data set column to la.
- void [setLineAttributes](#) (const [LineAttributes](#) &a)  
Sets the global line attributes to la.
- void [setModel](#) (QAbstractItemModel \*model)  
Associate a model with the diagram.
- void [setPen](#) (const QPen &pen)  
Set the pen to be used, for painting all datasets in the model.
- void [setPen](#) (int dataset, const QPen &pen)  
Set the pen to be used, for painting the given dataset.
- void [setPen](#) (const QModelIndex &index, const QPen &pen)  
Set the pen to be used, for painting the datapoint at the given index.
- void [setPercentMode](#) (bool percent)
- virtual void [setReferenceDiagram](#) ([AbstractCartesianDiagram](#) \*diagram, const QPointF &offset=QPointF())  
Makes this diagram use another diagram diagram as reference diagram with relative offset offset.
- void [setRootIndex](#) (const QModelIndex &index)  
Set the root index in the model, where the diagram starts referencing data for display.
- virtual void [setSelection](#) (const QRect &rect, QItemSelectionModel::SelectionFlags command)  
[reimplemented]
- void [setThreeDLineAttributes](#) (const QModelIndex &index, const [ThreeDLineAttributes](#) &a)  
Sets the 3D line attributes of model index index to la.
- void [setThreeDLineAttributes](#) (int column, const [ThreeDLineAttributes](#) &a)

*Sets the 3D line attributes of data set column to ta.*

- void [setThreeDLineAttributes](#) (const [ThreeDLineAttributes](#) &a)  
*Sets the global 3D line attributes to la.*
- void [setType](#) (const [LineType](#) type)  
*Sets the line diagram's type to type.*
- void [setUnitPrefix](#) (const QString &prefix, Qt::Orientation orientation)  
*Sets the unit prefix for all values.*
- void [setUnitPrefix](#) (const QString &prefix, int column, Qt::Orientation orientation)  
*Sets the unit prefix for one value.*
- void [setUnitSuffix](#) (const QString &suffix, Qt::Orientation orientation)  
*Sets the unit suffix for all values.*
- void [setUnitSuffix](#) (const QString &suffix, int column, Qt::Orientation orientation)  
*Sets the unit suffix for one value.*
- void [setValueTrackerAttributes](#) (const QModelIndex &index, const [ValueTrackerAttributes](#) &a)  
*Sets the value tracker attributes of the model index index to va.*
- virtual void [takeAxis](#) ([CartesianAxis](#) \*axis)  
*Removes the axis from the diagram, without deleting it.*
- [ThreeDLineAttributes](#) [threeDLineAttributes](#) (const QModelIndex &index) const  
**Returns:**  
*the 3D line attributes of the model index index*
- [ThreeDLineAttributes](#) [threeDLineAttributes](#) (int column) const  
**Returns:**  
*the 3D line attributes of data set column*
- [ThreeDLineAttributes](#) [threeDLineAttributes](#) () const  
**Returns:**  
*the global 3D line attributes*
- [LineType](#) [type](#) () const  
**Returns:**  
*the type of the line diagram*
- QString [unitPrefix](#) (Qt::Orientation orientation) const  
*Returns the global unit prefix.*
- QString [unitPrefix](#) (int column, Qt::Orientation orientation, bool fallback=false) const  
*Returns the unit prefix for a special value.*
- QString [unitSuffix](#) (Qt::Orientation orientation) const

*Returns the global unit suffix.*

- QString [unitSuffix](#) (int column, Qt::Orientation orientation, bool fallback=false) const  
*Returns the unit suffix for a special value.*
- void [update](#) () const
- void [useDefaultColors](#) ()  
*Set the palette to be used, for painting datasets to the default palette.*
- void [useRainbowColors](#) ()  
*Set the palette to be used, for painting datasets to the rainbow palette.*
- virtual bool [usesExternalAttributesModel](#) () const  
*Returns whether the diagram is using its own built-in attributes model or an attributes model that was set via setAttributesModel.*
- void [useSubduedColors](#) ()  
*Set the palette to be used, for painting datasets to the subdued palette.*
- [ValueTrackerAttributes](#) [valueTrackerAttributes](#) (const QModelIndex &index) const  
*Returns the value tracker attributes of the model index index.*
- virtual int [verticalOffset](#) () const  
*[reimplemented]*
- virtual QRect [visualRect](#) (const QModelIndex &index) const  
*[reimplemented]*
- virtual QRegion [visualRegionForSelection](#) (const QItemSelection &selection) const  
*[reimplemented]*
- virtual [~LineDiagram](#) ()

## Protected Member Functions

- QModelIndex [attributesModelRootIndex](#) () const
- virtual const QPair< QPointF, QPointF > [calculateDataBoundaries](#) () const  
*[reimplemented]*
- virtual bool [checkInvariants](#) (bool justReturnTheStatus=false) const
- QModelIndex [columnToIndex](#) (int column) const
- [LineAttributes::MissingValuesPolicy](#) [getCellValues](#) (int row, int column, bool shiftCountedXValues-ByHalfSection, double &valueX, double &valueY) const
- void [paint](#) (PaintContext \*paintContext)  
*Draw the diagram contents to the rectangle and painter, that are passed in as part of the paint context.*
- virtual void [paintDataValueTexts](#) (QPainter \*painter)
- void [paintEvent](#) (QPaintEvent \*)
- virtual void [paintMarkers](#) (QPainter \*painter)
- void [resizeEvent](#) (QResizeEvent \*)

- void [setAttributesModelRootIndex](#) (const QModelIndex &)
- void [setDataBoundariesDirty](#) () const
- virtual double [threeDItemDepth](#) (int column) const

**Returns:**

*the 3D item depth of the data set column*

- virtual double [threeDItemDepth](#) (const QModelIndex &index) const

**Returns:**

*the 3D item depth of the model index index*

- double [valueForCell](#) (int row, int column) const

*Helper method, retrieving the data value (DisplayRole) for a given row and column.*

- double [valueForCellTesting](#) (int row, int column, bool &bOK, bool showHiddenCellsAsInvalid=false) const

## 9.35.2 Member Enumeration Documentation

### 9.35.2.1 enum [KDChart::LineDiagram::LineType](#)

**Enumerator:**

*Normal*

*Stacked*

*Percent*

Definition at line 73 of file KDChartLineDiagram.h.

```

73      {
74      Normal = 0,
75      Stacked = 1,
76      Percent = 2
77      };

```

## 9.35.3 Constructor & Destructor Documentation

### 9.35.3.1 [LineDiagram::LineDiagram](#) ([QWidget](#) \* *parent* = 0, [CartesianCoordinatePlane](#) \* *plane* = 0)

Definition at line 61 of file KDChartLineDiagram.cpp.

Referenced by [clone\(\)](#).

```

61      :
62      AbstractCartesianDiagram( new Private(), parent, plane )
63  {
64      init();
65  }

```



### 9.35.3.2 LineDiagram::~~LineDiagram () [virtual]

Definition at line 76 of file KDChartLineDiagram.cpp.

```
77 {  
78 }
```

## 9.35.4 Member Function Documentation

### 9.35.4.1 void AbstractCartesianDiagram::addAxis ([CartesianAxis](#) \* *axis*) [virtual, inherited]

Add the axis to the diagram.

The diagram takes ownership of the axis and will delete it.

To gain back ownership (e.g. for assigning the axis to another diagram) use the takeAxis method, before calling addAxis on the other diagram.

**See also:**

[takeAxis](#)

Definition at line 91 of file KDChartAbstractCartesianDiagram.cpp.

References [d](#), and [KDChart::AbstractCartesianDiagram::layoutPlanes\(\)](#).

```
92 {  
93     if ( !d->axesList.contains( axis ) ) {  
94         d->axesList.append( axis );  
95         axis->createObserver( this );  
96         layoutPlanes();  
97     }  
98 }
```

### 9.35.4.2 bool AbstractDiagram::allowOverlappingDataValueTexts () const [inherited]

**Returns:**

Whether data value labels are allowed to overlap.

Definition at line 440 of file KDChartAbstractDiagram.cpp.

References [d](#).

Referenced by [KDChart::AbstractDiagram::compare\(\)](#).

```
441 {  
442     return d->allowOverlappingDataValueTexts;  
443 }
```

### 9.35.4.3 bool AbstractDiagram::antiAliasing () const [inherited]

**Returns:**

Whether anti-aliasing is to be used for rendering this diagram.

Definition at line 451 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::compare().

```
452 {
453     return d->antiAliasing;
454 }
```

#### 9.35.4.4 [AttributesModel](#) \* AbstractDiagram::attributesModel () const [virtual, inherited]

Returns the [AttributesModel](#), that is used by this diagram.

By default each diagram owns its own [AttributesModel](#), which should never be deleted. Only if a user-supplied [AttributesModel](#) has been set does the pointer returned here not belong to the diagram.

#### Returns:

The [AttributesModel](#) associated with the diagram.

#### See also:

[setAttributesModel](#)

Definition at line 280 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::brush(), KDChart::AbstractDiagram::compare(), KDChart::AbstractDiagram::datasetBrushes(), KDChart::AbstractDiagram::datasetLabels(), KDChart::AbstractDiagram::datasetMarkers(), KDChart::AbstractDiagram::datasetPens(), KDChart::AbstractDiagram::dataValueAttributes(), KDChart::AbstractDiagram::isHidden(), KDChart::AbstractDiagram::itemRowLabels(), KDChart::RingDiagram::paint(), KDChart::PolarDiagram::paint(), KDChart::AbstractDiagram::pen(), KDChart::AbstractCartesianDiagram::setAttributesModel(), KDChart::BarDiagram::setBarAttributes(), KDChart::AbstractDiagram::setBrush(), KDChart::AbstractCartesianDiagram::setModel(), KDChart::AbstractDiagram::setPen(), and KDChart::TernaryLineDiagram::TernaryLineDiagram().

```
281 {
282     return d->attributesModel;
283 }
```

#### 9.35.4.5 [QModelIndex](#) AbstractDiagram::attributesModelRootIndex () const [protected, inherited]

returns a [QModelIndex](#) pointing into the [AttributesModel](#) that corresponds to the root index of the diagram.

Definition at line 302 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::datasetBrushes(), KDChart::AbstractDiagram::datasetLabels(), KDChart::AbstractDiagram::datasetMarkers(), KDChart::AbstractDiagram::datasetPens(), KDChart::AbstractDiagram::itemRowLabels(), KDChart::Plotter::numberOfAbscissaSegments(), KDChart::BarDiagram::numberOfAbscissaSegments(),

KDChart::Plotter::numberOfOrdinateSegments(), numberOfOrdinateSegments(), KDChart::BarDiagram::numberOfOrdinateSegments(), KDChart::AbstractDiagram::valueForCell(), and valueForCellTesting().

```

303 {
304     if ( !d->attributesModelRootIndex.isValid() )
305         d->attributesModelRootIndex = d->attributesModel->mapFromSource( rootIndex() );
306     return d->attributesModelRootIndex;
307 }
```

#### 9.35.4.6 **KDChart::CartesianAxisList** AbstractCartesianDiagram::axes () const [virtual, inherited]

##### Returns:

a list of all axes added to the diagram

Definition at line 110 of file KDChartAbstractCartesianDiagram.cpp.

References d.

Referenced by KDChart::Widget::setType(), and KDChart::CartesianCoordinatePlane::sharedAxisMasterPlane().

```

111 {
112     return d->axesList;
113 }
```

#### 9.35.4.7 **QBrush** AbstractDiagram::brush (const QModelIndex & *index*) const [inherited]

Retrieve the brush to be used, for painting the datapoint at the given index in the model.

##### Parameters:

*index* The index of the datapoint in the model.

##### Returns:

The brush to use for painting.

Definition at line 838 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::brush(), KDChart::AttributesModel::data(), KDChart::DatasetBrushRole, and KDChart::AbstractDiagram::datasetDimension().

```

839 {
840     if( datasetDimension() > 1 )
841         return brush( index.column() );
842     return qVariantValue<QBrush>(
843         attributesModel()->data(
844             attributesModel()->mapFromSource( index ),
845             DatasetBrushRole ) );
846 }
```

#### 9.35.4.8 QBrush AbstractDiagram::brush (int *dataset*) const [inherited]

Retrieve the brush to be used for the given dataset.

This will fall back automatically to what was set at model level, if there are no dataset specific settings.

##### Parameters:

*dataset* The dataset to retrieve the brush for.

##### Returns:

The brush to use for painting.

Definition at line 828 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::columnToIndex(), KDChart::AttributesModel::data(), KDChart::DatasetBrushRole, and KDChart::AbstractDiagram::datasetDimension().

```

829 {
830     if( datasetDimension() > 1 )
831         dataset /= datasetDimension();
832     return qVariantValue<QBrush>(
833         attributesModel()->data(
834             attributesModel()->mapFromSource( columnToIndex( dataset ) ),
835             DatasetBrushRole ) );
836 }
```

#### 9.35.4.9 QBrush AbstractDiagram::brush () const [inherited]

Retrieve the brush to be used for painting datapoints globally.

This will fall back automatically to the default settings, if there are no specific settings.

##### Returns:

The brush to use for painting.

Definition at line 822 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AttributesModel::data(), and KDChart::DatasetBrushRole.

Referenced by KDChart::AbstractDiagram::brush(), KDChart::TernaryPointDiagram::paint(), KDChart::TernaryLineDiagram::paint(), KDChart::RingDiagram::paint(), KDChart::PolarDiagram::paint(), and KDChart::AbstractDiagram::paintMarker().

```

823 {
824     return qVariantValue<QBrush>(
825         attributesModel()->data( DatasetBrushRole ) );
826 }
```

#### 9.35.4.10 const QPair< QPointF, QPointF > LineDiagram::calculateDataBoundaries () const [protected, virtual]

[reimplemented]

Implements [KDChart::AbstractDiagram](#).

Definition at line 360 of file KDChartLineDiagram.cpp.

References [KDChart::AbstractDiagram::checkInvariants\(\)](#), and [d](#).

```

361 {
362     if ( !checkInvariants( true ) ) return QPair<QPointF, QPointF>( QPointF( 0, 0 ), QPointF( 0, 0 ) )
363
364     // note: calculateDataBoundaries() is ignoring the hidden flags.
365     //       That's not a bug but a feature: Hiding data does not mean removing them.
366     // For totally removing data from KD Chart's view people can use e.g. a proxy model ...
367
368     // calculate boundaries for different line types Normal - Stacked - Percent - Default Normal
369     return d->implementor->calculateDataBoundaries();
370 }
```

#### 9.35.4.11 **bool AbstractDiagram::checkInvariants (bool *justReturnTheStatus* = false) const** [protected, virtual, inherited]

Definition at line 1060 of file KDChartAbstractDiagram.cpp.

References [KDChart::AbstractDiagram::coordinatePlane\(\)](#).

Referenced by [KDChart::RingDiagram::calculateDataBoundaries\(\)](#), [KDChart::PolarDiagram::calculateDataBoundaries\(\)](#), [KDChart::Plotter::calculateDataBoundaries\(\)](#), [KDChart::PieDiagram::calculateDataBoundaries\(\)](#), [calculateDataBoundaries\(\)](#), [KDChart::BarDiagram::calculateDataBoundaries\(\)](#), [KDChart::RingDiagram::paint\(\)](#), [KDChart::PolarDiagram::paint\(\)](#), [KDChart::Plotter::paint\(\)](#), [KDChart::PieDiagram::paint\(\)](#), [paint\(\)](#), [KDChart::BarDiagram::paint\(\)](#), [KDChart::AbstractDiagram::paintDataValueTexts\(\)](#), [KDChart::AbstractDiagram::paintMarker\(\)](#), and [KDChart::AbstractDiagram::paintMarkers\(\)](#).

```

1061 {
1062     if( ! justReturnTheStatus ){
1063         Q_ASSERT_X ( model(), "AbstractDiagram::checkInvariants()",
1064                     "There is no usable model set, for the diagram." );
1065
1066         Q_ASSERT_X ( coordinatePlane(), "AbstractDiagram::checkInvariants()",
1067                     "There is no usable coordinate plane set, for the diagram." );
1068     }
1069     return model() && coordinatePlane();
1070 }
```

#### 9.35.4.12 **[LineDiagram](#) \* LineDiagram::clone () const** [virtual]

Creates an exact copy of this diagram.

Definition at line 83 of file KDChartLineDiagram.cpp.

References [d](#), [LineDiagram\(\)](#), [setType\(\)](#), and [type\(\)](#).

```

84 {
85     LineDiagram* newDiagram = new LineDiagram( new Private( *d ) );
86     newDiagram->setType( type() );
87     return newDiagram;
88 }
```

#### 9.35.4.13 QModelIndex AbstractDiagram::columnToIndex (int *column*) const [protected, inherited]

Definition at line 309 of file KDChartAbstractDiagram.cpp.

Referenced by KDChart::BarDiagram::barAttributes(), KDChart::AbstractDiagram::brush(), KDChart::AbstractDiagram::dataValueAttributes(), KDChart::AbstractDiagram::isHidden(), KDChart::Plotter::lineAttributes(), lineAttributes(), KDChart::AbstractDiagram::pen(), KDChart::AbstractPieDiagram::pieAttributes(), KDChart::BarDiagram::threeDBarAttributes(), KDChart::Plotter::threeDLineAttributes(), threeDLineAttributes(), and KDChart::AbstractPieDiagram::threeDPieAttributes().

```

310 {    // FIXME (Mirko): shouldn't this be headerData? instead of the index for the first row?
311     if( model() )
312         return QModelIndex( model()->index( 0, column, rootIndex() ) );
313     return QModelIndex();
314 }
```

#### 9.35.4.14 bool AbstractDiagram::compare (const AbstractDiagram \* *other*) const [inherited]

Returns true if both diagrams have the same settings.

Definition at line 129 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::allowOverlappingDataValueTexts(), KDChart::AbstractDiagram::antiAliasing(), KDChart::AbstractDiagram::attributesModel(), KDChart::AttributesModel::compare(), KDChart::AbstractDiagram::datasetDimension(), and KDChart::AbstractDiagram::percentMode().

```

130 {
131     if( other == this ) return true;
132     if( ! other ){
133         //qDebug() << "AbstractDiagram::compare() cannot compare to Null pointer";
134         return false;
135     }
136     /*
137     qDebug() << "\n                AbstractDiagram::compare() QAbstractScrollArea:";
138     // compare QAbstractScrollArea properties
139     qDebug() <<
140     ((horizontalScrollBarPolicy() == other->horizontalScrollBarPolicy()) &&
141     (verticalScrollBarPolicy() == other->verticalScrollBarPolicy()));
142     qDebug() << "AbstractDiagram::compare() QFrame:";
143     // compare QFrame properties
144     qDebug() <<
145     ((frameShadow() == other->frameShadow()) &&
146     (frameShape() == other->frameShape()) &&
147     (frameWidth() == other->frameWidth()) &&
148     (lineWidth() == other->lineWidth()) &&
149     (midLineWidth() == other->midLineWidth()));
150     qDebug() << "AbstractDiagram::compare() QAbstractItemView:";
151     // compare QAbstractItemView properties
152     qDebug() <<
153     ((alternatingRowColors() == other->alternatingRowColors()) &&
154     (hasAutoScroll() == other->hasAutoScroll()) &&
155 #if QT_VERSION > 0x040199
156     (dragDropMode() == other->dragDropMode()) &&
157     (dragDropOverwriteMode() == other->dragDropOverwriteMode()) &&
158     (horizontalScrollMode() == other->horizontalScrollMode()) &&
159     (verticalScrollMode() == other->verticalScrollMode()) &&
160 #endif

```

```

161         (dragEnabled() == other->dragEnabled()) &&
162         (editTriggers() == other->editTriggers()) &&
163         (iconSize() == other->iconSize()) &&
164         (selectionBehavior() == other->selectionBehavior()) &&
165         (selectionMode() == other->selectionMode()) &&
166         (showDropIndicator() == other->showDropIndicator()) &&
167         (tabKeyNavigation() == other->tabKeyNavigation()) &&
168         (textElideMode() == other->textElideMode()));
169     qDebug() << "AbstractDiagram::compare() AttributesModel: ";
170     // compare all of the properties stored in the attributes model
171     qDebug() << attributesModel()->compare( other->attributesModel() );
172     qDebug() << "AbstractDiagram::compare() own:";
173     // compare own properties
174     qDebug() <<
175         ((rootIndex().column() == other->rootIndex().column()) &&
176         (rootIndex().row() == other->rootIndex().row()) &&
177         (allowOverlappingDataValueTexts() == other->allowOverlappingDataValueTexts()) &&
178         (antiAliasing() == other->antiAliasing()) &&
179         (percentMode() == other->percentMode()) &&
180         (datasetDimension() == other->datasetDimension()));
181     */
182     return // compare QAbstractScrollArea properties
183         (horizontalScrollBarPolicy() == other->horizontalScrollBarPolicy()) &&
184         (verticalScrollBarPolicy() == other->verticalScrollBarPolicy()) &&
185         // compare QFrame properties
186         (frameShadow() == other->frameShadow()) &&
187         (frameShape() == other->frameShape()) &&
188     // frameWidth is a read-only property defined by the style, it should not be in here:
189     // (frameWidth() == other->frameWidth()) &&
190     (lineWidth() == other->lineWidth()) &&
191     (midLineWidth() == other->midLineWidth()) &&
192     // compare QAbstractItemView properties
193     (alternatingRowColors() == other->alternatingRowColors()) &&
194     (hasAutoScroll() == other->hasAutoScroll()) &&
195 #if QT_VERSION > 0x040199
196     (dragDropMode() == other->dragDropMode()) &&
197     (dragDropOverwriteMode() == other->dragDropOverwriteMode()) &&
198     (horizontalScrollMode() == other->horizontalScrollMode()) &&
199     (verticalScrollMode() == other->verticalScrollMode()) &&
200 #endif
201     (dragEnabled() == other->dragEnabled()) &&
202     (editTriggers() == other->editTriggers()) &&
203     (iconSize() == other->iconSize()) &&
204     (selectionBehavior() == other->selectionBehavior()) &&
205     (selectionMode() == other->selectionMode()) &&
206     (showDropIndicator() == other->showDropIndicator()) &&
207     (tabKeyNavigation() == other->tabKeyNavigation()) &&
208     (textElideMode() == other->textElideMode()) &&
209     // compare all of the properties stored in the attributes model
210     attributesModel()->compare( other->attributesModel() ) &&
211     // compare own properties
212     ((rootIndex().column() == other->rootIndex().column()) &&
213     (rootIndex().row() == other->rootIndex().row()) &&
214     (allowOverlappingDataValueTexts() == other->allowOverlappingDataValueTexts()) &&
215     (antiAliasing() == other->antiAliasing()) &&
216     (percentMode() == other->percentMode()) &&
217     (datasetDimension() == other->datasetDimension()));
218 }

```

#### 9.35.4.15 bool AbstractCartesianDiagram::compare (const [AbstractCartesianDiagram](#) \* other) const [inherited]

Returns true if both diagrams have the same settings.

Definition at line 46 of file KDChartAbstractCartesianDiagram.cpp.

References `KDChart::AbstractCartesianDiagram::referenceDiagram()`, and `KDChart::AbstractCartesianDiagram::referenceDiagramOffset()`.

```

47 {
48     if( other == this ) return true;
49     if( ! other ){
50         //qDebug() << "AbstractCartesianDiagram::compare() cannot compare to Null pointer";
51         return false;
52     }
53     /*
54     qDebug() << "\n                AbstractCartesianDiagram::compare() :";
55         // compare own properties
56     qDebug() <<
57         ((referenceDiagram() == other->referenceDiagram()) &&
58          (!! referenceDiagram()) || (referenceDiagramOffset() == other->referenceDiagramOffset()));
59     */
60     return // compare the base class
61         ( static_cast<const AbstractDiagram*>(this)->compare( other ) ) &&
62         // compare own properties
63         (referenceDiagram() == other->referenceDiagram()) &&
64         (!! referenceDiagram()) || (referenceDiagramOffset() == other->referenceDiagramOffset()));
65 }

```

#### 9.35.4.16 `bool LineDiagram::compare (const LineDiagram * other) const`

Returns true if both diagrams have the same settings.

Definition at line 91 of file `KDChartLineDiagram.cpp`.

References `type()`.

```

92 {
93     if( other == this ) return true;
94     if( ! other ){
95         return false;
96     }
97     /*
98     qDebug() << "\n                LineDiagram::compare() :";
99         // compare own properties
100     qDebug() << (type() == other->type());
101     */
102     return // compare the base class
103         ( static_cast<const AbstractCartesianDiagram*>(this)->compare( other ) ) &&
104         // compare own properties
105         (type() == other->type());
106 }

```

#### 9.35.4.17 `AbstractCoordinatePlane * AbstractDiagram::coordinatePlane () const` [inherited]

The coordinate plane associated with the diagram.

This determines how coordinates in value space are mapped into pixel space. By default this is a [Cartesian-CoordinatePlane](#).

#### Returns:

The coordinate plane associated with the diagram.



Definition at line 220 of file KDChartAbstractDiagram.cpp.

References `d`.

Referenced by `KDChart::AbstractDiagram::checkInvariants()`, `KDChart::AbstractCartesianDiagram::layoutPlanes()`, `KDChart::PolarDiagram::paint()`, `KDChart::AbstractDiagram::paintDataValueTexts()`, `KDChart::AbstractDiagram::paintMarkers()`, `KDChart::AbstractPolarDiagram::polarCoordinatePlane()`, and `KDChart::AbstractCartesianDiagram::setCoordinatePlane()`.

```
221 {
222     return d->plane;
223 }
```

#### 9.35.4.18 `const QPair< QPointF, QPointF > AbstractDiagram::dataBoundaries () const` [inherited]

Return the bottom left and top right data point, that the diagram will display (unless the grid adjusts these values).

This method returns a cached result of calculations done by `calculateDataBoundaries`. Classes derived from [AbstractDiagram](#) must implement the `calculateDataBoundaries` function, to specify their own way of calculating the data boundaries. If derived classes want to force recalculation of the data boundaries, they can call [setDataBoundariesDirty\(\)](#)

Returned value is in diagram coordinates.

Definition at line 225 of file KDChartAbstractDiagram.cpp.

References `KDChart::AbstractDiagram::calculateDataBoundaries()`, and `d`.

Referenced by `KDChart::CartesianCoordinatePlane::getRawDataBoundingRectFromDiagrams()`, `KDChart::PolarCoordinatePlane::layoutDiagrams()`, `KDChart::Plotter::paint()`, `paint()`, and `KDChart::BarDiagram::paint()`.

```
226 {
227     if( d->databoundariesDirty ){
228         d->databoundaries = calculateDataBoundaries ();
229         d->databoundariesDirty = false;
230     }
231     return d->databoundaries;
232 }
```

#### 9.35.4.19 `void AbstractDiagram::dataChanged (const QModelIndex & topLeft, const QModelIndex & bottomRight)` [virtual, inherited]

[reimplemented]

Definition at line 330 of file KDChartAbstractDiagram.cpp.

References `KDChart::AbstractDiagram::setDataBoundariesDirty()`.

```
332 {
333     Q_UNUSED( topLeft );
334     Q_UNUSED( bottomRight );
335     // We are still too dumb to do intelligent updates...
336     setDataBoundariesDirty();
337     scheduleDelayedItemsLayout();
338 }
```

#### 9.35.4.20 void KDChart::AbstractDiagram::dataHidden () [signal, inherited]

This signal is emitted, when the hidden status of at least one data cell was (un)set.

Referenced by KDChart::AbstractDiagram::setHidden().

#### 9.35.4.21 QList< QBrush > AbstractDiagram::datasetBrushes () const [inherited]

The set of dataset brushes currently used, for use in legends, etc.

#### Note:

Cell-level override brushes, if set, take precedence over the dataset values, so you might need to check these too, in order to find the brush, that is used for a single cell.

#### Returns:

The current set of dataset brushes.

Definition at line 1018 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::attributesModelRootIndex(), KDChart::AttributesModel::columnCount(), KDChart::DatasetBrushRole, and KDChart::AbstractDiagram::datasetDimension().

Referenced by KDChart::Legend::setBrushesFromDiagram().

```

1019 {
1020     QList<QBrush> ret;
1021     if( model() == 0 )
1022         return ret;
1023
1024     const int columnCount = attributesModel()->columnCount(attributesModelRootIndex());
1025     for( int i = 0; i < columnCount / datasetDimension(); ++i )
1026         ret << qVariantValue<QBrush>( attributesModel()->headerData( i, Qt::Vertical, DatasetBrushRole ));
1027
1028     return ret;
1029 }
```

#### 9.35.4.22 int AbstractDiagram::datasetDimension () const [inherited]

The dataset dimension of a diagram determines, how many value dimensions it expects each datapoint to have.

For each dimension it will expect one column of values in the model. If the dimensionality is 1, automatic values will be used for the abscissa.

For example a diagram with the default dimension of 1, will have one column per datapoint (the y values) and will use automatic values for the x axis (1, 2, 3, ... n). If the dimension is 2, the diagram will use the first, (and the third, fifth, etc) columns as X values, and the second, (and the fourth, sixth, etc) column as Y values.

#### Returns:

The dataset dimension of the diagram.

Definition at line 1072 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::brush(), KDChart::AbstractDiagram::compare(), KDChart::AbstractDiagram::datasetBrushes(), KDChart::AbstractDiagram::datasetLabels(), KDChart::AbstractDiagram::datasetMarkers(), KDChart::AbstractDiagram::datasetPens(), getCellValues(), KDChart::CartesianCoordinatePlane::getDataDimensionsList(), KDChart::TernaryPointDiagram::paint(), KDChart::TernaryLineDiagram::paint(), KDChart::AbstractDiagram::paintDataValueTexts(), KDChart::AbstractDiagram::paintMarkers(), KDChart::AbstractDiagram::pen(), KDChart::AbstractDiagram::setPen(), KDChart::Plotter::setType(), and setType().

```
1073 {
1074     return d->datasetDimension;
1075 }
```

#### 9.35.4.23 QStringList AbstractDiagram::datasetLabels () const [inherited]

The set of dataset labels currently displayed, for use in legends, etc.

##### Returns:

The set of dataset labels currently displayed.

Definition at line 1005 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::attributesModelRootIndex(), KDChart::AttributesModel::columnCount(), KDChart::AbstractDiagram::datasetDimension(), and KDChart::AttributesModel::headerData().

```
1006 {
1007     QStringList ret;
1008     if( model() == 0 )
1009         return ret;
1010
1011     const int columnCount = attributesModel()->columnCount(attributesModelRootIndex());
1012     for( int i = 0; i < columnCount / datasetDimension(); ++i )
1013         ret << attributesModel()->headerData( i, Qt::Horizontal, Qt::DisplayRole ).toString();
1014
1015     return ret;
1016 }
```

#### 9.35.4.24 QList< [MarkerAttributes](#) > AbstractDiagram::datasetMarkers () const [inherited]

The set of dataset markers currently used, for use in legends, etc.

##### Note:

Cell-level override markers, if set, take precedence over the dataset values, so you might need to check these too, in order to find the marker, that is shown for a single cell.

##### Returns:

The current set of dataset brushes.

Definition at line 1044 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::attributesModelRootIndex(), KDChart::AttributesModel::columnCount(), KDChart::AbstractDiagram::datasetDimension(), and KDChart::DataValueLabelAttributesRole.

```

1045 {
1046     QList<MarkerAttributes> ret;
1047     if( model() == 0 )
1048         return ret;
1049
1050     const int columnCount = attributesModel()->columnCount(attributesModelRootIndex());
1051     for( int i = 0; i < columnCount / datasetDimension(); ++i )
1052     {
1053         const DataValueAttributes a =
1054             qVariantValue<DataValueAttributes>( attributesModel()->headerData( i, Qt::Vertical, DataValueLabelAttributesRole ));
1055         ret << a.markerAttributes();
1056     }
1057     return ret;
1058 }
```

#### 9.35.4.25 `QList< QPen > AbstractDiagram::datasetPens () const` [inherited]

The set of dataset pens currently used, for use in legends, etc.

##### Note:

Cell-level override pens, if set, take precedence over the dataset values, so you might need to check these too, in order to find the pens, that is used for a single cell.

##### Returns:

The current set of dataset pens.

Definition at line 1031 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::attributesModelRootIndex(), KDChart::AttributesModel::columnCount(), KDChart::AbstractDiagram::datasetDimension(), and KDChart::DatasetPenRole.

```

1032 {
1033     QList<QPen> ret;
1034     if( model() == 0 )
1035         return ret;
1036
1037     const int columnCount = attributesModel()->columnCount(attributesModelRootIndex());
1038     for( int i = 0; i < columnCount / datasetDimension(); ++i )
1039         ret << qVariantValue<QPen>( attributesModel()->headerData( i, Qt::Vertical, DatasetPenRole ));
1040
1041     return ret;
1042 }
```

#### 9.35.4.26 `DataValueAttributes AbstractDiagram::dataValueAttributes (const QModelIndex & index) const` [inherited]

Retrieve the [DataValueAttributes](#) for the given index.

This will fall back automatically to what was set at dataset or model level, if there are no datapoint specific settings.

**Parameters:**

*index* The datapoint to retrieve the attributes for.

**Returns:**

The [DataValueAttributes](#) for the given index.

Definition at line 421 of file KDChartAbstractDiagram.cpp.

References [KDChart::AbstractDiagram::attributesModel\(\)](#), [KDChart::AttributesModel::data\(\)](#), and [KDChart::DataValueLabelAttributesRole](#).

```

422 {
423     return qVariantValue<DataValueAttributes>(
424         attributesModel()->data( attributesModel()->mapFromSource(index),
425                                 KDChart::DataValueLabelAttributesRole ));
426 }
```

#### 9.35.4.27 [DataValueAttributes](#) AbstractDiagram::dataValueAttributes (int *column*) const [inherited]

Retrieve the [DataValueAttributes](#) for the given dataset.

This will fall back automatically to what was set at model level, if there are no dataset specific settings.

**Parameters:**

*column* The dataset to retrieve the attributes for.

**Returns:**

The [DataValueAttributes](#) for the given dataset.

Definition at line 414 of file KDChartAbstractDiagram.cpp.

References [KDChart::AbstractDiagram::attributesModel\(\)](#), [KDChart::AbstractDiagram::columnToIndex\(\)](#), [KDChart::AttributesModel::data\(\)](#), and [KDChart::DataValueLabelAttributesRole](#).

```

415 {
416     return qVariantValue<DataValueAttributes>(
417         attributesModel()->data( attributesModel()->mapFromSource(columnToIndex( column )),
418                                 KDChart::DataValueLabelAttributesRole ));
419 }
```

#### 9.35.4.28 [DataValueAttributes](#) AbstractDiagram::dataValueAttributes () const [inherited]

Retrieve the [DataValueAttributes](#) specified globally.

This will fall back automatically to the default settings, if there are no specific settings.

**Returns:**

The global [DataValueAttributes](#).

Definition at line 408 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DataValueLabelAttributesRole, and KDChart::AttributesModel::modelData().

Referenced by KDChart::AbstractDiagram::paintDataValueText(), KDChart::AbstractDiagram::paintMarker(), and KDChart::TernaryLineDiagram::TernaryLineDiagram().

```

409 {
410     return qVariantValue<DataValueAttributes>(
411         attributesModel()->modelData( KDChart::DataValueLabelAttributesRole ) );
412 }
```

#### 9.35.4.29 void AbstractDiagram::doItemsLayout () [virtual, inherited]

[reimplemented]

Definition at line 321 of file KDChartAbstractDiagram.cpp.

References d, and KDChart::AbstractDiagram::update().

```

322 {
323     if ( d->plane ) {
324         d->plane->layoutDiagrams();
325         update();
326     }
327     QAbstractItemView::doItemsLayout();
328 }
```

#### 9.35.4.30 [LineAttributes::MissingValuesPolicy](#) LineDiagram::getCellValues (int row, int column, bool shiftCountedXValuesByHalfSection, double & valueX, double & valueY) const [protected]

Definition at line 399 of file KDChartLineDiagram.cpp.

References KDChart::AbstractDiagram::datasetDimension(), lineAttributes(), KDChart::LineAttributes::missingValuesPolicy(), KDChart::LineAttributes::MissingValuesPolicyIgnored, and valueForCellTesting().

```

403 {
404     LineAttributes::MissingValuesPolicy policy;
405
406     bool bOK = true;
407     valueX = ( datasetDimension() > 1 && column > 0 )
408             ? valueForCellTesting( row, column-1, bOK, true )
409             : ((shiftCountedXValuesByHalfSection ? 0.5 : 0.0) + row);
410     if( bOK )
411         valueY = valueForCellTesting( row, column, bOK, true );
412     if( bOK ){
413         policy = LineAttributes::MissingValuesPolicyIgnored;
414     }else{
415         // missing value: find out the policy
416         QModelIndex index = model()->index( row, column, rootIndex() );
417         LineAttributes la = lineAttributes( index );
418         policy = la.missingValuesPolicy();
419     }
420     return policy;
421 }
```

**9.35.4.31 int AbstractDiagram::horizontalOffset () const** [virtual, inherited]

[reimplemented]

Definition at line 950 of file KDChartAbstractDiagram.cpp.

```
951 { return 0; }
```

**9.35.4.32 QModelIndex AbstractDiagram::indexAt (const QPoint & point) const** [virtual, inherited]

[reimplemented]

Definition at line 1098 of file KDChartAbstractDiagram.cpp.

References d.

```
1099 {  
1100     return d->indexAt( point );  
1101 }
```

**9.35.4.33 QModelIndexList AbstractDiagram::indexesAt (const QPoint & point) const** [inherited]

This method is added alongside with indexAt from QAIM, since in kdchart multiple indexes can be displayed at the same spot.

Definition at line 1103 of file KDChartAbstractDiagram.cpp.

References d.

```
1104 {  
1105     return d->indexesAt( point );  
1106 }
```

**9.35.4.34 bool AbstractDiagram::isHidden (const QModelIndex & index) const** [inherited]

Retrieve the hidden status for the given index.

This will fall back automatically to what was set at dataset or diagram level, if there are no datapoint specific settings.

**Parameters:**

*index* The datapoint to retrieve the hidden status for.

**Returns:**

The hidden status for the given index.

Definition at line 380 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AttributesModel::data(), and KDChart::DataHiddenRole.

```

381 {
382     return qVariantValue<bool>(
383         attributesModel()->data(
384             attributesModel()->mapFromSource(index),
385             DataHiddenRole ) );
386 }

```

#### 9.35.4.35 bool AbstractDiagram::isHidden (int *column*) const [inherited]

Retrieve the hidden status for the given dataset.

This will fall back automatically to what was set at diagram level, if there are no dataset specific settings.

##### Parameters:

*column* The dataset to retrieve the hidden status for.

##### Returns:

The hidden status for the given dataset.

Definition at line 373 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::columnToIndex(), KDChart::AttributesModel::data(), and KDChart::DataHiddenRole.

```

374 {
375     return qVariantValue<bool>(
376         attributesModel()->data(
377             attributesModel()->mapFromSource(columnToIndex( column )),
378             DataHiddenRole ) );
379 }

```

#### 9.35.4.36 bool AbstractDiagram::isHidden () const [inherited]

Retrieve the hidden status specified globally.

This will fall back automatically to the default settings (= not hidden), if there are no specific settings.

##### Returns:

The global hidden status.

Definition at line 367 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DataHiddenRole, and KDChart::AttributesModel::modelData().

Referenced by valueForCellTesting().

```

368 {
369     return qVariantValue<bool>(
370         attributesModel()->modelData( DataHiddenRole ) );
371 }

```



#### 9.35.4.37 bool AbstractDiagram::isIndexHidden (const QModelIndex & *index*) const

[virtual, inherited]

[reimplemented]

Definition at line 956 of file KDChartAbstractDiagram.cpp.

```
957 { return true; }
```

#### 9.35.4.38 QStringList AbstractDiagram::itemRowLabels () const

[inherited]

The set of item row labels currently displayed, for use in Abscissa axes, etc.

##### Returns:

The set of item row labels currently displayed.

Definition at line 989 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::attributesModelRootIndex(), KDChart::AttributesModel::rowCount(), KDChart::AbstractDiagram::unitPrefix(), and KDChart::AbstractDiagram::unitSuffix().

```
990 {
991     QStringList ret;
992     if( model() ){
993         //qDebug() << "AbstractDiagram::itemRowLabels(): " << attributesModel()->rowCount(attributesModelRootIndex());
994         const int rowCount = attributesModel()->rowCount(attributesModelRootIndex());
995         for( int i = 0; i < rowCount; ++i ){
996             //qDebug() << "item row label: " << attributesModel()->headerData( i, Qt::Vertical, Qt::DisplayRole ).toString();
997             ret << unitPrefix( i, Qt::Horizontal, true ) +
998                 attributesModel()->headerData( i, Qt::Vertical, Qt::DisplayRole ).toString() +
999                 unitSuffix( i, Qt::Horizontal, true );
1000         }
1001     }
1002     return ret;
1003 }
```

#### 9.35.4.39 void KDChart::AbstractDiagram::layoutChanged ([AbstractDiagram \\*](#))

[signal, inherited]

Diagrams are supposed to emit this signal, when the layout of one of their element changes.

Layouts can change, for example, when axes are added or removed, or when the configuration was changed in a way that the axes or the diagram itself are displayed in a different geometry. Changes in the diagrams coordinate system also result in the [layoutChanged\(\)](#) signal being emitted.

Referenced by KDChart::AbstractDiagram::setDatasetDimension(), KDChart::AbstractPieDiagram::setPieAttributes(), KDChart::AbstractPieDiagram::setThreeDPieAttributes(), KDChart::Plotter::setType(), setType(), and KDChart::BarDiagram::setType().

#### 9.35.4.40 void KDChart::AbstractCartesianDiagram::layoutPlanes ()

[virtual, inherited]

Triggers layouting of all coordinate planes on the current chart.

Normally you don't need to call this method. It's handled automatically for you.

Definition at line 115 of file `KDChartAbstractCartesianDiagram.cpp`.

References `KDChart::AbstractDiagram::coordinatePlane()`, and `KDChart::AbstractCoordinatePlane::layoutPlanes()`.

Referenced by `KDChart::AbstractCartesianDiagram::addAxis()`, and `KDChart::AbstractCartesianDiagram::takeAxis()`.

```

116 {
117     //qDebug() << "KDChart::AbstractCartesianDiagram::layoutPlanes()";
118     AbstractCoordinatePlane* plane = coordinatePlane();
119     if( plane ){
120         plane->layoutPlanes();
121         //qDebug() << "KDChart::AbstractCartesianDiagram::layoutPlanes() OK";
122     }
123 }
```

#### 9.35.4.41 [LineAttributes](#) `LineDiagram::lineAttributes (const QModelIndex & index) const`

##### Returns:

the line attribute set of the model index *index*

Definition at line 235 of file `KDChartLineDiagram.cpp`.

References `d`, and `KDChart::LineAttributesRole`.

```

237 {
238     return qVariantValue<LineAttributes>(
239         d->attributesModel->data(
240             d->attributesModel->mapFromSource(index),
241             KDChart::LineAttributesRole ) );
242 }
```

#### 9.35.4.42 [LineAttributes](#) `LineDiagram::lineAttributes (int column) const`

##### Returns:

the line attribute set of data set *column*

Definition at line 224 of file `KDChartLineDiagram.cpp`.

References `KDChart::AbstractDiagram::columnToIndex()`, `d`, and `KDChart::LineAttributesRole`.

```

225 {
226     return qVariantValue<LineAttributes>(
227         d->attributesModel->data(
228             d->attributesModel->mapFromSource( columnToIndex( column ) ),
229             KDChart::LineAttributesRole ) );
230 }
```

#### 9.35.4.43 [LineAttributes](#) `LineDiagram::lineAttributes () const`

##### Returns:

the global line attribute set

Definition at line 215 of file KDChartLineDiagram.cpp.

References `d`, and `KDChart::LineAttributesRole`.

Referenced by `getCellValues()`.

```
216 {  
217     return qVariantValue<LineAttributes>(   
218         d->attributesModel->data( KDChart::LineAttributesRole ) );  
219 }
```

#### 9.35.4.44 void KDChart::AbstractDiagram::modelsChanged () [signal, inherited]

This signal is emitted, when either the model or the [AttributesModel](#) is replaced.

Referenced by `KDChart::AbstractDiagram::setAttributesModel()`, and `KDChart::AbstractDiagram::setModel()`.

#### 9.35.4.45 QModelIndex AbstractDiagram::moveCursor (CursorAction *cursorAction*, Qt::KeyboardModifiers *modifiers*) [virtual, inherited]

[reimplemented]

Definition at line 947 of file KDChartAbstractDiagram.cpp.

```
948 { return QModelIndex(); }
```

#### 9.35.4.46 const int LineDiagram::numberOfAbscissaSegments () const [virtual]

Implements [KDChart::AbstractCartesianDiagram](#).

Definition at line 450 of file KDChartLineDiagram.cpp.

References `KDChart::AbstractDiagram::attributesModelRootIndex()`, and `d`.

```
451 {  
452     return d->attributesModel->rowCount( attributesModelRootIndex() );  
453 }
```

#### 9.35.4.47 const int LineDiagram::numberOfOrdinateSegments () const [virtual]

Implements [KDChart::AbstractCartesianDiagram](#).

Definition at line 455 of file KDChartLineDiagram.cpp.

References `KDChart::AbstractDiagram::attributesModelRootIndex()`, and `d`.

```
456 {  
457     return d->attributesModel->columnCount( attributesModelRootIndex() );  
458 }
```

**9.35.4.48 void LineDiagram::paint (PaintContext \* paintContext) [protected, virtual]**

Draw the diagram contents to the rectangle and painter, that are passed in as part of the paint context.

**Parameters:**

*paintContext* All information needed for painting.

Implements [KDChart::AbstractDiagram](#).

Definition at line 423 of file KDChartLineDiagram.cpp.

References [KDChart::AbstractDiagram::checkInvariants\(\)](#), [KDChart::PaintContext::coordinatePlane\(\)](#), [d](#), [KDChart::AbstractDiagram::dataBoundaries\(\)](#), [KDChart::PaintContext::painter\(\)](#), [KDChart::PaintContext::setCoordinatePlane\(\)](#), and [KDChart::AbstractCoordinatePlane::sharedAxisMasterPlane\(\)](#).

Referenced by [paintEvent\(\)](#).

```

424 {
425     // note: Not having any data model assigned is no bug
426     //         but we can not draw a diagram then either.
427     if ( !checkInvariants( true ) ) return;
428     if ( !AbstractGrid::isBoundariesValid(dataBoundaries()) ) return;
429     const PainterSaver p( ctx->painter() );
430     if( model()->rowCount() == 0 || model()->columnCount() == 0 )
431         return; // nothing to paint for us
432
433     AbstractCoordinatePlane* const plane = ctx->coordinatePlane();
434     ctx->setCoordinatePlane( plane->sharedAxisMasterPlane( ctx->painter() ) );
435
436
437     // paint different line types Normal - Stacked - Percent - Default Normal
438     d->implementor->paint( ctx );
439
440     ctx->setCoordinatePlane( plane );
441 }
```

**9.35.4.49 void AbstractDiagram::paintDataValueText (QPainter \* painter, const QModelIndex & index, const QPointF & pos, double value) [inherited]**

Definition at line 468 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::DataValueAttributes::dataLabel\(\)](#), [KDChart::AbstractDiagram::dataValueAttributes\(\)](#), [KDChart::DataValueAttributes::decimalDigits\(\)](#), [KDChart::DataValueAttributes::isVisible\(\)](#), [KDChartEnums::MeasureOrientationMinimum](#), [KDChart::DataValueAttributes::position\(\)](#), [KDChart::DataValueAttributes::prefix\(\)](#), [KDChart::DataValueAttributes::suffix\(\)](#), and [KDChart::DataValueAttributes::textAttributes\(\)](#).

Referenced by [KDChart::RingDiagram::paint\(\)](#), [KDChart::PolarDiagram::paint\(\)](#), and [KDChart::AbstractDiagram::paintDataValueTexts\(\)](#).

```

472 {
473     // paint one data series
474     const DataValueAttributes a( dataValueAttributes(index) );
475     if ( !a.isVisible() ) return;
476
477     // handle decimal digits
478     int decimalDigits = a.decimalDigits();
479     int decimalPos = QString::number( value ).indexOf( QLatin1Char( '.' ) );
480     QString roundedValue;
```

```

481     if ( a.dataLabel().isNull() ) {
482         if ( decimalPos > 0 && value != 0 )
483             roundedValue = roundValues ( value, decimalPos, decimalDigits );
484         else
485             roundedValue = QString::number( value );
486     } else
487         roundedValue = a.dataLabel();
488     // handle prefix and suffix
489     if ( !a.prefix().isNull() )
490         roundedValue.prepend( a.prefix() );
491
492     if ( !a.suffix().isNull() )
493         roundedValue.append( a.suffix() );
494
495     const TextAttributes ta( a.textAttributes() );
496     // FIXME draw the non-text bits, background, etc
497     if ( ta.isVisible() ) {
498
499         QPointF pt( pos );
500         /* for debugging:
501         PainterSaver painterSaver( painter );
502         painter->setPen( Qt::black );
503         painter->drawLine( pos - QPointF( 1,1), pos + QPointF( 1,1) );
504         painter->drawLine( pos - QPointF(-1,1), pos + QPointF(-1,1) );
505         */
506
507         QTextDocument doc;
508         if( Qt::mightBeRichText( roundedValue ) )
509             doc.setHtml( roundedValue );
510         else
511             doc.setPlainText( roundedValue );
512
513         const RelativePosition relPos( a.position( value >= 0.0 ) );
514         const Qt::Alignment alignBottomLeft = Qt::AlignBottom | Qt::AlignLeft;
515         const QFont calculatedFont( ta.calculatedFont( d->plane, KDChartEnums::MeasureOrientationMinimum ) );
516         const QRectF boundRect( d->cachedFontMetrics( calculatedFont, painter->device() )->boundingRect( roundedValue ) );
517
518         // To place correctly
519         pt.ry() -= boundRect.height();
520
521         // qDebug() << "calculatedFont's point size:" << calculatedFont.pointSizeF();
522         // adjust the text start point position, if alignment is not Bottom/Left
523         if( (relPos.alignment() & alignBottomLeft) != alignBottomLeft ) {
524             if( relPos.alignment() & Qt::AlignRight )
525                 pt.rx() -= boundRect.width();
526             else if( relPos.alignment() & Qt::AlignHCenter )
527                 pt.rx() -= 0.5 * boundRect.width();
528
529             if( relPos.alignment() & Qt::AlignTop )
530                 pt.ry() += boundRect.height();
531             else if( relPos.alignment() & Qt::AlignVCenter )
532                 pt.ry() += 0.5 * boundRect.height();
533         }
534
535         // FIXME draw the non-text bits, background, etc
536
537         if ( a.showRepetitiveDataLabels() ||
538             pos.x() <= d->lastX ||
539             d->lastRoundedValue != roundedValue ) {
540             d->lastRoundedValue = roundedValue;
541             d->lastX = pos.x();
542             PainterSaver painterSaver( painter );
543
544             doc.setDefaultFont( calculatedFont );
545             QAbstractTextDocumentLayout::PaintContext context;
546             context.palette = palette();
547             context.palette.setColor( QPalette::Text, ta.pen().color() );

```

```

548
549         painter->translate( pt );
550         painter->rotate( ta.rotation() );
551
552         QAbstractTextDocumentLayout* layout = doc.documentLayout();
553         layout->draw( painter, context );
554     }
555 }
556 }

```

#### 9.35.4.50 void AbstractDiagram::paintDataValueTexts (QPainter \* *painter*) [protected, virtual, inherited]

Definition at line 584 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::checkInvariants(), KDChart::AbstractDiagram::coordinatePlane(), KDChart::AbstractDiagram::datasetDimension(), KDChart::AbstractDiagram::paintDataValueText(), and KDChart::AbstractCoordinatePlane::translate().

```

585 {
586     if ( !checkInvariants() ) return;
587     const int rowCount = model()->rowCount(rootIndex());
588     const int columnCount = model()->columnCount(rootIndex());
589     for ( int i=datasetDimension()-1; i<columnCount; i += datasetDimension() ) {
590         for ( int j=0; j< rowCount; ++j ) {
591             const QModelIndex index = model()->index( j, i, rootIndex() );
592             double value = model()->data( index ).toDouble();
593             const QPointF pos = coordinatePlane()->translate( QPointF( j, value ) );
594             paintDataValueText( painter, index, pos, value );
595         }
596     }
597 }

```

#### 9.35.4.51 void LineDiagram::paintEvent (QPaintEvent \*) [protected]

Definition at line 373 of file KDChartLineDiagram.cpp.

References paint(), KDChart::PaintContext::setPainter(), and KDChart::PaintContext::setRectangle().

```

374 {
375     //qDebug() << "starting LineDiagram::paintEvent ( QPaintEvent*)";
376     QPainter painter ( viewport() );
377     PaintContext ctx;
378     ctx.setPainter ( &painter );
379     ctx.setRectangle ( QRectF ( 0, 0, width(), height() ) );
380     paint ( &ctx );
381     //qDebug() << "          LineDiagram::paintEvent ( QPaintEvent*) ended.";
382 }

```

#### 9.35.4.52 void AbstractDiagram::paintMarker (QPainter \* *painter*, const QModelIndex & *index*, const QPointF & *pos*) [inherited]

Definition at line 600 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::brush(), KDChart::AbstractDiagram::checkInvariants(), d, KDChart::AbstractDiagram::dataValueAttributes(), KDChart::MarkerAttributes::isVisible(), KDChart::DataValueAttributes::isVisible(), and KDChart::DataValueAttributes::markerAttributes().

KDChart::MarkerAttributes::markerColor(), KDChart::MarkerAttributes::markerSize(),  
KDChart::AbstractDiagram::paintMarker(), and KDChart::MarkerAttributes::pen().

```

603 {
604     if ( !checkInvariants() ) return;
605     DataValueAttributes a = dataValueAttributes(index);
606     if ( !a.isVisible() ) return;
607     const MarkerAttributes &ma = a.markerAttributes();
608     if ( !ma.isVisible() ) return;
609
610     PainterSaver painterSaver( painter );
611     QSizeF maSize( ma.markerSize() );
612     QBrush indexBrush( brush( index ) );
613     QPen indexPen( ma.pen() );
614     if ( ma.markerColor().isValid() )
615         indexBrush.setColor( ma.markerColor() );
616
617     paintMarker( painter, ma, indexBrush, indexPen, pos, maSize );
618
619     // workaround: BC cannot be changed, otherwise we would pass the
620     // index down to next-lower paintMarker function. So far, we
621     // basically save a circle of radius maSize at pos in the
622     // reverseMapper. This means that ^^ this version of paintMarker
623     // needs to be called to reverse-map the marker.
624     d->reverseMapper.addCircle( index.row(), index.column(), pos, 2 * maSize );
625 }
```

#### 9.35.4.53 void AbstractDiagram::paintMarker (QPainter \* *painter*, const MarkerAttributes & *markerAttributes*, const QBrush & *brush*, const QPen &, const QPointF & *point*, const QSizeF & *size*) [virtual, inherited]

Definition at line 627 of file KDChartAbstractDiagram.cpp.

References KDChart::MarkerAttributes::Marker1Pixel, KDChart::MarkerAttributes::Marker4Pixels, KDChart::MarkerAttributes::MarkerCircle, KDChart::MarkerAttributes::MarkerCross, KDChart::MarkerAttributes::MarkerDiamond, KDChart::MarkerAttributes::MarkerFastCross, KDChart::MarkerAttributes::MarkerRing, KDChart::MarkerAttributes::MarkerSquare, and KDChart::MarkerAttributes::markerStyle().

Referenced by KDChart::TernaryPointDiagram::paint(), KDChart::TernaryLineDiagram::paint(), KDChart::MarkerLayoutItem::paintIntoRect(), KDChart::AbstractDiagram::paintMarker(), and KDChart::AbstractDiagram::paintMarkers().

```

633 {
634     const QPen oldPen( painter->pen() );
635     // Pen is used to paint 4Pixels - 1 Pixel - Ring and FastCross types.
636     // make sure to use the brush color - see above in those cases.
637     const bool isFourPixels = (markerAttributes.markerStyle() == MarkerAttributes::Marker4Pixels);
638     if( isFourPixels || (markerAttributes.markerStyle() == MarkerAttributes::Marker1Pixel) ){
639         // for high-performance point charts with tiny point markers:
640         painter->setPen( QPen( brush.color().light() ) );
641         if( isFourPixels ){
642             const qreal x = pos.x();
643             const qreal y = pos.y();
644             painter->drawLine( QPointF(x-1.0,y-1.0),
645                             QPointF(x+1.0,y-1.0) );
646             painter->drawLine( QPointF(x-1.0,y),
647                             QPointF(x+1.0,y) );
648             painter->drawLine( QPointF(x-1.0,y+1.0),
649                             QPointF(x+1.0,y+1.0) );
650         }
651         painter->drawPoint( pos );

```

```

652     }else{
653         PainterSaver painterSaver( painter );
654         // we only a solid line surrounding the markers
655         QPen painterPen( pen );
656         painterPen.setStyle( Qt::SolidLine );
657         painter->setPen( painterPen );
658         painter->setBrush( brush );
659         painter->setRenderHint ( QPainter::Antialiasing );
660         painter->translate( pos );
661         switch ( markerAttributes.markerStyle() ) {
662             case MarkerAttributes::MarkerCircle:
663                 painter->drawEllipse( QRectF( 0 - maSize.height()/2, 0 - maSize.width()/2,
664                     maSize.height(), maSize.width() ) );
665                 break;
666             case MarkerAttributes::MarkerSquare:
667                 {
668                     QRectF rect( 0 - maSize.width()/2, 0 - maSize.height()/2,
669                         maSize.width(), maSize.height() );
670                     painter->drawRect( rect );
671                     break;
672                 }
673             case MarkerAttributes::MarkerDiamond:
674                 {
675                     QVector <QPointF > diamondPoints;
676                     QPointF top, left, bottom, right;
677                     top = QPointF( 0, 0 - maSize.height()/2 );
678                     left = QPointF( 0 - maSize.width()/2, 0 );
679                     bottom = QPointF( 0, maSize.height()/2 );
680                     right = QPointF( maSize.width()/2, 0 );
681                     diamondPoints << top << left << bottom << right;
682                     painter->drawPolygon( diamondPoints );
683                     break;
684                 }
685             // both handled on top of the method:
686             case MarkerAttributes::Marker1Pixel:
687             case MarkerAttributes::Marker4Pixels:
688                 break;
689             case MarkerAttributes::MarkerRing:
690                 {
691                     painter->setPen( QPen( brush.color() ) );
692                     painter->setBrush( Qt::NoBrush );
693                     painter->drawEllipse( QRectF( 0 - maSize.height()/2, 0 - maSize.width()/2,
694                         maSize.height(), maSize.width() ) );
695                     break;
696                 }
697             case MarkerAttributes::MarkerCross:
698                 {
699                     QRectF rect( maSize.width()*-0.5, maSize.height()*-0.2,
700                         maSize.width(), maSize.height()*0.4 );
701                     painter->drawRect( rect );
702                     rect.setTopLeft(QPointF( maSize.width()*-0.2, maSize.height()*-0.5 ));
703                     rect.setSize(QSizeF( maSize.width()*0.4, maSize.height() ));
704                     painter->drawRect( rect );
705                     break;
706                 }
707             case MarkerAttributes::MarkerFastCross:
708                 {
709                     QPointF left, right, top, bottom;
710                     left = QPointF( -maSize.width()/2, 0 );
711                     right = QPointF( maSize.width()/2, 0 );
712                     top = QPointF( 0, -maSize.height()/2 );
713                     bottom= QPointF( 0, maSize.height()/2 );
714                     painter->setPen( QPen( brush.color() ) );
715                     painter->drawLine( left, right );
716                     painter->drawLine( top, bottom );
717                     break;
718                 }

```



```

719         default:
720             Q_ASSERT_X ( false, "paintMarkers()",
721                         "Type item does not match a defined Marker Type." );
722     }
723 }
724 painter->setPen( oldPen );
725 }

```

#### 9.35.4.54 void AbstractDiagram::paintMarkers (QPainter \* *painter*) [protected, virtual, inherited]

Definition at line 727 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::checkInvariants(), KDChart::AbstractDiagram::coordinatePlane(), KDChart::AbstractDiagram::datasetDimension(), KDChart::AbstractDiagram::paintMarker(), and KDChart::AbstractCoordinatePlane::translate().

```

728 {
729     if ( !checkInvariants() ) return;
730     const int rowCount = model()->rowCount( rootIndex() );
731     const int columnCount = model()->columnCount( rootIndex() );
732     for ( int i=datasetDimension()-1; i<columnCount; i += datasetDimension() ) {
733         for ( int j=0; j<rowCount; ++j ) {
734             const QModelIndex index = model()->index( j, i, rootIndex() );
735             double value = model()->data( index ).toDouble();
736             const QPointF pos = coordinatePlane()->translate( QPointF( j, value ) );
737             paintMarker( painter, index, pos );
738         }
739     }
740 }

```

#### 9.35.4.55 QPen AbstractDiagram::pen (const QModelIndex & *index*) const [inherited]

Retrieve the pen to be used, for painting the datapoint at the given index in the model.

##### Parameters:

*index* The index of the datapoint in the model.

##### Returns:

The pen to use for painting.

Definition at line 788 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AttributesModel::data(), KDChart::AbstractDiagram::datasetDimension(), KDChart::DatasetPenRole, and KDChart::AbstractDiagram::pen().

```

789 {
790     if( datasetDimension() > 1 )
791         return pen( index.column() );
792     return QVariantValue<QPen>(
793         attributesModel()->data(
794             attributesModel()->mapFromSource( index ),
795             DatasetPenRole ) );
796 }

```

**9.35.4.56 QPen AbstractDiagram::pen (int *dataset*) const** [inherited]

Retrieve the pen to be used for the given dataset.

This will fall back automatically to what was set at model level, if there are no dataset specific settings.

**Parameters:**

*dataset* The dataset to retrieve the pen for.

**Returns:**

The pen to use for painting.

Definition at line 777 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::columnToIndex(), KDChart::AttributesModel::data(), KDChart::AbstractDiagram::datasetDimension(), and KDChart::DatasetPenRole.

```

778 {
779     if( datasetDimension() > 1 )
780         dataset /= datasetDimension();
781
782     return qVariantValue<QPen>(
783         attributesModel()->data(
784             attributesModel()->mapFromSource( columnToIndex( dataset ) ),
785             DatasetPenRole ) );
786 }
```

**9.35.4.57 QPen AbstractDiagram::pen () const** [inherited]

Retrieve the pen to be used for painting datapoints globally.

This will fall back automatically to the default settings, if there are no specific settings.

**Returns:**

The pen to use for painting.

Definition at line 771 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AttributesModel::data(), and KDChart::DatasetPenRole.

Referenced by KDChart::TernaryPointDiagram::paint(), KDChart::TernaryLineDiagram::paint(), and KDChart::AbstractDiagram::pen().

```

772 {
773     return qVariantValue<QPen>(
774         attributesModel()->data( DatasetPenRole ) );
775 }
```

**9.35.4.58 bool AbstractDiagram::percentMode () const** [inherited]

Definition at line 462 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::compare(), and KDChart::CartesianCoordinatePlane::getDataDimensionsList().

```
463 {
464     return d->percent;
465 }
```

#### 9.35.4.59 void KDChart::AbstractDiagram::propertiesChanged () [signal, inherited]

Emitted upon change of a property of the Diagram.

Referenced by KDChart::Plotter::resetLineAttributes(), resetLineAttributes(), KDChart::AbstractDiagram::setAllowOverlappingDataValueTexts(), KDChart::AbstractDiagram::setAntiAliasing(), KDChart::BarDiagram::setBarAttributes(), KDChart::AbstractDiagram::setBrush(), KDChart::AbstractDiagram::setDataValueAttributes(), KDChart::Plotter::setLineAttributes(), setLineAttributes(), KDChart::AbstractDiagram::setPen(), KDChart::AbstractDiagram::setPercentMode(), KDChart::BarDiagram::setThreeDBarAttributes(), KDChart::Plotter::setThreeDLineAttributes(), setThreeDLineAttributes(), KDChart::Plotter::setType(), setType(), KDChart::BarDiagram::setType(), KDChart::Plotter::setValueTrackerAttributes(), and setValueTrackerAttributes().

#### 9.35.4.60 AbstractCartesianDiagram \* AbstractCartesianDiagram::referenceDiagram () const [virtual, inherited]

**Returns:**

this diagram's reference diagram

**See also:**

[setReferenceDiagram](#)

Definition at line 148 of file KDChartAbstractCartesianDiagram.cpp.

References d.

Referenced by KDChart::AbstractCartesianDiagram::compare(), and referenceDiagramIsBarDiagram().

```
149 {
150     return d->referenceDiagram;
151 }
```

#### 9.35.4.61 QPointF AbstractCartesianDiagram::referenceDiagramOffset () const [virtual, inherited]

**Returns:**

the relative offset of this diagram's reference diagram

**See also:**

[setReferenceDiagram](#)

Definition at line 153 of file KDChartAbstractCartesianDiagram.cpp.

References [d](#).

Referenced by [KDChart::AbstractCartesianDiagram::compare\(\)](#).

```
154 {
155     return d->referenceDiagramOffset;
156 }
```

#### 9.35.4.62 void LineDiagram::resetLineAttributes (const QModelIndex & *index*)

Remove any explicit line attributes settings that might have been specified before.

Definition at line 205 of file KDChartLineDiagram.cpp.

References [d](#), [KDChart::LineAttributesRole](#), and [KDChart::AbstractDiagram::propertiesChanged\(\)](#).

```
206 {
207     d->attributesModel->resetData(
208         d->attributesModel->mapFromSource(index), LineAttributesRole );
209     emit propertiesChanged();
210 }
```

#### 9.35.4.63 void LineDiagram::resetLineAttributes (int *column*)

Resets the line attributes of data set *column*.

Definition at line 181 of file KDChartLineDiagram.cpp.

References [d](#), [KDChart::LineAttributesRole](#), and [KDChart::AbstractDiagram::propertiesChanged\(\)](#).

```
182 {
183     d->attributesModel->resetHeaderData(
184         column, Qt::Vertical, LineAttributesRole );
185     emit propertiesChanged();
186 }
```

#### 9.35.4.64 void LineDiagram::resize (const QSizeF & *area*) [virtual]

Called by the widget's `sizeEvent`.

Adjust all internal structures, that are calculated, depending on the size of the widget.

**Parameters:**

*area*

Implements [KDChart::AbstractDiagram](#).

Definition at line 443 of file KDChartLineDiagram.cpp.

References [d](#), and [KDChart::AbstractDiagram::setDataBoundariesDirty\(\)](#).

```
444 {
445     d->compressor.setResolution( static_cast<int>( size.width() ),
446                               static_cast<int>( size.height() ) );
447     setDataBoundariesDirty();
448 }
```

**9.35.4.65 void LineDiagram::resizeEvent (QResizeEvent \*)** [protected]

Definition at line 356 of file KDChartLineDiagram.cpp.

```
357 {  
358 }
```

**9.35.4.66 void AbstractDiagram::scrollTo (const QModelIndex & index, ScrollHint hint = EnsureVisible)** [virtual, inherited]

[reimplemented]

Definition at line 942 of file KDChartAbstractDiagram.cpp.

```
943 {}
```

**9.35.4.67 void AbstractDiagram::setAllowOverlappingDataValueTexts (bool allow)** [inherited]

Set whether data value labels are allowed to overlap.

**Parameters:**

*allow* True means that overlapping labels are allowed.

Definition at line 434 of file KDChartAbstractDiagram.cpp.

References `d`, and `KDChart::AbstractDiagram::propertiesChanged()`.

```
435 {  
436     d->allowOverlappingDataValueTexts = allow;  
437     emit propertiesChanged();  
438 }
```

**9.35.4.68 void AbstractDiagram::setAntiAliasing (bool enabled)** [inherited]

Set whether anti-aliasing is to be used while rendering this diagram.

**Parameters:**

*enabled* True means that AA is enabled.

Definition at line 445 of file KDChartAbstractDiagram.cpp.

References `d`, and `KDChart::AbstractDiagram::propertiesChanged()`.

```
446 {  
447     d->antiAliasing = enabled;  
448     emit propertiesChanged();  
449 }
```

#### 9.35.4.69 void AbstractCartesianDiagram::setAttributesModel ([AttributesModel](#) \* *model*) [virtual, inherited]

Associate an [AttributesModel](#) with this diagram.

Note that the diagram does *\_not\_* take ownership of the [AttributesModel](#). This should thus only be used with AttributesModels that have been explicitly created by the user, and are owned by her. Setting an [AttributesModel](#) that is internal to another diagram is an error.

Correct:

```
AttributesModel *am = new AttributesModel( model, 0 );
diagram1->setAttributesModel( am );
diagram2->setAttributesModel( am );
```

Wrong:

```
diagram1->setAttributesModel( diagram2->attributesModel() );
```

#### Parameters:

*model* The [AttributesModel](#) to use for this diagram.

#### See also:

[AttributesModel](#), [usesExternalAttributesModel](#)

Reimplemented from [KDChart::AbstractDiagram](#).

Definition at line 170 of file `KDChartAbstractCartesianDiagram.cpp`.

References `KDChart::AbstractDiagram::attributesModel()`, `d`, and `KDChart::AbstractDiagram::setAttributesModel()`.

```
171 {
172     AbstractDiagram::setAttributesModel( model );
173     d->compressor.setModel( attributesModel() );
174 }
```

#### 9.35.4.70 void AbstractDiagram::setAttributesModelRootIndex (const QModelIndex &) [protected, inherited]

Definition at line 293 of file `KDChartAbstractDiagram.cpp`.

References `d`, and `KDChart::AbstractDiagram::setDataBoundariesDirty()`.

Referenced by `KDChart::AbstractDiagram::setRootIndex()`.

```
294 {
295     d->attributesModelRootIndex=idx;
296     setDataBoundariesDirty();
297     scheduleDelayedItemsLayout();
298 }
```

**9.35.4.71 void AbstractDiagram::setBrush (const QBrush & *brush*)** [inherited]

Set the brush to be used, for painting all datasets in the model.

**Parameters:**

***brush*** The brush to use.

Definition at line 806 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DatasetBrushRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setModelData().

```
807 {
808     attributesModel()->setModelData(
809         qVariantFromValue( brush ), DatasetBrushRole );
810     emit propertiesChanged();
811 }
```

**9.35.4.72 void AbstractDiagram::setBrush (int *dataset*, const QBrush & *brush*)** [inherited]

Set the brush to be used, for painting the given dataset.

**Parameters:**

***dataset*** The dataset's column in the model.

***brush*** The brush to use.

Definition at line 813 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DatasetBrushRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setHeaderData().

```
814 {
815     attributesModel()->setHeaderData(
816         column, Qt::Vertical,
817         qVariantFromValue( brush ),
818         DatasetBrushRole );
819     emit propertiesChanged();
820 }
```

**9.35.4.73 void AbstractDiagram::setBrush (const QModelIndex & *index*, const QBrush & *brush*)** [inherited]

Set the brush to be used, for painting the datapoint at the given index.

**Parameters:**

***index*** The datapoint's index in the model.

***brush*** The brush to use.

Definition at line 798 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DatasetBrushRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setData().

```

799 {
800     attributesModel()->setData(
801         attributesModel()->mapFromSource( index ),
802         qVariantFromValue( brush ), DatasetBrushRole );
803     emit propertiesChanged();
804 }

```

#### 9.35.4.74 void KDChart::AbstractCartesianDiagram::setCoordinatePlane ([AbstractCoordinatePlane](#) \*plane) [virtual, inherited]

[reimplemented]

Reimplemented from [KDChart::AbstractDiagram](#).

Definition at line 125 of file KDChartAbstractCartesianDiagram.cpp.

References [KDChart::AbstractDiagram::coordinatePlane\(\)](#), and [KDChart::AbstractDiagram::setCoordinatePlane\(\)](#).

```

126 {
127     if( coordinatePlane() ) disconnect( coordinatePlane() );
128     AbstractDiagram::setCoordinatePlane(plane);
129
130     // show the axes, after all have been adjusted
131     // (because they might be dependend on each other)
132     /*
133     if( plane )
134         Q_FOREACH( CartesianAxis* axis, d->axesList )
135             axis->show();
136     else
137         Q_FOREACH( CartesianAxis* axis, d->axesList )
138             axis->hide();
139     */
140 }

```

#### 9.35.4.75 void AbstractDiagram::setDataBoundariesDirty () const [protected, inherited]

Definition at line 234 of file KDChartAbstractDiagram.cpp.

References [d](#).

Referenced by [KDChart::AbstractDiagram::dataChanged\(\)](#), [KDChart::Plotter::resize\(\)](#), [resize\(\)](#), [KDChart::BarDiagram::resize\(\)](#), [KDChart::AbstractDiagram::setAttributesModel\(\)](#), [KDChart::AbstractDiagram::setAttributesModelRootIndex\(\)](#), [KDChart::AbstractDiagram::setDatasetDimension\(\)](#), [KDChart::AbstractDiagram::setModel\(\)](#), [KDChart::BarDiagram::setThreeDBarAttributes\(\)](#), [KDChart::Plotter::setThreeDLineAttributes\(\)](#), [setThreeDLineAttributes\(\)](#), [KDChart::Plotter::setType\(\)](#), [setType\(\)](#), and [KDChart::BarDiagram::setType\(\)](#).

```

235 {
236     d->databoundariesDirty = true;
237 }

```

#### 9.35.4.76 void AbstractDiagram::setDatasetDimension (int dimension) [inherited]

Sets the dataset dimension of the diagram.



See also:

[datasetDimension](#).

**Parameters:**

*dimension*

Definition at line 1077 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::AbstractDiagram::layoutChanged\(\)](#), and [KDChart::AbstractDiagram::setDataBoundariesDirty\(\)](#).

Referenced by [KDChart::Widget::setType\(\)](#), [KDChart::TernaryLineDiagram::TernaryLineDiagram\(\)](#), and [KDChart::TernaryPointDiagram::TernaryPointDiagram\(\)](#).

```
1078 {
1079     if ( d->datasetDimension == dimension ) return;
1080     d->datasetDimension = dimension;
1081     setDataBoundariesDirty();
1082     emit layoutChanged( this );
1083 }
```

#### 9.35.4.77 void AbstractDiagram::setDataValueAttributes (const [DataValueAttributes](#) & *a*) [inherited]

Set the [DataValueAttributes](#) for all datapoints in the model.

**Parameters:**

*a* The attributes to set.

Definition at line 428 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::DataValueLabelAttributesRole](#), and [KDChart::AbstractDiagram::propertiesChanged\(\)](#).

```
429 {
430     d->attributesModel->setModelData( qVariantFromValue( a ), DataValueLabelAttributesRole );
431     emit propertiesChanged();
432 }
```

#### 9.35.4.78 void AbstractDiagram::setDataValueAttributes (int *dataset*, const [DataValueAttributes](#) & *a*) [inherited]

Set the [DataValueAttributes](#) for the given dataset.

**Parameters:**

*dataset* The dataset to set the attributes for.

*a* The attributes to set.

Definition at line 400 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::DataValueLabelAttributesRole](#), and [KDChart::AbstractDiagram::propertiesChanged\(\)](#).

```

401 {
402     d->attributesModel->setHeaderData (
403         column, Qt::Vertical,
404         qVariantFromValue( a ), DataValueLabelAttributesRole );
405     emit propertiesChanged();
406 }

```

#### 9.35.4.79 void AbstractDiagram::setDataValueAttributes (const QModelIndex & *index*, const [DataValueAttributes](#) & *a*) [inherited]

Set the [DataValueAttributes](#) for the given index.

##### Parameters:

- index* The datapoint to set the attributes for.
- a* The attributes to set.

Definition at line 389 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::DataValueLabelAttributesRole](#), and [KDChart::AbstractDiagram::propertiesChanged\(\)](#).

```

391 {
392     d->attributesModel->setData (
393         d->attributesModel->mapFromSource( index ),
394         qVariantFromValue( a ),
395         DataValueLabelAttributesRole );
396     emit propertiesChanged();
397 }

```

#### 9.35.4.80 void AbstractDiagram::setHidden (bool *hidden*) [inherited]

Hide (or unhide, resp.

) all datapoints in the model.

##### Note:

Hidden data are still taken into account by the coordinate plane, so neither the grid nor your axes' ranges will change, when you hide data. For totally removing data from KD Chart's view you can use another approach: e.g. you could define a proxy model on top of your data model, and register the proxy model calling [setModel\(\)](#) instead of registering your real data model.

##### Parameters:

- hidden* The hidden status to set.

Definition at line 359 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::AbstractDiagram::dataHidden\(\)](#), and [KDChart::DataHiddenRole](#).

```

360 {
361     d->attributesModel->setModelData (
362         qVariantFromValue( hidden ),
363         DataHiddenRole );
364     emit dataHidden();
365 }

```

**9.35.4.81 void AbstractDiagram::setHidden (int *column*, bool *hidden*)** [inherited]

Hide (or unhide, resp.

) a dataset.

**Note:**

Hidden data are still taken into account by the coordinate plane, so neither the grid nor your axes' ranges will change, when you hide data. For totally removing data from KD Chart's view you can use another approach: e.g. you could define a proxy model on top of your data model, and register the proxy model calling [setModel\(\)](#) instead of registering your real data model.

**Parameters:**

***column*** The dataset to set the hidden status for.

***hidden*** The hidden status to set.

Definition at line 350 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::AbstractDiagram::dataHidden\(\)](#), and [KDChart::DataHiddenRole](#).

```
351 {  
352     d->attributesModel->setHeaderData(  
353         column, Qt::Vertical,  
354         qVariantFromValue( hidden ),  
355         DataHiddenRole );  
356     emit dataHidden();  
357 }
```

**9.35.4.82 void AbstractDiagram::setHidden (const QModelIndex & *index*, bool *hidden*)**  
[inherited]

Hide (or unhide, resp.

) a data cell.

**Note:**

Hidden data are still taken into account by the coordinate plane, so neither the grid nor your axes' ranges will change, when you hide data. For totally removing data from KD Chart's view you can use another approach: e.g. you could define a proxy model on top of your data model, and register the proxy model calling [setModel\(\)](#) instead of registering your real data model.

**Parameters:**

***index*** The datapoint to set the hidden status for.

***hidden*** The hidden status to set.

Definition at line 341 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::AbstractDiagram::dataHidden\(\)](#), and [KDChart::DataHiddenRole](#).

```
342 {  
343     d->attributesModel->setData(  
344         d->attributesModel->mapFromSource( index ),  
345         qVariantFromValue( hidden ),  
346         DataHiddenRole );  
347     emit dataHidden();  
348 }
```

#### 9.35.4.83 void LineDiagram::setLineAttributes (const QModelIndex & *index*, const [LineAttributes](#) & *a*)

Sets the line attributes for the model index *index* to *la*.

Definition at line 191 of file KDChartLineDiagram.cpp.

References [d](#), [KDChart::LineAttributesRole](#), and [KDChart::AbstractDiagram::propertiesChanged\(\)](#).

```

194 {
195     d->attributesModel->setData(
196         d->attributesModel->mapFromSource(index),
197         qVariantFromValue( la ),
198         LineAttributesRole );
199     emit propertiesChanged();
200 }
```

#### 9.35.4.84 void LineDiagram::setLineAttributes (int *column*, const [LineAttributes](#) & *a*)

Sets the line attributes of data set *column* to *la*.

Definition at line 166 of file KDChartLineDiagram.cpp.

References [d](#), [KDChart::LineAttributesRole](#), and [KDChart::AbstractDiagram::propertiesChanged\(\)](#).

```

169 {
170     d->attributesModel->setHeaderData(
171         column,
172         Qt::Vertical,
173         qVariantFromValue( la ),
174         LineAttributesRole );
175     emit propertiesChanged();
176 }
```

#### 9.35.4.85 void LineDiagram::setLineAttributes (const [LineAttributes](#) & *a*)

Sets the global line attributes to *la*.

Definition at line 155 of file KDChartLineDiagram.cpp.

References [d](#), [KDChart::LineAttributesRole](#), and [KDChart::AbstractDiagram::propertiesChanged\(\)](#).

```

156 {
157     d->attributesModel->setModelData(
158         qVariantFromValue( la ),
159         LineAttributesRole );
160     emit propertiesChanged();
161 }
```

#### 9.35.4.86 void AbstractCartesianDiagram::setModel (QAbstractItemModel \* *model*) [virtual, inherited]

Associate a model with the diagram.

Reimplemented from [KDChart::AbstractDiagram](#).

Definition at line 164 of file KDChartAbstractCartesianDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), *d*, and KDChart::AbstractDiagram::setModel().

```

165 {
166     AbstractDiagram::setModel( model );
167     d->compressor.setModel( attributesModel() );
168 }
```

#### 9.35.4.87 void AbstractDiagram::setPen (const QPen & *pen*) [inherited]

Set the pen to be used, for painting all datasets in the model.

##### Parameters:

*pen* The pen to use.

Definition at line 753 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DatasetPenRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setModelData().

```

754 {
755     attributesModel()->setModelData(
756         qVariantFromValue( pen ), DatasetPenRole );
757     emit propertiesChanged();
758 }
```

#### 9.35.4.88 void AbstractDiagram::setPen (int *dataset*, const QPen & *pen*) [inherited]

Set the pen to be used, for painting the given dataset.

##### Parameters:

*dataset* The dataset's row in the model.

*pen* The pen to use.

Definition at line 760 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::datasetDimension(), KDChart::DatasetPenRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setHeaderData().

```

761 {
762     if( datasetDimension() > 1 )
763         column *= datasetDimension();
764     attributesModel()->setHeaderData(
765         column, Qt::Vertical,
766         qVariantFromValue( pen ),
767         DatasetPenRole );
768     emit propertiesChanged();
769 }
```

### 9.35.4.89 void AbstractDiagram::setPen (const QModelIndex & *index*, const QPen & *pen*) [inherited]

Set the pen to be used, for painting the datapoint at the given index.

#### Parameters:

***index*** The datapoint's index in the model.

***pen*** The pen to use.

Definition at line 743 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::datasetDimension(), KDChart::DatasetPenRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setData().

```
744 {
745     if( datasetDimension() > 1 )
746         return setPen( index.column(), pen );
747     attributesModel()->setData(
748         attributesModel()->mapFromSource( index ),
749         qVariantFromValue( pen ), DatasetPenRole );
750     emit propertiesChanged();
751 }
```

### 9.35.4.90 void AbstractDiagram::setPercentMode (bool *percent*) [inherited]

Definition at line 456 of file KDChartAbstractDiagram.cpp.

References d, and KDChart::AbstractDiagram::propertiesChanged().

Referenced by setType(), and KDChart::BarDiagram::setType().

```
457 {
458     d->percent = percent;
459     emit propertiesChanged();
460 }
```

### 9.35.4.91 void AbstractCartesianDiagram::setReferenceDiagram ([AbstractCartesianDiagram](#) \* *diagram*, const QPointF & *offset* = QPointF()) [virtual, inherited]

Makes this diagram use another diagram *diagram* as reference diagram with relative offset *offset*.

To share cartesian axes between different diagrams there might be cases when you need that. Normally you don't.

#### See also:

examples/SharedAbscissa

Definition at line 142 of file KDChartAbstractCartesianDiagram.cpp.

References d.

```
143 {
144     d->referenceDiagram = diagram;
145     d->referenceDiagramOffset = offset;
146 }
```

#### 9.35.4.92 void AbstractCartesianDiagram::setRootIndex (const QModelIndex & *index*) [virtual, inherited]

Set the root index in the model, where the diagram starts referencing data for display.

[reimplemented]

Reimplemented from [KDChart::AbstractDiagram](#).

Definition at line 158 of file KDChartAbstractCartesianDiagram.cpp.

References [d](#), and [KDChart::AbstractDiagram::setRootIndex\(\)](#).

```
159 {
160     d->compressor.setRootIndex( index );
161     AbstractDiagram::setRootIndex( index );
162 }
```

#### 9.35.4.93 void AbstractDiagram::setSelection (const QRect & *rect*, QItemSelectionModel::SelectionFlags *command*) [virtual, inherited]

[reimplemented]

Definition at line 959 of file KDChartAbstractDiagram.cpp.

References [d](#).

```
960 {
961     const QModelIndexList indexes = d->indexesIn( rect );
962     QItemSelection selection;
963     KDAB_FOREACH( const QModelIndex& index, indexes )
964     {
965         selection.append( QItemSelectionRange( index ) );
966     }
967     selectionModel()->select( selection, command );
968 }
```

#### 9.35.4.94 void LineDiagram::setThreeDLineAttributes (const QModelIndex & *index*, const [ThreeDLineAttributes](#) & *a*)

Sets the 3D line attributes of model index *index* to *la*.

Definition at line 276 of file KDChartLineDiagram.cpp.

References [d](#), [KDChart::AbstractDiagram::propertiesChanged\(\)](#), [KDChart::AbstractDiagram::setDataBoundariesDirty\(\)](#), and [KDChart::ThreeDLineAttributesRole](#).

```
279 {
280     setDataBoundariesDirty();
281     d->attributesModel->setData(
282         d->attributesModel->mapFromSource( index ),
283         QVariantFromValue( la ),
284         ThreeDLineAttributesRole );
285     emit propertiesChanged();
286 }
```

#### 9.35.4.95 void LineDiagram::setThreeDLineAttributes (int *column*, const [ThreeDLineAttributes](#) & *a*)

Sets the 3D line attributes of data set *column* to *ta*.

Definition at line 260 of file KDChartLineDiagram.cpp.

References [d](#), [KDChart::AbstractDiagram::propertiesChanged\(\)](#), [KDChart::AbstractDiagram::setDataBoundariesDirty\(\)](#), and [KDChart::ThreeDLineAttributesRole](#).

```

263 {
264     setDataBoundariesDirty();
265     d->attributesModel->setHeaderData(
266         column,
267         Qt::Vertical,
268         qVariantFromValue( la ),
269         ThreeDLineAttributesRole );
270     emit propertiesChanged();
271 }
```

#### 9.35.4.96 void LineDiagram::setThreeDLineAttributes (const [ThreeDLineAttributes](#) & *a*)

Sets the global 3D line attributes to *la*.

Definition at line 247 of file KDChartLineDiagram.cpp.

References [d](#), [KDChart::AbstractDiagram::propertiesChanged\(\)](#), [KDChart::AbstractDiagram::setDataBoundariesDirty\(\)](#), and [KDChart::ThreeDLineAttributesRole](#).

```

249 {
250     setDataBoundariesDirty();
251     d->attributesModel->setModelData(
252         qVariantFromValue( la ),
253         ThreeDLineAttributesRole );
254     emit propertiesChanged();
255 }
```

#### 9.35.4.97 void LineDiagram::setType (const [LineType](#) *type*)

Sets the line diagram's type to *type*.

See also:

[LineDiagram::LineType](#)

Definition at line 112 of file KDChartLineDiagram.cpp.

References [d](#), [KDChart::AbstractDiagram::datasetDimension\(\)](#), [KDChart::AbstractDiagram::layoutChanged\(\)](#), [Normal](#), [Percent](#), [KDChart::AbstractDiagram::propertiesChanged\(\)](#), [KDChart::AbstractDiagram::setDataBoundariesDirty\(\)](#), [KDChart::AbstractDiagram::setPercentMode\(\)](#), and [Stacked](#).

Referenced by [clone\(\)](#).

```

113 {
114     if ( d->implementor->type() == type ) return;
115     if ( type != LineDiagram::Normal && datasetDimension() > 1 ) {
116         Q_ASSERT_X ( false, "setType()",
117                     "This line chart type can't be used with multi-dimensional data." );
```



```

118         return;
119     }
120     switch( type ) {
121     case Normal:
122         d->implementor = d->normalDiagram;
123         break;
124     case Stacked:
125         d->implementor = d->stackedDiagram;
126         break;
127     case Percent:
128         d->implementor = d->percentDiagram;
129         break;
130     default:
131         Q_ASSERT_X( false, "LineDiagram::setType", "unknown diagram subtype" );
132     };
133
134     // d->lineType = type;
135     Q_ASSERT( d->implementor->type() == type );
136
137     // AbstractAxis settings - see AbstractDiagram and CartesianAxis
138     setPercentMode( type == LineDiagram::Percent );
139     setDataBoundariesDirty();
140     emit layoutChanged( this );
141     emit propertiesChanged();
142 }

```

#### 9.35.4.98 void AbstractDiagram::setUnitPrefix (const QString & *prefix*, Qt::Orientation *orientation*) [inherited]

Sets the unit prefix for all values.

##### Parameters:

*prefix* the prefix to be set

*orientation* the orientation of the axis to set

Definition at line 864 of file KDChartAbstractDiagram.cpp.

References d.

```

865 {
866     d->unitPrefix[ orientation ] = prefix;
867 }

```

#### 9.35.4.99 void AbstractDiagram::setUnitPrefix (const QString & *prefix*, int *column*, Qt::Orientation *orientation*) [inherited]

Sets the unit prefix for one value.

##### Parameters:

*prefix* the prefix to be set

*column* the value using that prefix

*orientation* the orientation of the axis to set

Definition at line 854 of file KDChartAbstractDiagram.cpp.

References d.

```

855 {
856     d->unitPrefixMap[ column ][ orientation ]= prefix;
857 }

```

#### 9.35.4.100 void AbstractDiagram::setUnitSuffix (const QString & *suffix*, Qt::Orientation *orientation*) [inherited]

Sets the unit suffix for all values.

##### Parameters:

*suffix* the suffix to be set  
*orientation* the orientation of the axis to set

Definition at line 885 of file KDChartAbstractDiagram.cpp.

References d.

```

886 {
887     d->unitSuffix[ orientation ] = suffix;
888 }

```

#### 9.35.4.101 void AbstractDiagram::setUnitSuffix (const QString & *suffix*, int *column*, Qt::Orientation *orientation*) [inherited]

Sets the unit suffix for one value.

##### Parameters:

*suffix* the suffix to be set  
*column* the value using that suffix  
*orientation* the orientation of the axis to set

Definition at line 875 of file KDChartAbstractDiagram.cpp.

References d.

```

876 {
877     d->unitSuffixMap[ column ][ orientation ]= suffix;
878 }

```

#### 9.35.4.102 void LineDiagram::setValueTrackerAttributes (const QModelIndex & *index*, const ValueTrackerAttributes & *a*)

Sets the value tracker attributes of the model index *index* to *va*.

Definition at line 336 of file KDChartLineDiagram.cpp.

References d, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::ValueTrackerAttributesRole.

```

338 {
339     d->attributesModel->setData( d->attributesModel->mapFromSource(index),
340                               qVariantFromValue( va ),
341                               KDChart::ValueTrackerAttributesRole );
342     emit propertiesChanged();
343 }

```

**9.35.4.103** `void AbstractCartesianDiagram::takeAxis (CartesianAxis * axis)` [virtual, inherited]

Removes the axis from the diagram, without deleting it.

The diagram no longer owns the axis, so it is the caller's responsibility to delete the axis.

**See also:**

[addAxis](#)

Definition at line 100 of file KDChartAbstractCartesianDiagram.cpp.

References `d`, `KDChart::AbstractAxis::deleteObserver()`, `KDChart::AbstractCartesianDiagram::layoutPlanes()`, and `KDChart::AbstractLayoutItem::setParentWidget()`.

Referenced by `KDChart::Widget::setType()`, and `KDChart::CartesianAxis::~~CartesianAxis()`.

```
101 {
102     const int idx = d->axesList.indexOf( axis );
103     if( idx != -1 )
104         d->axesList.takeAt( idx );
105     axis->deleteObserver( this );
106     axis->setParentWidget( 0 );
107     layoutPlanes();
108 }
```

**9.35.4.104** `double LineDiagram::threeDItemDepth (int column) const` [protected, virtual]

**Returns:**

the 3D item depth of the data set *column*

Implements [KDChart::AbstractCartesianDiagram](#).

Definition at line 324 of file KDChartLineDiagram.cpp.

References `d`, and `KDChart::ThreeDLineAttributesRole`.

```
325 {
326     return qVariantValue<ThreeDLineAttributes>(
327         d->attributesModel->headerData (
328             column,
329             Qt::Vertical,
330             KDChart::ThreeDLineAttributesRole ) ).validDepth();
331 }
```

**9.35.4.105** `double LineDiagram::threeDItemDepth (const QModelIndex & index) const` [protected, virtual]

**Returns:**

the 3D item depth of the model index *index*

Implements [KDChart::AbstractCartesianDiagram](#).

Definition at line 319 of file KDChartLineDiagram.cpp.

References `threeDLineAttributes()`, and `KDChart::AbstractThreeDAttributes::validDepth()`.

```

320 {
321     return threeDLineAttributes( index ).validDepth();
322 }

```

#### 9.35.4.106 **ThreeDLineAttributes** **LineDiagram::threeDLineAttributes (const QModelIndex & index) const**

##### Returns:

the 3D line attributes of the model index *index*

Definition at line 311 of file KDChartLineDiagram.cpp.

References d, and KDChart::ThreeDLineAttributesRole.

```

312 {
313     return QVariantValue<ThreeDLineAttributes>(
314         d->attributesModel->data(
315             d->attributesModel->mapFromSource( index ),
316             KDChart::ThreeDLineAttributesRole ) );
317 }

```

#### 9.35.4.107 **ThreeDLineAttributes** **LineDiagram::threeDLineAttributes (int column) const**

##### Returns:

the 3D line attributes of data set *column*

Definition at line 300 of file KDChartLineDiagram.cpp.

References KDChart::AbstractDiagram::columnToIndex(), d, and KDChart::ThreeDLineAttributesRole.

```

301 {
302     return QVariantValue<ThreeDLineAttributes>(
303         d->attributesModel->data(
304             d->attributesModel->mapFromSource( columnToIndex( column ) ),
305             KDChart::ThreeDLineAttributesRole ) );
306 }

```

#### 9.35.4.108 **ThreeDLineAttributes** **LineDiagram::threeDLineAttributes () const**

##### Returns:

the global 3D line attributes

Definition at line 291 of file KDChartLineDiagram.cpp.

References d, and KDChart::ThreeDLineAttributesRole.

Referenced by threeDItemDepth().

```

292 {
293     return QVariantValue<ThreeDLineAttributes>(
294         d->attributesModel->data( KDChart::ThreeDLineAttributesRole ) );
295 }

```

**9.35.4.109** [LineDiagram::LineType](#) LineDiagram::type () const**Returns:**

the type of the line diagram

Definition at line 147 of file KDChartLineDiagram.cpp.

References d.

Referenced by clone(), and compare().

```
148 {
149     return d->implementor->type();
150 }
```

**9.35.4.110** [QString AbstractDiagram::unitPrefix \(Qt::Orientation \*orientation\*\)](#) const  
[inherited]

Returns the global unit prefix.

**Parameters:**

*orientation* the orientation of the axis

**Returns:**

the unit prefix

Definition at line 908 of file KDChartAbstractDiagram.cpp.

References d.

```
909 {
910     return d->unitPrefix[ orientation ];
911 }
```

**9.35.4.111** [QString AbstractDiagram::unitPrefix \(int \*column\*, Qt::Orientation \*orientation\*, bool \*fallback\* = false\)](#) const [inherited]

Returns the unit prefix for a special value.

**Parameters:**

*column* the value which's prefix is requested

*orientation* the orientation of the axis

*fallback* if true, the global prefix is return when no specific one is set for that value

**Returns:**

the unit prefix

Definition at line 897 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::itemRowLabels(), and KDChart::CartesianAxis::paintCtx().

```

898 {
899     if( !fallback || d->unitPrefixMap[ column ].contains( orientation ) )
900         return d->unitPrefixMap[ column ][ orientation ];
901     return d->unitPrefix[ orientation ];
902 }

```

#### 9.35.4.112 QString AbstractDiagram::unitSuffix (Qt::Orientation *orientation*) const [inherited]

Returns the global unit suffix.

##### Parameters:

*orientation* the orientation of the axis

##### Returns:

the unit suffix

Definition at line 931 of file KDChartAbstractDiagram.cpp.

References d.

```

932 {
933     return d->unitSuffix[ orientation ];
934 }

```

#### 9.35.4.113 QString AbstractDiagram::unitSuffix (int *column*, Qt::Orientation *orientation*, bool *fallback* = false) const [inherited]

Returns the unit suffix for a special value.

##### Parameters:

*column* the value which's suffix is requested

*orientation* the orientation of the axis

*fallback* if true, the global suffix is return when no specific one is set for that value

##### Returns:

the unit suffix

Definition at line 920 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::itemRowLabels(), and KDChart::CartesianAxis::paintCtx().

```

921 {
922     if( !fallback || d->unitSuffixMap[ column ].contains( orientation ) )
923         return d->unitSuffixMap[ column ][ orientation ];
924     return d->unitSuffix[ orientation ];
925 }

```

**9.35.4.114 void AbstractDiagram::update () const** [inherited]

Definition at line 1091 of file KDChartAbstractDiagram.cpp.

References [d](#).

Referenced by [KDChart::AbstractDiagram::doItemsLayout\(\)](#).

```
1092 {  
1093     //qDebug("KDChart::AbstractDiagram::update() called");  
1094     if( d->plane )  
1095         d->plane->update();  
1096 }
```

**9.35.4.115 void KDChart::AbstractDiagram::useDefaultColors ()** [inherited]

Set the palette to be used, for painting datasets to the default palette.

**See also:**

[KDChart::Palette](#). FIXME: fold into one usePalette ([KDChart::Palette&](#)) method

Definition at line 974 of file KDChartAbstractDiagram.cpp.

References [d](#), and [KDChart::AttributesModel::PaletteTypeDefault](#).

```
975 {  
976     d->attributesModel->setPaletteType( AttributesModel::PaletteTypeDefault );  
977 }
```

**9.35.4.116 void KDChart::AbstractDiagram::useRainbowColors ()** [inherited]

Set the palette to be used, for painting datasets to the rainbow palette.

**See also:**

[KDChart::Palette](#).

Definition at line 984 of file KDChartAbstractDiagram.cpp.

References [d](#), and [KDChart::AttributesModel::PaletteTypeRainbow](#).

```
985 {  
986     d->attributesModel->setPaletteType( AttributesModel::PaletteTypeRainbow );  
987 }
```

**9.35.4.117 bool AbstractDiagram::usesExternalAttributesModel () const** [virtual, inherited]

Returns whether the diagram is using its own built-in attributes model or an attributes model that was set via [setAttributesModel](#).

**See also:**

[setAttributesModel](#)

Definition at line 274 of file KDChartAbstractDiagram.cpp.

References [d](#).

```
275 {  
276     return d->usesExternalAttributesModel();  
277 }
```

#### 9.35.4.118 void KDChart::AbstractDiagram::useSubduedColors() [inherited]

Set the palette to be used, for painting datasets to the subdued palette.

**See also:**

[KDChart::Palette](#).

Definition at line 979 of file KDChartAbstractDiagram.cpp.

References [d](#), and [KDChart::AttributesModel::PaletteTypeSubdued](#).

```
980 {  
981     d->attributesModel->setPaletteType( AttributesModel::PaletteTypeSubdued );  
982 }
```

#### 9.35.4.119 double AbstractDiagram::valueForCell(int row, int column) const [protected, inherited]

Helper method, retrieving the data value (DisplayRole) for a given row and column.

**Parameters:**

*row* The row to query.

*column* The column to query.

**Returns:**

The value of the display role at the given row and column as a double.

**Deprecated**

Definition at line 1085 of file KDChartAbstractDiagram.cpp.

References [KDChart::AbstractDiagram::attributesModelRootIndex\(\)](#), and [d](#).

```
1086 {  
1087     return d->attributesModel->data(  
1088         d->attributesModel->index( row, column, attributesModelRootIndex() ) ).toDouble();  
1089 }
```



#### 9.35.4.120 **double** LineDiagram::valueForCellTesting (int row, int column, bool & bOK, bool showHiddenCellsAsInvalid = false) const [protected]

Definition at line 385 of file KDChartLineDiagram.cpp.

References KDChart::AbstractDiagram::attributesModelRootIndex(), d, and KDChart::AbstractDiagram::isHidden().

Referenced by getCellValues().

```

388 {
389     double value;
390     if( showHiddenCellsAsInvalid && isHidden( model()->index( row, column, rootIndex() ) ) )
391         bOK = false;
392     else
393         value = d->attributesModel->data(
394             d->attributesModel->index( row, column, attributesModelRootIndex() )
395             ).toDouble( &bOK );
396     return bOK ? value : 0.0;
397 }
```

#### 9.35.4.121 **ValueTrackerAttributes** LineDiagram::valueTrackerAttributes (const QModelIndex & index) const

Returns the value tracker attributes of the model index *index*.

Definition at line 348 of file KDChartLineDiagram.cpp.

References d, and KDChart::ValueTrackerAttributesRole.

```

350 {
351     return qVariantValue<ValueTrackerAttributes>( d->attributesModel->data(
352         d->attributesModel->mapFromSource( index ),
353         KDChart::ValueTrackerAttributesRole ) );
354 }
```

#### 9.35.4.122 **int** AbstractDiagram::verticalOffset () const [virtual, inherited]

[reimplemented]

Definition at line 953 of file KDChartAbstractDiagram.cpp.

```

954 { return 0; }
```

#### 9.35.4.123 **QRect** AbstractDiagram::visualRect (const QModelIndex & index) const [virtual, inherited]

[reimplemented]

Definition at line 937 of file KDChartAbstractDiagram.cpp.

```

938 {
939     return QRect();
940 }
```

**9.35.4.124 QRegion AbstractDiagram::visualRegionForSelection (const QItemSelection & selection) const** [virtual, inherited]

[reimplemented]

Definition at line 970 of file KDChartAbstractDiagram.cpp.

```
971 { return QRegion(); }
```

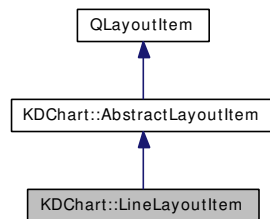
The documentation for this class was generated from the following files:

- [KDChartLineDiagram.h](#)
- [KDChartLineDiagram.cpp](#)

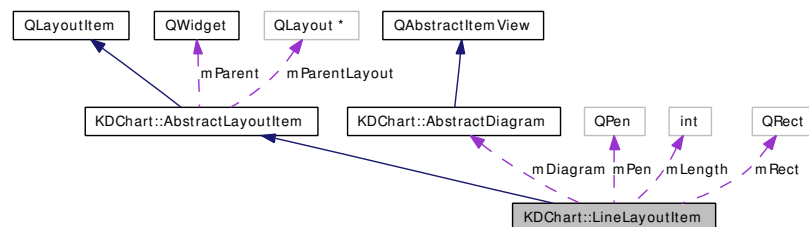
## 9.36 KDChart::LineLayoutItem Class Reference

```
#include <KDChartLayoutItems.h>
```

Inheritance diagram for KDChart::LineLayoutItem:



Collaboration diagram for KDChart::LineLayoutItem:



### 9.36.1 Detailed Description

Layout item showing a coloured line.

Definition at line 199 of file KDChartLayoutItems.h.

#### Public Member Functions

- virtual Qt::Orientations [expandingDirections](#) () const
- virtual QRect [geometry](#) () const
- virtual bool [isEmpty](#) () const
- [LineLayoutItem](#) ([AbstractDiagram](#) \**diagram*, int *length*, const QPen &*pen*, Qt::Alignment *alignment*=0)
- virtual QSize [maximumSize](#) () const
- virtual QSize [minimumSize](#) () const
- virtual void [paint](#) (QPainter \*)
- virtual void [paintAll](#) (QPainter &*painter*)  
*Default impl: just call paint.*
- virtual void [paintCtx](#) ([PaintContext](#) \**context*)  
*Default impl: Paint the complete item using its layouted position and size.*
- QLayout \* [parentLayout](#) ()
- void [removeFromParentLayout](#) ()
- virtual void [setGeometry](#) (const QRect &*r*)

- void [setParentLayout](#) (QLayout \*lay)
- virtual void [setParentWidget](#) (QWidget \*widget)

*Inform the item about its widget: This enables the item, to trigger that widget's update, whenever the size of the item's contents has changed.*

- virtual QSize [sizeHint](#) () const
- virtual void [sizeHintChanged](#) () const

*Report changed size hint: ask the parent widget to recalculate the layout.*

## Static Public Member Functions

- static void [paintIntoRect](#) (QPainter \*painter, const QRect &rect, const QPen &pen)

## Protected Attributes

- QWidget \* [mParent](#)
- QLayout \* [mParentLayout](#)

## 9.36.2 Constructor & Destructor Documentation

### 9.36.2.1 KDChart::LineLayoutItem::LineLayoutItem ([AbstractDiagram](#) \* *diagram*, int *length*, const QPen & *pen*, Qt::Alignment *alignment* = 0)

Definition at line 616 of file KDChartLayoutItems.cpp.

```

620     : AbstractLayoutItem( alignment )
621     , mDiagram( diagram )
622     , mLength( length )
623     , mPen( pen )
624 {
625     //have a mini pen width
626     if ( pen.width() < 2 )
627         mPen.setWidth( 2 );
628 }
```

## 9.36.3 Member Function Documentation

### 9.36.3.1 Qt::Orientations KDChart::LineLayoutItem::expandingDirections () const [virtual]

Definition at line 630 of file KDChartLayoutItems.cpp.

```

631 {
632     return 0; // Grow neither vertically nor horizontally
633 }
```

### 9.36.3.2 QRect KDChart::LineLayoutItem::geometry () const [virtual]

Definition at line 635 of file KDChartLayoutItems.cpp.

```
636 {  
637     return mRect;  
638 }
```

### 9.36.3.3 bool KDChart::LineLayoutItem::isEmpty () const [virtual]

Definition at line 640 of file KDChartLayoutItems.cpp.

```
641 {  
642     return false; // never empty, otherwise the layout item would not exist  
643 }
```

### 9.36.3.4 QSize KDChart::LineLayoutItem::maximumSize () const [virtual]

Definition at line 645 of file KDChartLayoutItems.cpp.

References [sizeHint\(\)](#).

```
646 {  
647     return sizeHint(); // PENDING(kalle) Review, quite inflexible  
648 }
```

### 9.36.3.5 QSize KDChart::LineLayoutItem::minimumSize () const [virtual]

Definition at line 650 of file KDChartLayoutItems.cpp.

References [sizeHint\(\)](#).

```
651 {  
652     return sizeHint(); // PENDING(kalle) Review, quite inflexible  
653 }
```

### 9.36.3.6 void KDChart::LineLayoutItem::paint (QPainter \*) [virtual]

Implements [KDChart::AbstractLayoutItem](#).

Definition at line 665 of file KDChartLayoutItems.cpp.

References [paintIntoRect\(\)](#).

```
666 {  
667     paintIntoRect( painter, mRect, mPen );  
668 }
```

### 9.36.3.7 void KDChart::AbstractLayoutItem::paintAll (QPainter & *painter*) [virtual, inherited]

Default impl: just call [paint](#).

Derived classes like [KDChart::AbstractArea](#) are providing additional action here.

Reimplemented in [KDChart::AbstractArea](#), [KDChart::TextArea](#), and [KDChart::TernaryAxis](#).

Definition at line 69 of file `KDChartLayoutItems.cpp`.

References `KDChart::AbstractLayoutItem::paint()`.

```
70 {
71     paint( &painter );
72 }
```

### 9.36.3.8 void KDChart::AbstractLayoutItem::paintCtx ([PaintContext](#) \* context) [virtual, inherited]

Default impl: Paint the complete item using its layouted position and size.

Reimplemented in [KDChart::CartesianAxis](#), and [KDChart::TernaryAxis](#).

Definition at line 77 of file `KDChartLayoutItems.cpp`.

References `KDChart::AbstractLayoutItem::paint()`, and `KDChart::PaintContext::painter()`.

```
78 {
79     if( context )
80         paint( context->painter() );
81 }
```

### 9.36.3.9 void KDChart::LineLayoutItem::paintIntoRect (QPainter \* painter, const QRect & rect, const QPen & pen) [static]

Definition at line 670 of file `KDChartLayoutItems.cpp`.

Referenced by `KDChart::LineWithMarkerLayoutItem::paint()`, and `paint()`.

```
674 {
675     if( ! rect.isValid() )
676         return;
677
678     const QPen oldPen = painter->pen();
679     painter->setPen( pen );
680     const qreal y = rect.center().y();
681     painter->drawLine( QPointF( rect.left(), y ),
682                     QPointF( rect.right(), y ) );
683     painter->setPen( oldPen );
684 }
```

### 9.36.3.10 QLayout\* KDChart::AbstractLayoutItem::parentLayout () [inherited]

Definition at line 76 of file `KDChartLayoutItems.h`.

```
77     {
78         return mParentLayout;
79     }
```

**9.36.3.11 void KDChart::AbstractLayoutItem::removeFromParentLayout ()** [inherited]

Definition at line 80 of file KDChartLayoutItems.h.

Referenced by KDChart::Chart::takeCoordinatePlane().

```
81         {
82             if( mParentLayout ){
83                 if( widget() )
84                     mParentLayout->removeWidget( widget() );
85             } else
86                 mParentLayout->removeItem( this );
87         }
88     }
```

**9.36.3.12 void KDChart::LineLayoutItem::setGeometry (const QRect &r)** [virtual]

Definition at line 655 of file KDChartLayoutItems.cpp.

```
656 {
657     mRect = r;
658 }
```

**9.36.3.13 void KDChart::AbstractLayoutItem::setParentLayout (QLayout \*lay)** [inherited]

Definition at line 72 of file KDChartLayoutItems.h.

```
73     {
74         mParentLayout = lay;
75     }
```

**9.36.3.14 void KDChart::AbstractLayoutItem::setParentWidget (QWidget \*widget)**  
[virtual, inherited]

Inform the item about its widget: This enables the item, to trigger that widget's update, whenever the size of the item's contents has changed.

Thus, you need to call setParentWidget on every item, that has a non-fixed size.

Definition at line 64 of file KDChartLayoutItems.cpp.

References KDChart::AbstractLayoutItem::mParent.

Referenced by KDChart::HeaderFooter::setParent(), and KDChart::AbstractCartesianDiagram::takeAxis().

```
65 {
66     mParent = widget;
67 }
```

### 9.36.3.15 QSize KDChart::LineLayoutItem::sizeHint () const [virtual]

Definition at line 660 of file KDChartLayoutItems.cpp.

Referenced by maxSize(), and minSize().

```
661 {  
662     return QSize( mLength, mPen.width()+2 );  
663 }
```

### 9.36.3.16 void KDChart::AbstractLayoutItem::sizeHintChanged () const [virtual, inherited]

Report changed size hint: ask the parent widget to recalculate the layout.

Definition at line 86 of file KDChartLayoutItems.cpp.

References KDChart::AbstractLayoutItem::mParent.

Referenced by KDChart::TextLayoutItem::sizeHint().

```
87 {  
88     // This is exactly like what QWidget::updateGeometry does.  
89     qDebug( "KDChart::AbstractLayoutItem::sizeHintChanged() called" );  
90     if( mParent ) {  
91         if ( mParent->layout() )  
92             mParent->layout()->invalidate();  
93         else  
94             QApplication::postEvent( mParent, new QEvent( QEvent::LayoutRequest ) );  
95     }  
96 }
```

## 9.36.4 Member Data Documentation

### 9.36.4.1 QWidget\* KDChart::AbstractLayoutItem::mParent [protected, inherited]

Definition at line 90 of file KDChartLayoutItems.h.

Referenced by KDChart::AbstractLayoutItem::setParentWidget(), KDChart::TextLayoutItem::setText(), KDChart::TextLayoutItem::setTextAttributes(), and KDChart::AbstractLayoutItem::sizeHintChanged().

### 9.36.4.2 QLayout\* KDChart::AbstractLayoutItem::mParentLayout [protected, inherited]

Definition at line 91 of file KDChartLayoutItems.h.

Referenced by KDChart::AutoSpacerLayoutItem::paint().

The documentation for this class was generated from the following files:

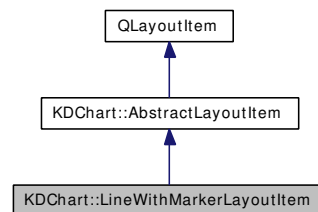
- [KDChartLayoutItems.h](#)
- [KDChartLayoutItems.cpp](#)



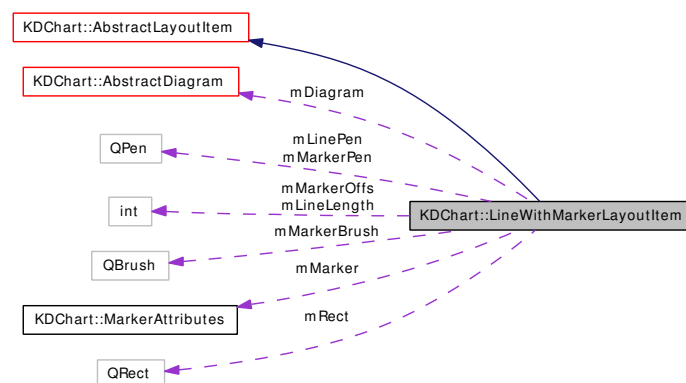
## 9.37 KDCart::LineWithMarkerLayoutItem Class Reference

```
#include <KDCartLayoutItems.h>
```

Inheritance diagram for KDCart::LineWithMarkerLayoutItem:



Collaboration diagram for KDCart::LineWithMarkerLayoutItem:



### 9.37.1 Detailed Description

Layout item showing a coloured line and a data point marker.

Definition at line 233 of file KDCartLayoutItems.h.

#### Public Member Functions

- virtual Qt::Orientations [expandingDirections](#) () const
- virtual QRect [geometry](#) () const
- virtual bool [isEmpty](#) () const
- [LineWithMarkerLayoutItem](#) ([AbstractDiagram](#) \**diagram*, int *lineLength*, const QPen &*linePen*, int *markerOffs*, const [MarkerAttributes](#) &*marker*, const QBrush &*markerBrush*, const QPen &*markerPen*, Qt::Alignment *alignment*=0)
- virtual QSize [maximumSize](#) () const
- virtual QSize [minimumSize](#) () const
- virtual void [paint](#) (QPainter \*)
- virtual void [paintAll](#) (QPainter &*painter*)

*Default impl: just call paint.*

- virtual void `paintCtx` (`PaintContext` \*context)

*Default impl: Paint the complete item using its layouted position and size.*

- `QLayout` \* `parentLayout` ()
- void `removeFromParentLayout` ()
- virtual void `setGeometry` (const `QRect` &r)
- void `setParentLayout` (`QLayout` \*lay)
- virtual void `setParentWidget` (`QWidget` \*widget)

*Inform the item about its widget: This enables the item, to trigger that widget's update, whenever the size of the item's contents has changed.*

- virtual `QSize` `sizeHint` () const
- virtual void `sizeHintChanged` () const

*Report changed size hint: ask the parent widget to recalculate the layout.*

## Protected Attributes

- `QWidget` \* `mParent`
- `QLayout` \* `mParentLayout`

## 9.37.2 Constructor & Destructor Documentation

### 9.37.2.1 `KDChart::LineWithMarkerLayoutItem::LineWithMarkerLayoutItem` (`AbstractDiagram` \* *diagram*, int *lineLength*, const `QPen` & *linePen*, int *markerOffs*, const `MarkerAttributes` & *marker*, const `QBrush` & *markerBrush*, const `QPen` & *markerPen*, `Qt::Alignment` *alignment* = 0)

Definition at line 687 of file `KDChartLayoutItems.cpp`.

```

696     : AbstractLayoutItem( alignment )
697     , mDiagram(      diagram )
698     , mLineLength(   lineLength )
699     , mLinePen(      linePen )
700     , mMarkerOffs(   markerOffs )
701     , mMarker(       marker )
702     , mMarkerBrush(  markerBrush )
703     , mMarkerPen(    markerPen )
704 {
705 }
```

## 9.37.3 Member Function Documentation

### 9.37.3.1 `Qt::Orientations KDChart::LineWithMarkerLayoutItem::expandingDirections` () const [virtual]

Definition at line 707 of file `KDChartLayoutItems.cpp`.

```

708 {
709     return 0; // Grow neither vertically nor horizontally
710 }
```

**9.37.3.2 QRect KDChart::LineWithMarkerLayoutItem::geometry () const** [virtual]

Definition at line 712 of file KDChartLayoutItems.cpp.

```
713 {  
714     return mRect;  
715 }
```

**9.37.3.3 bool KDChart::LineWithMarkerLayoutItem::isEmpty () const** [virtual]

Definition at line 717 of file KDChartLayoutItems.cpp.

```
718 {  
719     return false; // never empty, otherwise the layout item would not exist  
720 }
```

**9.37.3.4 QSize KDChart::LineWithMarkerLayoutItem::maximumSize () const** [virtual]

Definition at line 722 of file KDChartLayoutItems.cpp.

References [sizeHint\(\)](#).

```
723 {  
724     return sizeHint(); // PENDING(kalle) Review, quite inflexible  
725 }
```

**9.37.3.5 QSize KDChart::LineWithMarkerLayoutItem::minimumSize () const** [virtual]

Definition at line 727 of file KDChartLayoutItems.cpp.

References [sizeHint\(\)](#).

```
728 {  
729     return sizeHint(); // PENDING(kalle) Review, quite inflexible  
730 }
```

**9.37.3.6 void KDChart::LineWithMarkerLayoutItem::paint (QPainter \*)** [virtual]

Implements [KDChart::AbstractLayoutItem](#).

Definition at line 745 of file KDChartLayoutItems.cpp.

References [KDChart::MarkerAttributes::markerSize\(\)](#), [KDChart::MarkerLayoutItem::paintIntoRect\(\)](#), and [KDChart::LineLayoutItem::paintIntoRect\(\)](#).

```
746 {  
747     // paint the line over the full width, into the vertical middle of the rect  
748     LineLayoutItem::paintIntoRect( painter, mRect, mLinePen );  
749  
750     // paint the marker with the given offset from the left side of the line  
751     const QRect r(  
752         QPoint( mRect.x()+mMarkerOffs, mRect.y() ),  
753         QSize( mMarker.markerSize().toSize().width(), mRect.height() ) );  
754     MarkerLayoutItem::paintIntoRect(  
755         painter, r, mDiagram, mMarker, mMarkerBrush, mMarkerPen );  
756 }
```

### 9.37.3.7 void KDChart::AbstractLayoutItem::paintAll (QPainter & *painter*) [virtual, inherited]

Default impl: just call paint.

Derived classes like [KDChart::AbstractArea](#) are providing additional action here.

Reimplemented in [KDChart::AbstractArea](#), [KDChart::TextArea](#), and [KDChart::TernaryAxis](#).

Definition at line 69 of file KDChartLayoutItems.cpp.

References KDChart::AbstractLayoutItem::paint().

```
70 {
71     paint( &painter );
72 }
```

### 9.37.3.8 void KDChart::AbstractLayoutItem::paintCtx ([PaintContext](#) \* *context*) [virtual, inherited]

Default impl: Paint the complete item using its layouted position and size.

Reimplemented in [KDChart::CartesianAxis](#), and [KDChart::TernaryAxis](#).

Definition at line 77 of file KDChartLayoutItems.cpp.

References KDChart::AbstractLayoutItem::paint(), and KDChart::PaintContext::painter().

```
78 {
79     if( context )
80         paint( context->painter() );
81 }
```

### 9.37.3.9 QLayout\* KDChart::AbstractLayoutItem::parentLayout () [inherited]

Definition at line 76 of file KDChartLayoutItems.h.

```
77     {
78         return mParentLayout;
79     }
```

### 9.37.3.10 void KDChart::AbstractLayoutItem::removeFromParentLayout () [inherited]

Definition at line 80 of file KDChartLayoutItems.h.

Referenced by KDChart::Chart::takeCoordinatePlane().

```
81     {
82         if( mParentLayout ){
83             if( widget() )
84                 mParentLayout->removeWidget( widget() );
85             else
86                 mParentLayout->removeItem( this );
87         }
88     }
```

**9.37.3.11 void KDChart::LineWithMarkerLayoutItem::setGeometry (const QRect & r)**  
[virtual]

Definition at line 732 of file KDChartLayoutItems.cpp.

```
733 {  
734     mRect = r;  
735 }
```

**9.37.3.12 void KDChart::AbstractLayoutItem::setParentLayout (QLayout \* lay)** [inherited]

Definition at line 72 of file KDChartLayoutItems.h.

```
73     {  
74         mParentLayout = lay;  
75     }
```

**9.37.3.13 void KDChart::AbstractLayoutItem::setParentWidget (QWidget \* widget)**  
[virtual, inherited]

Inform the item about its widget: This enables the item, to trigger that widget's update, whenever the size of the item's contents has changed.

Thus, you need to call setParentWidget on every item, that has a non-fixed size.

Definition at line 64 of file KDChartLayoutItems.cpp.

References KDChart::AbstractLayoutItem::mParent.

Referenced by KDChart::HeaderFooter::setParent(), and KDChart::AbstractCartesianDiagram::takeAxis().

```
65 {  
66     mParent = widget;  
67 }
```

**9.37.3.14 QSize KDChart::LineWithMarkerLayoutItem::sizeHint () const** [virtual]

Definition at line 737 of file KDChartLayoutItems.cpp.

References KDChart::MarkerAttributes::markerSize().

Referenced by maxSize(), and minSize().

```
738 {  
739     const QSize sizeM = mMarker.markerSize().toSize();  
740     const QSize sizeL = QSize( mLineLength, mLinePen.width()+2 );  
741     return QSize( qMax(sizeM.width(), sizeL.width()),  
742                 qMax(sizeM.height(), sizeL.height()) );  
743 }
```

### 9.37.3.15 void KDChart::AbstractLayoutItem::sizeHintChanged () const [virtual, inherited]

Report changed size hint: ask the parent widget to recalculate the layout.

Definition at line 86 of file KDChartLayoutItems.cpp.

References KDChart::AbstractLayoutItem::mParent.

Referenced by KDChart::TextLayoutItem::sizeHint().

```
87 {
88     // This is exactly like what QWidget::updateGeometry does.
89     // qDebug("KDChart::AbstractLayoutItem::sizeHintChanged() called");
90     if( mParent ) {
91         if ( mParent->layout() )
92             mParent->layout()->invalidate();
93         else
94             QApplication::postEvent( mParent, new QEvent( QEvent::LayoutRequest ) );
95     }
96 }
```

## 9.37.4 Member Data Documentation

### 9.37.4.1 QWidget\* KDChart::AbstractLayoutItem::mParent [protected, inherited]

Definition at line 90 of file KDChartLayoutItems.h.

Referenced by KDChart::AbstractLayoutItem::setParentWidget(), KDChart::TextLayoutItem::setText(), KDChart::TextLayoutItem::setTextAttributes(), and KDChart::AbstractLayoutItem::sizeHintChanged().

### 9.37.4.2 QLayout\* KDChart::AbstractLayoutItem::mParentLayout [protected, inherited]

Definition at line 91 of file KDChartLayoutItems.h.

Referenced by KDChart::AutoSpacerLayoutItem::paint().

The documentation for this class was generated from the following files:

- [KDChartLayoutItems.h](#)
- [KDChartLayoutItems.cpp](#)

## 9.38 KDChart::MarkerAttributes Class Reference

```
#include <KDChartMarkerAttributes.h>
```

### 9.38.1 Detailed Description

A set of attributes controlling the appearance of data set markers.

Definition at line 47 of file KDChartMarkerAttributes.h.

#### Public Types

- enum [MarkerStyle](#) {  
    [MarkerCircle](#) = 0,  
    [MarkerSquare](#) = 1,  
    [MarkerDiamond](#) = 2,  
    [Marker1Pixel](#) = 3,  
    [Marker4Pixels](#) = 4,  
    [MarkerRing](#) = 5,  
    [MarkerCross](#) = 6,  
    [MarkerFastCross](#) = 7 }  
• typedef QMap< uint, [MarkerStyle](#) > [MarkerStylesMap](#)

#### Public Member Functions

- bool [isVisible](#) () const
- [MarkerAttributes](#) (const [MarkerAttributes](#) &)
- [MarkerAttributes](#) ()
- QColor [markerColor](#) () const
- QSizeF [markerSize](#) () const
- [MarkerStyle](#) [markerStyle](#) () const
- [MarkerStylesMap](#) [markerStylesMap](#) () const
- bool [operator!=](#) (const [MarkerAttributes](#) &) const
- [MarkerAttributes](#) & [operator=](#) (const [MarkerAttributes](#) &)
- bool [operator==](#) (const [MarkerAttributes](#) &) const
- QPen [pen](#) () const
- void [setMarkerColor](#) (const QColor &color)
- void [setMarkerSize](#) (const QSizeF &size)

*Normally you need to specify a valid QSizeF here, but for Legends you can use the invalid size QSizeF(), to enable automatic marker size calculation:.*

- void [setMarkerStyle](#) ([MarkerStyle](#) style)
- void [setMarkerStylesMap](#) (const [MarkerStylesMap](#) &map)
- void [setPen](#) (const QPen &pen)
- void [setVisible](#) (bool visible)
- ~[MarkerAttributes](#) ()

## 9.38.2 Member Typedef Documentation

### 9.38.2.1 typedef QMap<uint, [MarkerStyle](#)> [KDChart::MarkerAttributes::MarkerStylesMap](#)

Definition at line 68 of file KDChartMarkerAttributes.h.

## 9.38.3 Member Enumeration Documentation

### 9.38.3.1 enum [KDChart::MarkerAttributes::MarkerStyle](#)

Enumerator:

*MarkerCircle*  
*MarkerSquare*  
*MarkerDiamond*  
*Marker1Pixel*  
*Marker4Pixels*  
*MarkerRing*  
*MarkerCross*  
*MarkerFastCross*

Definition at line 56 of file KDChartMarkerAttributes.h.

```

56             { MarkerCircle  = 0,
57               MarkerSquare  = 1,
58               MarkerDiamond = 2,
59               Marker1Pixel  = 3,
60               Marker4Pixels = 4,
61               MarkerRing    = 5,
62               MarkerCross   = 6,
63               MarkerFastCross = 7 };

```

## 9.38.4 Constructor & Destructor Documentation

### 9.38.4.1 [MarkerAttributes::MarkerAttributes\(\)](#)

Definition at line 61 of file KDChartMarkerAttributes.cpp.

```

62     : _d( new Private )
63 {
64
65 }

```

### 9.38.4.2 [MarkerAttributes::MarkerAttributes\(const \[MarkerAttributes\]\(#\) &\)](#)

Definition at line 67 of file KDChartMarkerAttributes.cpp.

```

68     : _d( new Private( *r._d ) )
69 {
70
71 }

```



### 9.38.4.3 MarkerAttributes::~~MarkerAttributes ()

Definition at line 80 of file KDCartMarkerAttributes.cpp.

```
81 {  
82     delete _d; _d = 0;  
83 }
```

## 9.38.5 Member Function Documentation

### 9.38.5.1 bool MarkerAttributes::isVisible () const

Definition at line 114 of file KDCartMarkerAttributes.cpp.

References [d](#).

Referenced by [operator<<\(\)](#), [operator==\(\)](#), and [KDCart::AbstractDiagram::paintMarker\(\)](#).

```
115 {  
116     return d->visible;  
117 }
```

### 9.38.5.2 QColor MarkerAttributes::markerColor () const

Definition at line 154 of file KDCartMarkerAttributes.cpp.

References [d](#).

Referenced by [operator<<\(\)](#), [operator==\(\)](#), and [KDCart::AbstractDiagram::paintMarker\(\)](#).

```
155 {  
156     return d->markerColor;  
157 }
```

### 9.38.5.3 QSizeF MarkerAttributes::markerSize () const

Definition at line 144 of file KDCartMarkerAttributes.cpp.

References [d](#).

Referenced by [operator==\(\)](#), [KDCart::LineWithMarkerLayoutItem::paint\(\)](#), [KDCart::MarkerLayoutItem::paintIntoRect\(\)](#), [KDCart::AbstractDiagram::paintMarker\(\)](#), [KDCart::LineWithMarkerLayoutItem::sizeHint\(\)](#), and [KDCart::MarkerLayoutItem::sizeHint\(\)](#).

```
145 {  
146     return d->markerSize;  
147 }
```

### 9.38.5.4 [MarkerAttributes::MarkerStyle](#) MarkerAttributes::markerStyle () const

Definition at line 134 of file KDCartMarkerAttributes.cpp.

References [d](#).

Referenced by [operator<<\(\)](#), [operator==\(\)](#), and [KDCart::AbstractDiagram::paintMarker\(\)](#).

```

135 {
136     return d->markerStyle;
137 }

```

#### 9.38.5.5 [MarkerAttributes::MarkerStylesMap](#) MarkerAttributes::markerStylesMap () const

Definition at line 124 of file KDChartMarkerAttributes.cpp.

References [d](#).

Referenced by [operator<<\(\)](#), and [operator==\(\)](#).

```

125 {
126     return d->markerStylesMap;
127 }

```

#### 9.38.5.6 bool KDChart::MarkerAttributes::operator!= (const [MarkerAttributes](#) &) const

Definition at line 98 of file KDChartMarkerAttributes.h.

References [operator==\(\)](#).

```

98 { return !operator==( other ); }

```

#### 9.38.5.7 [MarkerAttributes](#) & MarkerAttributes::operator= (const [MarkerAttributes](#) &)

Definition at line 73 of file KDChartMarkerAttributes.cpp.

```

74 {
75     MarkerAttributes copy( r );
76     copy.swap( *this );
77     return *this;
78 }

```

#### 9.38.5.8 bool MarkerAttributes::operator== (const [MarkerAttributes](#) &) const

Definition at line 87 of file KDChartMarkerAttributes.cpp.

References [isVisible\(\)](#), [markerColor\(\)](#), [markerSize\(\)](#), [markerStyle\(\)](#), [markerStylesMap\(\)](#), and [pen\(\)](#).

Referenced by [operator!=\(\)](#).

```

88 {
89     /*
90     qDebug() << "MarkerAttributes::operator== finds"
91         << "b" << (isVisible() == r.isVisible())
92         << "c" << (markerStylesMap() == r.markerStylesMap())
93         << "d" << (markerStyle() == r.markerStyle()) << markerStyle() << r.markerStyle()
94         << "e" << (markerSize() == r.markerSize())
95         << "f" << (markerColor() == r.markerColor())
96         << "g" << (pen() == r.pen())
97         << "h" << (markerColor() == r.markerColor()) << markerColor() << r.markerColor();
98     */
99     return ( isVisible() == r.isVisible() &&

```

```
100         markerStylesMap() == r.markerStylesMap() &&
101         markerStyle() == r.markerStyle() &&
102         markerSize() == r.markerSize() &&
103         markerColor() == r.markerColor() &&
104         pen() == r.pen() );
105 }
```

#### 9.38.5.9 QPen MarkerAttributes::pen () const

Definition at line 164 of file KDChartMarkerAttributes.cpp.

References [d](#).

Referenced by [operator<<\(\)](#), [operator==\(\)](#), and [KDChart::AbstractDiagram::paintMarker\(\)](#).

```
165 {
166     return d->markerPen;
167 }
```

#### 9.38.5.10 void MarkerAttributes::setMarkerColor (const QColor & color)

Definition at line 149 of file KDChartMarkerAttributes.cpp.

References [d](#).

```
150 {
151     d->markerColor = color;
152 }
```

#### 9.38.5.11 void MarkerAttributes::setMarkerSize (const QSizeF & size)

Normally you need to specify a valid QSizeF here, but for Legends you can use the invalid size QSizeF(), to enable automatic marker size calculation:

For Markers shown in a [Legend](#) this means the marker size will be equal to the font height used for the labels that are shown next to the markers.

Definition at line 139 of file KDChartMarkerAttributes.cpp.

References [d](#).

```
140 {
141     d->markerSize = size;
142 }
```

#### 9.38.5.12 void MarkerAttributes::setMarkerStyle ([MarkerStyle](#) style)

Definition at line 129 of file KDChartMarkerAttributes.cpp.

References [d](#).

Referenced by [KDChart::TernaryLineDiagram::TernaryLineDiagram\(\)](#).

```
130 {
131     d->markerStyle = style;
132 }
```

**9.38.5.13 void MarkerAttributes::setMarkerStylesMap (const [MarkerStylesMap](#) & *map*)**

Definition at line 119 of file KDChartMarkerAttributes.cpp.

References [d](#).

```
120 {  
121     d->markerStylesMap = map;  
122 }
```

**9.38.5.14 void MarkerAttributes::setPen (const QPen & *pen*)**

Definition at line 159 of file KDChartMarkerAttributes.cpp.

References [d](#).

```
160 {  
161     d->markerPen = pen;  
162 }
```

**9.38.5.15 void MarkerAttributes::setVisible (bool *visible*)**

Definition at line 109 of file KDChartMarkerAttributes.cpp.

References [d](#).

Referenced by [KDChart::TernaryLineDiagram::TernaryLineDiagram\(\)](#).

```
110 {  
111     d->visible = visible;  
112 }
```

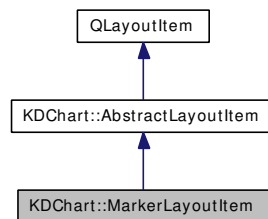
The documentation for this class was generated from the following files:

- [KDChartMarkerAttributes.h](#)
- [KDChartMarkerAttributes.cpp](#)

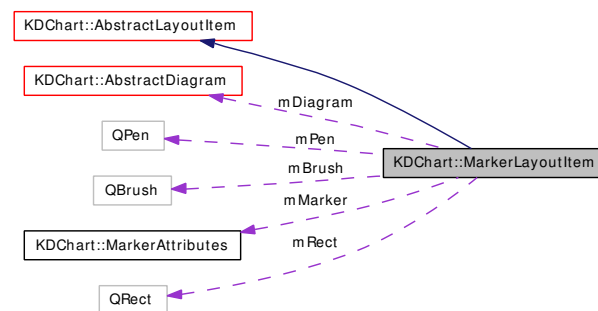
## 9.39 KDChart::MarkerLayoutItem Class Reference

```
#include <KDChartLayoutItems.h>
```

Inheritance diagram for KDChart::MarkerLayoutItem:



Collaboration diagram for KDChart::MarkerLayoutItem:



### 9.39.1 Detailed Description

Layout item showing a data point marker.

Definition at line 160 of file KDChartLayoutItems.h.

#### Public Member Functions

- virtual Qt::Orientations [expandingDirections](#) () const
- virtual QRect [geometry](#) () const
- virtual bool [isEmpty](#) () const
- [MarkerLayoutItem](#) ([AbstractDiagram](#) \**diagram*, const [MarkerAttributes](#) &*marker*, const QBrush &*brush*, const QPen &*pen*, Qt::Alignment *alignment*=0)
- virtual QSize [maximumSize](#) () const
- virtual QSize [minimumSize](#) () const
- virtual void [paint](#) (QPainter \*)
- virtual void [paintAll](#) (QPainter &*painter*)

*Default impl: just call paint.*

- virtual void [paintCtx](#) ([PaintContext](#) \**context*)

*Default impl: Paint the complete item using its layouted position and size.*

- `QLayout * parentLayout ()`
- `void removeFromParentLayout ()`
- `virtual void setGeometry (const QRect &r)`
- `void setParentLayout (QLayout *lay)`
- `virtual void setParentWidget (QWidget *widget)`

*Inform the item about its widget: This enables the item, to trigger that widget's update, whenever the size of the item's contents has changed.*

- `virtual QSize sizeHint () const`
- `virtual void sizeHintChanged () const`

*Report changed size hint: ask the parent widget to recalculate the layout.*

## Static Public Member Functions

- `static void paintIntoRect (QPainter *painter, const QRect &rect, AbstractDiagram *diagram, const MarkerAttributes &marker, const QBrush &brush, const QPen &pen)`

## Protected Attributes

- `QWidget * mParent`
- `QLayout * mParentLayout`

## 9.39.2 Constructor & Destructor Documentation

### 9.39.2.1 KDChart::MarkerLayoutItem::MarkerLayoutItem (AbstractDiagram \* diagram, const MarkerAttributes & marker, const QBrush & brush, const QPen & pen, Qt::Alignment alignment = 0)

Definition at line 525 of file KDChartLayoutItems.cpp.

```

529     : AbstractLayoutItem( alignment )
530     , mDiagram( diagram )
531     , mMarker( marker )
532     , mBrush( brush )
533     , mPen( pen )
534 {
535 }
```

## 9.39.3 Member Function Documentation

### 9.39.3.1 Qt::Orientations KDChart::MarkerLayoutItem::expandingDirections () const [virtual]

Definition at line 537 of file KDChartLayoutItems.cpp.

```

538 {
539     return 0; // Grow neither vertically nor horizontally
540 }
```

### 9.39.3.2 QRect KDChart::MarkerLayoutItem::geometry () const [virtual]

Definition at line 542 of file KDChartLayoutItems.cpp.

```
543 {  
544     return mRect;  
545 }
```

### 9.39.3.3 bool KDChart::MarkerLayoutItem::isEmpty () const [virtual]

Definition at line 547 of file KDChartLayoutItems.cpp.

```
548 {  
549     return false; // never empty, otherwise the layout item would not exist  
550 }
```

### 9.39.3.4 QSize KDChart::MarkerLayoutItem::maximumSize () const [virtual]

Definition at line 552 of file KDChartLayoutItems.cpp.

References [sizeHint\(\)](#).

```
553 {  
554     return sizeHint(); // PENDING(kalle) Review, quite inflexible  
555 }
```

### 9.39.3.5 QSize KDChart::MarkerLayoutItem::minimumSize () const [virtual]

Definition at line 557 of file KDChartLayoutItems.cpp.

References [sizeHint\(\)](#).

```
558 {  
559     return sizeHint(); // PENDING(kalle) Review, quite inflexible  
560 }
```

### 9.39.3.6 void KDChart::MarkerLayoutItem::paint (QPainter \*) [virtual]

Implements [KDChart::AbstractLayoutItem](#).

Definition at line 573 of file KDChartLayoutItems.cpp.

References [paintIntoRect\(\)](#).

```
574 {  
575     paintIntoRect( painter, mRect, mDiagram, mMarker, mBrush, mPen );  
576 }
```

### 9.39.3.7 void KDChart::AbstractLayoutItem::paintAll (QPainter & *painter*) [virtual, inherited]

Default impl: just call paint.

Derived classes like [KDChart::AbstractArea](#) are providing additional action here.

Reimplemented in [KDChart::AbstractArea](#), [KDChart::TextArea](#), and [KDChart::TernaryAxis](#).

Definition at line 69 of file KDChartLayoutItems.cpp.

References KDChart::AbstractLayoutItem::paint().

```
70 {
71     paint( &painter );
72 }
```

### 9.39.3.8 void KDChart::AbstractLayoutItem::paintCtx ([PaintContext](#) \* *context*) [virtual, inherited]

Default impl: Paint the complete item using its layouted position and size.

Reimplemented in [KDChart::CartesianAxis](#), and [KDChart::TernaryAxis](#).

Definition at line 77 of file KDChartLayoutItems.cpp.

References KDChart::AbstractLayoutItem::paint(), and KDChart::PaintContext::painter().

```
78 {
79     if( context )
80         paint( context->painter() );
81 }
```

### 9.39.3.9 void KDChart::MarkerLayoutItem::paintIntoRect (QPainter \* *painter*, const QRect & *rect*, [AbstractDiagram](#) \* *diagram*, const [MarkerAttributes](#) & *marker*, const QBrush & *brush*, const QPen & *pen*) [static]

Definition at line 578 of file KDChartLayoutItems.cpp.

References KDChart::MarkerAttributes::markerSize(), and KDChart::AbstractDiagram::paintMarker().

Referenced by KDChart::LineWithMarkerLayoutItem::paint(), and paint().

```
585 {
586     if( ! rect.isValid() )
587         return;
588
589     // The layout management may assign a larger rect than what we
590     // wanted. We need to adjust the position.
591     const QSize siz = marker.markerSize().toSize();
592     QPointF pos = rect.topLeft();
593     pos += QPointF( static_cast<qreal>(( rect.width() - siz.width() ) / 2.0 ),
594                   static_cast<qreal>(( rect.height() - siz.height() ) / 2.0 ) );
595
596     #ifdef DEBUG_ITEMS_PAINT
597         QPointF oldPos = pos;
598     #endif
599
600     // And finally, drawMarker() assumes the position to be the center
601     // of the marker, adjust again.
```



```

602     pos += QPointF( static_cast<qreal>( siz.width() ) / 2.0,
603                    static_cast<qreal>( siz.height() ) / 2.0 );
604
605     diagram->paintMarker( painter, marker, brush, pen, pos.toPoint(), siz );
606
607 #ifdef DEBUG_ITEMS_PAINT
608     const QPen oldPen( painter->pen() );
609     painter->setPen( Qt::red );
610     painter->drawRect( QRect( oldPos.toPoint(), siz ) );
611     painter->setPen( oldPen );
612 #endif
613 }

```

### 9.39.3.10 QLayout\* KDChart::AbstractLayoutItem::parentLayout () [inherited]

Definition at line 76 of file KDChartLayoutItems.h.

```

77     {
78         return mParentLayout;
79     }

```

### 9.39.3.11 void KDChart::AbstractLayoutItem::removeFromParentLayout () [inherited]

Definition at line 80 of file KDChartLayoutItems.h.

Referenced by KDChart::Chart::takeCoordinatePlane().

```

81     {
82         if( mParentLayout ){
83             if( widget() )
84                 mParentLayout->removeWidget( widget() );
85             else
86                 mParentLayout->removeItem( this );
87         }
88     }

```

### 9.39.3.12 void KDChart::MarkerLayoutItem::setGeometry (const QRect & r) [virtual]

Definition at line 562 of file KDChartLayoutItems.cpp.

```

563 {
564     mRect = r;
565 }

```

### 9.39.3.13 void KDChart::AbstractLayoutItem::setParentLayout (QLayout \* lay) [inherited]

Definition at line 72 of file KDChartLayoutItems.h.

```

73     {
74         mParentLayout = lay;
75     }

```

### 9.39.3.14 void KDChart::AbstractLayoutItem::setParentWidget (QWidget \* widget) [virtual, inherited]

Inform the item about its widget: This enables the item, to trigger that widget's update, whenever the size of the item's contents has changed.

Thus, you need to call setParentWidget on every item, that has a non-fixed size.

Definition at line 64 of file KDChartLayoutItems.cpp.

References KDChart::AbstractLayoutItem::mParent.

Referenced by KDChart::HeaderFooter::setParent(), and KDChart::AbstractCartesianDiagram::takeAxis().

```
65 {
66     mParent = widget;
67 }
```

### 9.39.3.15 QSize KDChart::MarkerLayoutItem::sizeHint () const [virtual]

Definition at line 567 of file KDChartLayoutItems.cpp.

References KDChart::MarkerAttributes::markerSize().

Referenced by maximumSize(), and minimumSize().

```
568 {
569     //qDebug() << "KDChart::MarkerLayoutItem::sizeHint() returns:"<<mMarker.markerSize().toSize();
570     return mMarker.markerSize().toSize();
571 }
```

### 9.39.3.16 void KDChart::AbstractLayoutItem::sizeHintChanged () const [virtual, inherited]

Report changed size hint: ask the parent widget to recalculate the layout.

Definition at line 86 of file KDChartLayoutItems.cpp.

References KDChart::AbstractLayoutItem::mParent.

Referenced by KDChart::TextLayoutItem::sizeHint().

```
87 {
88     // This is exactly like what QWidget::updateGeometry does.
89     // qDebug("KDChart::AbstractLayoutItem::sizeHintChanged() called");
90     if( mParent ) {
91         if ( mParent->layout() )
92             mParent->layout()->invalidate();
93         else
94             QApplication::postEvent( mParent, new QEvent( QEvent::LayoutRequest ) );
95     }
96 }
```

## 9.39.4 Member Data Documentation

### 9.39.4.1 QWidget\* KDChart::AbstractLayoutItem::mParent [protected, inherited]

Definition at line 90 of file KDChartLayoutItems.h.

Referenced by KDChart::AbstractLayoutItem::setParentWidget(), KDChart::TextLayoutItem::setText(), KDChart::TextLayoutItem::setTextAttributes(), and KDChart::AbstractLayoutItem::sizeHintChanged().

#### 9.39.4.2 QLayout\* [KDChart::AbstractLayoutItem::mParentLayout](#) [protected, inherited]

Definition at line 91 of file KDChartLayoutItems.h.

Referenced by KDChart::AutoSpacerLayoutItem::paint().

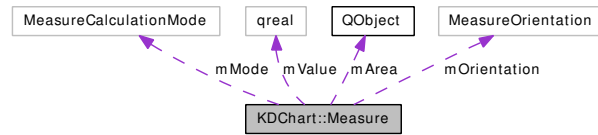
The documentation for this class was generated from the following files:

- [KDChartLayoutItems.h](#)
- [KDChartLayoutItems.cpp](#)

## 9.40 KDChart::Measure Class Reference

```
#include <KDChartMeasure>
```

Collaboration diagram for KDChart::Measure:



### 9.40.1 Detailed Description

**Measure** is used to specify all relative and/or absolute measures in **KDChart**, e.g. font sizes.

Definition at line 57 of file KDChartMeasure.h.

#### Public Member Functions

- qreal **calculatedValue** (const QSizeF &autoSize, **KDChartEnums::MeasureOrientation** autoOrientation) const
- qreal **calculatedValue** (const QObject \*autoArea, **KDChartEnums::MeasureOrientation** autoOrientation) const

*The reference area must either be derived from **AbstractArea** or be derived from **QWidget**, so e.g.*

- **KDChartEnums::MeasureCalculationMode** **calculationMode** () const
- **Measure** (const **Measure** &)
- **Measure** (qreal value, **KDChartEnums::MeasureCalculationMode** mode=**KDChartEnums::MeasureCalculationModeAuto**, **KDChartEnums::MeasureOrientation** orientation=**KDChartEnums::MeasureOrientationAuto**)
- **Measure** ()
- bool **operator!=** (const **Measure** &other) const
- **Measure** & **operator=** (const **Measure** &)
- bool **operator==** (const **Measure** &) const
- const QObject \* **referenceArea** () const

*The returned reference area will either be derived from **AbstractArea** or be derived from **QWidget**.*

- **KDChartEnums::MeasureOrientation** **referenceOrientation** () const
- void **setAbsoluteValue** (qreal val)

*This is a convenience method for specifying a value, with implicitly setting the calculation mode to **MeasureCalculationModeAbsolute**.*

- void **setCalculationMode** (**KDChartEnums::MeasureCalculationMode** mode)
- void **setReferenceArea** (const QObject \*area)

*The reference area must either be derived from **AbstractArea** or be derived from **QWidget**, so e.g.*

- void **setReferenceOrientation** (**KDChartEnums::MeasureOrientation** orientation)
- void **setRelativeMode** (const QObject \*area, **KDChartEnums::MeasureOrientation** orientation)

*The reference area must either be derived from [AbstractArea](#) or be derived from [QWidget](#), so e.g.*

- void [setValue](#) (qreal val)
- const QSizeF [sizeOfArea](#) (const [QObject](#) \*area) const
- qreal [value](#) () const

## 9.40.2 Constructor & Destructor Documentation

### 9.40.2.1 KDCart::Measure::Measure ()

Definition at line 41 of file KDCartMeasure.cpp.

```
42 : mValue( 0.0 ),
43   mMode( KDCartEnums::MeasureCalculationModeAuto ),
44   mArea( 0 ),
45   mOrientation( KDCartEnums::MeasureOrientationAuto )
46 {
47     // this bloc left empty intentionally
48 }
```

### 9.40.2.2 KDCart::Measure::Measure (qreal *value*, [KDCartEnums::MeasureCalculationMode](#) *mode* = [KDCartEnums::MeasureCalculationModeAuto](#), [KDCartEnums::MeasureOrientation](#) *orientation* = [KDCartEnums::MeasureOrientationAuto](#))

Definition at line 50 of file KDCartMeasure.cpp.

```
53 : mValue( value ),
54   mMode( mode ),
55   mArea( 0 ),
56   mOrientation( orientation )
57 {
58     // this bloc left empty intentionally
59 }
```

### 9.40.2.3 KDCart::Measure::Measure (const [Measure](#) &)

Definition at line 61 of file KDCartMeasure.cpp.

```
62 : mValue( r.value() ),
63   mMode( r.calculationMode() ),
64   mArea( r.referenceArea() ),
65   mOrientation( r.referenceOrientation() )
66 {
67     // this bloc left empty intentionally
68 }
```

## 9.40.3 Member Function Documentation

### 9.40.3.1 qreal KDCart::Measure::calculatedValue (const QSizeF & *autoSize*, [KDCartEnums::MeasureOrientation](#) *autoOrientation*) const

Definition at line 83 of file KDCartMeasure.cpp.

References `KDChartEnums::MeasureCalculationModeAbsolute`, `KDChartEnums::MeasureCalculationModeAuto`, `KDChartEnums::MeasureCalculationModeAutoArea`, `KDChartEnums::MeasureCalculationModeAutoOrientation`, `KDChartEnums::MeasureCalculationModeRelative`, `KDChartEnums::MeasureOrientationAuto`, `KDChartEnums::MeasureOrientationHorizontal`, `KDChartEnums::MeasureOrientationMaximum`, `KDChartEnums::MeasureOrientationMinimum`, `KDChartEnums::MeasureOrientationVertical`, `sizeofArea()`, and `value()`.

```

85 {
86     if( mMode == KDChartEnums::MeasureCalculationModeAbsolute ){
87         return mValue;
88     }else{
89         qreal value = 0.0;
90         const QObject theAutoArea;
91         const QObject* autoArea = &theAutoArea;
92         const QObject* area = mArea ? mArea : autoArea;
93         KDChartEnums::MeasureOrientation orientation = mOrientation;
94         switch( mMode ){
95             case KDChartEnums::MeasureCalculationModeAuto:
96                 area = autoArea;
97                 orientation = autoOrientation;
98                 break;
99             case KDChartEnums::MeasureCalculationModeAutoArea:
100                 area = autoArea;
101                 break;
102             case KDChartEnums::MeasureCalculationModeAutoOrientation:
103                 orientation = autoOrientation;
104                 break;
105             case KDChartEnums::MeasureCalculationModeAbsolute: // fall through intended
106             case KDChartEnums::MeasureCalculationModeRelative:
107                 break;
108         }
109         if( area ){
110             QSizeF size;
111             if( area == autoArea )
112                 size = autoSize;
113             else
114                 size = sizeofArea( area );
115             //qDebug() << "size" << size;
116             qreal referenceValue;
117             switch( orientation ){
118                 case KDChartEnums::MeasureOrientationAuto: // fall through intended
119                 case KDChartEnums::MeasureOrientationMinimum:
120                     referenceValue = qMin( size.width(), size.height() );
121                     break;
122                 case KDChartEnums::MeasureOrientationMaximum:
123                     referenceValue = qMax( size.width(), size.height() );
124                     break;
125                 case KDChartEnums::MeasureOrientationHorizontal:
126                     referenceValue = size.width();
127                     break;
128                 case KDChartEnums::MeasureOrientationVertical:
129                     referenceValue = size.height();
130                     break;
131             }
132             value = mValue / 1000.0 * referenceValue;
133         }
134         return value;
135     }
136 }

```

### 9.40.3.2 `qreal KDChart::Measure::calculatedValue (const QObject * autoArea, KDChartEnums::MeasureOrientation autoOrientation) const`

The reference area must either be derived from [AbstractArea](#) or be derived from [QWidget](#), so e.g.

it could be derived from [AbstractAreaWidget](#) too.

Definition at line 139 of file `KDChartMeasure.cpp`.

References `sizeOfArea()`.

Referenced by `KDChart::TextAttributes::calculatedFontSize()`, and `KDChart::Relative-Position::calculatedPoint()`.

```
141 {
142     return calculatedValue( sizeOfArea( autoArea ), autoOrientation);
143 }
```

### 9.40.3.3 `KDChartEnums::MeasureCalculationMode KDChart::Measure::calculationMode () const`

Definition at line 71 of file `KDChartMeasure.h`.

Referenced by `operator<<()`, `operator=()`, and `operator==()`.

```
71 { return mMode; }
```

### 9.40.3.4 `bool KDChart::Measure::operator!= (const Measure & other) const`

Definition at line 127 of file `KDChartMeasure.h`.

```
127 { return !operator==(other); }
```

### 9.40.3.5 `Measure & KDChart::Measure::operator= (const Measure &)`

Definition at line 70 of file `KDChartMeasure.cpp`.

References `calculationMode()`, `referenceArea()`, `referenceOrientation()`, and `value()`.

```
71 {
72     if( this != &r ){
73         mValue = r.value();
74         mMode = r.calculationMode();
75         mArea = r.referenceArea();
76         mOrientation = r.referenceOrientation();
77     }
78
79     return *this;
80 }
```

#### 9.40.3.6 **bool KDChart::Measure::operator==(const [Measure](#) &) const**

Definition at line 177 of file KDChartMeasure.cpp.

References [calculationMode\(\)](#), [referenceArea\(\)](#), [referenceOrientation\(\)](#), and [value\(\)](#).

```

178 {
179     return( mValue == r.value() &&
180            mMode == r.calculationMode() &&
181            mArea == r.referenceArea() &&
182            mOrientation == r.referenceOrientation() );
183 }
```

#### 9.40.3.7 **const [QObject](#)\* KDChart::Measure::referenceArea () const**

The returned reference area will either be derived from [AbstractArea](#) or be derived from [QWidget](#).

Definition at line 112 of file KDChartMeasure.h.

Referenced by [operator<<\(\)](#), [operator=\(\)](#), and [operator==\(\)](#).

```

112 { return mArea; }
```

#### 9.40.3.8 **[KDChartEnums::MeasureOrientation](#) KDChart::Measure::referenceOrientation () const**

Definition at line 115 of file KDChartMeasure.h.

Referenced by [operator<<\(\)](#), [operator=\(\)](#), and [operator==\(\)](#).

```

115 { return mOrientation; }
```

#### 9.40.3.9 **void KDChart::Measure::setAbsoluteValue (qreal *val*)**

This is a convenience method for specifying a value, with implicitly setting the calculation mode to [MeasureCalculationModeAbsolute](#).

Calling [setAbsoluteValue\( value \)](#) is the same as calling

```

setValue( value );
setCalculationMode( KDChartEnums::MeasureCalculationModeAbsolute );
```

Definition at line 96 of file KDChartMeasure.h.

References [KDChartEnums::MeasureCalculationModeAbsolute](#).

```

97     {
98         mMode = KDChartEnums::MeasureCalculationModeAbsolute;
99         mValue = val;
100     }
```



**9.40.3.10 void KDChart::Measure::setCalculationMode ([KDChartEnums::MeasureCalculationMode](#) *mode*)**

Definition at line 70 of file KDChartMeasure.h.

```
70 { mMode = mode; }
```

**9.40.3.11 void KDChart::Measure::setReferenceArea (const [QObject](#) \* *area*)**

The reference area must either be derived from [AbstractArea](#) or be derived from [QWidget](#), so e.g. it could be derived from [AbstractAreaWidget](#) too.

Definition at line 107 of file KDChartMeasure.h.

```
107 { mArea = area; }
```

**9.40.3.12 void KDChart::Measure::setReferenceOrientation ([KDChartEnums::MeasureOrientation](#) *orientation*)**

Definition at line 114 of file KDChartMeasure.h.

```
114 { mOrientation = orientation; }
```

**9.40.3.13 void KDChart::Measure::setRelativeMode (const [QObject](#) \* *area*, [KDChartEnums::MeasureOrientation](#) *orientation*)**

The reference area must either be derived from [AbstractArea](#) or be derived from [QWidget](#), so e.g. it could be derived from [AbstractAreaWidget](#) too.

Definition at line 78 of file KDChartMeasure.h.

References [KDChartEnums::MeasureCalculationModeRelative](#).

```
80 {  
81     mMode = KDChartEnums::MeasureCalculationModeRelative;  
82     mArea = area;  
83     mOrientation = orientation;  
84 }
```

**9.40.3.14 void KDChart::Measure::setValue (qreal *val*)**

Definition at line 67 of file KDChartMeasure.h.

```
67 { mValue = val; }
```

### 9.40.3.15 `const QSizeF KDChart::Measure::sizeOfArea (const QObject * area) const`

Definition at line 146 of file `KDChartMeasure.cpp`.

References `KDChart::GlobalMeasureScaling::currentFactors()`, `KDChart::GlobalMeasureScaling::instance()`, and `KDChartEnums::MeasureCalculationModeAbsolute`.

Referenced by `calculatedValue()`.

```

147 {
148     QSizeF size;
149     const AbstractArea* kdcArea = dynamic_cast<const AbstractArea*>(area);
150     if( kdcArea ){
151         size = kdcArea->geometry().size();
152         //qDebug() << "Measure::sizeOfArea() found kdcArea with size" << size;
153     }else{
154         const QWidget* widget = dynamic_cast<const QWidget*>(area);
155         if( widget ){
156             /* ATTENTION: Using the layout does not work: The Legend will never get the right size the
157             const QLayout * layout = widget->layout();
158             if( layout ){
159                 size = layout->geometry().size();
160                 //qDebug() << "Measure::sizeOfArea() found widget with layout size" << size;
161             }else*/
162             {
163                 size = widget->geometry().size();
164                 //qDebug() << "Measure::sizeOfArea() found widget with size" << size;
165             }
166         }else if( mMode != KDChartEnums::MeasureCalculationModeAbsolute ){
167             size = QSizeF(1.0, 1.0);
168             //qDebug("Measure::sizeOfArea() got no valid area.");
169         }
170     }
171     const QPair< qreal, qreal > factors
172         = GlobalMeasureScaling::instance()->currentFactors();
173     return QSizeF(size.width() * factors.first, size.height() * factors.second);
174 }
```

### 9.40.3.16 `qreal KDChart::Measure::value () const`

Definition at line 68 of file `KDChartMeasure.h`.

Referenced by `calculatedValue()`, `operator<<()`, `operator=()`, and `operator==()`.

```

68 { return mValue; }
```

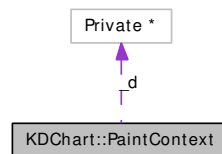
The documentation for this class was generated from the following files:

- [KDChartMeasure.h](#)
- [KDChartMeasure.cpp](#)

## 9.41 KDChart::PaintContext Class Reference

```
#include <KDChartPaintContext.h>
```

Collaboration diagram for KDChart::PaintContext:



### 9.41.1 Detailed Description

Stores information about painting diagrams.

Definition at line 43 of file KDChartPaintContext.h.

### Public Member Functions

- [AbstractCoordinatePlane](#) \* [coordinatePlane](#) () const
- [PaintContext](#) ()
- [QPainter](#) \* [painter](#) () const
- const [QRectF](#) [rectangle](#) () const
- void [setCoordinatePlane](#) ([AbstractCoordinatePlane](#) \*plane)
- void [setPainter](#) ([QPainter](#) \*painter)
- void [setRectangle](#) (const [QRectF](#) &rect)
- [~PaintContext](#) ()

### 9.41.2 Constructor & Destructor Documentation

#### 9.41.2.1 PaintContext::PaintContext ()

Definition at line 51 of file KDChartPaintContext.cpp.

```

52     : _d ( new Private() )
53 {
54 }
```

#### 9.41.2.2 PaintContext::~PaintContext ()

Definition at line 56 of file KDChartPaintContext.cpp.

```

57 {
58     delete _d;
59 }
```

### 9.41.3 Member Function Documentation

#### 9.41.3.1 [AbstractCoordinatePlane](#) \* [PaintContext::coordinatePlane](#) () const

Definition at line 81 of file `KDChartPaintContext.cpp`.

References d.

Referenced by `KDChart::TernaryPointDiagram::paint()`, `KDChart::TernaryLineDiagram::paint()`, `KDChart::Plotter::paint()`, `KDChart::LineDiagram::paint()`, `KDChart::BarDiagram::paint()`, `KDChart::TernaryAxis::paintCtx()`, and `KDChart::CartesianAxis::paintCtx()`.

```
82 {
83     return d->plane;
84 }
```

#### 9.41.3.2 [QPainter](#) \* [PaintContext::painter](#) () const

Definition at line 71 of file `KDChartPaintContext.cpp`.

References d.

Referenced by `KDChart::TernaryPointDiagram::paint()`, `KDChart::TernaryLineDiagram::paint()`, `KDChart::RingDiagram::paint()`, `KDChart::PolarDiagram::paint()`, `KDChart::Plotter::paint()`, `KDChart::PieDiagram::paint()`, `KDChart::LineDiagram::paint()`, `KDChart::BarDiagram::paint()`, `KDChart::TernaryAxis::paintCtx()`, `KDChart::AbstractLayoutItem::paintCtx()`, `KDChart::CartesianAxis::paintCtx()`, and `KDChart::PolarDiagram::paintPolarMarkers()`.

```
72 {
73     return d->painter;
74 }
```

#### 9.41.3.3 [const QRectF](#) [PaintContext::rectangle](#) () const

Definition at line 61 of file `KDChartPaintContext.cpp`.

References d.

Referenced by `KDChart::PolarDiagram::paint()`, `KDChart::PieDiagram::paint()`, and `KDChart::TernaryAxis::paintCtx()`.

```
62 {
63     return d->rect;
64 }
```

#### 9.41.3.4 [void](#) [PaintContext::setCoordinatePlane](#) ([AbstractCoordinatePlane](#) \* *plane*)

Definition at line 86 of file `KDChartPaintContext.cpp`.

References d.

Referenced by `KDChart::Plotter::paint()`, `KDChart::LineDiagram::paint()`, `KDChart::CartesianAxis::paint()`, and `KDChart::BarDiagram::paint()`.

```
87 {
88     d->plane = plane;
89 }
```

### 9.41.3.5 void PaintContext::setPainter (QPainter \* *painter*)

Definition at line 76 of file KDChartPaintContext.cpp.

References `d`.

Referenced by `KDChart::CartesianAxis::paint()`, `KDChart::RingDiagram::paintEvent()`, `KDChart::PolarDiagram::paintEvent()`, `KDChart::Plotter::paintEvent()`, `KDChart::PieDiagram::paintEvent()`, `KDChart::LineDiagram::paintEvent()`, and `KDChart::BarDiagram::paintEvent()`.

```
77 {  
78     d->painter = painter;  
79 }
```

### 9.41.3.6 void PaintContext::setRectangle (const QRectF & *rect*)

Definition at line 66 of file KDChartPaintContext.cpp.

References `d`.

Referenced by `KDChart::CartesianAxis::paint()`, `KDChart::RingDiagram::paintEvent()`, `KDChart::PolarDiagram::paintEvent()`, `KDChart::Plotter::paintEvent()`, `KDChart::PieDiagram::paintEvent()`, `KDChart::LineDiagram::paintEvent()`, and `KDChart::BarDiagram::paintEvent()`.

```
67 {  
68     d->rect = rect;  
69 }
```

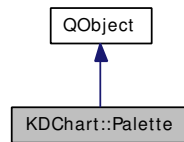
The documentation for this class was generated from the following files:

- [KDChartPaintContext.h](#)
- [KDChartPaintContext.cpp](#)

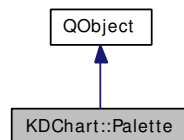
## 9.42 KDChart::Palette Class Reference

```
#include <KDChartPalette.h>
```

Inheritance diagram for KDChart::Palette:



Collaboration diagram for KDChart::Palette:



### 9.42.1 Detailed Description

A [Palette](#) is a set of brushes (or colors) to be used for painting data sets.

The palette class encapsulates a collection of brushes, which in the simplest case are colors, to be used for painting a series of data sets. When asked for the m-th color, a palette of size n will wrap around and thus cycle through the available colors.

Three builtin palettes are provided for convenience, one with a default set of colors, one with a subdued color selection, one with rainbow colors.

When a palette changes, it emits a [changed\(\)](#) signal. Hook up to it, if you want to repaint when the color selection changes.

Definition at line 55 of file KDChartPalette.h.

### Signals

- void [changed](#) ()  
*Emitted whenever the palette changes.*

### Public Member Functions

- void [addBrush](#) (const QBrush &brush, int position=-1)  
*Adds brush to the palette.*
- QBrush [getBrush](#) (int position) const  
*Query the palette for a brush at the specified position.*
- bool [isValid](#) () const

**Returns:**

*wether this represents a valid palette.*

- [Palette](#) & [operator=](#) (const [Palette](#) &)
- [Palette](#) (const [Palette](#) &)
- [Palette](#) ([QObject](#) \*parent=0)
- void [removeBrush](#) (int position)

*Remove the brush at position position, if there is one.*

- int [size](#) () const

**Returns:**

*the number of brushed in the palette.*

- [~Palette](#) ()

**Static Public Member Functions**

- static const [Palette](#) & [defaultPalette](#) ()

*Provide access to the three builtin palettes, one with standard bright colors, one with more subdued colors, and one with rainbow colors.*

- static const [Palette](#) & [rainbowPalette](#) ()
- static const [Palette](#) & [subduedPalette](#) ()

**9.42.2 Constructor & Destructor Documentation****9.42.2.1 [Palette::Palette](#) ([QObject](#) \*parent = 0) [explicit]**

Definition at line 132 of file [KDChartPalette.cpp](#).

```
133 : QObject( parent ), _d( new Private )
134 {
135
136 }
```

**9.42.2.2 [Palette::Palette](#) (const [Palette](#) &)**

Definition at line 145 of file [KDChartPalette.cpp](#).

```
146 : QObject(), _d( new Private( *r.d ) )
147 {
148 }
```

**9.42.2.3 [Palette::~~Palette](#) ()**

Definition at line 138 of file [KDChartPalette.cpp](#).

```
139 {
140     delete _d; _d = 0;
141 }
```

### 9.42.3 Member Function Documentation

#### 9.42.3.1 void Palette::addBrush (const QBrush & *brush*, int *position* = -1)

Adds *brush* to the palette.

If no *position* is specified, the brush is appended.

Definition at line 169 of file KDChartPalette.cpp.

References changed(), d, and size().

Referenced by makeDefaultPalette(), makeRainbowPalette(), and makeSubduedPalette().

```
170 {
171     if ( position < 0 || position >= size() ) {
172         d->brushes.append( brush );
173     } else {
174         d->brushes.insert( position, brush );
175     }
176     emit changed();
177 }
```

#### 9.42.3.2 void KDChart::Palette::changed () [signal]

Emitted whenever the palette changes.

Views listen to this and repainting.

Referenced by addBrush(), and removeBrush().

#### 9.42.3.3 const [Palette](#) & Palette::defaultPalette () [static]

Provide access to the three builtin palettes, one with standard bright colors, one with more subdued colors, and one with rainbow colors.

Definition at line 114 of file KDChartPalette.cpp.

References makeDefaultPalette().

Referenced by KDChart::AttributesModel::headerData().

```
115 {
116     static const Palette palette = makeDefaultPalette();
117     return palette;
118 }
```

#### 9.42.3.4 QBrush Palette::getBrush (int *position*) const

Query the palette for a brush at the specified position.

If the position exceeds the size of the palette, it wraps around.

Definition at line 179 of file KDChartPalette.cpp.

References d, isValid(), and size().

Referenced by KDChart::AttributesModel::headerData(), and makeRainbowPalette().



```
180 {  
181     if ( !isValid() ) return QBrush();  
182     return d->brushes.at( position % size() );  
183 }
```

#### 9.42.3.5 bool Palette::isValid () const

##### Returns:

wether this represents a valid palette.

For a palette to be valid it needs to have at least one brush associated.

Definition at line 159 of file KDChartPalette.cpp.

References d.

Referenced by getBrush().

```
160 {  
161     return d->brushes.size() >= 1;  
162 }
```

#### 9.42.3.6 Palette & Palette::operator= (const Palette &)

Definition at line 150 of file KDChartPalette.cpp.

```
151 {  
152     Palette copy( r );  
153     copy.swap( *this );  
154  
155     // emit changed() ?  
156     return *this;  
157 }
```

#### 9.42.3.7 const Palette & Palette::rainbowPalette () [static]

Definition at line 126 of file KDChartPalette.cpp.

References makeRainbowPalette().

Referenced by KDChart::AttributesModel::headerData().

```
127 {  
128     static const Palette palette = makeRainbowPalette();  
129     return palette;  
130 }
```

#### 9.42.3.8 void Palette::removeBrush (int position)

Remove the brush at position *position*, if there is one.

Definition at line 185 of file KDChartPalette.cpp.

References changed(), d, and size().

```
186 {  
187     if ( position < 0 || position >= size() ) return;  
188     d->brushes.remove( position );  
189     emit changed();  
190 }
```

#### 9.42.3.9 int Palette::size () const

##### Returns:

the number of brushed in the palette.

Definition at line 164 of file KDChartPalette.cpp.

References d.

Referenced by addBrush(), getBrush(), and removeBrush().

```
165 {  
166     return d->brushes.size();  
167 }
```

#### 9.42.3.10 const [Palette](#) & Palette::subduedPalette () [static]

Definition at line 120 of file KDChartPalette.cpp.

References makeSubduedPalette().

Referenced by KDChart::AttributesModel::headerData().

```
121 {  
122     static const Palette palette = makeSubduedPalette();  
123     return palette;  
124 }
```

The documentation for this class was generated from the following files:

- [KDChartPalette.h](#)
- [KDChartPalette.cpp](#)

## 9.43 KDCart::PieAttributes Class Reference

```
#include <KDCartPieAttributes.h>
```

### 9.43.1 Detailed Description

A set of attributes controlling the appearance of pie charts.

Definition at line 38 of file KDCartPieAttributes.h.

### Public Member Functions

- bool [explode](#) () const  
*Returns:*  
*whether the respective pie piece(s) will be exploded.*
- qreal [explodeFactor](#) () const  
*Returns:*  
*the explode factor set by [setExplode](#) or by [setExplodeFactor](#).*
- bool [operator!=](#) (const [PieAttributes](#) &other) const
- [PieAttributes](#) & [operator=](#) (const [PieAttributes](#) &)
- bool [operator==](#) (const [PieAttributes](#) &) const
- [PieAttributes](#) (const [PieAttributes](#) &)
- [PieAttributes](#) ()
- void [setExplode](#) (bool explode)  
*Enable or disable exploding the respective pie piece(s).*
- void [setExplodeFactor](#) (qreal factor)  
*Set the explode factor.*
- [~PieAttributes](#) ()

### 9.43.2 Constructor & Destructor Documentation

#### 9.43.2.1 PieAttributes::PieAttributes ()

Definition at line 45 of file KDCartPieAttributes.cpp.

```
46      : _d( new Private() )
47  {
48  }
```

#### 9.43.2.2 PieAttributes::PieAttributes (const [PieAttributes](#) &)

Definition at line 50 of file KDCartPieAttributes.cpp.

```
51      : _d( new Private( *r.d ) )
52  {
53  }
```

### 9.43.2.3 PieAttributes::~~PieAttributes ()

Definition at line 65 of file KDChartPieAttributes.cpp.

```
66 {  
67     delete _d; _d = 0;  
68 }
```

## 9.43.3 Member Function Documentation

### 9.43.3.1 bool PieAttributes::explode () const

#### Returns:

whether the respective pie piece(s) will be exploded.

Definition at line 90 of file KDChartPieAttributes.cpp.

References [d](#).

```
91 {  
92     return (d->explodeFactor != 0.0);  
93 }
```

### 9.43.3.2 qreal PieAttributes::explodeFactor () const

#### Returns:

the explode factor set by setExplode or by setExplodeFactor.

Definition at line 100 of file KDChartPieAttributes.cpp.

References [d](#).

Referenced by [operator<<\(\)](#), and [operator==\(\)](#).

```
101 {  
102     return d->explodeFactor;  
103 }
```

### 9.43.3.3 bool KDChart::PieAttributes::operator!= (const [PieAttributes](#) & *other*) const

Definition at line 75 of file KDChartPieAttributes.h.

```
75 { return !operator==(other); }
```

### 9.43.3.4 [PieAttributes](#) & PieAttributes::operator= (const [PieAttributes](#) &)

Definition at line 55 of file KDChartPieAttributes.cpp.

References [d](#).

```
56 {  
57     if( this == &r )  
58         return *this;  
59  
60     *d = *r.d;  
61  
62     return *this;  
63 }
```

#### 9.43.3.5 bool PieAttributes::operator==(const [PieAttributes](#) &) const

Definition at line 71 of file KDChartPieAttributes.cpp.

References [explodeFactor\(\)](#).

```
72 {  
73     if( explodeFactor() == r.explodeFactor() )  
74         return true;  
75     else  
76         return false;  
77 }
```

#### 9.43.3.6 void PieAttributes::setExplode (bool *explode*)

Enable or disable exploding the respective pie piece(s).

The default explode factor is 10 percent; use [setExplodeFactor](#) to specify a different factor.

##### Note:

This is a convenience function: Calling [setExplode](#)( true ) does the same as calling [setExplodeFactor](#)( 0.1 ), and calling [setExplode](#)( false ) does the same as calling [setExplodeFactor](#)( 0.0 ).

##### See also:

[setExplodeFactor](#)

Definition at line 85 of file KDChartPieAttributes.cpp.

References [d](#).

```
86 {  
87     d->explodeFactor = (enabled ? 0.1 : 0.0);  
88 }
```

#### 9.43.3.7 void PieAttributes::setExplodeFactor (qreal *factor*)

Set the explode factor.

The explode factor is a qreal between 0 and 1, and is interpreted as a percentage of the total available radius of the pie.

##### See also:

[setExplode](#)

Definition at line 95 of file KDChartPieAttributes.cpp.

References d.

```
96 {  
97     d->explodeFactor = factor;  
98 }
```

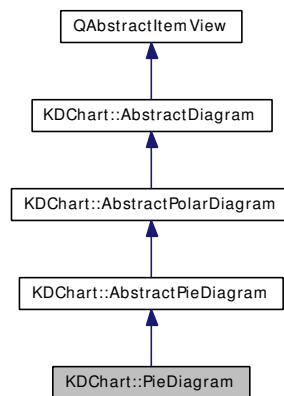
The documentation for this class was generated from the following files:

- [KDChartPieAttributes.h](#)
- [KDChartPieAttributes.cpp](#)

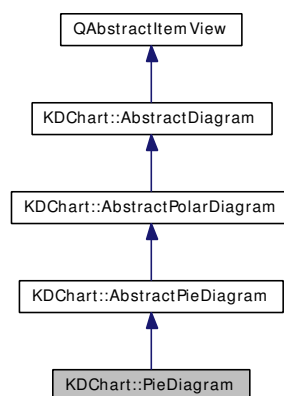
## 9.44 KDChart::PieDiagram Class Reference

```
#include <KDChartPieDiagram.h>
```

Inheritance diagram for KDChart::PieDiagram:



Collaboration diagram for KDChart::PieDiagram:



### 9.44.1 Detailed Description

[PieDiagram](#) defines a common pie diagram.

Definition at line 40 of file KDChartPieDiagram.h.

#### Signals

- void [dataHidden](#) ()  
*This signal is emitted, when the hidden status of at least one data cell was (un)set.*
- void [layoutChanged](#) ([AbstractDiagram](#) \*)  
*Diagrams are supposed to emit this signal, when the layout of one of their element changes.*
- void [modelsChanged](#) ()

*This signal is emitted, when either the model or the [AttributesModel](#) is replaced.*

- void [propertiesChanged](#) ()  
*Emitted upon change of a property of the Diagram.*

## Public Member Functions

- bool [allowOverlappingDataValueTexts](#) () const  
**Returns:**  
*Whether data value labels are allowed to overlap.*
- bool [antiAliasing](#) () const  
**Returns:**  
*Whether anti-aliasing is to be used for rendering this diagram.*
- virtual [AttributesModel](#) \* [attributesModel](#) () const  
*Returns the [AttributesModel](#), that is used by this diagram.*
- QBrush [brush](#) (const QModelIndex &index) const  
*Retrieve the brush to be used, for painting the datapoint at the given index in the model.*
- QBrush [brush](#) (int dataset) const  
*Retrieve the brush to be used for the given dataset.*
- QBrush [brush](#) () const  
*Retrieve the brush to be used for painting datapoints globally.*
- virtual [PieDiagram](#) \* [clone](#) () const  
*Creates an exact copy of this diagram.*
- int [columnCount](#) () const
- bool [compare](#) (const [AbstractDiagram](#) \*other) const  
*Returns true if both diagrams have the same settings.*
- [AbstractCoordinatePlane](#) \* [coordinatePlane](#) () const  
*The coordinate plane associated with the diagram.*
- const QPair< QPointF, QPointF > [dataBoundaries](#) () const  
*Return the bottom left and top right data point, that the diagram will display (unless the grid adjusts these values).*
- virtual void [dataChanged](#) (const QModelIndex &topLeft, const QModelIndex &bottomRight)  
*[reimplemented]*
- QList< QBrush > [datasetBrushes](#) () const  
*The set of dataset brushes currently used, for use in legends, etc.*
- int [datasetDimension](#) () const



*The dataset dimension of a diagram determines, how many value dimensions it expects each datapoint to have.*

- QStringList [datasetLabels](#) () const  
*The set of dataset labels currently displayed, for use in legends, etc.*
- QList< [MarkerAttributes](#) > [datasetMarkers](#) () const  
*The set of dataset markers currently used, for use in legends, etc.*
- QList< QPen > [datasetPens](#) () const  
*The set of dataset pens currently used, for use in legends, etc.*
- [DataValueAttributes](#) [dataValueAttributes](#) (const QModelIndex &index) const  
*Retrieve the [DataValueAttributes](#) for the given index.*
- [DataValueAttributes](#) [dataValueAttributes](#) (int column) const  
*Retrieve the [DataValueAttributes](#) for the given dataset.*
- [DataValueAttributes](#) [dataValueAttributes](#) () const  
*Retrieve the [DataValueAttributes](#) specified globally.*
- virtual void [doItemsLayout](#) ()  
*[reimplemented]*
- qreal [granularity](#) () const  
**Returns:**  
*the granularity.*
- virtual int [horizontalOffset](#) () const  
*[reimplemented]*
- virtual QModelIndex [indexAt](#) (const QPoint &point) const  
*[reimplemented]*
- QModelIndexList [indexesAt](#) (const QPoint &point) const  
*This method is added alongside with [indexAt](#) from QAIM, since in kdchart multiple indexes can be displayed at the same spot.*
- bool [isHidden](#) (const QModelIndex &index) const  
*Retrieve the hidden status for the given index.*
- bool [isHidden](#) (int column) const  
*Retrieve the hidden status for the given dataset.*
- bool [isHidden](#) () const  
*Retrieve the hidden status specified globally.*
- virtual bool [isIndexHidden](#) (const QModelIndex &index) const  
*[reimplemented]*

- QStringList [itemRowLabels](#) () const  
*The set of item row labels currently displayed, for use in Abscissa axes, etc.*
- virtual QModelIndex [moveCursor](#) (CursorAction cursorAction, Qt::KeyboardModifiers modifiers)  
*[reimplemented]*
- virtual double [numberOfGridRings](#) () const  
*[reimplemented]*
- virtual double [numberOfValuesPerDataset](#) () const  
*[reimplemented]*
- void [paintDataValueText](#) (QPainter \*painter, const QModelIndex &index, const QPointF &pos, double value)
- void [paintMarker](#) (QPainter \*painter, const QModelIndex &index, const QPointF &pos)
- virtual void [paintMarker](#) (QPainter \*painter, const [MarkerAttributes](#) &markerAttributes, const QBrush &brush, const QPen &, const QPointF &point, const QSizeF &size)
- QPen [pen](#) (const QModelIndex &index) const  
*Retrieve the pen to be used, for painting the datapoint at the given index in the model.*
- QPen [pen](#) (int dataset) const  
*Retrieve the pen to be used for the given dataset.*
- QPen [pen](#) () const  
*Retrieve the pen to be used for painting datapoints globally.*
- bool [percentMode](#) () const
- [PieAttributes](#) [pieAttributes](#) (const QModelIndex &index) const
- [PieAttributes](#) [pieAttributes](#) (int column) const
- [PieAttributes](#) [pieAttributes](#) () const
- [PieDiagram](#) (QWidget \*parent=0, [PolarCoordinatePlane](#) \*plane=0)
- const [PolarCoordinatePlane](#) \* [polarCoordinatePlane](#) () const
- virtual void [resize](#) (const QSizeF &area)  
*[reimplemented]*
- virtual void [scrollTo](#) (const QModelIndex &index, ScrollHint hint=EnsureVisible)  
*[reimplemented]*
- void [setAllowOverlappingDataValueTexts](#) (bool allow)  
*Set whether data value labels are allowed to overlap.*
- void [setAntiAliasing](#) (bool enabled)  
*Set whether anti-aliasing is to be used while rendering this diagram.*
- virtual void [setAttributesModel](#) ([AttributesModel](#) \*model)  
*Associate an [AttributesModel](#) with this diagram.*
- void [setBrush](#) (const QBrush &brush)  
*Set the brush to be used, for painting all datasets in the model.*

- void [setBrush](#) (int dataset, const QBrush &brush)  
*Set the brush to be used, for painting the given dataset.*
- void [setBrush](#) (const QModelIndex &index, const QBrush &brush)  
*Set the brush to be used, for painting the datapoint at the given index.*
- virtual void [setCoordinatePlane](#) ([AbstractCoordinatePlane](#) \*plane)  
*Set the coordinate plane associated with the diagram.*
- void [setDatasetDimension](#) (int dimension)  
*Sets the dataset dimension of the diagram.*
- void [setDataValueAttributes](#) (const [DataValueAttributes](#) &a)  
*Set the [DataValueAttributes](#) for all datapoints in the model.*
- void [setDataValueAttributes](#) (int dataset, const [DataValueAttributes](#) &a)  
*Set the [DataValueAttributes](#) for the given dataset.*
- void [setDataValueAttributes](#) (const QModelIndex &index, const [DataValueAttributes](#) &a)  
*Set the [DataValueAttributes](#) for the given index.*
- void [setGranularity](#) (qreal value)  
*Set the granularity: the smaller the granularity the more your diagram segments will show facettes instead of rounded segments.*
- void [setHidden](#) (bool hidden)  
*Hide (or unhide, resp.*
- void [setHidden](#) (int column, bool hidden)  
*Hide (or unhide, resp.*
- void [setHidden](#) (const QModelIndex &index, bool hidden)  
*Hide (or unhide, resp.*
- virtual void [setModel](#) (QAbstractItemModel \*model)  
*Associate a model with the diagram.*
- void [setPen](#) (const QPen &pen)  
*Set the pen to be used, for painting all datasets in the model.*
- void [setPen](#) (int dataset, const QPen &pen)  
*Set the pen to be used, for painting the given dataset.*
- void [setPen](#) (const QModelIndex &index, const QPen &pen)  
*Set the pen to be used, for painting the datapoint at the given index.*
- void [setPercentMode](#) (bool percent)
- void [setPieAttributes](#) (int column, const [PieAttributes](#) &a)
- void [setPieAttributes](#) (const [PieAttributes](#) &a)
- virtual void [setRootIndex](#) (const QModelIndex &idx)

*Set the root index in the model, where the diagram starts referencing data for display.*

- virtual void [setSelection](#) (const QRect &rect, QItemSelectionModel::SelectionFlags command)  
*[reimplemented]*

- void [setStartPosition](#) (int degrees)
- void [setThreeDPieAttributes](#) (const QModelIndex &index, const [ThreeDPieAttributes](#) &a)
- void [setThreeDPieAttributes](#) (int column, const [ThreeDPieAttributes](#) &a)
- void [setThreeDPieAttributes](#) (const [ThreeDPieAttributes](#) &a)
- void [setUnitPrefix](#) (const QString &prefix, Qt::Orientation orientation)

*Sets the unit prefix for all values.*

- void [setUnitPrefix](#) (const QString &prefix, int column, Qt::Orientation orientation)  
*Sets the unit prefix for one value.*

- void [setUnitSuffix](#) (const QString &suffix, Qt::Orientation orientation)  
*Sets the unit suffix for all values.*

- void [setUnitSuffix](#) (const QString &suffix, int column, Qt::Orientation orientation)  
*Sets the unit suffix for one value.*

- int [startPosition](#) () const
- [ThreeDPieAttributes](#) [threeDPieAttributes](#) (const QModelIndex &index) const
- [ThreeDPieAttributes](#) [threeDPieAttributes](#) (int column) const
- [ThreeDPieAttributes](#) [threeDPieAttributes](#) () const
- QString [unitPrefix](#) (Qt::Orientation orientation) const  
*Returns the global unit prefix.*

- QString [unitPrefix](#) (int column, Qt::Orientation orientation, bool fallback=false) const  
*Returns the unit prefix for a special value.*

- QString [unitSuffix](#) (Qt::Orientation orientation) const  
*Returns the global unit suffix.*

- QString [unitSuffix](#) (int column, Qt::Orientation orientation, bool fallback=false) const  
*Returns the unit suffix for a special value.*

- void [update](#) () const
- void [useDefaultColors](#) ()  
*Set the palette to be used, for painting datasets to the default palette.*

- void [useRainbowColors](#) ()  
*Set the palette to be used, for painting datasets to the rainbow palette.*

- virtual bool [usesExternalAttributesModel](#) () const  
*Returns whether the diagram is using its own built-in attributes model or an attributes model that was set via [setAttributesModel](#).*

- void [useSubduedColors](#) ()  
*Set the palette to be used, for painting datasets to the subdued palette.*

- virtual double [valueTotals](#) () const  
*[reimplemented]*
- virtual int [verticalOffset](#) () const  
*[reimplemented]*
- virtual QRect [visualRect](#) (const QModelIndex &index) const  
*[reimplemented]*
- virtual QRegion [visualRegionForSelection](#) (const QItemSelection &selection) const  
*[reimplemented]*
- virtual [~PieDiagram](#) ()

## Protected Member Functions

- QModelIndex [attributesModelRootIndex](#) () const
- virtual const QPair< QPointF, QPointF > [calculateDataBoundaries](#) () const  
*[reimplemented]*
- virtual bool [checkInvariants](#) (bool justReturnTheStatus=false) const
- QModelIndex [columnToIndex](#) (int column) const
- virtual void [paint](#) (PaintContext \*paintContext)  
*[reimplemented]*
- virtual void [paintDataValueTexts](#) (QPainter \*painter)
- void [paintEvent](#) (QPaintEvent \*)
- virtual void [paintMarkers](#) (QPainter \*painter)
- void [resizeEvent](#) (QResizeEvent \*)
- void [setAttributesModelRootIndex](#) (const QModelIndex &)
- void [setDataBoundariesDirty](#) () const
- double [valueForCell](#) (int row, int column) const  
*Helper method, retrieving the data value (DisplayRole) for a given row and column.*

## 9.44.2 Constructor & Destructor Documentation

### 9.44.2.1 PieDiagram::PieDiagram (QWidget \*parent = 0, PolarCoordinatePlane \*plane = 0) *[explicit]*

Definition at line 52 of file KDChartPieDiagram.cpp.

Referenced by clone().

```

52                                     :
53     AbstractPieDiagram( new Private(), parent, plane )
54 {
55     init();
56 }
```

#### 9.44.2.2 PieDiagram::~PieDiagram () [virtual]

Definition at line 58 of file KDChartPieDiagram.cpp.

```
59 {  
60 }
```

### 9.44.3 Member Function Documentation

#### 9.44.3.1 bool AbstractDiagram::allowOverlappingDataValueTexts () const [inherited]

##### Returns:

Whether data value labels are allowed to overlap.

Definition at line 440 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::compare().

```
441 {  
442     return d->allowOverlappingDataValueTexts;  
443 }
```

#### 9.44.3.2 bool AbstractDiagram::antiAliasing () const [inherited]

##### Returns:

Whether anti-aliasing is to be used for rendering this diagram.

Definition at line 451 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::compare().

```
452 {  
453     return d->antiAliasing;  
454 }
```

#### 9.44.3.3 [AttributesModel](#) \* AbstractDiagram::attributesModel () const [virtual, inherited]

Returns the [AttributesModel](#), that is used by this diagram.

By default each diagram owns its own [AttributesModel](#), which should never be deleted. Only if a user-supplied [AttributesModel](#) has been set does the pointer returned here not belong to the diagram.

##### Returns:

The [AttributesModel](#) associated with the diagram.

##### See also:

[setAttributesModel](#)

Definition at line 280 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::brush(), KDChart::AbstractDiagram::compare(), KDChart::AbstractDiagram::datasetBrushes(), KDChart::AbstractDiagram::datasetLabels(), KDChart::AbstractDiagram::datasetMarkers(), KDChart::AbstractDiagram::datasetPens(), KDChart::AbstractDiagram::dataValueAttributes(), KDChart::AbstractDiagram::isHidden(), KDChart::AbstractDiagram::itemRowLabels(), KDChart::RingDiagram::paint(), KDChart::PolarDiagram::paint(), KDChart::AbstractDiagram::pen(), KDChart::AbstractCartesianDiagram::setAttributesModel(), KDChart::BarDiagram::setBarAttributes(), KDChart::AbstractDiagram::setBrush(), KDChart::AbstractCartesianDiagram::setModel(), KDChart::AbstractDiagram::setPen(), and KDChart::TernaryLineDiagram::TernaryLineDiagram().

```
281 {
282     return d->attributesModel;
283 }
```

#### 9.44.3.4 QModelIndex AbstractDiagram::attributesModelRootIndex () const [protected, inherited]

returns a QModelIndex pointing into the [AttributesModel](#) that corresponds to the root index of the diagram.

Definition at line 302 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::datasetBrushes(), KDChart::AbstractDiagram::datasetLabels(), KDChart::AbstractDiagram::datasetMarkers(), KDChart::AbstractDiagram::datasetPens(), KDChart::AbstractDiagram::itemRowLabels(), KDChart::Plotter::numberOfAbscissaSegments(), KDChart::LineDiagram::numberOfAbscissaSegments(), KDChart::BarDiagram::numberOfAbscissaSegments(), KDChart::Plotter::numberOfOrdinateSegments(), KDChart::LineDiagram::numberOfOrdinateSegments(), KDChart::BarDiagram::numberOfOrdinateSegments(), KDChart::AbstractDiagram::valueForCell(), and KDChart::LineDiagram::valueForCellTesting().

```
303 {
304     if ( !d->attributesModelRootIndex.isValid() )
305         d->attributesModelRootIndex = d->attributesModel->mapFromSource( rootIndex() );
306     return d->attributesModelRootIndex;
307 }
```

#### 9.44.3.5 QBrush AbstractDiagram::brush (const QModelIndex & index) const [inherited]

Retrieve the brush to be used, for painting the datapoint at the given index in the model.

##### Parameters:

*index* The index of the datapoint in the model.

##### Returns:

The brush to use for painting.

Definition at line 838 of file KDChartAbstractDiagram.cpp.

References `KDChart::AbstractDiagram::attributesModel()`, `KDChart::AbstractDiagram::brush()`, `KDChart::AttributesModel::data()`, `KDChart::DatasetBrushRole`, and `KDChart::AbstractDiagram::datasetDimension()`.

```

839 {
840     if( datasetDimension() > 1 )
841         return brush( index.column() );
842     return qVariantValue<QBrush>(
843         attributesModel()->data(
844             attributesModel()->mapFromSource( index ),
845             DatasetBrushRole ) );
846 }
```

#### 9.44.3.6 `QBrush AbstractDiagram::brush (int dataset) const` [inherited]

Retrieve the brush to be used for the given dataset.

This will fall back automatically to what was set at model level, if there are no dataset specific settings.

##### Parameters:

*dataset* The dataset to retrieve the brush for.

##### Returns:

The brush to use for painting.

Definition at line 828 of file `KDChartAbstractDiagram.cpp`.

References `KDChart::AbstractDiagram::attributesModel()`, `KDChart::AbstractDiagram::columnToIndex()`, `KDChart::AttributesModel::data()`, `KDChart::DatasetBrushRole`, and `KDChart::AbstractDiagram::datasetDimension()`.

```

829 {
830     if( datasetDimension() > 1 )
831         dataset /= datasetDimension();
832     return qVariantValue<QBrush>(
833         attributesModel()->data(
834             attributesModel()->mapFromSource( columnToIndex( dataset ) ),
835             DatasetBrushRole ) );
836 }
```

#### 9.44.3.7 `QBrush AbstractDiagram::brush () const` [inherited]

Retrieve the brush to be used for painting datapoints globally.

This will fall back automatically to the default settings, if there are no specific settings.

##### Returns:

The brush to use for painting.

Definition at line 822 of file `KDChartAbstractDiagram.cpp`.

References `KDChart::AbstractDiagram::attributesModel()`, `KDChart::AttributesModel::data()`, and `KDChart::DatasetBrushRole`.

Referenced by `KDChart::AbstractDiagram::brush()`, `KDChart::TernaryPointDiagram::paint()`, `KDChart::TernaryLineDiagram::paint()`, `KDChart::RingDiagram::paint()`, `KDChart::PolarDiagram::paint()`, and `KDChart::AbstractDiagram::paintMarker()`.



```

823 {
824     return qVariantValue<QBrush>(
825         attributesModel()->data( DatasetBrushRole ) );
826 }

```

#### 9.44.3.8 const QPair< QPointF, QPointF > PieDiagram::calculateDataBoundaries () const [protected, virtual]

[reimplemented]

Implements [KDChart::AbstractDiagram](#).

Definition at line 74 of file KDChartPieDiagram.cpp.

References [KDChart::AbstractDiagram::checkInvariants\(\)](#), [KDChart::AbstractPolarDiagram::columnCount\(\)](#), and [KDChart::AbstractPieDiagram::pieAttributes\(\)](#).

```

75 {
76     if ( !checkInvariants( true ) ) return QPair<QPointF, QPointF>( QPointF( 0, 0 ), QPointF( 0, 0 ) );
77
78     const PieAttributes attrs( pieAttributes( model()->index( 0, 0, rootIndex() ) ) );
79
80     QPointF bottomLeft ( QPointF( 0, 0 ) );
81     QPointF topRight;
82     // If we explode, we need extra space for the pie slice that has
83     // the largest explosion distance.
84     if ( attrs.explode() ) {
85         const int colCount = columnCount();
86         qreal maxExplode = 0.0;
87         for( int j = 0; j < colCount; ++j ){
88             const PieAttributes columnAttrs( pieAttributes( model()->index( 0, j, rootIndex() ) ) );
89             maxExplode = qMax( maxExplode, columnAttrs.explodeFactor() );
90         }
91         topRight = QPointF( 1.0+maxExplode, 1.0+maxExplode );
92     }else{
93         topRight = QPointF( 1.0, 1.0 );
94     }
95     return QPair<QPointF, QPointF> ( bottomLeft, topRight );
96 }

```

#### 9.44.3.9 bool AbstractDiagram::checkInvariants (bool *justReturnTheStatus* = false) const [protected, virtual, inherited]

Definition at line 1060 of file KDChartAbstractDiagram.cpp.

References [KDChart::AbstractDiagram::coordinatePlane\(\)](#).

Referenced by [KDChart::RingDiagram::calculateDataBoundaries\(\)](#), [KDChart::PolarDiagram::calculateDataBoundaries\(\)](#), [KDChart::Plotter::calculateDataBoundaries\(\)](#), [calculateDataBoundaries\(\)](#), [KDChart::LineDiagram::calculateDataBoundaries\(\)](#), [KDChart::BarDiagram::calculateDataBoundaries\(\)](#), [KDChart::RingDiagram::paint\(\)](#), [KDChart::PolarDiagram::paint\(\)](#), [KDChart::Plotter::paint\(\)](#), [paint\(\)](#), [KDChart::LineDiagram::paint\(\)](#), [KDChart::BarDiagram::paint\(\)](#), [KDChart::AbstractDiagram::paintDataValueTexts\(\)](#), [KDChart::AbstractDiagram::paintMarker\(\)](#), and [KDChart::AbstractDiagram::paintMarkers\(\)](#).

```

1061 {
1062     if( ! justReturnTheStatus ){
1063         Q_ASSERT_X ( model(), "AbstractDiagram::checkInvariants()",
1064             "There is no usable model set, for the diagram." );

```

```

1065
1066         Q_ASSERT_X ( coordinatePlane(), "AbstractDiagram::checkInvariants()",
1067                     "There is no usable coordinate plane set, for the diagram." );
1068     }
1069     return model() && coordinatePlane();
1070 }

```

#### 9.44.3.10 **PieDiagram \* PieDiagram::clone () const** [virtual]

Creates an exact copy of this diagram.

Definition at line 69 of file KDChartPieDiagram.cpp.

References d, and PieDiagram().

```

70 {
71     return new PieDiagram( new Private( *d ) );
72 }

```

#### 9.44.3.11 **int AbstractPolarDiagram::columnCount () const** [inherited]

Definition at line 60 of file KDChartAbstractPolarDiagram.cpp.

References KDChart::AbstractPolarDiagram::numberOfValuesPerDataset().

Referenced by calculateDataBoundaries(), paint(), and valueTotals().

```

61 {
62     return static_cast<int>( numberOfValuesPerDataset() );
63 }

```

#### 9.44.3.12 **QModelIndex AbstractDiagram::columnToIndex (int *column*) const** [protected, inherited]

Definition at line 309 of file KDChartAbstractDiagram.cpp.

Referenced by KDChart::BarDiagram::barAttributes(), KDChart::AbstractDiagram::brush(), KDChart::AbstractDiagram::dataValueAttributes(), KDChart::AbstractDiagram::isHidden(), KDChart::Plotter::lineAttributes(), KDChart::LineDiagram::lineAttributes(), KDChart::AbstractDiagram::pen(), KDChart::AbstractPieDiagram::pieAttributes(), KDChart::BarDiagram::threeDBarAttributes(), KDChart::Plotter::threeDLineAttributes(), KDChart::LineDiagram::threeDLineAttributes(), and KDChart::AbstractPieDiagram::threeDPieAttributes().

```

310 {    // FIXME (Mirko): shouldn't this be headerData? instead of the index for the first row?
311     if( model() )
312         return QModelIndex( model()->index( 0, column, rootIndex() ) );
313     return QModelIndex();
314 }

```

#### 9.44.3.13 **bool AbstractDiagram::compare (const **AbstractDiagram** \* *other*) const** [inherited]

Returns true if both diagrams have the same settings.

Definition at line 129 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::allowOverlappingDataValueTexts(), KDChart::AbstractDiagram::antiAliasing(), KDChart::AbstractDiagram::attributesModel(), KDChart::AttributesModel::compare(), KDChart::AbstractDiagram::datasetDimension(), and KDChart::AbstractDiagram::percentMode().

```

130 {
131     if( other == this ) return true;
132     if( ! other ){
133         //qDebug() << "AbstractDiagram::compare() cannot compare to Null pointer";
134         return false;
135     }
136     /*
137     qDebug() << "\n                AbstractDiagram::compare() QAbstractScrollArea:";
138         // compare QAbstractScrollArea properties
139     qDebug() <<
140         ((horizontalScrollBarPolicy() == other->horizontalScrollBarPolicy()) &&
141         (verticalScrollBarPolicy() == other->verticalScrollBarPolicy()));
142     qDebug() << "AbstractDiagram::compare() QFrame:";
143         // compare QFrame properties
144     qDebug() <<
145         ((frameShadow() == other->frameShadow()) &&
146         (frameShape() == other->frameShape()) &&
147         (frameWidth() == other->frameWidth()) &&
148         (lineWidth() == other->lineWidth()) &&
149         (midLineWidth() == other->midLineWidth()));
150     qDebug() << "AbstractDiagram::compare() QAbstractItemView:";
151         // compare QAbstractItemView properties
152     qDebug() <<
153         ((alternatingRowColors() == other->alternatingRowColors()) &&
154         (hasAutoScroll() == other->hasAutoScroll()) &&
155 #if QT_VERSION > 0x040199
156         (dragDropMode() == other->dragDropMode()) &&
157         (dragDropOverwriteMode() == other->dragDropOverwriteMode()) &&
158         (horizontalScrollMode() == other->horizontalScrollMode()) &&
159         (verticalScrollMode() == other->verticalScrollMode()) &&
160 #endif
161         (dragEnabled() == other->dragEnabled()) &&
162         (editTriggers() == other->editTriggers()) &&
163         (iconSize() == other->iconSize()) &&
164         (selectionBehavior() == other->selectionBehavior()) &&
165         (selectionMode() == other->selectionMode()) &&
166         (showDropIndicator() == other->showDropIndicator()) &&
167         (tabKeyNavigation() == other->tabKeyNavigation()) &&
168         (textElideMode() == other->textElideMode()));
169     qDebug() << "AbstractDiagram::compare() AttributesModel: ";
170         // compare all of the properties stored in the attributes model
171     qDebug() << attributesModel()->compare( other->attributesModel() );
172     qDebug() << "AbstractDiagram::compare() own:";
173         // compare own properties
174     qDebug() <<
175         ((rootIndex().column() == other->rootIndex().column()) &&
176         (rootIndex().row() == other->rootIndex().row()) &&
177         (allowOverlappingDataValueTexts() == other->allowOverlappingDataValueTexts()) &&
178         (antiAliasing() == other->antiAliasing()) &&
179         (percentMode() == other->percentMode()) &&
180         (datasetDimension() == other->datasetDimension()));
181     */
182     return // compare QAbstractScrollArea properties
183         (horizontalScrollBarPolicy() == other->horizontalScrollBarPolicy()) &&
184         (verticalScrollBarPolicy() == other->verticalScrollBarPolicy()) &&
185         // compare QFrame properties
186         (frameShadow() == other->frameShadow()) &&
187         (frameShape() == other->frameShape()) &&
188         // frameWidth is a read-only property defined by the style, it should not be in here:
189         (frameWidth() == other->frameWidth()) &&

```

```

190         (lineWidth()      == other->lineWidth()) &&
191         (midLineWidth()   == other->midLineWidth()) &&
192         // compare QAbstractItemView properties
193         (alternatingRowColors() == other->alternatingRowColors()) &&
194         (hasAutoScroll()    == other->hasAutoScroll()) &&
195 #if QT_VERSION > 0x040199
196         (dragDropMode()    == other->dragDropMode()) &&
197         (dragDropOverwriteMode() == other->dragDropOverwriteMode()) &&
198         (horizontalScrollMode() == other->horizontalScrollMode()) &&
199         (verticalScrollMode() == other->verticalScrollMode()) &&
200 #endif
201         (dragEnabled()      == other->dragEnabled()) &&
202         (editTriggers()     == other->editTriggers()) &&
203         (iconSize()        == other->iconSize()) &&
204         (selectionBehavior() == other->selectionBehavior()) &&
205         (selectionMode()    == other->selectionMode()) &&
206         (showDropIndicator() == other->showDropIndicator()) &&
207         (tabKeyNavigation() == other->tabKeyNavigation()) &&
208         (textElideMode()    == other->textElideMode()) &&
209         // compare all of the properties stored in the attributes model
210         attributesModel()->compare( other->attributesModel() ) &&
211         // compare own properties
212         (rootIndex().column() == other->rootIndex().column()) &&
213         (rootIndex().row()    == other->rootIndex().row()) &&
214         (allowOverlappingDataValueTexts() == other->allowOverlappingDataValueTexts()) &&
215         (antiAliasing()       == other->antiAliasing()) &&
216         (percentMode()        == other->percentMode()) &&
217         (datasetDimension()   == other->datasetDimension());
218 }

```

#### 9.44.3.14 **AbstractCoordinatePlane \* AbstractDiagram::coordinatePlane () const** [inherited]

The coordinate plane associated with the diagram.

This determines how coordinates in value space are mapped into pixel space. By default this is a [Cartesian-CoordinatePlane](#).

##### Returns:

The coordinate plane associated with the diagram.

Definition at line 220 of file `KDChartAbstractDiagram.cpp`.

References d.

Referenced by `KDChart::AbstractDiagram::checkInvariants()`, `KDChart::AbstractCartesianDiagram::layoutPlanes()`, `KDChart::PolarDiagram::paint()`, `KDChart::AbstractDiagram::paintDataValueTexts()`, `KDChart::AbstractDiagram::paintMarkers()`, `KDChart::AbstractPolarDiagram::polarCoordinatePlane()`, and `KDChart::AbstractCartesianDiagram::setCoordinatePlane()`.

```

221 {
222     return d->plane;
223 }

```

#### 9.44.3.15 **const QPair< QPointF, QPointF > AbstractDiagram::dataBoundaries () const** [inherited]

Return the bottom left and top right data point, that the diagram will display (unless the grid adjusts these values).

This method returns a cached result of calculations done by `calculateDataBoundaries`. Classes derived from [AbstractDiagram](#) must implement the `calculateDataBoundaries` function, to specify their own way of calculating the data boundaries. If derived classes want to force recalculation of the data boundaries, they can call [setDataBoundariesDirty\(\)](#)

Returned value is in diagram coordinates.

Definition at line 225 of file `KDChartAbstractDiagram.cpp`.

References `KDChart::AbstractDiagram::calculateDataBoundaries()`, and `d`.

Referenced by `KDChart::CartesianCoordinatePlane::getRawDataBoundingRectFromDiagrams()`, `KDChart::PolarCoordinatePlane::layoutDiagrams()`, `KDChart::Plotter::paint()`, `KDChart::LineDiagram::paint()`, and `KDChart::BarDiagram::paint()`.

```
226 {
227     if( d->databoundariesDirty ){
228         d->databoundaries = calculateDataBoundaries ();
229         d->databoundariesDirty = false;
230     }
231     return d->databoundaries;
232 }
```

#### 9.44.3.16 void AbstractDiagram::dataChanged (const QModelIndex & *topLeft*, const QModelIndex & *bottomRight*) [virtual, inherited]

[reimplemented]

Definition at line 330 of file `KDChartAbstractDiagram.cpp`.

References `KDChart::AbstractDiagram::setDataBoundariesDirty()`.

```
332 {
333     Q_UNUSED( topLeft );
334     Q_UNUSED( bottomRight );
335     // We are still too dumb to do intelligent updates...
336     setDataBoundariesDirty();
337     scheduleDelayedItemsLayout();
338 }
```

#### 9.44.3.17 void KDChart::AbstractDiagram::dataHidden () [signal, inherited]

This signal is emitted, when the hidden status of at least one data cell was (un)set.

Referenced by `KDChart::AbstractDiagram::setHidden()`.

#### 9.44.3.18 QList< QBrush > AbstractDiagram::datasetBrushes () const [inherited]

The set of dataset brushes currently used, for use in legends, etc.

##### Note:

Cell-level override brushes, if set, take precedence over the dataset values, so you might need to check these too, in order to find the brush, that is used for a single cell.

##### Returns:

The current set of dataset brushes.

Definition at line 1018 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::attributesModelRootIndex(), KDChart::AttributesModel::columnCount(), KDChart::DatasetBrushRole, and KDChart::AbstractDiagram::datasetDimension().

Referenced by KDChart::Legend::setBrushesFromDiagram().

```

1019 {
1020     QList<QBrush> ret;
1021     if( model() == 0 )
1022         return ret;
1023
1024     const int columnCount = attributesModel()->columnCount(attributesModelRootIndex());
1025     for( int i = 0; i < columnCount / datasetDimension(); ++i )
1026         ret << QVariantValue<QBrush>( attributesModel()->headerData( i, Qt::Vertical, DatasetBrushRole ),
1027
1028     return ret;
1029 }
```

#### 9.44.3.19 int AbstractDiagram::datasetDimension () const [inherited]

The dataset dimension of a diagram determines, how many value dimensions it expects each datapoint to have.

For each dimension it will expect one column of values in the model. If the dimensionality is 1, automatic values will be used for the abscissa.

For example a diagram with the default dimension of 1, will have one column per datapoint (the y values) and will use automatic values for the x axis (1, 2, 3, ... n). If the dimension is 2, the diagram will use the first, (and the third, fifth, etc) columns as X values, and the second, (and the fourth, sixth, etc) column as Y values.

##### Returns:

The dataset dimension of the diagram.

Definition at line 1072 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::brush(), KDChart::AbstractDiagram::compare(), KDChart::AbstractDiagram::datasetBrushes(), KDChart::AbstractDiagram::datasetLabels(), KDChart::AbstractDiagram::datasetMarkers(), KDChart::AbstractDiagram::datasetPens(), KDChart::LineDiagram::getCellValues(), KDChart::CartesianCoordinatePlane::getDataDimensionsList(), KDChart::TernaryPointDiagram::paint(), KDChart::TernaryLineDiagram::paint(), KDChart::AbstractDiagram::paintDataValueTexts(), KDChart::AbstractDiagram::paintMarkers(), KDChart::AbstractDiagram::pen(), KDChart::AbstractDiagram::setPen(), KDChart::Plotter::setType(), and KDChart::LineDiagram::setType().

```

1073 {
1074     return d->datasetDimension;
1075 }
```

#### 9.44.3.20 QStringList AbstractDiagram::datasetLabels () const [inherited]

The set of dataset labels currently displayed, for use in legends, etc.

**Returns:**

The set of dataset labels currently displayed.

Definition at line 1005 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::attributesModelRootIndex(), KDChart::AttributesModel::columnCount(), KDChart::AbstractDiagram::datasetDimension(), and KDChart::AttributesModel::headerData().

```

1006 {
1007     QStringList ret;
1008     if( model() == 0 )
1009         return ret;
1010
1011     const int columnCount = attributesModel()->columnCount(attributesModelRootIndex());
1012     for( int i = 0; i < columnCount / datasetDimension(); ++i )
1013         ret << attributesModel()->headerData( i, Qt::Horizontal, Qt::DisplayRole ).toString();
1014
1015     return ret;
1016 }
```

#### 9.44.3.21 QList< [MarkerAttributes](#) > AbstractDiagram::datasetMarkers () const

[inherited]

The set of dataset markers currently used, for use in legends, etc.

**Note:**

Cell-level override markers, if set, take precedence over the dataset values, so you might need to check these too, in order to find the marker, that is shown for a single cell.

**Returns:**

The current set of dataset brushes.

Definition at line 1044 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::attributesModelRootIndex(), KDChart::AttributesModel::columnCount(), KDChart::AbstractDiagram::datasetDimension(), and KDChart::DataValueLabelAttributesRole.

```

1045 {
1046     QList<MarkerAttributes> ret;
1047     if( model() == 0 )
1048         return ret;
1049
1050     const int columnCount = attributesModel()->columnCount(attributesModelRootIndex());
1051     for( int i = 0; i < columnCount / datasetDimension(); ++i )
1052     {
1053         const DataValueAttributes a =
1054             qVariantValue<DataValueAttributes>( attributesModel()->headerData( i, Qt::Vertical, DataValueLabelAttributesRole ) );
1055         ret << a.markerAttributes();
1056     }
1057     return ret;
1058 }
```

**9.44.3.22 QList< QPen > AbstractDiagram::datasetPens () const** [inherited]

The set of dataset pens currently used, for use in legends, etc.

**Note:**

Cell-level override pens, if set, take precedence over the dataset values, so you might need to check these too, in order to find the pens, that is used for a single cell.

**Returns:**

The current set of dataset pens.

Definition at line 1031 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::attributesModelRootIndex(), KDChart::AttributesModel::columnCount(), KDChart::AbstractDiagram::datasetDimension(), and KDChart::DatasetPenRole.

```

1032 {
1033     QList<QPen> ret;
1034     if( model() == 0 )
1035         return ret;
1036
1037     const int columnCount = attributesModel()->columnCount(attributesModelRootIndex());
1038     for( int i = 0; i < columnCount / datasetDimension(); ++i )
1039         ret << qVariantValue<QPen>( attributesModel()->headerData( i, Qt::Vertical, DatasetPenRole ) );
1040
1041     return ret;
1042 }
```

**9.44.3.23 DataValueAttributes AbstractDiagram::dataValueAttributes (const QModelIndex & index) const** [inherited]

Retrieve the [DataValueAttributes](#) for the given index.

This will fall back automatically to what was set at dataset or model level, if there are no datapoint specific settings.

**Parameters:**

*index* The datapoint to retrieve the attributes for.

**Returns:**

The [DataValueAttributes](#) for the given index.

Definition at line 421 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AttributesModel::data(), and KDChart::DataValueLabelAttributesRole.

```

422 {
423     return qVariantValue<DataValueAttributes>(
424         attributesModel()->data( attributesModel()->mapFromSource(index),
425                                 KDChart::DataValueLabelAttributesRole ) );
426 }
```



### 9.44.3.24 [DataValueAttributes](#) AbstractDiagram::dataValueAttributes (int *column*) const [inherited]

Retrieve the [DataValueAttributes](#) for the given dataset.

This will fall back automatically to what was set at model level, if there are no dataset specific settings.

#### Parameters:

*column* The dataset to retrieve the attributes for.

#### Returns:

The [DataValueAttributes](#) for the given dataset.

Definition at line 414 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::columnToIndex(), KDChart::AttributesModel::data(), and KDChart::DataValueLabelAttributesRole.

```
415 {
416     return qVariantValue<DataValueAttributes>(
417         attributesModel()->data( attributesModel()->mapFromSource(columnToIndex( column )),
418         KDChart::DataValueLabelAttributesRole ) );
419 }
```

### 9.44.3.25 [DataValueAttributes](#) AbstractDiagram::dataValueAttributes () const [inherited]

Retrieve the [DataValueAttributes](#) specified globally.

This will fall back automatically to the default settings, if there are no specific settings.

#### Returns:

The global [DataValueAttributes](#).

Definition at line 408 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DataValueLabelAttributesRole, and KDChart::AttributesModel::modelData().

Referenced by KDChart::AbstractDiagram::paintDataValueText(), KDChart::AbstractDiagram::paintMarker(), and KDChart::TernaryLineDiagram::TernaryLineDiagram().

```
409 {
410     return qVariantValue<DataValueAttributes>(
411         attributesModel()->modelData( KDChart::DataValueLabelAttributesRole ) );
412 }
```

### 9.44.3.26 void AbstractDiagram::doItemsLayout () [virtual, inherited]

[reimplemented]

Definition at line 321 of file KDChartAbstractDiagram.cpp.

References d, and KDChart::AbstractDiagram::update().

```

322 {
323     if ( d->plane ) {
324         d->plane->layoutDiagrams();
325         update();
326     }
327     QAbstractItemView::doItemsLayout();
328 }

```

#### 9.44.3.27 qreal AbstractPieDiagram::granularity () const [inherited]

##### Returns:

the granularity.

Definition at line 69 of file KDChartAbstractPieDiagram.cpp.

References d.

Referenced by paint().

```

70 {
71     return (d->granularity < 0.05 || d->granularity > 36.0)
72           ? 1.0
73           : d->granularity;
74 }

```

#### 9.44.3.28 int AbstractDiagram::horizontalOffset () const [virtual, inherited]

[reimplemented]

Definition at line 950 of file KDChartAbstractDiagram.cpp.

```

951 { return 0; }

```

#### 9.44.3.29 QModelIndex AbstractDiagram::indexAt (const QPoint & point) const [virtual, inherited]

[reimplemented]

Definition at line 1098 of file KDChartAbstractDiagram.cpp.

References d.

```

1099 {
1100     return d->indexAt( point );
1101 }

```

#### 9.44.3.30 QModelIndexList AbstractDiagram::indexesAt (const QPoint & point) const [inherited]

This method is added alongside with indexAt from QAIM, since in kdchart multiple indexes can be displayed at the same spot.

Definition at line 1103 of file KDChartAbstractDiagram.cpp.

References d.

```

1104 {
1105     return d->indexesAt( point );
1106 }

```

#### 9.44.3.31 bool AbstractDiagram::isHidden (const QModelIndex & *index*) const [inherited]

Retrieve the hidden status for the given index.

This will fall back automatically to what was set at dataset or diagram level, if there are no datapoint specific settings.

##### Parameters:

***index*** The datapoint to retrieve the hidden status for.

##### Returns:

The hidden status for the given index.

Definition at line 380 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AttributesModel::data(), and KDChart::DataHiddenRole.

```

381 {
382     return qVariantValue<bool>(
383         attributesModel()->data(
384             attributesModel()->mapFromSource(index),
385             DataHiddenRole ) );
386 }

```

#### 9.44.3.32 bool AbstractDiagram::isHidden (int *column*) const [inherited]

Retrieve the hidden status for the given dataset.

This will fall back automatically to what was set at diagram level, if there are no dataset specific settings.

##### Parameters:

***column*** The dataset to retrieve the hidden status for.

##### Returns:

The hidden status for the given dataset.

Definition at line 373 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::columnToIndex(), KDChart::AttributesModel::data(), and KDChart::DataHiddenRole.

```

374 {
375     return qVariantValue<bool>(
376         attributesModel()->data(
377             attributesModel()->mapFromSource(columnToIndex( column ) ),
378             DataHiddenRole ) );
379 }

```

**9.44.3.33 bool AbstractDiagram::isHidden () const** [inherited]

Retrieve the hidden status specified globally.

This will fall back automatically to the default settings (= not hidden), if there are no specific settings.

**Returns:**

The global hidden status.

Definition at line 367 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DataHiddenRole, and KDChart::AttributesModel::modelData().

Referenced by KDChart::LineDiagram::valueForCellTesting().

```

368 {
369     return qVariantValue<bool>(
370         attributesModel()->modelData( DataHiddenRole ) );
371 }
```

**9.44.3.34 bool AbstractDiagram::isIndexHidden (const QModelIndex & index) const**  
[virtual, inherited]

[reimplemented]

Definition at line 956 of file KDChartAbstractDiagram.cpp.

```

957 { return true; }
```

**9.44.3.35 QStringList AbstractDiagram::itemRowLabels () const** [inherited]

The set of item row labels currently displayed, for use in Abscissa axes, etc.

**Returns:**

The set of item row labels currently displayed.

Definition at line 989 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::attributesModelRootIndex(), KDChart::AttributesModel::rowCount(), KDChart::AbstractDiagram::unitPrefix(), and KDChart::AbstractDiagram::unitSuffix().

```

990 {
991     QStringList ret;
992     if( model() ){
993         //qDebug() << "AbstractDiagram::itemRowLabels(): " << attributesModel()->rowCount(attributesModelRootIndex());
994         const int rowCount = attributesModel()->rowCount(attributesModelRootIndex());
995         for( int i = 0; i < rowCount; ++i ){
996             //qDebug() << "item row label: " << attributesModel()->headerData( i, Qt::Vertical, Qt::DisplayRole ).toString();
997             ret << unitPrefix( i, Qt::Horizontal, true ) +
998                 attributesModel()->headerData( i, Qt::Vertical, Qt::DisplayRole ).toString() +
999                 unitSuffix( i, Qt::Horizontal, true );
1000         }
1001     }
1002     return ret;
1003 }
```

**9.44.3.36** `void KDChart::AbstractDiagram::layoutChanged (AbstractDiagram *)` [signal, inherited]

Diagrams are supposed to emit this signal, when the layout of one of their element changes.

Layouts can change, for example, when axes are added or removed, or when the configuration was changed in a way that the axes or the diagram itself are displayed in a different geometry. Changes in the diagrams coordinate system also result in the [layoutChanged\(\)](#) signal being emitted.

Referenced by `KDChart::AbstractDiagram::setDatasetDimension()`, `KDChart::AbstractPieDiagram::setPieAttributes()`, `KDChart::AbstractPieDiagram::setThreeDPieAttributes()`, `KDChart::Plotter::setType()`, `KDChart::LineDiagram::setType()`, and `KDChart::BarDiagram::setType()`.

**9.44.3.37** `void KDChart::AbstractDiagram::modelsChanged ()` [signal, inherited]

This signal is emitted, when either the model or the [AttributesModel](#) is replaced.

Referenced by `KDChart::AbstractDiagram::setAttributesModel()`, and `KDChart::AbstractDiagram::setModel()`.

**9.44.3.38** `QModelIndex AbstractDiagram::moveCursor (CursorAction cursorAction, Qt::KeyboardModifiers modifiers)` [virtual, inherited]

[reimplemented]

Definition at line 947 of file `KDChartAbstractDiagram.cpp`.

```
948 { return QModelIndex(); }
```

**9.44.3.39** `double PieDiagram::numberOfGridRings () const` [virtual]

[reimplemented]

Implements [KDChart::AbstractPolarDiagram](#).

Definition at line 987 of file `KDChartPieDiagram.cpp`.

```
988 {
989     return 1;
990 }
```

**9.44.3.40** `double PieDiagram::numberOfValuesPerDataset () const` [virtual]

[reimplemented]

Implements [KDChart::AbstractPolarDiagram](#).

Definition at line 981 of file `KDChartPieDiagram.cpp`.

```
982 {
983     return model() ? model()->columnCount( rootIndex() ) : 0.0;
984 }
```

**9.44.3.41 void PieDiagram::paint (PaintContext \* paintContext)** [protected, virtual]

[reimplemented]

Implements [KDChart::AbstractDiagram](#).

Definition at line 172 of file KDChartPieDiagram.cpp.

References [buildReferenceRect\(\)](#), [KDChart::AbstractDiagram::checkInvariants\(\)](#), [KDChart::AbstractPolarDiagram::columnCount\(\)](#), [d](#), [KDChart::AbstractThreeDAttributes::depth\(\)](#), [KDChart::AbstractPieDiagram::granularity\(\)](#), [KDChart::AbstractThreeDAttributes::isEnabled\(\)](#), [KDChart::PaintContext::painter\(\)](#), [KDChart::AbstractPieDiagram::pieAttributes\(\)](#), [KDChart::AbstractPolarDiagram::polarCoordinatePlane\(\)](#), [KDChart::PaintContext::rectangle\(\)](#), [KDChart::AbstractPieDiagram::threeDPieAttributes\(\)](#), and [valueTotals\(\)](#).

Referenced by [paintEvent\(\)](#).

```

173 {
174     // note: Not having any data model assigned is no bug
175     //       but we can not draw a diagram then either.
176     if ( !checkInvariants(true) )
177         return;
178
179     const PieAttributes attrs( pieAttributes() );
180     const ThreeDPieAttributes threeDAttrs( threeDPieAttributes( model()->index( 0, 0, rootIndex() ) ) );
181
182     const int colCount = columnCount();
183
184     QRectF contentsRect( buildReferenceRect( polarCoordinatePlane() ) );
185     contentsRect = ctx->rectangle();
186     // contentsRect = geometry();
187     // qDebug() << contentsRect;
188     if( contentsRect.isEmpty() )
189         return;
190
191     DataValueTextInfoList list;
192     const qreal sum = valueTotals();
193
194     if( sum == 0.0 ) //nothing to draw
195         return;
196
197     d->startAngles.resize( colCount );
198     d->angleLens.resize( colCount );
199
200     // compute position
201     d->size = qMin( contentsRect.width(), contentsRect.height() ); // initial size
202
203     // if the pies explode, we need to give them additional space =>
204     // make the basic size smaller
205     qreal maxExplode = 0.0;
206     for( int j = 0; j < colCount; ++j ){
207         const PieAttributes columnAttrs( pieAttributes( model()->index( 0, j, rootIndex() ) ) );
208         maxExplode = qMax( maxExplode, columnAttrs.explodeFactor() );
209     }
210     d->size /= ( 1.0 + 2.0 * maxExplode );
211
212     qreal sizeFor3DEffect = 0.0;
213     if ( ! threeDAttrs.isEnabled() ) {
214
215         qreal x = ( contentsRect.width() == d->size ) ? 0.0 : ( ( contentsRect.width() - d->size ) / 2
216         qreal y = ( contentsRect.height() == d->size ) ? 0.0 : ( ( contentsRect.height() - d->size ) / 2
217         d->position = QRectF( x, y, d->size, d->size );
218         d->position.translate( contentsRect.left(), contentsRect.top() );
219     } else {
220         // threeD: width is the maximum possible width; height is 1/2 of that

```

```

222     qreal x = ( contentsRect.width() == d->size ) ? 0.0 : ( ( contentsRect.width() - d->size ) / 2
223     qreal height = d->size;
224     // make sure that the height plus the threeDheight is not more than the
225     // available size
226     if ( threeDAttrs.depth() >= 0.0 ) {
227         // positive pie height: absolute value
228         sizeFor3DEffect = threeDAttrs.depth();
229         height = d->size - sizeFor3DEffect;
230     } else {
231         // negative pie height: relative value
232         sizeFor3DEffect = - threeDAttrs.depth() / 100.0 * height;
233         height = d->size - sizeFor3DEffect;
234     }
235     qreal y = ( contentsRect.height() == height ) ? 0.0 : ( ( contentsRect.height() - height - siz
236
237     d->position = QRectF( contentsRect.left() + x, contentsRect.top() + y,
238         d->size, height );
239     // d->position.moveBy( contentsRect.left(), contentsRect.top() );
240 }
241
242 const PolarCoordinatePlane * plane = polarCoordinatePlane();
243 const qreal sectorsPerValue = 360.0 / sum;
244 qreal currentValue = plane ? plane->startPosition() : 0.0;
245
246 bool atLeastOneValue = false; // guard against completely empty tables
247 QVariant vValY;
248 for ( int iColumn = 0; iColumn < colCount; ++iColumn ) {
249     // is there anything at all at this column?
250     bool bOK;
251     const double cellValue = qAbs( model()->data( model()->index( 0, iColumn, rootIndex() ) )
252         .toDouble( &bOK ) );
253
254     if( bOK ){
255         d->startAngles[ iColumn ] = currentValue;
256         d->angleLens[ iColumn ] = cellValue * sectorsPerValue;
257         atLeastOneValue = true;
258     } else { // mark as non-existent
259         d->angleLens[ iColumn ] = 0.0;
260         if ( iColumn > 0.0 )
261             d->startAngles[ iColumn ] = d->startAngles[ iColumn - 1 ];
262         else
263             d->startAngles[ iColumn ] = currentValue;
264     }
265     //qDebug() << "d->startAngles["<<iColumn<<" == " << d->startAngles[ iColumn ]
266     // << " + d->angleLens["<<iColumn<<" << d->angleLens[ iColumn ]
267     // << " = " << d->startAngles[ iColumn ]+d->angleLens[ iColumn ];
268
269     currentValue = d->startAngles[ iColumn ] + d->angleLens[ iColumn ];
270 }
271
272 // If there was no value at all, bail out, to avoid endless loops
273 // later on (e.g. in findPieAt()).
274 if( ! atLeastOneValue )
275     return;
276
277
278 // Find the backmost pie which is at +90° and needs to be drawn
279 // first
280 int backmostpie = findPieAt( 90, colCount );
281 // Find the frontmost pie (at -90°/+270°) that should be drawn last
282 int frontmostpie = findPieAt( 270, colCount );
283 // the right- and the leftmost (only needed in some special cases...)
284 int rightmostpie = findPieAt( 0, colCount );
285 int leftmostpie = findPieAt( 180, colCount );
286
287
288 int currentLeftPie = backmostpie;

```

```

289     int currentRightPie = backmostpie;
290
291     drawOnePie( ctx->painter(), 0, backmostpie, granularity(), sizeFor3DEffect );
292
293     if( backmostpie == frontmostpie )
294     {
295         if( backmostpie == leftmostpie )
296             currentLeftPie = findLeftPie( currentLeftPie, colCount );
297         if( backmostpie == rightmostpie )
298             currentRightPie = findRightPie( currentRightPie, colCount );
299     }
300     while( currentLeftPie != frontmostpie )
301     {
302         if( currentLeftPie != backmostpie )
303             drawOnePie( ctx->painter(), 0, currentLeftPie, granularity(), sizeFor3DEffect );
304         currentLeftPie = findLeftPie( currentLeftPie, colCount );
305     }
306     while( currentRightPie != frontmostpie )
307     {
308         if( currentRightPie != backmostpie )
309             drawOnePie( ctx->painter(), 0, currentRightPie, granularity(), sizeFor3DEffect );
310         currentRightPie = findRightPie( currentRightPie, colCount );
311     }
312
313     // if the backmost pie is not the frontmost pie, we draw the frontmost at last
314     if( backmostpie != frontmostpie || ! threeDPieAttributes().isEnabled() )
315     {
316         drawOnePie( ctx->painter(), 0, frontmostpie, granularity(), sizeFor3DEffect );
317         // otherwise, this gets a bit more complicated...
318     /* } else if( threeDPieAttributes().isEnabled() ) {
319         //drawPieSurface( ctx->painter(), 0, frontmostpie, granularity() );
320         const QModelIndex index = model()->index( 0, frontmostpie, rootIndex() );
321         QPen pen = this->pen( index );
322         ctx->painter()->setBrush( brush( index ) );
323         if ( threeDAttrs.isEnabled() )
324             pen.setColor( QColor( 0, 0, 0 ) );
325         ctx->painter()->setPen( pen );
326
327         qreal startAngle = d->startAngles[ frontmostpie ];
328         if( startAngle > 360 )
329             startAngle -= 360;
330
331         qreal endAngle = startAngle + d->angleLens[ frontmostpie ];
332         startAngle = qMax( startAngle, 180.0 );
333
334         drawArcEffectSegment( ctx->painter(), piePosition( 0, frontmostpie),
335             sizeFor3DEffect, startAngle, endAngle, granularity() );*/
336     }
337 }

```

#### 9.44.3.42 void AbstractDiagram::paintDataValueText (QPainter \*painter, const QModelIndex &index, const QPointF &pos, double value) [inherited]

Definition at line 468 of file KDChartAbstractDiagram.cpp.

References `d`, `KDChart::DataValueAttributes::dataLabel()`, `KDChart::AbstractDiagram::dataValueAttributes()`, `KDChart::DataValueAttributes::decimalDigits()`, `KDChart::DataValueAttributes::isVisible()`, `KDChartEnums::MeasureOrientationMinimum`, `KDChart::DataValueAttributes::position()`, `KDChart::DataValueAttributes::prefix()`, `KDChart::DataValueAttributes::suffix()`, and `KDChart::DataValueAttributes::textAttributes()`.

Referenced by `KDChart::RingDiagram::paint()`, `KDChart::PolarDiagram::paint()`, and `KDChart::AbstractDiagram::paintDataValueTexts()`.



```

472 {
473     // paint one data series
474     const DataValueAttributes a( dataValueAttributes(index) );
475     if ( !a.isVisible() ) return;
476
477     // handle decimal digits
478     int decimalDigits = a.decimalDigits();
479     int decimalPos = QString::number( value ).indexOf( QLatin1Char( '.' ) );
480     QString roundedValue;
481     if ( a.dataLabel().isNull() ) {
482         if ( decimalPos > 0 && value != 0 )
483             roundedValue = roundValues ( value, decimalPos, decimalDigits );
484         else
485             roundedValue = QString::number( value );
486     } else
487         roundedValue = a.dataLabel();
488     // handle prefix and suffix
489     if ( !a.prefix().isNull() )
490         roundedValue.prepend( a.prefix() );
491
492     if ( !a.suffix().isNull() )
493         roundedValue.append( a.suffix() );
494
495     const TextAttributes ta( a.textAttributes() );
496     // FIXME draw the non-text bits, background, etc
497     if ( ta.isVisible() ) {
498
499         QPointF pt( pos );
500         /* for debugging:
501         PainterSaver painterSaver( painter );
502         painter->setPen( Qt::black );
503         painter->drawLine( pos - QPointF( 1,1), pos + QPointF( 1,1) );
504         painter->drawLine( pos - QPointF(-1,1), pos + QPointF(-1,1) );
505         */
506
507         QTextDocument doc;
508         if( Qt::mightBeRichText( roundedValue ) )
509             doc.setHtml( roundedValue );
510         else
511             doc.setPlainText( roundedValue );
512
513         const RelativePosition relPos( a.position( value >= 0.0 ) );
514         const Qt::Alignment alignBottomLeft = Qt::AlignBottom | Qt::AlignLeft;
515         const QFont calculatedFont( ta.calculatedFont( d->plane, KDChartEnums::MeasureOrientationMinimum ) );
516         const QRectF boundRect( d->cachedFontMetrics( calculatedFont, painter->device() )->boundingRect( roundedValue, calculatedFont ) );
517
518         // To place correctly
519         pt.ry() -= boundRect.height();
520
521         // qDebug() << "calculatedFont's point size:" << calculatedFont.pointSizeF();
522         // adjust the text start point position, if alignment is not Bottom/Left
523         if( (relPos.alignment() & alignBottomLeft) != alignBottomLeft ){
524             if( relPos.alignment() & Qt::AlignRight )
525                 pt.rx() -= boundRect.width();
526             else if( relPos.alignment() & Qt::AlignHCenter )
527                 pt.rx() -= 0.5 * boundRect.width();
528
529             if( relPos.alignment() & Qt::AlignTop )
530                 pt.ry() += boundRect.height();
531             else if( relPos.alignment() & Qt::AlignVCenter )
532                 pt.ry() += 0.5 * boundRect.height();
533         }
534
535         // FIXME draw the non-text bits, background, etc
536
537         if ( a.showRepetitiveDataLabels() ||
538             pos.x() <= d->lastX ||

```

```

539         d->lastRoundedValue != roundedValue ) {
540             d->lastRoundedValue = roundedValue;
541             d->lastX = pos.x();
542             PainterSaver painterSaver( painter );
543
544             doc.setDefaultFont( calculatedFont );
545             QAbstractTextDocumentLayout::PaintContext context;
546             context.palette = palette();
547             context.palette.setColor( QPalette::Text, ta.pen().color() );
548
549             painter->translate( pt );
550             painter->rotate( ta.rotation() );
551
552             QAbstractTextDocumentLayout* layout = doc.documentLayout();
553             layout->draw( painter, context );
554         }
555     }
556 }

```

#### 9.44.3.43 void AbstractDiagram::paintDataValueTexts (QPainter \*painter) [protected, virtual, inherited]

Definition at line 584 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::checkInvariants(), KDChart::AbstractDiagram::coordinatePlane(), KDChart::AbstractDiagram::datasetDimension(), KDChart::AbstractDiagram::paintDataValueText(), and KDChart::AbstractCoordinatePlane::translate().

```

585 {
586     if ( !checkInvariants() ) return;
587     const int rowCount = model()->rowCount( rootIndex() );
588     const int columnCount = model()->columnCount( rootIndex() );
589     for ( int i=datasetDimension()-1; i<columnCount; i += datasetDimension() ) {
590         for ( int j=0; j< rowCount; ++j ) {
591             const QModelIndex index = model()->index( j, i, rootIndex() );
592             double value = model()->data( index ).toDouble();
593             const QPointF pos = coordinatePlane()->translate( QPointF( j, value ) );
594             paintDataValueText( painter, index, pos, value );
595         }
596     }
597 }

```

#### 9.44.3.44 void PieDiagram::paintEvent (QPaintEvent \*) [protected]

Definition at line 99 of file KDChartPieDiagram.cpp.

References paint(), KDChart::PaintContext::setPainter(), and KDChart::PaintContext::setRectangle().

```

100 {
101     QPainter painter ( viewport() );
102     PaintContext ctx;
103     ctx.setPainter ( &painter );
104     ctx.setRectangle( QRectF ( 0, 0, width(), height() ) );
105     paint ( &ctx );
106 }

```

### 9.44.3.45 void AbstractDiagram::paintMarker (QPainter \* *painter*, const QModelIndex & *index*, const QPointF & *pos*) [inherited]

Definition at line 600 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::brush(), KDChart::AbstractDiagram::checkInvariants(), d, KDChart::AbstractDiagram::dataValueAttributes(), KDChart::MarkerAttributes::isVisible(), KDChart::DataValueAttributes::isVisible(), KDChart::DataValueAttributes::markerAttributes(), KDChart::MarkerAttributes::markerColor(), KDChart::MarkerAttributes::markerSize(), KDChart::AbstractDiagram::paintMarker(), and KDChart::MarkerAttributes::pen().

```

603 {
604     if ( !checkInvariants() ) return;
605     DataValueAttributes a = dataValueAttributes(index);
606     if ( !a.isVisible() ) return;
607     const MarkerAttributes &ma = a.markerAttributes();
608     if ( !ma.isVisible() ) return;
609
610     PainterSaver painterSaver( painter );
611     QSizeF maSize( ma.markerSize() );
612     QBrush indexBrush( brush( index ) );
613     QPen indexPen( ma.pen() );
614     if ( ma.markerColor().isValid() )
615         indexBrush.setColor( ma.markerColor() );
616
617     paintMarker( painter, ma, indexBrush, indexPen, pos, maSize );
618
619     // workaround: BC cannot be changed, otherwise we would pass the
620     // index down to next-lower paintMarker function. So far, we
621     // basically save a circle of radius maSize at pos in the
622     // reverseMapper. This means that ^^^ this version of paintMarker
623     // needs to be called to reverse-map the marker.
624     d->reverseMapper.addCircle( index.row(), index.column(), pos, 2 * maSize );
625 }
```

### 9.44.3.46 void AbstractDiagram::paintMarker (QPainter \* *painter*, const MarkerAttributes & *markerAttributes*, const QBrush & *brush*, const QPen & *pen*, const QPointF & *point*, const QSizeF & *size*) [virtual, inherited]

Definition at line 627 of file KDChartAbstractDiagram.cpp.

References KDChart::MarkerAttributes::Marker1Pixel, KDChart::MarkerAttributes::Marker4Pixels, KDChart::MarkerAttributes::MarkerCircle, KDChart::MarkerAttributes::MarkerCross, KDChart::MarkerAttributes::MarkerDiamond, KDChart::MarkerAttributes::MarkerFastCross, KDChart::MarkerAttributes::MarkerRing, KDChart::MarkerAttributes::MarkerSquare, and KDChart::MarkerAttributes::markerStyle().

Referenced by KDChart::TernaryPointDiagram::paint(), KDChart::TernaryLineDiagram::paint(), KDChart::MarkerLayoutItem::paintIntoRect(), KDChart::AbstractDiagram::paintMarker(), and KDChart::AbstractDiagram::paintMarkers().

```

633 {
634     const QPen oldPen( painter->pen() );
635     // Pen is used to paint 4Pixels - 1 Pixel - Ring and FastCross types.
636     // make sure to use the brush color - see above in those cases.
637     const bool isFourPixels = (markerAttributes.markerStyle() == MarkerAttributes::Marker4Pixels);
638     if( isFourPixels || (markerAttributes.markerStyle() == MarkerAttributes::Marker1Pixel) ){
639         // for high-performance point charts with tiny point markers:
640         painter->setPen( QPen( brush.color().light() ) );
641         if( isFourPixels ){
```

```

642         const qreal x = pos.x();
643         const qreal y = pos.y();
644         painter->drawLine( QPointF(x-1.0,y-1.0),
645                           QPointF(x+1.0,y-1.0) );
646         painter->drawLine( QPointF(x-1.0,y),
647                           QPointF(x+1.0,y) );
648         painter->drawLine( QPointF(x-1.0,y+1.0),
649                           QPointF(x+1.0,y+1.0) );
650     }
651     painter->drawPoint( pos );
652 }else{
653     PainterSaver painterSaver( painter );
654     // we only a solid line surrounding the markers
655     QPen painterPen( pen );
656     painterPen.setStyle( Qt::SolidLine );
657     painter->setPen( painterPen );
658     painter->setBrush( brush );
659     painter->setRenderHint ( QPainter::Antialiasing );
660     painter->translate( pos );
661     switch ( markerAttributes.markerStyle() ) {
662     case MarkerAttributes::MarkerCircle:
663         painter->drawEllipse( QRectF( 0 - maSize.height()/2, 0 - maSize.width()/2,
664                                     maSize.height(), maSize.width() ) );
665         break;
666     case MarkerAttributes::MarkerSquare:
667     {
668         QRectF rect( 0 - maSize.width()/2, 0 - maSize.height()/2,
669                     maSize.width(), maSize.height() );
670         painter->drawRect( rect );
671         break;
672     }
673     case MarkerAttributes::MarkerDiamond:
674     {
675         QVector <QPointF > diamondPoints;
676         QPointF top, left, bottom, right;
677         top = QPointF( 0, 0 - maSize.height()/2 );
678         left = QPointF( 0 - maSize.width()/2, 0 );
679         bottom = QPointF( 0, maSize.height()/2 );
680         right = QPointF( maSize.width()/2, 0 );
681         diamondPoints << top << left << bottom << right;
682         painter->drawPolygon( diamondPoints );
683         break;
684     }
685     // both handled on top of the method:
686     case MarkerAttributes::Marker1Pixel:
687     case MarkerAttributes::Marker4Pixels:
688         break;
689     case MarkerAttributes::MarkerRing:
690     {
691         painter->setPen( QPen( brush.color() ) );
692         painter->setBrush( Qt::NoBrush );
693         painter->drawEllipse( QRectF( 0 - maSize.height()/2, 0 - maSize.width()/2,
694                                     maSize.height(), maSize.width() ) );
695         break;
696     }
697     case MarkerAttributes::MarkerCross:
698     {
699         QRectF rect( maSize.width()*-0.5, maSize.height()*-0.2,
700                     maSize.width(), maSize.height()*0.4 );
701         painter->drawRect( rect );
702         rect.setTopLeft( QPointF( maSize.width()*-0.2, maSize.height()*-0.5 ) );
703         rect.setSize( QSizeF( maSize.width()*0.4, maSize.height() ) );
704         painter->drawRect( rect );
705         break;
706     }
707     case MarkerAttributes::MarkerFastCross:
708     {

```

```

709             QPointF left, right, top, bottom;
710             left = QPointF( -maSize.width()/2, 0 );
711             right = QPointF( maSize.width()/2, 0 );
712             top = QPointF( 0, -maSize.height()/2 );
713             bottom= QPointF( 0, maSize.height()/2 );
714             painter->setPen( QPen( brush.color() ) );
715             painter->drawLine( left, right );
716             painter->drawLine( top, bottom );
717             break;
718         }
719         default:
720             Q_ASSERT_X ( false, "paintMarkers()",
721                 "Type item does not match a defined Marker Type." );
722     }
723 }
724 painter->setPen( oldPen );
725 }

```

#### 9.44.3.47 void AbstractDiagram::paintMarkers (QPainter \* *painter*) [protected, virtual, inherited]

Definition at line 727 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::checkInvariants(), KDChart::AbstractDiagram::coordinatePlane(), KDChart::AbstractDiagram::datasetDimension(), KDChart::AbstractDiagram::paintMarker(), and KDChart::AbstractCoordinatePlane::translate().

```

728 {
729     if ( !checkInvariants() ) return;
730     const int rowCount = model()->rowCount( rootIndex() );
731     const int columnCount = model()->columnCount( rootIndex() );
732     for ( int i=datasetDimension()-1; i<columnCount; i += datasetDimension() ) {
733         for ( int j=0; j<rowCount; ++j ) {
734             const QModelIndex index = model()->index( j, i, rootIndex() );
735             double value = model()->data( index ).toDouble();
736             const QPointF pos = coordinatePlane()->translate( QPointF( j, value ) );
737             paintMarker( painter, index, pos );
738         }
739     }
740 }

```

#### 9.44.3.48 QPen AbstractDiagram::pen (const QModelIndex & *index*) const [inherited]

Retrieve the pen to be used, for painting the datapoint at the given index in the model.

##### Parameters:

*index* The index of the datapoint in the model.

##### Returns:

The pen to use for painting.

Definition at line 788 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AttributesModel::data(), KDChart::AbstractDiagram::datasetDimension(), KDChart::DatasetPenRole, and KDChart::AbstractDiagram::pen().

```

789 {
790     if( datasetDimension() > 1 )
791         return pen( index.column() );
792     return qVariantValue<QPen>(
793         attributesModel()->data(
794             attributesModel()->mapFromSource( index ),
795             DatasetPenRole ) );
796 }

```

#### 9.44.3.49 QPen AbstractDiagram::pen (int *dataset*) const [inherited]

Retrieve the pen to be used for the given dataset.

This will fall back automatically to what was set at model level, if there are no dataset specific settings.

##### Parameters:

*dataset* The dataset to retrieve the pen for.

##### Returns:

The pen to use for painting.

Definition at line 777 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::columnToIndex(), KDChart::AttributesModel::data(), KDChart::AbstractDiagram::datasetDimension(), and KDChart::DatasetPenRole.

```

778 {
779     if( datasetDimension() > 1 )
780         dataset /= datasetDimension();
781
782     return qVariantValue<QPen>(
783         attributesModel()->data(
784             attributesModel()->mapFromSource( columnToIndex( dataset ) ),
785             DatasetPenRole ) );
786 }

```

#### 9.44.3.50 QPen AbstractDiagram::pen () const [inherited]

Retrieve the pen to be used for painting datapoints globally.

This will fall back automatically to the default settings, if there are no specific settings.

##### Returns:

The pen to use for painting.

Definition at line 771 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AttributesModel::data(), and KDChart::DatasetPenRole.

Referenced by KDChart::TernaryPointDiagram::paint(), KDChart::TernaryLineDiagram::paint(), and KDChart::AbstractDiagram::pen().

```

772 {
773     return qVariantValue<QPen>(
774         attributesModel()->data( DatasetPenRole ) );
775 }

```

**9.44.3.51** `bool AbstractDiagram::percentMode () const` [inherited]

Definition at line 462 of file KDChartAbstractDiagram.cpp.

References `d`.

Referenced by `KDChart::AbstractDiagram::compare()`, and `KDChart::CartesianCoordinatePlane::getDataDimensionsList()`.

```
463 {  
464     return d->percent;  
465 }
```

**9.44.3.52** `PieAttributes AbstractPieDiagram::pieAttributes (const QModelIndex & index) const` [inherited]

Definition at line 122 of file KDChartAbstractPieDiagram.cpp.

References `d`, and `KDChart::PieAttributesRole`.

```
123 {  
124     return qVariantValue<PieAttributes>(   
125         d->attributesModel->data(   
126             d->attributesModel->mapFromSource( index ),   
127             PieAttributesRole ) );  
128 }
```

**9.44.3.53** `PieAttributes AbstractPieDiagram::pieAttributes (int column) const` [inherited]

Definition at line 114 of file KDChartAbstractPieDiagram.cpp.

References `KDChart::AbstractDiagram::columnToIndex()`, `d`, and `KDChart::PieAttributesRole`.

```
115 {  
116     return qVariantValue<PieAttributes>(   
117         d->attributesModel->data(   
118             d->attributesModel->mapFromSource( columnToIndex( column ) ).column(),   
119             PieAttributesRole ) );  
120 }
```

**9.44.3.54** `PieAttributes AbstractPieDiagram::pieAttributes () const` [inherited]

Definition at line 105 of file KDChartAbstractPieDiagram.cpp.

References `d`, and `KDChart::PieAttributesRole`.

Referenced by `calculateDataBoundaries()`, and `paint()`.

```
106 {  
107     return qVariantValue<PieAttributes>(   
108         d->attributesModel->data( PieAttributesRole ) );  
109 }
```

#### 9.44.3.55 `const PolarCoordinatePlane * AbstractPolarDiagram::polarCoordinatePlane () const` [inherited]

Definition at line 55 of file `KDChartAbstractPolarDiagram.cpp`.

References `KDChart::AbstractDiagram::coordinatePlane()`.

Referenced by `paint()`.

```
56 {
57     return dynamic_cast<const PolarCoordinatePlane>( coordinatePlane() );
58 }
```

#### 9.44.3.56 `void KDChart::AbstractDiagram::propertiesChanged ()` [signal, inherited]

Emitted upon change of a property of the Diagram.

Referenced by `KDChart::Plotter::resetLineAttributes()`, `KDChart::LineDiagram::resetLineAttributes()`, `KDChart::AbstractDiagram::setAllowOverlappingDataValueTexts()`, `KDChart::AbstractDiagram::setAntiAliasing()`, `KDChart::BarDiagram::setBarAttributes()`, `KDChart::AbstractDiagram::setBrush()`, `KDChart::AbstractDiagram::setDataValueAttributes()`, `KDChart::Plotter::setLineAttributes()`, `KDChart::LineDiagram::setLineAttributes()`, `KDChart::AbstractDiagram::setPen()`, `KDChart::AbstractDiagram::setPercentMode()`, `KDChart::BarDiagram::setThreeDBarAttributes()`, `KDChart::Plotter::setThreeDLineAttributes()`, `KDChart::LineDiagram::setThreeDLineAttributes()`, `KDChart::Plotter::setType()`, `KDChart::LineDiagram::setType()`, `KDChart::BarDiagram::setType()`, `KDChart::Plotter::setValueTrackerAttributes()`, and `KDChart::LineDiagram::setValueTrackerAttributes()`.

#### 9.44.3.57 `void PieDiagram::resize (const QSizeF & area)` [virtual]

[reimplemented]

Implements [KDChart::AbstractDiagram](#).

Definition at line 112 of file `KDChartPieDiagram.cpp`.

```
113 {
114 }
```

#### 9.44.3.58 `void PieDiagram::resizeEvent (QResizeEvent *)` [protected]

Definition at line 108 of file `KDChartPieDiagram.cpp`.

```
109 {
110 }
```

#### 9.44.3.59 `void AbstractDiagram::scrollTo (const QModelIndex & index, ScrollHint hint = EnsureVisible)` [virtual, inherited]

[reimplemented]

Definition at line 942 of file `KDChartAbstractDiagram.cpp`.

```
943 {}
```



**9.44.3.60 void AbstractDiagram::setAllowOverlappingDataValueTexts (bool *allow*)**  
[inherited]

Set whether data value labels are allowed to overlap.

**Parameters:**

*allow* True means that overlapping labels are allowed.

Definition at line 434 of file KDChartAbstractDiagram.cpp.

References [d](#), and [KDChart::AbstractDiagram::propertiesChanged\(\)](#).

```
435 {  
436     d->allowOverlappingDataValueTexts = allow;  
437     emit propertiesChanged();  
438 }
```

**9.44.3.61 void AbstractDiagram::setAntiAliasing (bool *enabled*)** [inherited]

Set whether anti-aliasing is to be used while rendering this diagram.

**Parameters:**

*enabled* True means that AA is enabled.

Definition at line 445 of file KDChartAbstractDiagram.cpp.

References [d](#), and [KDChart::AbstractDiagram::propertiesChanged\(\)](#).

```
446 {  
447     d->antiAliasing = enabled;  
448     emit propertiesChanged();  
449 }
```

**9.44.3.62 void AbstractDiagram::setAttributesModel ([AttributesModel](#) \* *model*)** [virtual, inherited]

Associate an [AttributesModel](#) with this diagram.

Note that the diagram does `_not_` take ownership of the [AttributesModel](#). This should thus only be used with [AttributesModels](#) that have been explicitly created by the user, and are owned by her. Setting an [AttributesModel](#) that is internal to another diagram is an error.

Correct:

```
AttributesModel *am = new AttributesModel( model, 0 );  
diagram1->setAttributesModel( am );  
diagram2->setAttributesModel( am );
```

Wrong:

```
diagram1->setAttributesModel( diagram2->attributesModel() );
```

**Parameters:**

*model* The [AttributesModel](#) to use for this diagram.

**See also:**

[AttributesModel](#), [usesExternalAttributesModel](#)

Reimplemented in [KDChart::AbstractCartesianDiagram](#).

Definition at line 255 of file `KDChartAbstractDiagram.cpp`.

References `d`, `KDChart::AbstractDiagram::modelsChanged()`, and `KDChart::AbstractDiagram::setDataBoundariesDirty()`.

Referenced by `KDChart::AbstractCartesianDiagram::setAttributesModel()`.

```

256 {
257     if( amodel->sourceModel() != model() ) {
258         qWarning("KDChart::AbstractDiagram::setAttributesModel() failed: "
259                 "Trying to set an attributesmodel which works on a different "
260                 "model than the diagram.");
261         return;
262     }
263     if( qobject_cast<PrivateAttributesModel*>(amodel) ) {
264         qWarning("KDChart::AbstractDiagram::setAttributesModel() failed: "
265                 "Trying to set an attributesmodel that is private to another diagram.");
266         return;
267     }
268     d->setAttributesModel(amodel);
269     scheduleDelayedItemsLayout();
270     setDataBoundariesDirty();
271     emit modelsChanged();
272 }
```

#### 9.44.3.63 void AbstractDiagram::setAttributesModelRootIndex (const QModelIndex &) [protected, inherited]

Definition at line 293 of file `KDChartAbstractDiagram.cpp`.

References `d`, and `KDChart::AbstractDiagram::setDataBoundariesDirty()`.

Referenced by `KDChart::AbstractDiagram::setRootIndex()`.

```

294 {
295     d->attributesModelRootIndex=idx;
296     setDataBoundariesDirty();
297     scheduleDelayedItemsLayout();
298 }
```

#### 9.44.3.64 void AbstractDiagram::setBrush (const QBrush & brush) [inherited]

Set the brush to be used, for painting all datasets in the model.

**Parameters:**

*brush* The brush to use.

Definition at line 806 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DatasetBrushRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setModelData().

```
807 {
808     attributesModel()->setModelData(
809         qVariantFromValue( brush ), DatasetBrushRole );
810     emit propertiesChanged();
811 }
```

#### 9.44.3.65 void AbstractDiagram::setBrush (int *dataset*, const QBrush & *brush*) [inherited]

Set the brush to be used, for painting the given dataset.

##### Parameters:

*dataset* The dataset's column in the model.

*brush* The brush to use.

Definition at line 813 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DatasetBrushRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setHeaderData().

```
814 {
815     attributesModel()->setHeaderData(
816         column, Qt::Vertical,
817         qVariantFromValue( brush ),
818         DatasetBrushRole );
819     emit propertiesChanged();
820 }
```

#### 9.44.3.66 void AbstractDiagram::setBrush (const QModelIndex & *index*, const QBrush & *brush*) [inherited]

Set the brush to be used, for painting the datapoint at the given index.

##### Parameters:

*index* The datapoint's index in the model.

*brush* The brush to use.

Definition at line 798 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DatasetBrushRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setData().

```
799 {
800     attributesModel()->setData(
801         attributesModel()->mapFromSource( index ),
802         qVariantFromValue( brush ), DatasetBrushRole );
803     emit propertiesChanged();
804 }
```

**9.44.3.67** `void AbstractDiagram::setCoordinatePlane (AbstractCoordinatePlane * plane)`  
 [virtual, inherited]

Set the coordinate plane associated with the diagram.

This determines how coordinates in value space are mapped into pixel space. The chart takes ownership.

**Returns:**

The coordinate plane associated with the diagram.

Reimplemented in [KDChart::AbstractCartesianDiagram](#).

Definition at line 316 of file `KDChartAbstractDiagram.cpp`.

References d.

Referenced by `KDChart::AbstractCoordinatePlane::addDiagram()`, `KDChart::AbstractCartesianDiagram::setCoordinatePlane()`, and `KDChart::AbstractCoordinatePlane::takeDiagram()`.

```
317 {
318     d->plane = parent;
319 }
```

**9.44.3.68** `void AbstractDiagram::setDataBoundariesDirty () const` [protected, inherited]

Definition at line 234 of file `KDChartAbstractDiagram.cpp`.

References d.

Referenced by `KDChart::AbstractDiagram::dataChanged()`, `KDChart::Plotter::resize()`, `KDChart::LineDiagram::resize()`, `KDChart::BarDiagram::resize()`, `KDChart::AbstractDiagram::setAttributesModel()`, `KDChart::AbstractDiagram::setAttributesModelRootIndex()`, `KDChart::AbstractDiagram::setDatasetDimension()`, `KDChart::AbstractDiagram::setModel()`, `KDChart::BarDiagram::setThreeDBarAttributes()`, `KDChart::Plotter::setThreeDLineAttributes()`, `KDChart::LineDiagram::setThreeDLineAttributes()`, `KDChart::Plotter::setType()`, `KDChart::LineDiagram::setType()`, and `KDChart::BarDiagram::setType()`.

```
235 {
236     d->databoundariesDirty = true;
237 }
```

**9.44.3.69** `void AbstractDiagram::setDatasetDimension (int dimension)` [inherited]

Sets the dataset dimension of the diagram.

**See also:**

[datasetDimension](#).

**Parameters:**

*dimension*

Definition at line 1077 of file `KDChartAbstractDiagram.cpp`.

References `d`, `KDChart::AbstractDiagram::layoutChanged()`, and `KDChart::AbstractDiagram::setDataBoundariesDirty()`.

Referenced by `KDChart::Widget::setType()`, `KDChart::TernaryLineDiagram::TernaryLineDiagram()`, and `KDChart::TernaryPointDiagram::TernaryPointDiagram()`.

```
1078 {
1079     if ( d->datasetDimension == dimension ) return;
1080     d->datasetDimension = dimension;
1081     setDataBoundariesDirty();
1082     emit layoutChanged( this );
1083 }
```

#### 9.44.3.70 void AbstractDiagram::setDataValueAttributes (const [DataValueAttributes](#) & *a*) [inherited]

Set the [DataValueAttributes](#) for all datapoints in the model.

##### Parameters:

*a* The attributes to set.

Definition at line 428 of file `KDChartAbstractDiagram.cpp`.

References `d`, `KDChart::DataValueLabelAttributesRole`, and `KDChart::AbstractDiagram::propertiesChanged()`.

```
429 {
430     d->attributesModel->setModelData( qVariantFromValue( a ), DataValueLabelAttributesRole );
431     emit propertiesChanged();
432 }
```

#### 9.44.3.71 void AbstractDiagram::setDataValueAttributes (int *dataset*, const [DataValueAttributes](#) & *a*) [inherited]

Set the [DataValueAttributes](#) for the given dataset.

##### Parameters:

*dataset* The dataset to set the attributes for.

*a* The attributes to set.

Definition at line 400 of file `KDChartAbstractDiagram.cpp`.

References `d`, `KDChart::DataValueLabelAttributesRole`, and `KDChart::AbstractDiagram::propertiesChanged()`.

```
401 {
402     d->attributesModel->setHeaderData(
403         column, Qt::Vertical,
404         qVariantFromValue( a ), DataValueLabelAttributesRole );
405     emit propertiesChanged();
406 }
```

#### 9.44.3.72 void AbstractDiagram::setDataValueAttributes (const QModelIndex & *index*, const [DataValueAttributes](#) & *a*) [inherited]

Set the [DataValueAttributes](#) for the given index.

##### Parameters:

- index* The datapoint to set the attributes for.
- a* The attributes to set.

Definition at line 389 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::DataValueLabelAttributesRole](#), and [KDChart::AbstractDiagram::propertiesChanged\(\)](#).

```

391 {
392     d->attributesModel->setData(
393         d->attributesModel->mapFromSource( index ),
394         QVariantFromValue( a ),
395         DataValueLabelAttributesRole );
396     emit propertiesChanged();
397 }
```

#### 9.44.3.73 void AbstractPieDiagram::setGranularity (qreal *value*) [inherited]

Set the granularity: the smaller the granularity the more your diagram segments will show facettes instead of rounded segments.

##### Parameters:

- value* the granularity value between 0.05 (one twentieth of a degree) and 36.0 (one tenth of a full circle), other values will be interpreted as 1.0.

Definition at line 64 of file KDChartAbstractPieDiagram.cpp.

References [d](#).

```

65 {
66     d->granularity = value;
67 }
```

#### 9.44.3.74 void AbstractDiagram::setHidden (bool *hidden*) [inherited]

Hide (or unhide, resp.

) all datapoints in the model.

##### Note:

Hidden data are still taken into account by the coordinate plane, so neither the grid nor your axes' ranges will change, when you hide data. For totally removing data from KD Chart's view you can use another approach: e.g. you could define a proxy model on top of your data model, and register the proxy model calling [setModel\(\)](#) instead of registering your real data model.

**Parameters:**

***hidden*** The hidden status to set.

Definition at line 359 of file KDChartAbstractDiagram.cpp.

References `d`, `KDChart::AbstractDiagram::dataHidden()`, and `KDChart::DataHiddenRole`.

```
360 {  
361     d->attributesModel->setModelData(  
362         qVariantFromValue( hidden ),  
363         DataHiddenRole );  
364     emit dataHidden();  
365 }
```

**9.44.3.75 void AbstractDiagram::setHidden (int *column*, bool *hidden*)** [inherited]

Hide (or unhide, resp.

) a dataset.

**Note:**

Hidden data are still taken into account by the coordinate plane, so neither the grid nor your axes' ranges will change, when you hide data. For totally removing data from KD Chart's view you can use another approach: e.g. you could define a proxy model on top of your data model, and register the proxy model calling [setModel\(\)](#) instead of registering your real data model.

**Parameters:**

***column*** The dataset to set the hidden status for.

***hidden*** The hidden status to set.

Definition at line 350 of file KDChartAbstractDiagram.cpp.

References `d`, `KDChart::AbstractDiagram::dataHidden()`, and `KDChart::DataHiddenRole`.

```
351 {  
352     d->attributesModel->setHeaderData(  
353         column, Qt::Vertical,  
354         qVariantFromValue( hidden ),  
355         DataHiddenRole );  
356     emit dataHidden();  
357 }
```

**9.44.3.76 void AbstractDiagram::setHidden (const QModelIndex & *index*, bool *hidden*)**  
[inherited]

Hide (or unhide, resp.

) a data cell.

**Note:**

Hidden data are still taken into account by the coordinate plane, so neither the grid nor your axes' ranges will change, when you hide data. For totally removing data from KD Chart's view you can use another approach: e.g. you could define a proxy model on top of your data model, and register the proxy model calling [setModel\(\)](#) instead of registering your real data model.

**Parameters:**

***index*** The datapoint to set the hidden status for.

***hidden*** The hidden status to set.

Definition at line 341 of file KDChartAbstractDiagram.cpp.

References `d`, `KDChart::AbstractDiagram::dataHidden()`, and `KDChart::DataHiddenRole`.

```

342 {
343     d->attributesModel->setData(
344         d->attributesModel->mapFromSource( index ),
345         qVariantFromValue( hidden ),
346         DataHiddenRole );
347     emit dataHidden();
348 }
```

**9.44.3.77 void AbstractDiagram::setModel (QAbstractItemModel \* *model*)** [virtual, inherited]

Associate a model with the diagram.

Reimplemented in [KDChart::AbstractCartesianDiagram](#).

Definition at line 239 of file KDChartAbstractDiagram.cpp.

References `d`, `KDChart::AttributesModel::initFrom()`, `KDChart::AbstractDiagram::modelsChanged()`, and `KDChart::AbstractDiagram::setDataBoundariesDirty()`.

Referenced by `KDChart::AbstractCartesianDiagram::setModel()`, and `KDChart::Widget::setType()`.

```

240 {
241     QAbstractItemView::setModel( newModel );
242     AttributesModel* amodel = new PrivateAttributesModel( newModel, this );
243     amodel->initFrom( d->attributesModel );
244     d->setAttributesModel( amodel );
245     scheduleDelayedItemsLayout();
246     setDataBoundariesDirty();
247     emit modelsChanged();
248 }
```

**9.44.3.78 void AbstractDiagram::setPen (const QPen & *pen*)** [inherited]

Set the pen to be used, for painting all datasets in the model.

**Parameters:**

***pen*** The pen to use.

Definition at line 753 of file KDChartAbstractDiagram.cpp.

References `KDChart::AbstractDiagram::attributesModel()`, `KDChart::DatasetPenRole`, `KDChart::AbstractDiagram::propertiesChanged()`, and `KDChart::AttributesModel::setModelData()`.

```

754 {
755     attributesModel()->setModelData(
756         qVariantFromValue( pen ), DatasetPenRole );
757     emit propertiesChanged();
758 }
```



**9.44.3.79 void AbstractDiagram::setPen (int *dataset*, const QPen & *pen*)** [inherited]

Set the pen to be used, for painting the given dataset.

**Parameters:**

*dataset* The dataset's row in the model.

*pen* The pen to use.

Definition at line 760 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::datasetDimension(), KDChart::DatasetPenRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setHeaderData().

```

761 {
762     if( datasetDimension() > 1 )
763         column *= datasetDimension();
764     attributesModel()->setHeaderData(
765         column, Qt::Vertical,
766         qVariantFromValue( pen ),
767         DatasetPenRole );
768     emit propertiesChanged();
769 }
```

**9.44.3.80 void AbstractDiagram::setPen (const QModelIndex & *index*, const QPen & *pen*)** [inherited]

Set the pen to be used, for painting the datapoint at the given index.

**Parameters:**

*index* The datapoint's index in the model.

*pen* The pen to use.

Definition at line 743 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::datasetDimension(), KDChart::DatasetPenRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setData().

```

744 {
745     if( datasetDimension() > 1 )
746         return setPen( index.column(), pen );
747     attributesModel()->setData(
748         attributesModel()->mapFromSource( index ),
749         qVariantFromValue( pen ), DatasetPenRole );
750     emit propertiesChanged();
751 }
```

**9.44.3.81 void AbstractDiagram::setPercentMode (bool *percent*)** [inherited]

Definition at line 456 of file KDChartAbstractDiagram.cpp.

References d, and KDChart::AbstractDiagram::propertiesChanged().

Referenced by KDChart::LineDiagram::setType(), and KDChart::BarDiagram::setType().

```

457 {
458     d->percent = percent;
459     emit propertiesChanged();
460 }

```

#### 9.44.3.82 void AbstractPieDiagram::setPieAttributes (int *column*, const [PieAttributes](#) & *a*) [inherited]

Definition at line 95 of file `KDChartAbstractPieDiagram.cpp`.

References `d`, `KDChart::AbstractDiagram::layoutChanged()`, and `KDChart::PieAttributesRole`.

```

96 {
97     d->attributesModel->setHeaderData(
98         column, Qt::Vertical, qVariantFromValue( attrs ), PieAttributesRole );
99     emit layoutChanged( this );
100 }

```

#### 9.44.3.83 void AbstractPieDiagram::setPieAttributes (const [PieAttributes](#) & *a*) [inherited]

Definition at line 89 of file `KDChartAbstractPieDiagram.cpp`.

References `d`, `KDChart::AbstractDiagram::layoutChanged()`, and `KDChart::PieAttributesRole`.

```

90 {
91     d->attributesModel->setModelData( qVariantFromValue( attrs ), PieAttributesRole );
92     emit layoutChanged( this );
93 }

```

#### 9.44.3.84 void AbstractDiagram::setRootIndex (const [QModelIndex](#) & *idx*) [virtual, inherited]

Set the root index in the model, where the diagram starts referencing data for display.

[reimplemented]

Reimplemented in [KDChart::AbstractCartesianDiagram](#).

Definition at line 286 of file `KDChartAbstractDiagram.cpp`.

References `d`, and `KDChart::AbstractDiagram::setAttributesModelRootIndex()`.

Referenced by `KDChart::AbstractCartesianDiagram::setRootIndex()`.

```

287 {
288     QAbstractItemView::setRootIndex(idx);
289     setAttributesModelRootIndex( d->attributesModel->mapFromSource(idx) );
290 }

```

#### 9.44.3.85 void AbstractDiagram::setSelection (const [QRect](#) & *rect*, [QItemSelectionModel::SelectionFlags](#) *command*) [virtual, inherited]

[reimplemented]

Definition at line 959 of file `KDChartAbstractDiagram.cpp`.

References `d`.

```

960 {
961     const QModelIndexList indexes = d->indexesIn( rect );
962     QItemSelection selection;
963     KDAB_FOREACH( const QModelIndex& index, indexes )
964     {
965         selection.append( QItemSelectionRange( index ) );
966     }
967     selectionModel()->select( selection, command );
968 }

```

#### 9.44.3.86 void AbstractPieDiagram::setStartPosition (int *degrees*) [inherited]

##### Deprecated

Use [PolarCoordinatePlane::setStartPosition\( qreal degrees \)](#) instead.

Definition at line 77 of file KDChartAbstractPieDiagram.cpp.

```

78 {
79     Q_UNUSED( degrees );
80     qWarning() << "Deprecated AbstractPieDiagram::setStartPosition() called, setting ignored.";
81 }

```

#### 9.44.3.87 void AbstractPieDiagram::setThreeDPieAttributes (const QModelIndex & *index*, const [ThreeDPieAttributes](#) & *a*) [inherited]

Definition at line 144 of file KDChartAbstractPieDiagram.cpp.

References [KDChart::AbstractDiagram::layoutChanged\(\)](#), and [KDChart::ThreeDPieAttributesRole](#).

```

145 {
146     model()->setData( index, qVariantFromValue( tda ), ThreeDPieAttributesRole );
147     emit layoutChanged( this );
148 }

```

#### 9.44.3.88 void AbstractPieDiagram::setThreeDPieAttributes (int *column*, const [ThreeDPieAttributes](#) & *a*) [inherited]

Definition at line 137 of file KDChartAbstractPieDiagram.cpp.

References [d](#), [KDChart::AbstractDiagram::layoutChanged\(\)](#), and [KDChart::ThreeDPieAttributesRole](#).

```

138 {
139     d->attributesModel->setHeaderData(
140         column, Qt::Vertical, qVariantFromValue( tda ), ThreeDPieAttributesRole );
141     emit layoutChanged( this );
142 }

```

#### 9.44.3.89 void AbstractPieDiagram::setThreeDPieAttributes (const [ThreeDPieAttributes](#) & *a*) [inherited]

Definition at line 131 of file KDChartAbstractPieDiagram.cpp.

References [d](#), [KDChart::AbstractDiagram::layoutChanged\(\)](#), and [KDChart::ThreeDPieAttributesRole](#).

```
132 {  
133     d->attributesModel->setModelData( qVariantFromValue( tda ), ThreeDPieAttributesRole );  
134     emit layoutChanged( this );  
135 }
```

#### 9.44.3.90 void AbstractDiagram::setUnitPrefix (const QString & *prefix*, Qt::Orientation *orientation*) [inherited]

Sets the unit prefix for all values.

##### Parameters:

*prefix* the prefix to be set

*orientation* the orientation of the axis to set

Definition at line 864 of file KDChartAbstractDiagram.cpp.

References d.

```
865 {  
866     d->unitPrefix[ orientation ] = prefix;  
867 }
```

#### 9.44.3.91 void AbstractDiagram::setUnitPrefix (const QString & *prefix*, int *column*, Qt::Orientation *orientation*) [inherited]

Sets the unit prefix for one value.

##### Parameters:

*prefix* the prefix to be set

*column* the value using that prefix

*orientation* the orientation of the axis to set

Definition at line 854 of file KDChartAbstractDiagram.cpp.

References d.

```
855 {  
856     d->unitPrefixMap[ column ][ orientation ]= prefix;  
857 }
```

#### 9.44.3.92 void AbstractDiagram::setUnitSuffix (const QString & *suffix*, Qt::Orientation *orientation*) [inherited]

Sets the unit suffix for all values.

##### Parameters:

*suffix* the suffix to be set

*orientation* the orientation of the axis to set

Definition at line 885 of file KDChartAbstractDiagram.cpp.

References [d](#).

```
886 {
887     d->unitSuffix[ orientation ] = suffix;
888 }
```

#### 9.44.3.93 void AbstractDiagram::setUnitSuffix (const QString & *suffix*, int *column*, Qt::Orientation *orientation*) [inherited]

Sets the unit suffix for one value.

##### Parameters:

*suffix* the suffix to be set

*column* the value using that suffix

*orientation* the orientation of the axis to set

Definition at line 875 of file KDChartAbstractDiagram.cpp.

References [d](#).

```
876 {
877     d->unitSuffixMap[ column ][ orientation ]= suffix;
878 }
```

#### 9.44.3.94 int AbstractPieDiagram::startPosition () const [inherited]

##### Deprecated

Use [qreal PolarCoordinatePlane::startPosition](#) instead.

Definition at line 83 of file KDChartAbstractPieDiagram.cpp.

```
84 {
85     qWarning() << "Deprecated AbstractPieDiagram::startPosition() called.";
86     return 0;
87 }
```

#### 9.44.3.95 [ThreeDPieAttributes](#) AbstractPieDiagram::threeDPieAttributes (const QModelIndex & *index*) const [inherited]

Definition at line 170 of file KDChartAbstractPieDiagram.cpp.

References [d](#), and [KDChart::ThreeDPieAttributesRole](#).

```
171 {
172     return qVariantValue<ThreeDPieAttributes>(
173         d->attributesModel->data(
174             d->attributesModel->mapFromSource( index ),
175             ThreeDPieAttributesRole ) );
176 }
```

#### 9.44.3.96 **ThreeDPieAttributes** AbstractPieDiagram::threeDPieAttributes (int *column*) const [inherited]

Definition at line 162 of file KDChartAbstractPieDiagram.cpp.

References KDChart::AbstractDiagram::columnToIndex(), d, and KDChart::ThreeDPieAttributesRole.

```
163 {
164     return qVariantValue<ThreeDPieAttributes>(
165         d->attributesModel->data(
166             d->attributesModel->mapFromSource( columnToIndex( column ) ).column(),
167             ThreeDPieAttributesRole ) );
168 }
```

#### 9.44.3.97 **ThreeDPieAttributes** AbstractPieDiagram::threeDPieAttributes () const [inherited]

Definition at line 153 of file KDChartAbstractPieDiagram.cpp.

References d, and KDChart::ThreeDPieAttributesRole.

Referenced by paint().

```
154 {
155     return qVariantValue<ThreeDPieAttributes>(
156         d->attributesModel->data( ThreeDPieAttributesRole ) );
157 }
```

#### 9.44.3.98 **QString** AbstractDiagram::unitPrefix (Qt::Orientation *orientation*) const [inherited]

Returns the global unit prefix.

##### Parameters:

***orientation*** the orientation of the axis

##### Returns:

the unit prefix

Definition at line 908 of file KDChartAbstractDiagram.cpp.

References d.

```
909 {
910     return d->unitPrefix[ orientation ];
911 }
```

#### 9.44.3.99 **QString** AbstractDiagram::unitPrefix (int *column*, Qt::Orientation *orientation*, bool *fallback* = false) const [inherited]

Returns the unit prefix for a special value.

**Parameters:**

*column* the value which's prefix is requested  
*orientation* the orientation of the axis  
*fallback* if true, the global prefix is return when no specific one is set for that value

**Returns:**

the unit prefix

Definition at line 897 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::itemRowLabels(), and KDChart::CartesianAxis::paintCtx().

```

898 {
899     if( !fallback || d->unitPrefixMap[ column ].contains( orientation ) )
900         return d->unitPrefixMap[ column ][ orientation ];
901     return d->unitPrefix[ orientation ];
902 }
```

#### 9.44.3.100 QString AbstractDiagram::unitSuffix (Qt::Orientation *orientation*) const [inherited]

Returns the global unit suffix.

**Parameters:**

*orientation* the orientation of the axis

**Returns:**

the unit suffix

Definition at line 931 of file KDChartAbstractDiagram.cpp.

References d.

```

932 {
933     return d->unitSuffix[ orientation ];
934 }
```

#### 9.44.3.101 QString AbstractDiagram::unitSuffix (int *column*, Qt::Orientation *orientation*, bool *fallback* = false) const [inherited]

Returns the unit suffix for a special value.

**Parameters:**

*column* the value which's suffix is requested  
*orientation* the orientation of the axis  
*fallback* if true, the global suffix is return when no specific one is set for that value

**Returns:**

the unit suffix

Definition at line 920 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::itemRowLabels(), and KDChart::CartesianAxis::paintCtx().

```

921 {
922     if( !fallback || d->unitSuffixMap[ column ].contains( orientation ) )
923         return d->unitSuffixMap[ column ][ orientation ];
924     return d->unitSuffix[ orientation ];
925 }
```

**9.44.3.102 void AbstractDiagram::update () const [inherited]**

Definition at line 1091 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::doItemsLayout().

```

1092 {
1093     //qDebug("KDChart::AbstractDiagram::update() called");
1094     if( d->plane )
1095         d->plane->update();
1096 }
```

**9.44.3.103 void KDChart::AbstractDiagram::useDefaultColors () [inherited]**

Set the palette to be used, for painting datasets to the default palette.

**See also:**

[KDChart::Palette](#). FIXME: fold into one usePalette ([KDChart::Palette&](#)) method

Definition at line 974 of file KDChartAbstractDiagram.cpp.

References d, and KDChart::AttributesModel::PaletteTypeDefault.

```

975 {
976     d->attributesModel->setPaletteType( AttributesModel::PaletteTypeDefault );
977 }
```

**9.44.3.104 void KDChart::AbstractDiagram::useRainbowColors () [inherited]**

Set the palette to be used, for painting datasets to the rainbow palette.

**See also:**

[KDChart::Palette](#).



Definition at line 984 of file KDChartAbstractDiagram.cpp.

References [d](#), and [KDChart::AttributesModel::PaletteTypeRainbow](#).

```
985 {  
986     d->attributesModel->setPaletteType( AttributesModel::PaletteTypeRainbow );  
987 }
```

#### 9.44.3.105 bool AbstractDiagram::usesExternalAttributesModel () const [virtual, inherited]

Returns whether the diagram is using its own built-in attributes model or an attributes model that was set via [setAttributesModel](#).

**See also:**

[setAttributesModel](#)

Definition at line 274 of file KDChartAbstractDiagram.cpp.

References [d](#).

```
275 {  
276     return d->usesExternalAttributesModel();  
277 }
```

#### 9.44.3.106 void KDChart::AbstractDiagram::useSubduedColors () [inherited]

Set the palette to be used, for painting datasets to the subdued palette.

**See also:**

[KDChart::Palette](#).

Definition at line 979 of file KDChartAbstractDiagram.cpp.

References [d](#), and [KDChart::AttributesModel::PaletteTypeSubdued](#).

```
980 {  
981     d->attributesModel->setPaletteType( AttributesModel::PaletteTypeSubdued );  
982 }
```

#### 9.44.3.107 double AbstractDiagram::valueForCell (int row, int column) const [protected, inherited]

Helper method, retrieving the data value (DisplayRole) for a given row and column.

**Parameters:**

*row* The row to query.

*column* The column to query.

**Returns:**

The value of the display role at the given row and column as a double.

**Deprecated**

Definition at line 1085 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModelRootIndex(), and d.

```
1086 {
1087     return d->attributesModel->data(
1088         d->attributesModel->index( row, column, attributesModelRootIndex() ) ).toDouble();
1089 }
```

**9.44.3.108 double PieDiagram::valueTotals () const [virtual]**

[reimplemented]

Implements [KDChart::AbstractPolarDiagram](#).

Definition at line 969 of file KDChartPieDiagram.cpp.

References KDChart::AbstractPolarDiagram::columnCount().

Referenced by paint().

```
970 {
971     const int colCount = columnCount();
972     double total = 0.0;
973     for ( int j = 0; j < colCount; ++j ) {
974         total += qAbs(model()->data( model()->index( 0, j, rootIndex() ) ).toDouble());
975         //qDebug() << model()->data( model()->index( 0, j, rootIndex() ) ).toDouble();
976     }
977     return total;
978 }
```

**9.44.3.109 int AbstractDiagram::verticalOffset () const [virtual, inherited]**

[reimplemented]

Definition at line 953 of file KDChartAbstractDiagram.cpp.

```
954 { return 0; }
```

**9.44.3.110 QRect AbstractDiagram::visualRect (const QModelIndex & *index*) const [virtual, inherited]**

[reimplemented]

Definition at line 937 of file KDChartAbstractDiagram.cpp.

```
938 {
939     return QRect();
940 }
```

**9.44.3.111 QRegion AbstractDiagram::visualRegionForSelection (const QItemSelection & *selection*) const** [virtual, inherited]

[reimplemented]

Definition at line 970 of file KDChartAbstractDiagram.cpp.

```
971 { return QRegion(); }
```

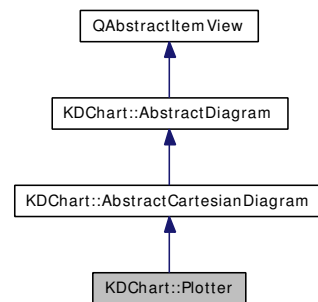
The documentation for this class was generated from the following files:

- [KDChartPieDiagram.h](#)
- [KDChartPieDiagram.cpp](#)

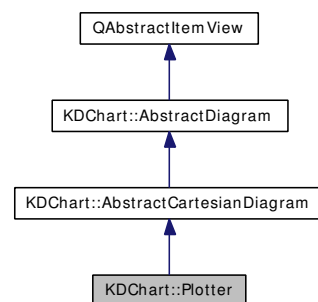
## 9.45 KDChart::Plotter Class Reference

```
#include <KDChartPlotter.h>
```

Inheritance diagram for KDChart::Plotter:



Collaboration diagram for KDChart::Plotter:



### 9.45.1 Detailed Description

[Plotter](#) defines a diagram type plotting two-dimensional data.

Definition at line 45 of file `KDChartPlotter.h`.

### Public Types

- enum [PlotType](#) { [Normal](#) = 0 }

### Signals

- void [dataHidden](#) ()  
*This signal is emitted, when the hidden status of at least one data cell was (un)set.*
- void [layoutChanged](#) ([AbstractDiagram](#) \*)  
*Diagrams are supposed to emit this signal, when the layout of one of their element changes.*
- void [modelsChanged](#) ()  
*This signal is emitted, when either the model or the [AttributesModel](#) is replaced.*

- void [propertiesChanged](#) ()  
*Emitted upon change of a property of the Diagram.*

## Public Member Functions

- virtual void [addAxis](#) ([CartesianAxis](#) \*axis)  
*Add the axis to the diagram.*
- bool [allowOverlappingDataValueTexts](#) () const  
**Returns:**  
*Whether data value labels are allowed to overlap.*
- bool [antiAliasing](#) () const  
**Returns:**  
*Whether anti-aliasing is to be used for rendering this diagram.*
- virtual [AttributesModel](#) \* [attributesModel](#) () const  
*Returns the [AttributesModel](#), that is used by this diagram.*
- virtual [KDChart::CartesianAxisList](#) [axes](#) () const  
**Returns:**  
*a list of all axes added to the diagram*
- [QBrush](#) [brush](#) (const [QModelIndex](#) &index) const  
*Retrieve the brush to be used, for painting the datapoint at the given index in the model.*
- [QBrush](#) [brush](#) (int dataset) const  
*Retrieve the brush to be used for the given dataset.*
- [QBrush](#) [brush](#) () const  
*Retrieve the brush to be used for painting datapoints globally.*
- virtual [Plotter](#) \* [clone](#) () const  
*Creates an exact copy of this diagram.*
- bool [compare](#) (const [AbstractDiagram](#) \*other) const  
*Returns true if both diagrams have the same settings.*
- bool [compare](#) (const [AbstractCartesianDiagram](#) \*other) const  
*Returns true if both diagrams have the same settings.*
- bool [compare](#) (const [Plotter](#) \*other) const  
*Returns true if both diagrams have the same settings.*
- [AbstractCoordinatePlane](#) \* [coordinatePlane](#) () const  
*The coordinate plane associated with the diagram.*

- `const QPair< QPointF, QPointF > dataBoundaries () const`  
*Return the bottom left and top right data point, that the diagram will display (unless the grid adjusts these values).*
- `virtual void dataChanged (const QModelIndex &topLeft, const QModelIndex &bottomRight)`  
*[reimplemented]*
- `QList< QBrush > datasetBrushes () const`  
*The set of dataset brushes currently used, for use in legends, etc.*
- `int datasetDimension () const`  
*The dataset dimension of a diagram determines, how many value dimensions it expects each datapoint to have.*
- `QStringList datasetLabels () const`  
*The set of dataset labels currently displayed, for use in legends, etc.*
- `QList< MarkerAttributes > datasetMarkers () const`  
*The set of dataset markers currently used, for use in legends, etc.*
- `QList< QPen > datasetPens () const`  
*The set of dataset pens currently used, for use in legends, etc.*
- `DataValueAttributes dataValueAttributes (const QModelIndex &index) const`  
*Retrieve the [DataValueAttributes](#) for the given index.*
- `DataValueAttributes dataValueAttributes (int column) const`  
*Retrieve the [DataValueAttributes](#) for the given dataset.*
- `DataValueAttributes dataValueAttributes () const`  
*Retrieve the [DataValueAttributes](#) specified globally.*
- `virtual void doItemsLayout ()`  
*[reimplemented]*
- `virtual int horizontalOffset () const`  
*[reimplemented]*
- `virtual QModelIndex indexAt (const QPoint &point) const`  
*[reimplemented]*
- `QModelIndexList indexesAt (const QPoint &point) const`  
*This method is added alongside with `indexAt` from *QAIM*, since in *kdchart* multiple indexes can be displayed at the same spot.*
- `bool isHidden (const QModelIndex &index) const`  
*Retrieve the hidden status for the given index.*
- `bool isHidden (int column) const`  
*Retrieve the hidden status for the given dataset.*

- bool [isHidden](#) () const  
*Retrieve the hidden status specified globally.*
- virtual bool [isIndexHidden](#) (const QModelIndex &index) const  
*[reimplemented]*
- QStringList [itemRowLabels](#) () const  
*The set of item row labels currently displayed, for use in Abscissa axes, etc.*
- virtual void [layoutPlanes](#) ()  
*Triggers layouting of all coordinate planes on the current chart.*
- [LineAttributes](#) [lineAttributes](#) (const QModelIndex &index) const  
**Returns:**  
*the line attribute set of the model index index*
- [LineAttributes](#) [lineAttributes](#) (int column) const  
**Returns:**  
*the line attribute set of data set column*
- [LineAttributes](#) [lineAttributes](#) () const  
**Returns:**  
*the global line attribute set*
- virtual QModelIndex [moveCursor](#) (CursorAction cursorAction, Qt::KeyboardModifiers modifiers)  
*[reimplemented]*
- const int [numberOfAbscissaSegments](#) () const
- const int [numberOfOrdinateSegments](#) () const
- void [paintDataValueText](#) (QPainter \*painter, const QModelIndex &index, const QPointF &pos, double value)
- void [paintMarker](#) (QPainter \*painter, const QModelIndex &index, const QPointF &pos)
- virtual void [paintMarker](#) (QPainter \*painter, const [MarkerAttributes](#) &markerAttributes, const QBrush &brush, const QPen &, const QPointF &point, const QSizeF &size)
- QPen [pen](#) (const QModelIndex &index) const  
*Retrieve the pen to be used, for painting the datapoint at the given index in the model.*
- QPen [pen](#) (int dataset) const  
*Retrieve the pen to be used for the given dataset.*
- QPen [pen](#) () const  
*Retrieve the pen to be used for painting datapoints globally.*
- bool [percentMode](#) () const
- [Plotter](#) (QWidget \*parent=0, [CartesianCoordinatePlane](#) \*plane=0)
- virtual [AbstractCartesianDiagram](#) \* [referenceDiagram](#) () const  
**Returns:**  
*this diagram's reference diagram*

- virtual QPointF [referenceDiagramOffset](#) () const  
*Returns:*  
*the relative offset of this diagram's reference diagram*
- void [resetLineAttributes](#) (const QModelIndex &index)  
*Remove any explicit line attributes settings that might have been specified before.*
- void [resetLineAttributes](#) (int column)  
*Resets the line attributes of data set column.*
- void [resize](#) (const QSizeF &area)  
*Called by the widget's sizeEvent.*
- virtual void [scrollTo](#) (const QModelIndex &index, ScrollHint hint=EnsureVisible)  
*[reimplemented]*
- void [setAllowOverlappingDataValueTexts](#) (bool allow)  
*Set whether data value labels are allowed to overlap.*
- void [setAntiAliasing](#) (bool enabled)  
*Set whether anti-aliasing is to be used while rendering this diagram.*
- void [setAttributesModel](#) ([AttributesModel](#) \*model)  
*Associate an [AttributesModel](#) with this diagram.*
- void [setBrush](#) (const QBrush &brush)  
*Set the brush to be used, for painting all datasets in the model.*
- void [setBrush](#) (int dataset, const QBrush &brush)  
*Set the brush to be used, for painting the given dataset.*
- void [setBrush](#) (const QModelIndex &index, const QBrush &brush)  
*Set the brush to be used, for painting the datapoint at the given index.*
- virtual void [setCoordinatePlane](#) ([AbstractCoordinatePlane](#) \*plane)  
*[reimplemented]*
- void [setDatasetDimension](#) (int dimension)  
*Sets the dataset dimension of the diagram.*
- void [setDataValueAttributes](#) (const [DataValueAttributes](#) &a)  
*Set the [DataValueAttributes](#) for all datapoints in the model.*
- void [setDataValueAttributes](#) (int dataset, const [DataValueAttributes](#) &a)  
*Set the [DataValueAttributes](#) for the given dataset.*
- void [setDataValueAttributes](#) (const QModelIndex &index, const [DataValueAttributes](#) &a)  
*Set the [DataValueAttributes](#) for the given index.*



- void [setHidden](#) (bool hidden)  
*Hide (or unhide, resp).*
- void [setHidden](#) (int column, bool hidden)  
*Hide (or unhide, resp).*
- void [setHidden](#) (const QModelIndex &index, bool hidden)  
*Hide (or unhide, resp).*
- void [setLineAttributes](#) (const QModelIndex &index, const [LineAttributes](#) &a)  
*Sets the line attributes for the model index index to la.*
- void [setLineAttributes](#) (int column, const [LineAttributes](#) &a)  
*Sets the line attributes of data set column to la.*
- void [setLineAttributes](#) (const [LineAttributes](#) &a)  
*Sets the global line attributes to la.*
- void [setModel](#) (QAbstractItemModel \*model)  
*Associate a model with the diagram.*
- void [setPen](#) (const QPen &pen)  
*Set the pen to be used, for painting all datasets in the model.*
- void [setPen](#) (int dataset, const QPen &pen)  
*Set the pen to be used, for painting the given dataset.*
- void [setPen](#) (const QModelIndex &index, const QPen &pen)  
*Set the pen to be used, for painting the datapoint at the given index.*
- void [setPercentMode](#) (bool percent)
- virtual void [setReferenceDiagram](#) ([AbstractCartesianDiagram](#) \*diagram, const QPointF &offset=QPointF())  
*Makes this diagram use another diagram diagram as reference diagram with relative offset offset.*
- void [setRootIndex](#) (const QModelIndex &index)  
*Set the root index in the model, where the diagram starts referencing data for display.*
- virtual void [setSelection](#) (const QRect &rect, QItemSelectionModel::SelectionFlags command)  
*[reimplemented]*
- void [setThreeDLineAttributes](#) (const QModelIndex &index, const [ThreeDLineAttributes](#) &a)  
*Sets the 3D line attributes of model index index to la.*
- void [setThreeDLineAttributes](#) (int column, const [ThreeDLineAttributes](#) &a)  
*Sets the 3D line attributes of data set column to la.*
- void [setThreeDLineAttributes](#) (const [ThreeDLineAttributes](#) &a)  
*Sets the global 3D line attributes to la.*

- void [setType](#) (const [PlotType](#) type)
 

*Sets the plotter's type to type. Currently, there's no other type than [PlotType::Normal](#), but others might be added in future.*
- void [setUnitPrefix](#) (const QString &prefix, Qt::Orientation orientation)
 

*Sets the unit prefix for all values.*
- void [setUnitPrefix](#) (const QString &prefix, int column, Qt::Orientation orientation)
 

*Sets the unit prefix for one value.*
- void [setUnitSuffix](#) (const QString &suffix, Qt::Orientation orientation)
 

*Sets the unit suffix for all values.*
- void [setUnitSuffix](#) (const QString &suffix, int column, Qt::Orientation orientation)
 

*Sets the unit suffix for one value.*
- void [setValueTrackerAttributes](#) (const QModelIndex &index, const [ValueTrackerAttributes](#) &a)
 

*Sets the value tracker attributes of the model index index to va.*
- virtual void [takeAxis](#) ([CartesianAxis](#) \*axis)
 

*Removes the axis from the diagram, without deleting it.*
- [ThreeDLineAttributes](#) [threeDLineAttributes](#) (const QModelIndex &index) const
 

**Returns:**

*the 3D line attributes of the model index index*
- [ThreeDLineAttributes](#) [threeDLineAttributes](#) (int column) const
 

**Returns:**

*the 3D line attributes of data set column*
- [ThreeDLineAttributes](#) [threeDLineAttributes](#) () const
 

**Returns:**

*the global 3D line attributes*
- [PlotType](#) type () const
 

**Returns:**

*the type of the plotter*
- QString [unitPrefix](#) (Qt::Orientation orientation) const
 

*Returns the global unit prefix.*
- QString [unitPrefix](#) (int column, Qt::Orientation orientation, bool fallback=false) const
 

*Returns the unit prefix for a special value.*
- QString [unitSuffix](#) (Qt::Orientation orientation) const
 

*Returns the global unit suffix.*
- QString [unitSuffix](#) (int column, Qt::Orientation orientation, bool fallback=false) const
 

*Returns the unit suffix for a special value.*

- void [update](#) () const
- void [useDefaultColors](#) ()  
*Set the palette to be used, for painting datasets to the default palette.*
- void [useRainbowColors](#) ()  
*Set the palette to be used, for painting datasets to the rainbow palette.*
- virtual bool [usesExternalAttributesModel](#) () const  
*Returns whether the diagram is using its own built-in attributes model or an attributes model that was set via [setAttributesModel](#).*
- void [useSubduedColors](#) ()  
*Set the palette to be used, for painting datasets to the subdued palette.*
- [ValueTrackerAttributes](#) [valueTrackerAttributes](#) (const QModelIndex &index) const  
*Returns the value tracker attributes of the model index index.*
- virtual int [verticalOffset](#) () const  
*[reimplemented]*
- virtual QRect [visualRect](#) (const QModelIndex &index) const  
*[reimplemented]*
- virtual QRegion [visualRegionForSelection](#) (const QItemSelection &selection) const  
*[reimplemented]*
- virtual [~Plotter](#) ()

## Protected Member Functions

- QModelIndex [attributesModelRootIndex](#) () const
- virtual const QPair< QPointF, QPointF > [calculateDataBoundaries](#) () const  
*[reimplemented]*
- virtual bool [checkInvariants](#) (bool justReturnTheStatus=false) const
- QModelIndex [columnToIndex](#) (int column) const
- void [paint](#) (PaintContext \*paintContext)  
*Draw the diagram contents to the rectangle and painter, that are passed in as part of the paint context.*
- virtual void [paintDataValueTexts](#) (QPainter \*painter)
- void [paintEvent](#) (QPaintEvent \*)
- virtual void [paintMarkers](#) (QPainter \*painter)
- void [resizeEvent](#) (QResizeEvent \*)
- void [setAttributesModelRootIndex](#) (const QModelIndex &)
- void [setDataBoundariesDirty](#) () const
- virtual double [threeDItemDepth](#) (int column) const

### Returns:

*the 3D item depth of the data set column*

- virtual double [threeDItemDepth](#) (const QModelIndex &index) const

*Returns:*

*the 3D item depth of the model index index*

- double [valueForCell](#) (int row, int column) const

*Helper method, retrieving the data value (DisplayRole) for a given row and column.*

## 9.45.2 Member Enumeration Documentation

### 9.45.2.1 enum [KDChart::Plotter::PlotType](#)

Enumerator:

*Normal*

Definition at line 68 of file KDChartPlotter.h.

```
68         {
69             Normal = 0
70         };
```

## 9.45.3 Constructor & Destructor Documentation

### 9.45.3.1 [Plotter::Plotter](#) ([QWidget](#) \* *parent* = 0, [CartesianCoordinatePlane](#) \* *plane* = 0)

Definition at line 47 of file KDChartPlotter.cpp.

Referenced by clone().

```
47                                     :
48     AbstractCartesianDiagram( new Private(), parent, plane )
49 {
50     init();
51 }
```

### 9.45.3.2 [Plotter::~~Plotter](#) () [virtual]

Definition at line 62 of file KDChartPlotter.cpp.

```
63 {
64 }
```

## 9.45.4 Member Function Documentation

### 9.45.4.1 void [AbstractCartesianDiagram::addAxis](#) ([CartesianAxis](#) \* *axis*) [virtual, inherited]

Add the axis to the diagram.

The diagram takes ownership of the axis and will delete it.

To gain back ownership (e.g. for assigning the axis to another diagram) use the `takeAxis` method, before calling `addAxis` on the other diagram.

**See also:**

[takeAxis](#)

Definition at line 91 of file `KDChartAbstractCartesianDiagram.cpp`.

References `d`, and `KDChart::AbstractCartesianDiagram::layoutPlanes()`.

```
92 {
93     if ( !d->axesList.contains( axis ) ) {
94         d->axesList.append( axis );
95         axis->createObserver( this );
96         layoutPlanes();
97     }
98 }
```

#### 9.45.4.2 `bool AbstractDiagram::allowOverlappingDataValueTexts () const` [inherited]

**Returns:**

Whether data value labels are allowed to overlap.

Definition at line 440 of file `KDChartAbstractDiagram.cpp`.

References `d`.

Referenced by `KDChart::AbstractDiagram::compare()`.

```
441 {
442     return d->allowOverlappingDataValueTexts;
443 }
```

#### 9.45.4.3 `bool AbstractDiagram::antiAliasing () const` [inherited]

**Returns:**

Whether anti-aliasing is to be used for rendering this diagram.

Definition at line 451 of file `KDChartAbstractDiagram.cpp`.

References `d`.

Referenced by `KDChart::AbstractDiagram::compare()`.

```
452 {
453     return d->antiAliasing;
454 }
```

#### 9.45.4.4 **AttributesModel** \* **AbstractDiagram::attributesModel () const** [virtual, inherited]

Returns the [AttributesModel](#), that is used by this diagram.

By default each diagram owns its own [AttributesModel](#), which should never be deleted. Only if a user-supplied [AttributesModel](#) has been set does the pointer returned here not belong to the diagram.

#### Returns:

The [AttributesModel](#) associated with the diagram.

#### See also:

[setAttributesModel](#)

Definition at line 280 of file `KDChartAbstractDiagram.cpp`.

References d.

Referenced by `KDChart::AbstractDiagram::brush()`, `KDChart::AbstractDiagram::compare()`, `KDChart::AbstractDiagram::datasetBrushes()`, `KDChart::AbstractDiagram::datasetLabels()`, `KDChart::AbstractDiagram::datasetMarkers()`, `KDChart::AbstractDiagram::datasetPens()`, `KDChart::AbstractDiagram::dataValueAttributes()`, `KDChart::AbstractDiagram::isHidden()`, `KDChart::AbstractDiagram::itemRowLabels()`, `KDChart::RingDiagram::paint()`, `KDChart::PolarDiagram::paint()`, `KDChart::AbstractDiagram::pen()`, `KDChart::AbstractCartesianDiagram::setAttributesModel()`, `KDChart::BarDiagram::setBarAttributes()`, `KDChart::AbstractDiagram::setBrush()`, `KDChart::AbstractCartesianDiagram::setModel()`, `KDChart::AbstractDiagram::setPen()`, and `KDChart::TernaryLineDiagram::TernaryLineDiagram()`.

```
281 {
282     return d->attributesModel;
283 }
```

#### 9.45.4.5 **QModelIndex** **AbstractDiagram::attributesModelRootIndex () const** [protected, inherited]

returns a `QModelIndex` pointing into the [AttributesModel](#) that corresponds to the root index of the diagram.

Definition at line 302 of file `KDChartAbstractDiagram.cpp`.

References d.

Referenced by `KDChart::AbstractDiagram::datasetBrushes()`, `KDChart::AbstractDiagram::datasetLabels()`, `KDChart::AbstractDiagram::datasetMarkers()`, `KDChart::AbstractDiagram::datasetPens()`, `KDChart::AbstractDiagram::itemRowLabels()`, `numberOfAbscissaSegments()`, `KDChart::LineDiagram::numberOfAbscissaSegments()`, `KDChart::BarDiagram::numberOfAbscissaSegments()`, `numberOfOrdinateSegments()`, `KDChart::LineDiagram::numberOfOrdinateSegments()`, `KDChart::BarDiagram::numberOfOrdinateSegments()`, `KDChart::AbstractDiagram::valueForCell()`, and `KDChart::LineDiagram::valueForCellTesting()`.

```
303 {
304     if ( !d->attributesModelRootIndex.isValid() )
305         d->attributesModelRootIndex = d->attributesModel->mapFromSource( rootIndex() );
306     return d->attributesModelRootIndex;
307 }
```

#### 9.45.4.6 **KDChart::CartesianAxisList** AbstractCartesianDiagram::axes () const [virtual, inherited]

##### Returns:

a list of all axes added to the diagram

Definition at line 110 of file KDChartAbstractCartesianDiagram.cpp.

References d.

Referenced by KDChart::Widget::setType(), and KDChart::CartesianCoordinatePlane::sharedAxisMasterPlane().

```
111 {
112     return d->axesList;
113 }
```

#### 9.45.4.7 **QBrush** AbstractDiagram::brush (const QModelIndex & *index*) const [inherited]

Retrieve the brush to be used, for painting the datapoint at the given index in the model.

##### Parameters:

*index* The index of the datapoint in the model.

##### Returns:

The brush to use for painting.

Definition at line 838 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::brush(), KDChart::AttributesModel::data(), KDChart::DatasetBrushRole, and KDChart::AbstractDiagram::datasetDimension().

```
839 {
840     if( datasetDimension() > 1 )
841         return brush( index.column() );
842     return qVariantValue<QBrush>(
843         attributesModel()->data(
844             attributesModel()->mapFromSource( index ),
845             DatasetBrushRole ) );
846 }
```

#### 9.45.4.8 **QBrush** AbstractDiagram::brush (int *dataset*) const [inherited]

Retrieve the brush to be used for the given dataset.

This will fall back automatically to what was set at model level, if there are no dataset specific settings.

##### Parameters:

*dataset* The dataset to retrieve the brush for.

##### Returns:

The brush to use for painting.

Definition at line 828 of file KDChartAbstractDiagram.cpp.

References [KDChart::AbstractDiagram::attributesModel\(\)](#), [KDChart::AbstractDiagram::columnToIndex\(\)](#), [KDChart::AttributesModel::data\(\)](#), [KDChart::DatasetBrushRole](#), and [KDChart::AbstractDiagram::datasetDimension\(\)](#).

```

829 {
830     if( datasetDimension() > 1 )
831         dataset /= datasetDimension();
832     return qVariantValue<QBrush>(
833         attributesModel()->data(
834             attributesModel()->mapFromSource( columnToIndex( dataset ) ),
835             DatasetBrushRole ) );
836 }
```

#### 9.45.4.9 QBrush AbstractDiagram::brush () const [inherited]

Retrieve the brush to be used for painting datapoints globally.

This will fall back automatically to the default settings, if there are no specific settings.

##### Returns:

The brush to use for painting.

Definition at line 822 of file KDChartAbstractDiagram.cpp.

References [KDChart::AbstractDiagram::attributesModel\(\)](#), [KDChart::AttributesModel::data\(\)](#), and [KDChart::DatasetBrushRole](#).

Referenced by [KDChart::AbstractDiagram::brush\(\)](#), [KDChart::TernaryPointDiagram::paint\(\)](#), [KDChart::TernaryLineDiagram::paint\(\)](#), [KDChart::RingDiagram::paint\(\)](#), [KDChart::PolarDiagram::paint\(\)](#), and [KDChart::AbstractDiagram::paintMarker\(\)](#).

```

823 {
824     return qVariantValue<QBrush>(
825         attributesModel()->data( DatasetBrushRole ) );
826 }
```

#### 9.45.4.10 const QPair< QPointF, QPointF > Plotter::calculateDataBoundaries () const [protected, virtual]

[reimplemented]

Implements [KDChart::AbstractDiagram](#).

Definition at line 335 of file KDChartPlotter.cpp.

References [KDChart::AbstractDiagram::checkInvariants\(\)](#), and [d](#).

```

336 {
337     if ( !checkInvariants( true ) )
338         return QPair< QPointF, QPointF >( QPointF( 0, 0 ), QPointF( 0, 0 ) );
339
340     // note: calculateDataBoundaries() is ignoring the hidden flags.
341     //       That's not a bug but a feature: Hiding data does not mean removing them.
342     // For totally removing data from KD Chart's view people can use e.g. a proxy model ...
343
344     // calculate boundaries for different line types Normal - Stacked - Percent - Default Normal
345     return d->implementor->calculateDataBoundaries();
346 }
```



#### 9.45.4.11 `bool AbstractDiagram::checkInvariants (bool justReturnTheStatus = false) const` `[protected, virtual, inherited]`

Definition at line 1060 of file KdChartAbstractDiagram.cpp.

References KdChart::AbstractDiagram::coordinatePlane().

Referenced by KdChart::RingDiagram::calculateDataBoundaries(), KdChart::PolarDiagram::calculateDataBoundaries(), calculateDataBoundaries(), KdChart::PieDiagram::calculateDataBoundaries(), KdChart::LineDiagram::calculateDataBoundaries(), KdChart::BarDiagram::calculateDataBoundaries(), KdChart::RingDiagram::paint(), KdChart::PolarDiagram::paint(), paint(), KdChart::PieDiagram::paint(), KdChart::LineDiagram::paint(), KdChart::BarDiagram::paint(), KdChart::AbstractDiagram::paintDataValueTexts(), KdChart::AbstractDiagram::paintMarker(), and KdChart::AbstractDiagram::paintMarkers().

```

1061 {
1062     if( ! justReturnTheStatus ){
1063         Q_ASSERT_X ( model(), "AbstractDiagram::checkInvariants()",
1064                     "There is no usable model set, for the diagram." );
1065
1066         Q_ASSERT_X ( coordinatePlane(), "AbstractDiagram::checkInvariants()",
1067                     "There is no usable coordinate plane set, for the diagram." );
1068     }
1069     return model() && coordinatePlane();
1070 }

```

#### 9.45.4.12 `Plotter * Plotter::clone () const` `[virtual]`

Creates an exact copy of this diagram.

Definition at line 69 of file KdChartPlotter.cpp.

References d, Plotter(), setType(), and type().

```

70 {
71     Plotter* newDiagram = new Plotter( new Private( *d ) );
72     newDiagram->setType( type() );
73     return newDiagram;
74 }

```

#### 9.45.4.13 `QModelIndex AbstractDiagram::columnToIndex (int column) const` `[protected, inherited]`

Definition at line 309 of file KdChartAbstractDiagram.cpp.

Referenced by KdChart::BarDiagram::barAttributes(), KdChart::AbstractDiagram::brush(), KdChart::AbstractDiagram::dataValueAttributes(), KdChart::AbstractDiagram::isHidden(), lineAttributes(), KdChart::LineDiagram::lineAttributes(), KdChart::AbstractDiagram::pen(), KdChart::AbstractPieDiagram::pieAttributes(), KdChart::BarDiagram::threeDBarAttributes(), threeDLineAttributes(), KdChart::LineDiagram::threeDLineAttributes(), and KdChart::AbstractPieDiagram::threeDPieAttributes().

```

310 {    // FIXME (Mirko): shouldn't this be headerData? instead of the index for the first row?
311     if( model() )
312         return QModelIndex( model()->index( 0, column, rootIndex() ) );
313     return QModelIndex();
314 }

```

#### 9.45.4.14 bool AbstractDiagram::compare (const AbstractDiagram \* other) const [inherited]

Returns true if both diagrams have the same settings.

Definition at line 129 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::allowOverlappingDataValueTexts(), KDChart::AbstractDiagram::antiAliasing(), KDChart::AbstractDiagram::attributesModel(), KDChart::AttributesModel::compare(), KDChart::AbstractDiagram::datasetDimension(), and KDChart::AbstractDiagram::percentMode().

```

130 {
131     if( other == this ) return true;
132     if( ! other ){
133         //qDebug() << "AbstractDiagram::compare() cannot compare to Null pointer";
134         return false;
135     }
136     /*
137     qDebug() << "\n                AbstractDiagram::compare() QAbstractScrollArea:";
138         // compare QAbstractScrollArea properties
139     qDebug() <<
140         ((horizontalScrollBarPolicy() == other->horizontalScrollBarPolicy()) &&
141         (verticalScrollBarPolicy() == other->verticalScrollBarPolicy()));
142     qDebug() << "AbstractDiagram::compare() QFrame:";
143         // compare QFrame properties
144     qDebug() <<
145         ((frameShadow() == other->frameShadow()) &&
146         (frameShape() == other->frameShape()) &&
147         (frameWidth() == other->frameWidth()) &&
148         (lineWidth() == other->lineWidth()) &&
149         (midLineWidth() == other->midLineWidth()));
150     qDebug() << "AbstractDiagram::compare() QAbstractItemView:";
151         // compare QAbstractItemView properties
152     qDebug() <<
153         ((alternatingRowColors() == other->alternatingRowColors()) &&
154         (hasAutoScroll() == other->hasAutoScroll()) &&
155 #if QT_VERSION > 0x040199
156         (dragDropMode() == other->dragDropMode()) &&
157         (dragDropOverwriteMode() == other->dragDropOverwriteMode()) &&
158         (horizontalScrollMode() == other->horizontalScrollMode()) &&
159         (verticalScrollMode() == other->verticalScrollMode()) &&
160 #endif
161         (dragEnabled() == other->dragEnabled()) &&
162         (editTriggers() == other->editTriggers()) &&
163         (iconSize() == other->iconSize()) &&
164         (selectionBehavior() == other->selectionBehavior()) &&
165         (selectionMode() == other->selectionMode()) &&
166         (showDropIndicator() == other->showDropIndicator()) &&
167         (tabKeyNavigation() == other->tabKeyNavigation()) &&
168         (textElideMode() == other->textElideMode()));
169     qDebug() << "AbstractDiagram::compare() AttributesModel: ";
170         // compare all of the properties stored in the attributes model
171     qDebug() << attributesModel()->compare( other->attributesModel() );
172     qDebug() << "AbstractDiagram::compare() own:";
173         // compare own properties
174     qDebug() <<
175         ((rootIndex().column() == other->rootIndex().column()) &&
176         (rootIndex().row() == other->rootIndex().row()) &&
177         (allowOverlappingDataValueTexts() == other->allowOverlappingDataValueTexts()) &&
178         (antiAliasing() == other->antiAliasing()) &&
179         (percentMode() == other->percentMode()) &&
180         (datasetDimension() == other->datasetDimension()));
181     */
182     return // compare QAbstractScrollArea properties
183         (horizontalScrollBarPolicy() == other->horizontalScrollBarPolicy()) &&

```

```

184         (verticalScrollBarPolicy() == other->verticalScrollBarPolicy()) &&
185         // compare QFrame properties
186         (frameShadow() == other->frameShadow()) &&
187         (frameShape() == other->frameShape()) &&
188 // frameWidth is a read-only property defined by the style, it should not be in here:
189         // (frameWidth() == other->frameWidth()) &&
190         (lineWidth() == other->lineWidth()) &&
191         (midLineWidth() == other->midLineWidth()) &&
192         // compare QAbstractItemView properties
193         (alternatingRowColors() == other->alternatingRowColors()) &&
194         (hasAutoScroll() == other->hasAutoScroll()) &&
195 #if QT_VERSION > 0x040199
196         (dragDropMode() == other->dragDropMode()) &&
197         (dragDropOverwriteMode() == other->dragDropOverwriteMode()) &&
198         (horizontalScrollMode() == other->horizontalScrollMode()) &&
199         (verticalScrollMode() == other->verticalScrollMode()) &&
200 #endif
201         (dragEnabled() == other->dragEnabled()) &&
202         (editTriggers() == other->editTriggers()) &&
203         (iconSize() == other->iconSize()) &&
204         (selectionBehavior() == other->selectionBehavior()) &&
205         (selectionMode() == other->selectionMode()) &&
206         (showDropIndicator() == other->showDropIndicator()) &&
207         (tabKeyNavigation() == other->tabKeyNavigation()) &&
208         (textElideMode() == other->textElideMode()) &&
209         // compare all of the properties stored in the attributes model
210         attributesModel()->compare( other->attributesModel() ) &&
211         // compare own properties
212         (rootIndex().column() == other->rootIndex().column()) &&
213         (rootIndex().row() == other->rootIndex().row()) &&
214         (allowOverlappingDataValueTexts() == other->allowOverlappingDataValueTexts()) &&
215         (antiAliasing() == other->antiAliasing()) &&
216         (percentMode() == other->percentMode()) &&
217         (datasetDimension() == other->datasetDimension());
218 }

```

#### 9.45.4.15 bool AbstractCartesianDiagram::compare (const [AbstractCartesianDiagram](#) \* other) const [inherited]

Returns true if both diagrams have the same settings.

Definition at line 46 of file KDChartAbstractCartesianDiagram.cpp.

References [KDChart::AbstractCartesianDiagram::referenceDiagram\(\)](#), and [KDChart::AbstractCartesianDiagram::referenceDiagramOffset\(\)](#).

```

47 {
48     if( other == this ) return true;
49     if( ! other ){
50         //qDebug() << "AbstractCartesianDiagram::compare() cannot compare to Null pointer";
51         return false;
52     }
53     /*
54     qDebug() << "\n                AbstractCartesianDiagram::compare() : ";
55     // compare own properties
56     qDebug() <<
57         ((referenceDiagram() == other->referenceDiagram()) &&
58         ((! referenceDiagram()) || (referenceDiagramOffset() == other->referenceDiagramOffset())));
59     */
60     return // compare the base class
61         ( static_cast<const AbstractDiagram*>(this)->compare( other ) ) &&
62         // compare own properties
63         (referenceDiagram() == other->referenceDiagram()) &&
64         ((! referenceDiagram()) || (referenceDiagramOffset() == other->referenceDiagramOffset()));

```

```
65 }
```

#### 9.45.4.16 bool `Plotter::compare (const Plotter * other) const`

Returns true if both diagrams have the same settings.

Definition at line 76 of file `KDChartPlotter.cpp`.

References `type()`.

```
77 {
78     if( other == this )
79         return true;
80     if( other == 0 )
81         return false;
82     return // compare the base class
83           ( static_cast< const AbstractCartesianDiagram* >( this )->compare( other ) ) &&
84           // compare own properties
85           ( type() == other->type() );
86 }
```

#### 9.45.4.17 [AbstractCoordinatePlane](#) \* `AbstractDiagram::coordinatePlane () const` [inherited]

The coordinate plane associated with the diagram.

This determines how coordinates in value space are mapped into pixel space. By default this is a [Cartesian-CoordinatePlane](#).

##### Returns:

The coordinate plane associated with the diagram.

Definition at line 220 of file `KDChartAbstractDiagram.cpp`.

References `d`.

Referenced by `KDChart::AbstractDiagram::checkInvariants()`, `KDChart::AbstractCartesianDiagram::layoutPlanes()`, `KDChart::PolarDiagram::paint()`, `KDChart::AbstractDiagram::paintData-ValueTexts()`, `KDChart::AbstractDiagram::paintMarkers()`, `KDChart::AbstractPolarDiagram::polar-CoordinatePlane()`, and `KDChart::AbstractCartesianDiagram::setCoordinatePlane()`.

```
221 {
222     return d->plane;
223 }
```

#### 9.45.4.18 const `QPair< QPointF, QPointF > AbstractDiagram::dataBoundaries () const` [inherited]

Return the bottom left and top right data point, that the diagram will display (unless the grid adjusts these values).

This method returns a cached result of calculations done by `calculateDataBoundaries`. Classes derived from [AbstractDiagram](#) must implement the `calculateDataBoundaries` function, to specify their own way of calculating the data boundaries. If derived classes want to force recalculation of the data boundaries, they can call `setDataBoundariesDirty()`

Returned value is in diagram coordinates.

Definition at line 225 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::calculateDataBoundaries(), and d.

Referenced by KDChart::CartesianCoordinatePlane::getRawDataBoundingRectFromDiagrams(), KDChart::PolarCoordinatePlane::layoutDiagrams(), paint(), KDChart::LineDiagram::paint(), and KDChart::BarDiagram::paint().

```

226 {
227     if( d->databoundariesDirty ){
228         d->databoundaries = calculateDataBoundaries ();
229         d->databoundariesDirty = false;
230     }
231     return d->databoundaries;
232 }
```

#### 9.45.4.19 void AbstractDiagram::dataChanged (const QModelIndex & *topLeft*, const QModelIndex & *bottomRight*) [virtual, inherited]

[reimplemented]

Definition at line 330 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::setDataBoundariesDirty().

```

332 {
333     Q_UNUSED( topLeft );
334     Q_UNUSED( bottomRight );
335     // We are still too dumb to do intelligent updates...
336     setDataBoundariesDirty();
337     scheduleDelayedItemsLayout();
338 }
```

#### 9.45.4.20 void KDChart::AbstractDiagram::dataHidden () [signal, inherited]

This signal is emitted, when the hidden status of at least one data cell was (un)set.

Referenced by KDChart::AbstractDiagram::setHidden().

#### 9.45.4.21 QList< QBrush > AbstractDiagram::datasetBrushes () const [inherited]

The set of dataset brushes currently used, for use in legends, etc.

##### Note:

Cell-level override brushes, if set, take precedence over the dataset values, so you might need to check these too, in order to find the brush, that is used for a single cell.

##### Returns:

The current set of dataset brushes.

Definition at line 1018 of file KDChartAbstractDiagram.cpp.

References `KDChart::AbstractDiagram::attributesModel()`, `KDChart::AbstractDiagram::attributesModelRootIndex()`, `KDChart::AttributesModel::columnCount()`, `KDChart::DatasetBrushRole`, and `KDChart::AbstractDiagram::datasetDimension()`.

Referenced by `KDChart::Legend::setBrushesFromDiagram()`.

```

1019 {
1020     QList<QBrush> ret;
1021     if( model() == 0 )
1022         return ret;
1023
1024     const int columnCount = attributesModel()->columnCount(attributesModelRootIndex());
1025     for( int i = 0; i < columnCount / datasetDimension(); ++i )
1026         ret << qVariantValue<QBrush>( attributesModel()->headerData( i, Qt::Vertical, DatasetBrushRole ));
1027
1028     return ret;
1029 }
```

#### 9.45.4.22 `int AbstractDiagram::datasetDimension () const` [inherited]

The dataset dimension of a diagram determines, how many value dimensions it expects each datapoint to have.

For each dimension it will expect one column of values in the model. If the dimensionality is 1, automatic values will be used for the abscissa.

For example a diagram with the default dimension of 1, will have one column per datapoint (the y values) and will use automatic values for the x axis (1, 2, 3, ... n). If the dimension is 2, the diagram will use the first, (and the third, fifth, etc) columns as X values, and the second, (and the fourth, sixth, etc) column as Y values.

##### Returns:

The dataset dimension of the diagram.

Definition at line 1072 of file `KDChartAbstractDiagram.cpp`.

References `d`.

Referenced by `KDChart::AbstractDiagram::brush()`, `KDChart::AbstractDiagram::compare()`, `KDChart::AbstractDiagram::datasetBrushes()`, `KDChart::AbstractDiagram::datasetLabels()`, `KDChart::AbstractDiagram::datasetMarkers()`, `KDChart::AbstractDiagram::datasetPens()`, `KDChart::LineDiagram::getCellValues()`, `KDChart::CartesianCoordinatePlane::getDataDimensionsList()`, `KDChart::TernaryPointDiagram::paint()`, `KDChart::TernaryLineDiagram::paint()`, `KDChart::AbstractDiagram::paintDataValueTexts()`, `KDChart::AbstractDiagram::paintMarkers()`, `KDChart::AbstractDiagram::pen()`, `KDChart::AbstractDiagram::setPen()`, `setType()`, and `KDChart::LineDiagram::setType()`.

```

1073 {
1074     return d->datasetDimension;
1075 }
```

#### 9.45.4.23 `QStringList AbstractDiagram::datasetLabels () const` [inherited]

The set of dataset labels currently displayed, for use in legends, etc.

##### Returns:

The set of dataset labels currently displayed.

Definition at line 1005 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::attributesModelRootIndex(), KDChart::AttributesModel::columnCount(), KDChart::AbstractDiagram::datasetDimension(), and KDChart::AttributesModel::headerData().

```

1006 {
1007     QStringList ret;
1008     if( model() == 0 )
1009         return ret;
1010
1011     const int columnCount = attributesModel()->columnCount(attributesModelRootIndex());
1012     for( int i = 0; i < columnCount / datasetDimension(); ++i )
1013         ret << attributesModel()->headerData( i, Qt::Horizontal, Qt::DisplayRole ).toString();
1014
1015     return ret;
1016 }
```

#### 9.45.4.24 QList< [MarkerAttributes](#) > AbstractDiagram::datasetMarkers () const [inherited]

The set of dataset markers currently used, for use in legends, etc.

##### Note:

Cell-level override markers, if set, take precedence over the dataset values, so you might need to check these too, in order to find the marker, that is shown for a single cell.

##### Returns:

The current set of dataset brushes.

Definition at line 1044 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::attributesModelRootIndex(), KDChart::AttributesModel::columnCount(), KDChart::AbstractDiagram::datasetDimension(), and KDChart::DataValueLabelAttributesRole.

```

1045 {
1046     QList<MarkerAttributes> ret;
1047     if( model() == 0 )
1048         return ret;
1049
1050     const int columnCount = attributesModel()->columnCount(attributesModelRootIndex());
1051     for( int i = 0; i < columnCount / datasetDimension(); ++i )
1052     {
1053         const DataValueAttributes a =
1054             qVariantValue<DataValueAttributes>( attributesModel()->headerData( i, Qt::Vertical, DataValueLabelAttributesRole ));
1055         ret << a.markerAttributes();
1056     }
1057     return ret;
1058 }
```

#### 9.45.4.25 QList< QPen > AbstractDiagram::datasetPens () const [inherited]

The set of dataset pens currently used, for use in legends, etc.

**Note:**

Cell-level override pens, if set, take precedence over the dataset values, so you might need to check these too, in order to find the pens, that is used for a single cell.

**Returns:**

The current set of dataset pens.

Definition at line 1031 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::attributesModelRootIndex(), KDChart::AttributesModel::columnCount(), KDChart::AbstractDiagram::datasetDimension(), and KDChart::DatasetPenRole.

```

1032 {
1033     QList<QPen> ret;
1034     if( model() == 0 )
1035         return ret;
1036
1037     const int columnCount = attributesModel()->columnCount(attributesModelRootIndex());
1038     for( int i = 0; i < columnCount / datasetDimension(); ++i )
1039         ret << qVariantValue<QPen>( attributesModel()->headerData( i, Qt::Vertical, DatasetPenRole ) );
1040
1041     return ret;
1042 }
```

#### 9.45.4.26 [DataValueAttributes](#) AbstractDiagram::dataValueAttributes (const QModelIndex & *index*) const [inherited]

Retrieve the [DataValueAttributes](#) for the given index.

This will fall back automatically to what was set at dataset or model level, if there are no datapoint specific settings.

**Parameters:**

*index* The datapoint to retrieve the attributes for.

**Returns:**

The [DataValueAttributes](#) for the given index.

Definition at line 421 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AttributesModel::data(), and KDChart::DataValueLabelAttributesRole.

```

422 {
423     return qVariantValue<DataValueAttributes>(
424         attributesModel()->data( attributesModel()->mapFromSource(index),
425                                 KDChart::DataValueLabelAttributesRole ) );
426 }
```

#### 9.45.4.27 [DataValueAttributes](#) AbstractDiagram::dataValueAttributes (int *column*) const [inherited]

Retrieve the [DataValueAttributes](#) for the given dataset.

This will fall back automatically to what was set at model level, if there are no dataset specific settings.



**Parameters:**

*column* The dataset to retrieve the attributes for.

**Returns:**

The [DataValueAttributes](#) for the given dataset.

Definition at line 414 of file KDChartAbstractDiagram.cpp.

References [KDChart::AbstractDiagram::attributesModel\(\)](#), [KDChart::AbstractDiagram::columnToIndex\(\)](#), [KDChart::AttributesModel::data\(\)](#), and [KDChart::DataValueLabelAttributesRole](#).

```
415 {
416     return qVariantValue<DataValueAttributes>(
417         attributesModel()->data( attributesModel()->mapFromSource(columnToIndex( column )),
418         KDChart::DataValueLabelAttributesRole ) );
419 }
```

**9.45.4.28 [DataValueAttributes](#) AbstractDiagram::dataValueAttributes () const** [inherited]

Retrieve the [DataValueAttributes](#) specified globally.

This will fall back automatically to the default settings, if there are no specific settings.

**Returns:**

The global [DataValueAttributes](#).

Definition at line 408 of file KDChartAbstractDiagram.cpp.

References [KDChart::AbstractDiagram::attributesModel\(\)](#), [KDChart::DataValueLabelAttributesRole](#), and [KDChart::AttributesModel::modelData\(\)](#).

Referenced by [KDChart::AbstractDiagram::paintDataValueText\(\)](#), [KDChart::AbstractDiagram::paintMarker\(\)](#), and [KDChart::TernaryLineDiagram::TernaryLineDiagram\(\)](#).

```
409 {
410     return qVariantValue<DataValueAttributes>(
411         attributesModel()->modelData( KDChart::DataValueLabelAttributesRole ) );
412 }
```

**9.45.4.29 void AbstractDiagram::doItemsLayout ()** [virtual, inherited]

[reimplemented]

Definition at line 321 of file KDChartAbstractDiagram.cpp.

References [d](#), and [KDChart::AbstractDiagram::update\(\)](#).

```
322 {
323     if ( d->plane ) {
324         d->plane->layoutDiagrams();
325         update();
326     }
327     QAbstractItemView::doItemsLayout();
328 }
```

**9.45.4.30 int AbstractDiagram::horizontalOffset () const** [virtual, inherited]

[reimplemented]

Definition at line 950 of file KDChartAbstractDiagram.cpp.

```
951 { return 0; }
```

**9.45.4.31 QModelIndex AbstractDiagram::indexAt (const QPoint & point) const** [virtual, inherited]

[reimplemented]

Definition at line 1098 of file KDChartAbstractDiagram.cpp.

References d.

```
1099 {
1100     return d->indexAt( point );
1101 }
```

**9.45.4.32 QModelIndexList AbstractDiagram::indexesAt (const QPoint & point) const** [inherited]

This method is added alongside with indexAt from QAIM, since in kdchart multiple indexes can be displayed at the same spot.

Definition at line 1103 of file KDChartAbstractDiagram.cpp.

References d.

```
1104 {
1105     return d->indexesAt( point );
1106 }
```

**9.45.4.33 bool AbstractDiagram::isHidden (const QModelIndex & index) const** [inherited]

Retrieve the hidden status for the given index.

This will fall back automatically to what was set at dataset or diagram level, if there are no datapoint specific settings.

**Parameters:**

*index* The datapoint to retrieve the hidden status for.

**Returns:**

The hidden status for the given index.

Definition at line 380 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AttributesModel::data(), and KDChart::DataHiddenRole.

```

381 {
382     return qVariantValue<bool>(
383         attributesModel()->data(
384             attributesModel()->mapFromSource(index),
385             DataHiddenRole ) );
386 }

```

#### 9.45.4.34 bool AbstractDiagram::isHidden (int *column*) const [inherited]

Retrieve the hidden status for the given dataset.

This will fall back automatically to what was set at diagram level, if there are no dataset specific settings.

##### Parameters:

*column* The dataset to retrieve the hidden status for.

##### Returns:

The hidden status for the given dataset.

Definition at line 373 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::columnToIndex(), KDChart::AttributesModel::data(), and KDChart::DataHiddenRole.

```

374 {
375     return qVariantValue<bool>(
376         attributesModel()->data(
377             attributesModel()->mapFromSource(columnToIndex( column ) ),
378             DataHiddenRole ) );
379 }

```

#### 9.45.4.35 bool AbstractDiagram::isHidden () const [inherited]

Retrieve the hidden status specified globally.

This will fall back automatically to the default settings (= not hidden), if there are no specific settings.

##### Returns:

The global hidden status.

Definition at line 367 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DataHiddenRole, and KDChart::AttributesModel::modelData().

Referenced by KDChart::LineDiagram::valueForCellTesting().

```

368 {
369     return qVariantValue<bool>(
370         attributesModel()->modelData( DataHiddenRole ) );
371 }

```

**9.45.4.36** `bool AbstractDiagram::isIndexHidden (const QModelIndex & index) const`  
[virtual, inherited]

[reimplemented]

Definition at line 956 of file KDChartAbstractDiagram.cpp.

```
957 { return true; }
```

**9.45.4.37** `QStringList AbstractDiagram::itemRowLabels () const` [inherited]

The set of item row labels currently displayed, for use in Abscissa axes, etc.

#### Returns:

The set of item row labels currently displayed.

Definition at line 989 of file KDChartAbstractDiagram.cpp.

References `KDChart::AbstractDiagram::attributesModel()`, `KDChart::AbstractDiagram::attributesModelRootIndex()`, `KDChart::AttributesModel::rowCount()`, `KDChart::AbstractDiagram::unitPrefix()`, and `KDChart::AbstractDiagram::unitSuffix()`.

```
990 {
991     QStringList ret;
992     if( model() ){
993         qDebug() << "AbstractDiagram::itemRowLabels(): " << attributesModel()->rowCount(attributesModelRootIndex());
994         const int rowCount = attributesModel()->rowCount(attributesModelRootIndex());
995         for( int i = 0; i < rowCount; ++i ){
996             qDebug() << "item row label: " << attributesModel()->headerData( i, Qt::Vertical, Qt::DisplayRole );
997             ret << unitPrefix( i, Qt::Horizontal, true ) +
998                 attributesModel()->headerData( i, Qt::Vertical, Qt::DisplayRole ).toString() +
999                 unitSuffix( i, Qt::Horizontal, true );
1000         }
1001     }
1002     return ret;
1003 }
```

**9.45.4.38** `void KDChart::AbstractDiagram::layoutChanged (AbstractDiagram *)` [signal, inherited]

Diagrams are supposed to emit this signal, when the layout of one of their element changes.

Layouts can change, for example, when axes are added or removed, or when the configuration was changed in a way that the axes or the diagram itself are displayed in a different geometry. Changes in the diagrams coordinate system also result in the [layoutChanged\(\)](#) signal being emitted.

Referenced by `KDChart::AbstractDiagram::setDatasetDimension()`, `KDChart::AbstractPieDiagram::setPieAttributes()`, `KDChart::AbstractPieDiagram::setThreeDPieAttributes()`, `setType()`, `KDChart::LineDiagram::setType()`, and `KDChart::BarDiagram::setType()`.

**9.45.4.39** `void KDChart::AbstractCartesianDiagram::layoutPlanes ()` [virtual, inherited]

Triggers layouting of all coordinate planes on the current chart.

Normally you don't need to call this method. It's handled automatically for you.

Definition at line 115 of file KDChartAbstractCartesianDiagram.cpp.

References KDChart::AbstractDiagram::coordinatePlane(), and KDChart::AbstractCoordinatePlane::layoutPlanes().

Referenced by KDChart::AbstractCartesianDiagram::addAxis(), and KDChart::AbstractCartesianDiagram::takeAxis().

```

116 {
117     //qDebug() << "KDChart::AbstractCartesianDiagram::layoutPlanes()";
118     AbstractCoordinatePlane* plane = coordinatePlane();
119     if( plane ){
120         plane->layoutPlanes();
121         //qDebug() << "KDChart::AbstractCartesianDiagram::layoutPlanes() OK";
122     }
123 }
```

#### 9.45.4.40 **LineAttributes** Plotter::lineAttributes (const QModelIndex & *index*) const

##### Returns:

the line attribute set of the model index *index*

Definition at line 209 of file KDChartPlotter.cpp.

References d, and KDChart::LineAttributesRole.

```

211 {
212     return qVariantValue<LineAttributes>(
213         d->attributesModel->data(
214             d->attributesModel->mapFromSource(index),
215             KDChart::LineAttributesRole ) );
216 }
```

#### 9.45.4.41 **LineAttributes** Plotter::lineAttributes (int *column*) const

##### Returns:

the line attribute set of data set *column*

Definition at line 198 of file KDChartPlotter.cpp.

References KDChart::AbstractDiagram::columnToIndex(), d, and KDChart::LineAttributesRole.

```

199 {
200     return qVariantValue<LineAttributes>(
201         d->attributesModel->data(
202             d->attributesModel->mapFromSource( columnToIndex( column ) ),
203             KDChart::LineAttributesRole ) );
204 }
```

#### 9.45.4.42 **LineAttributes** Plotter::lineAttributes () const

##### Returns:

the global line attribute set

Definition at line 189 of file KDChartPlotter.cpp.

References `d`, and `KDChart::LineAttributesRole`.

```
190 {
191     return qVariantValue<LineAttributes>(
192         d->attributesModel->data( KDChart::LineAttributesRole ) );
193 }
```

#### 9.45.4.43 void KDChart::AbstractDiagram::modelsChanged () [signal, inherited]

This signal is emitted, when either the model or the [AttributesModel](#) is replaced.

Referenced by `KDChart::AbstractDiagram::setAttributesModel()`, and `KDChart::AbstractDiagram::setModel()`.

#### 9.45.4.44 QModelIndex AbstractDiagram::moveCursor (CursorAction *cursorAction*, Qt::KeyboardModifiers *modifiers*) [virtual, inherited]

[reimplemented]

Definition at line 947 of file KDChartAbstractDiagram.cpp.

```
948 { return QModelIndex(); }
```

#### 9.45.4.45 const int Plotter::numberOfAbscissaSegments () const [virtual]

Implements [KDChart::AbstractCartesianDiagram](#).

Definition at line 385 of file KDChartPlotter.cpp.

References `KDChart::AbstractDiagram::attributesModelRootIndex()`, and `d`.

```
386 {
387     return d->attributesModel->rowCount( attributesModelRootIndex() );
388 }
```

#### 9.45.4.46 const int Plotter::numberOfOrdinateSegments () const [virtual]

Implements [KDChart::AbstractCartesianDiagram](#).

Definition at line 390 of file KDChartPlotter.cpp.

References `KDChart::AbstractDiagram::attributesModelRootIndex()`, and `d`.

```
391 {
392     return d->attributesModel->columnCount( attributesModelRootIndex() );
393 }
```

**9.45.4.47 void Plotter::paint (PaintContext \*paintContext) [protected, virtual]**

Draw the diagram contents to the rectangle and painter, that are passed in as part of the paint context.

**Parameters:**

*paintContext* All information needed for painting.

Implements [KDChart::AbstractDiagram](#).

Definition at line 358 of file KDChartPlotter.cpp.

References [KDChart::AbstractDiagram::checkInvariants\(\)](#), [KDChart::PaintContext::coordinatePlane\(\)](#), [d](#), [KDChart::AbstractDiagram::dataBoundaries\(\)](#), [KDChart::PaintContext::painter\(\)](#), [KDChart::PaintContext::setCoordinatePlane\(\)](#), and [KDChart::AbstractCoordinatePlane::sharedAxisMasterPlane\(\)](#).

Referenced by [paintEvent\(\)](#).

```

359 {
360     // note: Not having any data model assigned is no bug
361     //         but we can not draw a diagram then either.
362     if ( !checkInvariants( true ) ) return;
363     if ( !AbstractGrid::isBoundariesValid(dataBoundaries()) ) return;
364     const PainterSaver p( ctx->painter() );
365     if( model()->rowCount() == 0 || model()->columnCount() == 0 )
366         return; // nothing to paint for us
367
368     AbstractCoordinatePlane* const plane = ctx->coordinatePlane();
369     ctx->setCoordinatePlane( plane->sharedAxisMasterPlane( ctx->painter() ) );
370
371
372     // paint different line types Normal - Stacked - Percent - Default Normal
373     d->implementor->paint( ctx );
374
375     ctx->setCoordinatePlane( plane );
376 }
```

**9.45.4.48 void AbstractDiagram::paintDataValueText (QPainter \*painter, const QModelIndex &index, const QPointF &pos, double value) [inherited]**

Definition at line 468 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::DataValueAttributes::dataLabel\(\)](#), [KDChart::AbstractDiagram::dataValueAttributes\(\)](#), [KDChart::DataValueAttributes::decimalDigits\(\)](#), [KDChart::DataValueAttributes::isVisible\(\)](#), [KDChartEnums::MeasureOrientationMinimum](#), [KDChart::DataValueAttributes::position\(\)](#), [KDChart::DataValueAttributes::prefix\(\)](#), [KDChart::DataValueAttributes::suffix\(\)](#), and [KDChart::DataValueAttributes::textAttributes\(\)](#).

Referenced by [KDChart::RingDiagram::paint\(\)](#), [KDChart::PolarDiagram::paint\(\)](#), and [KDChart::AbstractDiagram::paintDataValueTexts\(\)](#).

```

472 {
473     // paint one data series
474     const DataValueAttributes a( dataValueAttributes(index) );
475     if ( !a.isVisible() ) return;
476
477     // handle decimal digits
478     int decimalDigits = a.decimalDigits();
479     int decimalPos = QString::number( value ).indexOf( QLatin1Char( '.' ) );
480     QString roundedValue;
```

```

481     if ( a.dataLabel().isNull() ) {
482         if ( decimalPos > 0 && value != 0 )
483             roundedValue = roundValues ( value, decimalPos, decimalDigits );
484         else
485             roundedValue = QString::number( value );
486     } else
487         roundedValue = a.dataLabel();
488     // handle prefix and suffix
489     if ( !a.prefix().isNull() )
490         roundedValue.prepend( a.prefix() );
491
492     if ( !a.suffix().isNull() )
493         roundedValue.append( a.suffix() );
494
495     const TextAttributes ta( a.textAttributes() );
496     // FIXME draw the non-text bits, background, etc
497     if ( ta.isVisible() ) {
498
499         QPointF pt( pos );
500         /* for debugging:
501         PainterSaver painterSaver( painter );
502         painter->setPen( Qt::black );
503         painter->drawLine( pos - QPointF( 1,1), pos + QPointF( 1,1) );
504         painter->drawLine( pos - QPointF(-1,1), pos + QPointF(-1,1) );
505         */
506
507         QTextDocument doc;
508         if( Qt::mightBeRichText( roundedValue ) )
509             doc.setHtml( roundedValue );
510         else
511             doc.setPlainText( roundedValue );
512
513         const RelativePosition relPos( a.position( value >= 0.0 ) );
514         const Qt::Alignment alignBottomLeft = Qt::AlignBottom | Qt::AlignLeft;
515         const QFont calculatedFont( ta.calculatedFont( d->plane, KDChartEnums::MeasureOrientationMinimum ) );
516         const QRectF boundRect( d->cachedFontMetrics( calculatedFont, painter->device() )->boundingRect( roundedValue ) );
517
518         // To place correctly
519         pt.ry() -= boundRect.height();
520
521         // qDebug() << "calculatedFont's point size:" << calculatedFont.pointSizeF();
522         // adjust the text start point position, if alignment is not Bottom/Left
523         if( (relPos.alignment() & alignBottomLeft) != alignBottomLeft ){
524             if( relPos.alignment() & Qt::AlignRight )
525                 pt.rx() -= boundRect.width();
526             else if( relPos.alignment() & Qt::AlignHCenter )
527                 pt.rx() -= 0.5 * boundRect.width();
528
529             if( relPos.alignment() & Qt::AlignTop )
530                 pt.ry() += boundRect.height();
531             else if( relPos.alignment() & Qt::AlignVCenter )
532                 pt.ry() += 0.5 * boundRect.height();
533         }
534
535         // FIXME draw the non-text bits, background, etc
536
537         if ( a.showRepetitiveDataLabels() ||
538             pos.x() <= d->lastX ||
539             d->lastRoundedValue != roundedValue ) {
540             d->lastRoundedValue = roundedValue;
541             d->lastX = pos.x();
542             PainterSaver painterSaver( painter );
543
544             doc.setDefaultFont( calculatedFont );
545             QAbstractTextDocumentLayout::PaintContext context;
546             context.palette = palette();
547             context.palette.setColor( QPalette::Text, ta.pen().color() );

```



```

548
549         painter->translate( pt );
550         painter->rotate( ta.rotation() );
551
552         QAbstractTextDocumentLayout* layout = doc.documentLayout();
553         layout->draw( painter, context );
554     }
555 }
556 }

```

#### 9.45.4.49 void AbstractDiagram::paintDataValueTexts (QPainter \* *painter*) [protected, virtual, inherited]

Definition at line 584 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::checkInvariants(), KDChart::AbstractDiagram::coordinatePlane(), KDChart::AbstractDiagram::datasetDimension(), KDChart::AbstractDiagram::paintDataValueText(), and KDChart::AbstractCoordinatePlane::translate().

```

585 {
586     if ( !checkInvariants() ) return;
587     const int rowCount = model()->rowCount(rootIndex());
588     const int columnCount = model()->columnCount(rootIndex());
589     for ( int i=datasetDimension()-1; i<columnCount; i += datasetDimension() ) {
590         for ( int j=0; j< rowCount; ++j ) {
591             const QModelIndex index = model()->index( j, i, rootIndex() );
592             double value = model()->data( index ).toDouble();
593             const QPointF pos = coordinatePlane()->translate( QPointF( j, value ) );
594             paintDataValueText( painter, index, pos, value );
595         }
596     }
597 }

```

#### 9.45.4.50 void Plotter::paintEvent (QPaintEvent \*) [protected]

Definition at line 349 of file KDChartPlotter.cpp.

References paint(), KDChart::PaintContext::setPainter(), and KDChart::PaintContext::setRectangle().

```

350 {
351     QPainter painter ( viewport() );
352     PaintContext ctx;
353     ctx.setPainter ( &painter );
354     ctx.setRectangle ( QRectF ( 0, 0, width(), height() ) );
355     paint ( &ctx );
356 }

```

#### 9.45.4.51 void AbstractDiagram::paintMarker (QPainter \* *painter*, const QModelIndex & *index*, const QPointF & *pos*) [inherited]

Definition at line 600 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::brush(), KDChart::AbstractDiagram::checkInvariants(), d, KDChart::AbstractDiagram::dataValueAttributes(), KDChart::MarkerAttributes::isVisible(), KDChart::DataValueAttributes::isVisible(), KDChart::DataValueAttributes::markerAttributes(), KDChart::MarkerAttributes::markerColor(), KDChart::MarkerAttributes::markerSize(), KDChart::AbstractDiagram::paintMarker(), and KDChart::MarkerAttributes::pen().

```

603 {
604     if ( !checkInvariants() ) return;
605     DataValueAttributes a = dataValueAttributes(index);
606     if ( !a.isVisible() ) return;
607     const MarkerAttributes &ma = a.markerAttributes();
608     if ( !ma.isVisible() ) return;
609
610     PainterSaver painterSaver( painter );
611     QSizeF maSize( ma.markerSize() );
612     QBrush indexBrush( brush( index ) );
613     QPen indexPen( ma.pen() );
614     if ( ma.markerColor().isValid() )
615         indexBrush.setColor( ma.markerColor() );
616
617     paintMarker( painter, ma, indexBrush, indexPen, pos, maSize );
618
619     // workaround: BC cannot be changed, otherwise we would pass the
620     // index down to next-lower paintMarker function. So far, we
621     // basically save a circle of radius maSize at pos in the
622     // reverseMapper. This means that ^^ this version of paintMarker
623     // needs to be called to reverse-map the marker.
624     d->reverseMapper.addCircle( index.row(), index.column(), pos, 2 * maSize );
625 }

```

**9.45.4.52 void AbstractDiagram::paintMarker** (QPainter \* *painter*, const [MarkerAttributes](#) & *markerAttributes*, const QBrush & *brush*, const QPen &, const QPointF & *point*, const QSizeF & *size*) [virtual, inherited]

Definition at line 627 of file KDChartAbstractDiagram.cpp.

References KDChart::MarkerAttributes::Marker1Pixel, KDChart::MarkerAttributes::Marker4Pixels, KDChart::MarkerAttributes::MarkerCircle, KDChart::MarkerAttributes::MarkerCross, KDChart::MarkerAttributes::MarkerDiamond, KDChart::MarkerAttributes::MarkerFastCross, KDChart::MarkerAttributes::MarkerRing, KDChart::MarkerAttributes::MarkerSquare, and KDChart::MarkerAttributes::markerStyle().

Referenced by KDChart::TernaryPointDiagram::paint(), KDChart::TernaryLineDiagram::paint(), KDChart::MarkerLayoutItem::paintIntoRect(), KDChart::AbstractDiagram::paintMarker(), and KDChart::AbstractDiagram::paintMarkers().

```

633 {
634     const QPen oldPen( painter->pen() );
635     // Pen is used to paint 4Pixels - 1 Pixel - Ring and FastCross types.
636     // make sure to use the brush color - see above in those cases.
637     const bool isFourPixels = (markerAttributes.markerStyle() == MarkerAttributes::Marker4Pixels);
638     if( isFourPixels || (markerAttributes.markerStyle() == MarkerAttributes::Marker1Pixel) ){
639         // for high-performance point charts with tiny point markers:
640         painter->setPen( QPen( brush.color().light() ) );
641         if( isFourPixels ){
642             const qreal x = pos.x();
643             const qreal y = pos.y();
644             painter->drawLine( QPointF(x-1.0,y-1.0),
645                               QPointF(x+1.0,y-1.0) );
646             painter->drawLine( QPointF(x-1.0,y),
647                               QPointF(x+1.0,y) );
648             painter->drawLine( QPointF(x-1.0,y+1.0),
649                               QPointF(x+1.0,y+1.0) );
650         }
651         painter->drawPoint( pos );
652     }else{
653         PainterSaver painterSaver( painter );
654         // we only a solid line surrounding the markers

```

```

655     QPen painterPen( pen );
656     painterPen.setStyle( Qt::SolidLine );
657     painter->setPen( painterPen );
658     painter->setBrush( brush );
659     painter->setRenderHint ( QPainter::Antialiasing );
660     painter->translate( pos );
661     switch ( markerAttributes.markerStyle() ) {
662     case MarkerAttributes::MarkerCircle:
663         painter->drawEllipse( QRectF( 0 - maSize.height()/2, 0 - maSize.width()/2,
664             maSize.height(), maSize.width() ) );
665         break;
666     case MarkerAttributes::MarkerSquare:
667     {
668         QRectF rect( 0 - maSize.width()/2, 0 - maSize.height()/2,
669             maSize.width(), maSize.height() );
670         painter->drawRect( rect );
671         break;
672     }
673     case MarkerAttributes::MarkerDiamond:
674     {
675         QVector <QPointF > diamondPoints;
676         QPointF top, left, bottom, right;
677         top = QPointF( 0, 0 - maSize.height()/2 );
678         left = QPointF( 0 - maSize.width()/2, 0 );
679         bottom = QPointF( 0, maSize.height()/2 );
680         right = QPointF( maSize.width()/2, 0 );
681         diamondPoints << top << left << bottom << right;
682         painter->drawPolygon( diamondPoints );
683         break;
684     }
685     // both handled on top of the method:
686     case MarkerAttributes::Marker1Pixel:
687     case MarkerAttributes::Marker4Pixels:
688         break;
689     case MarkerAttributes::MarkerRing:
690     {
691         painter->setPen( QPen( brush.color() ) );
692         painter->setBrush( Qt::NoBrush );
693         painter->drawEllipse( QRectF( 0 - maSize.height()/2, 0 - maSize.width()/2,
694             maSize.height(), maSize.width() ) );
695         break;
696     }
697     case MarkerAttributes::MarkerCross:
698     {
699         QRectF rect( maSize.width()*-0.5, maSize.height()*-0.2,
700             maSize.width(), maSize.height()*0.4 );
701         painter->drawRect( rect );
702         rect.setTopLeft(QPointF( maSize.width()*-0.2, maSize.height()*-0.5 ));
703         rect.setSize(QSizeF( maSize.width()*0.4, maSize.height() ));
704         painter->drawRect( rect );
705         break;
706     }
707     case MarkerAttributes::MarkerFastCross:
708     {
709         QPointF left, right, top, bottom;
710         left = QPointF( -maSize.width()/2, 0 );
711         right = QPointF( maSize.width()/2, 0 );
712         top = QPointF( 0, -maSize.height()/2 );
713         bottom= QPointF( 0, maSize.height()/2 );
714         painter->setPen( QPen( brush.color() ) );
715         painter->drawLine( left, right );
716         painter->drawLine( top, bottom );
717         break;
718     }
719     default:
720         Q_ASSERT_X ( false, "paintMarkers()",
721             "Type item does not match a defined Marker Type." );

```

```

722     }
723 }
724 painter->setPen( oldPen );
725 }

```

#### 9.45.4.53 void AbstractDiagram::paintMarkers (QPainter \* *painter*) [protected, virtual, inherited]

Definition at line 727 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::checkInvariants(), KDChart::AbstractDiagram::coordinatePlane(), KDChart::AbstractDiagram::datasetDimension(), KDChart::AbstractDiagram::paintMarker(), and KDChart::AbstractCoordinatePlane::translate().

```

728 {
729     if ( !checkInvariants() ) return;
730     const int rowCount = model()->rowCount(rootIndex());
731     const int columnCount = model()->columnCount(rootIndex());
732     for ( int i=datasetDimension()-1; i<columnCount; i += datasetDimension() ) {
733         for ( int j=0; j< rowCount; ++j ) {
734             const QModelIndex index = model()->index( j, i, rootIndex() );
735             double value = model()->data( index ).toDouble();
736             const QPointF pos = coordinatePlane()->translate( QPointF( j, value ) );
737             paintMarker( painter, index, pos );
738         }
739     }
740 }

```

#### 9.45.4.54 QPen AbstractDiagram::pen (const QModelIndex & *index*) const [inherited]

Retrieve the pen to be used, for painting the datapoint at the given index in the model.

##### Parameters:

***index*** The index of the datapoint in the model.

##### Returns:

The pen to use for painting.

Definition at line 788 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AttributesModel::data(), KDChart::AbstractDiagram::datasetDimension(), KDChart::DatasetPenRole, and KDChart::AbstractDiagram::pen().

```

789 {
790     if( datasetDimension() > 1 )
791         return pen( index.column() );
792     return qVariantValue<QPen>(
793         attributesModel()->data(
794             attributesModel()->mapFromSource( index ),
795             DatasetPenRole ) );
796 }

```

**9.45.4.55 QPen AbstractDiagram::pen (int *dataset*) const** [inherited]

Retrieve the pen to be used for the given dataset.

This will fall back automatically to what was set at model level, if there are no dataset specific settings.

**Parameters:**

*dataset* The dataset to retrieve the pen for.

**Returns:**

The pen to use for painting.

Definition at line 777 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::columnToIndex(), KDChart::AttributesModel::data(), KDChart::AbstractDiagram::datasetDimension(), and KDChart::DatasetPenRole.

```

778 {
779     if( datasetDimension() > 1 )
780         dataset /= datasetDimension();
781
782     return qVariantValue<QPen>(
783         attributesModel()->data(
784             attributesModel()->mapFromSource( columnToIndex( dataset ) ),
785             DatasetPenRole ) );
786 }
```

**9.45.4.56 QPen AbstractDiagram::pen () const** [inherited]

Retrieve the pen to be used for painting datapoints globally.

This will fall back automatically to the default settings, if there are no specific settings.

**Returns:**

The pen to use for painting.

Definition at line 771 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AttributesModel::data(), and KDChart::DatasetPenRole.

Referenced by KDChart::TernaryPointDiagram::paint(), KDChart::TernaryLineDiagram::paint(), and KDChart::AbstractDiagram::pen().

```

772 {
773     return qVariantValue<QPen>(
774         attributesModel()->data( DatasetPenRole ) );
775 }
```

**9.45.4.57 bool AbstractDiagram::percentMode () const** [inherited]

Definition at line 462 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by `KDChart::AbstractDiagram::compare()`, and `KDChart::CartesianCoordinatePlane::getDataDimensionsList()`.

```
463 {
464     return d->percent;
465 }
```

#### 9.45.4.58 `void KDChart::AbstractDiagram::propertiesChanged ()` [signal, inherited]

Emitted upon change of a property of the Diagram.

Referenced by `resetLineAttributes()`, `KDChart::LineDiagram::resetLineAttributes()`, `KDChart::AbstractDiagram::setAllowOverlappingDataValueTexts()`, `KDChart::AbstractDiagram::setAntiAliasing()`, `KDChart::BarDiagram::setBarAttributes()`, `KDChart::AbstractDiagram::setBrush()`, `KDChart::AbstractDiagram::setDataValueAttributes()`, `setLineAttributes()`, `KDChart::LineDiagram::setLineAttributes()`, `KDChart::AbstractDiagram::setPen()`, `KDChart::AbstractDiagram::setPercentMode()`, `KDChart::BarDiagram::setThreeDBarAttributes()`, `setThreeDLineAttributes()`, `KDChart::LineDiagram::setThreeDLineAttributes()`, `setType()`, `KDChart::LineDiagram::setType()`, `KDChart::BarDiagram::setType()`, `setValueTrackerAttributes()`, and `KDChart::LineDiagram::setValueTrackerAttributes()`.

#### 9.45.4.59 `AbstractCartesianDiagram * AbstractCartesianDiagram::referenceDiagram () const` [virtual, inherited]

**Returns:**

this diagram's reference diagram

**See also:**

[setReferenceDiagram](#)

Definition at line 148 of file `KDChartAbstractCartesianDiagram.cpp`.

References d.

Referenced by `KDChart::AbstractCartesianDiagram::compare()`, and `referenceDiagramIsBarDiagram()`.

```
149 {
150     return d->referenceDiagram;
151 }
```

#### 9.45.4.60 `QPointF AbstractCartesianDiagram::referenceDiagramOffset () const` [virtual, inherited]

**Returns:**

the relative offset of this diagram's reference diagram

**See also:**

[setReferenceDiagram](#)

Definition at line 153 of file KDChartAbstractCartesianDiagram.cpp.

References [d](#).

Referenced by [KDChart::AbstractCartesianDiagram::compare\(\)](#).

```
154 {  
155     return d->referenceDiagramOffset;  
156 }
```

#### 9.45.4.61 void Plotter::resetLineAttributes (const QModelIndex & index)

Remove any explicit line attributes settings that might have been specified before.

Definition at line 179 of file KDChartPlotter.cpp.

References [d](#), [KDChart::LineAttributesRole](#), and [KDChart::AbstractDiagram::propertiesChanged\(\)](#).

```
180 {  
181     d->attributesModel->resetData(  
182         d->attributesModel->mapFromSource(index), LineAttributesRole );  
183     emit propertiesChanged();  
184 }
```

#### 9.45.4.62 void Plotter::resetLineAttributes (int column)

Resets the line attributes of data set *column*.

Definition at line 155 of file KDChartPlotter.cpp.

References [d](#), [KDChart::LineAttributesRole](#), and [KDChart::AbstractDiagram::propertiesChanged\(\)](#).

```
156 {  
157     d->attributesModel->resetHeaderData(  
158         column, Qt::Vertical, LineAttributesRole );  
159     emit propertiesChanged();  
160 }
```

#### 9.45.4.63 void Plotter::resize (const QSizeF & area) [virtual]

Called by the widget's `sizeEvent`.

Adjust all internal structures, that are calculated, depending on the size of the widget.

##### Parameters:

*area*

Implements [KDChart::AbstractDiagram](#).

Definition at line 378 of file KDChartPlotter.cpp.

References [d](#), and [KDChart::AbstractDiagram::setDataBoundariesDirty\(\)](#).

```
379 {  
380     d->compressor.setResolution( static_cast<int>( size.width() ),  
381                               static_cast<int>( size.height() ) );  
382     setDataBoundariesDirty();  
383 }
```

**9.45.4.64 void Plotter::resizeEvent (QResizeEvent \*)** [protected]

Definition at line 331 of file KDChartPlotter.cpp.

```
332 {  
333 }
```

**9.45.4.65 void AbstractDiagram::scrollTo (const QModelIndex & index, ScrollHint hint = EnsureVisible)** [virtual, inherited]

[reimplemented]

Definition at line 942 of file KDChartAbstractDiagram.cpp.

```
943 {}
```

**9.45.4.66 void AbstractDiagram::setAllowOverlappingDataValueTexts (bool allow)** [inherited]

Set whether data value labels are allowed to overlap.

**Parameters:**

*allow* True means that overlapping labels are allowed.

Definition at line 434 of file KDChartAbstractDiagram.cpp.

References `d`, and `KDChart::AbstractDiagram::propertiesChanged()`.

```
435 {  
436     d->allowOverlappingDataValueTexts = allow;  
437     emit propertiesChanged();  
438 }
```

**9.45.4.67 void AbstractDiagram::setAntiAliasing (bool enabled)** [inherited]

Set whether anti-aliasing is to be used while rendering this diagram.

**Parameters:**

*enabled* True means that AA is enabled.

Definition at line 445 of file KDChartAbstractDiagram.cpp.

References `d`, and `KDChart::AbstractDiagram::propertiesChanged()`.

```
446 {  
447     d->antiAliasing = enabled;  
448     emit propertiesChanged();  
449 }
```



**9.45.4.68 void AbstractCartesianDiagram::setAttributesModel ([AttributesModel](#) \* *model*)**  
[virtual, inherited]

Associate an [AttributesModel](#) with this diagram.

Note that the diagram does *\_not\_* take ownership of the [AttributesModel](#). This should thus only be used with AttributesModels that have been explicitly created by the user, and are owned by her. Setting an [AttributesModel](#) that is internal to another diagram is an error.

Correct:

```
AttributesModel *am = new AttributesModel( model, 0 );
diagram1->setAttributesModel( am );
diagram2->setAttributesModel( am );
```

Wrong:

```
diagram1->setAttributesModel( diagram2->attributesModel() );
```

**Parameters:**

*model* The [AttributesModel](#) to use for this diagram.

**See also:**

[AttributesModel](#), [usesExternalAttributesModel](#)

Reimplemented from [KDChart::AbstractDiagram](#).

Definition at line 170 of file KDChartAbstractCartesianDiagram.cpp.

References [KDChart::AbstractDiagram::attributesModel\(\)](#), [d](#), and [KDChart::AbstractDiagram::setAttributesModel\(\)](#).

```
171 {
172     AbstractDiagram::setAttributesModel( model );
173     d->compressor.setModel( attributesModel() );
174 }
```

**9.45.4.69 void AbstractDiagram::setAttributesModelRootIndex (const QModelIndex &)**  
[protected, inherited]

Definition at line 293 of file KDChartAbstractDiagram.cpp.

References [d](#), and [KDChart::AbstractDiagram::setDataBoundariesDirty\(\)](#).

Referenced by [KDChart::AbstractDiagram::setRootIndex\(\)](#).

```
294 {
295     d->attributesModelRootIndex=idx;
296     setDataBoundariesDirty();
297     scheduleDelayedItemsLayout();
298 }
```

**9.45.4.70 void AbstractDiagram::setBrush (const QBrush & *brush*)** [inherited]

Set the brush to be used, for painting all datasets in the model.

**Parameters:**

***brush*** The brush to use.

Definition at line 806 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DatasetBrushRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setModelData().

```
807 {
808     attributesModel()->setModelData(
809         qVariantFromValue( brush ), DatasetBrushRole );
810     emit propertiesChanged();
811 }
```

**9.45.4.71 void AbstractDiagram::setBrush (int *dataset*, const QBrush & *brush*)** [inherited]

Set the brush to be used, for painting the given dataset.

**Parameters:**

***dataset*** The dataset's column in the model.

***brush*** The brush to use.

Definition at line 813 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DatasetBrushRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setHeaderData().

```
814 {
815     attributesModel()->setHeaderData(
816         column, Qt::Vertical,
817         qVariantFromValue( brush ),
818         DatasetBrushRole );
819     emit propertiesChanged();
820 }
```

**9.45.4.72 void AbstractDiagram::setBrush (const QModelIndex & *index*, const QBrush & *brush*)** [inherited]

Set the brush to be used, for painting the datapoint at the given index.

**Parameters:**

***index*** The datapoint's index in the model.

***brush*** The brush to use.

Definition at line 798 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DatasetBrushRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setData().

```

799 {
800     attributesModel()->setData(
801         attributesModel()->mapFromSource( index ),
802         qVariantFromValue( brush ), DatasetBrushRole );
803     emit propertiesChanged();
804 }

```

#### 9.45.4.73 void KDChart::AbstractCartesianDiagram::setCoordinatePlane (AbstractCoordinatePlane \*plane) [virtual, inherited]

[reimplemented]

Reimplemented from [KDChart::AbstractDiagram](#).

Definition at line 125 of file [KDChartAbstractCartesianDiagram.cpp](#).

References [KDChart::AbstractDiagram::coordinatePlane\(\)](#), and [KDChart::AbstractDiagram::setCoordinatePlane\(\)](#).

```

126 {
127     if( coordinatePlane() ) disconnect( coordinatePlane() );
128     AbstractDiagram::setCoordinatePlane(plane);
129
130     // show the axes, after all have been adjusted
131     // (because they might be dependend on each other)
132     /*
133     if( plane )
134         Q_FOREACH( CartesianAxis* axis, d->axesList )
135             axis->show();
136     else
137         Q_FOREACH( CartesianAxis* axis, d->axesList )
138             axis->hide();
139     */
140 }

```

#### 9.45.4.74 void AbstractDiagram::setDataBoundariesDirty () const [protected, inherited]

Definition at line 234 of file [KDChartAbstractDiagram.cpp](#).

References [d](#).

Referenced by [KDChart::AbstractDiagram::dataChanged\(\)](#), [resize\(\)](#), [KDChart::LineDiagram::resize\(\)](#), [KDChart::BarDiagram::resize\(\)](#), [KDChart::AbstractDiagram::setAttributesModel\(\)](#), [KDChart::AbstractDiagram::setAttributesModelRootIndex\(\)](#), [KDChart::AbstractDiagram::setDatasetDimension\(\)](#), [KDChart::AbstractDiagram::setModel\(\)](#), [KDChart::BarDiagram::setThreeDBarAttributes\(\)](#), [setThreeDLineAttributes\(\)](#), [KDChart::LineDiagram::setThreeDLineAttributes\(\)](#), [setType\(\)](#), [KDChart::LineDiagram::setType\(\)](#), and [KDChart::BarDiagram::setType\(\)](#).

```

235 {
236     d->databoundariesDirty = true;
237 }

```

#### 9.45.4.75 void AbstractDiagram::setDatasetDimension (int dimension) [inherited]

Sets the dataset dimension of the diagram.

See also:

[datasetDimension](#).

Parameters:

*dimension*

Definition at line 1077 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::AbstractDiagram::layoutChanged\(\)](#), and [KDChart::AbstractDiagram::setDataBoundariesDirty\(\)](#).

Referenced by [KDChart::Widget::setType\(\)](#), [KDChart::TernaryLineDiagram::TernaryLineDiagram\(\)](#), and [KDChart::TernaryPointDiagram::TernaryPointDiagram\(\)](#).

```
1078 {
1079     if ( d->datasetDimension == dimension ) return;
1080     d->datasetDimension = dimension;
1081     setDataBoundariesDirty();
1082     emit layoutChanged( this );
1083 }
```

#### 9.45.4.76 void AbstractDiagram::setDataValueAttributes (const [DataValueAttributes](#) & *a*) [inherited]

Set the [DataValueAttributes](#) for all datapoints in the model.

Parameters:

*a* The attributes to set.

Definition at line 428 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::DataValueLabelAttributesRole](#), and [KDChart::AbstractDiagram::propertiesChanged\(\)](#).

```
429 {
430     d->attributesModel->setModelData( qVariantFromValue( a ), DataValueLabelAttributesRole );
431     emit propertiesChanged();
432 }
```

#### 9.45.4.77 void AbstractDiagram::setDataValueAttributes (int *dataset*, const [DataValueAttributes](#) & *a*) [inherited]

Set the [DataValueAttributes](#) for the given dataset.

Parameters:

*dataset* The dataset to set the attributes for.

*a* The attributes to set.

Definition at line 400 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::DataValueLabelAttributesRole](#), and [KDChart::AbstractDiagram::propertiesChanged\(\)](#).

```
401 {  
402     d->attributesModel->setHeaderData(  
403         column, Qt::Vertical,  
404         qVariantFromValue( a ), DataValueLabelAttributesRole );  
405     emit propertiesChanged();  
406 }
```

#### 9.45.4.78 void AbstractDiagram::setDataValueAttributes (const QModelIndex & *index*, const [DataValueAttributes](#) & *a*) [inherited]

Set the [DataValueAttributes](#) for the given index.

##### Parameters:

- index* The datapoint to set the attributes for.
- a* The attributes to set.

Definition at line 389 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::DataValueLabelAttributesRole](#), and [KDChart::AbstractDiagram::propertiesChanged\(\)](#).

```
391 {  
392     d->attributesModel->setData(  
393         d->attributesModel->mapFromSource( index ),  
394         qVariantFromValue( a ),  
395         DataValueLabelAttributesRole );  
396     emit propertiesChanged();  
397 }
```

#### 9.45.4.79 void AbstractDiagram::setHidden (bool *hidden*) [inherited]

Hide (or unhide, resp.

) all datapoints in the model.

##### Note:

Hidden data are still taken into account by the coordinate plane, so neither the grid nor your axes' ranges will change, when you hide data. For totally removing data from KD Chart's view you can use another approach: e.g. you could define a proxy model on top of your data model, and register the proxy model calling [setModel\(\)](#) instead of registering your real data model.

##### Parameters:

- hidden* The hidden status to set.

Definition at line 359 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::AbstractDiagram::dataHidden\(\)](#), and [KDChart::DataHiddenRole](#).

```
360 {  
361     d->attributesModel->setModelData(  
362         qVariantFromValue( hidden ),  
363         DataHiddenRole );  
364     emit dataHidden();  
365 }
```

**9.45.4.80 void AbstractDiagram::setHidden (int *column*, bool *hidden*)** [inherited]

Hide (or unhide, resp.

) a dataset.

**Note:**

Hidden data are still taken into account by the coordinate plane, so neither the grid nor your axes' ranges will change, when you hide data. For totally removing data from KD Chart's view you can use another approach: e.g. you could define a proxy model on top of your data model, and register the proxy model calling [setModel\(\)](#) instead of registering your real data model.

**Parameters:**

***column*** The dataset to set the hidden status for.

***hidden*** The hidden status to set.

Definition at line 350 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::AbstractDiagram::dataHidden\(\)](#), and [KDChart::DataHiddenRole](#).

```
351 {  
352     d->attributesModel->setHeaderData(  
353         column, Qt::Vertical,  
354         qVariantFromValue( hidden ),  
355         DataHiddenRole );  
356     emit dataHidden();  
357 }
```

**9.45.4.81 void AbstractDiagram::setHidden (const QModelIndex & *index*, bool *hidden*)**  
[inherited]

Hide (or unhide, resp.

) a data cell.

**Note:**

Hidden data are still taken into account by the coordinate plane, so neither the grid nor your axes' ranges will change, when you hide data. For totally removing data from KD Chart's view you can use another approach: e.g. you could define a proxy model on top of your data model, and register the proxy model calling [setModel\(\)](#) instead of registering your real data model.

**Parameters:**

***index*** The datapoint to set the hidden status for.

***hidden*** The hidden status to set.

Definition at line 341 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::AbstractDiagram::dataHidden\(\)](#), and [KDChart::DataHiddenRole](#).

```
342 {  
343     d->attributesModel->setData(  
344         d->attributesModel->mapFromSource( index ),  
345         qVariantFromValue( hidden ),  
346         DataHiddenRole );  
347     emit dataHidden();  
348 }
```

**9.45.4.82 void Plotter::setLineAttributes (const QModelIndex & *index*, const [LineAttributes](#) & *a*)**

Sets the line attributes for the model index *index* to *la*.

Definition at line 165 of file KDChartPlotter.cpp.

References *d*, KDChart::LineAttributesRole, and KDChart::AbstractDiagram::propertiesChanged().

```
168 {
169     d->attributesModel->setData (
170         d->attributesModel->mapFromSource (index),
171         qVariantFromValue ( la ),
172         LineAttributesRole );
173     emit propertiesChanged();
174 }
```

**9.45.4.83 void Plotter::setLineAttributes (int *column*, const [LineAttributes](#) & *a*)**

Sets the line attributes of data set *column* to *la*.

Definition at line 140 of file KDChartPlotter.cpp.

References *d*, KDChart::LineAttributesRole, and KDChart::AbstractDiagram::propertiesChanged().

```
143 {
144     d->attributesModel->setHeaderData (
145         column,
146         Qt::Vertical,
147         qVariantFromValue ( la ),
148         LineAttributesRole );
149     emit propertiesChanged();
150 }
```

**9.45.4.84 void Plotter::setLineAttributes (const [LineAttributes](#) & *a*)**

Sets the global line attributes to *la*.

Definition at line 129 of file KDChartPlotter.cpp.

References *d*, KDChart::LineAttributesRole, and KDChart::AbstractDiagram::propertiesChanged().

```
130 {
131     d->attributesModel->setModelData (
132         qVariantFromValue ( la ),
133         LineAttributesRole );
134     emit propertiesChanged();
135 }
```

**9.45.4.85 void AbstractCartesianDiagram::setModel (QAbstractItemModel \* *model*)**  
[virtual, inherited]

Associate a model with the diagram.

Reimplemented from [KDChart::AbstractDiagram](#).

Definition at line 164 of file KDChartAbstractCartesianDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), *d*, and KDChart::AbstractDiagram::setModel().

```

165 {
166     AbstractDiagram::setModel( model );
167     d->compressor.setModel( attributesModel() );
168 }

```

#### 9.45.4.86 void AbstractDiagram::setPen (const QPen & *pen*) [inherited]

Set the pen to be used, for painting all datasets in the model.

##### Parameters:

*pen* The pen to use.

Definition at line 753 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DatasetPenRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setModelData().

```

754 {
755     attributesModel()->setModelData(
756         qVariantFromValue( pen ), DatasetPenRole );
757     emit propertiesChanged();
758 }

```

#### 9.45.4.87 void AbstractDiagram::setPen (int *dataset*, const QPen & *pen*) [inherited]

Set the pen to be used, for painting the given dataset.

##### Parameters:

*dataset* The dataset's row in the model.

*pen* The pen to use.

Definition at line 760 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::datasetDimension(), KDChart::DatasetPenRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setHeaderData().

```

761 {
762     if( datasetDimension() > 1 )
763         column *= datasetDimension();
764     attributesModel()->setHeaderData(
765         column, Qt::Vertical,
766         qVariantFromValue( pen ),
767         DatasetPenRole );
768     emit propertiesChanged();
769 }

```

#### 9.45.4.88 void AbstractDiagram::setPen (const QModelIndex & *index*, const QPen & *pen*) [inherited]

Set the pen to be used, for painting the datapoint at the given index.



**Parameters:**

*index* The datapoint's index in the model.

*pen* The pen to use.

Definition at line 743 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::datasetDimension(), KDChart::DatasetPenRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setData().

```

744 {
745     if( datasetDimension() > 1 )
746         return setPen( index.column(), pen );
747     attributesModel()->setData(
748         attributesModel()->mapFromSource( index ),
749         qVariantFromValue( pen ), DatasetPenRole );
750     emit propertiesChanged();
751 }
```

**9.45.4.89 void AbstractDiagram::setPercentMode (bool *percent*)** [inherited]

Definition at line 456 of file KDChartAbstractDiagram.cpp.

References d, and KDChart::AbstractDiagram::propertiesChanged().

Referenced by KDChart::LineDiagram::setType(), and KDChart::BarDiagram::setType().

```

457 {
458     d->percent = percent;
459     emit propertiesChanged();
460 }
```

**9.45.4.90 void AbstractCartesianDiagram::setReferenceDiagram ([AbstractCartesianDiagram](#) \* *diagram*, const QPointF & *offset* = QPointF())** [virtual, inherited]

Makes this diagram use another diagram *diagram* as reference diagram with relative offset *offset*.

To share cartesian axes between different diagrams there might be cases when you need that. Normally you don't.

**See also:**

examples/SharedAbscissa

Definition at line 142 of file KDChartAbstractCartesianDiagram.cpp.

References d.

```

143 {
144     d->referenceDiagram = diagram;
145     d->referenceDiagramOffset = offset;
146 }
```

#### 9.45.4.91 void AbstractCartesianDiagram::setRootIndex (const QModelIndex & *index*) [virtual, inherited]

Set the root index in the model, where the diagram starts referencing data for display.

[reimplemented]

Reimplemented from [KDChart::AbstractDiagram](#).

Definition at line 158 of file `KDChartAbstractCartesianDiagram.cpp`.

References `d`, and `KDChart::AbstractDiagram::setRootIndex()`.

```
159 {
160     d->compressor.setRootIndex( index );
161     AbstractDiagram::setRootIndex( index );
162 }
```

#### 9.45.4.92 void AbstractDiagram::setSelection (const QRect & *rect*, QItemSelectionModel::SelectionFlags *command*) [virtual, inherited]

[reimplemented]

Definition at line 959 of file `KDChartAbstractDiagram.cpp`.

References `d`.

```
960 {
961     const QModelIndexList indexes = d->indexesIn( rect );
962     QItemSelection selection;
963     KDAB_FOREACH( const QModelIndex& index, indexes )
964     {
965         selection.append( QItemSelectionRange( index ) );
966     }
967     selectionModel()->select( selection, command );
968 }
```

#### 9.45.4.93 void Plotter::setThreeDLineAttributes (const QModelIndex & *index*, const [ThreeDLineAttributes](#) & *a*)

Sets the 3D line attributes of model index *index* to *la*.

Definition at line 250 of file `KDChartPlotter.cpp`.

References `d`, `KDChart::AbstractDiagram::propertiesChanged()`, `KDChart::AbstractDiagram::setDataBoundariesDirty()`, and `KDChart::ThreeDLineAttributesRole`.

```
253 {
254     setDataBoundariesDirty();
255     d->attributesModel->setData(
256         d->attributesModel->mapFromSource( index ),
257         qVariantFromValue( la ),
258         ThreeDLineAttributesRole );
259     emit propertiesChanged();
260 }
```

**9.45.4.94 void Plotter::setThreeDLineAttributes (int *column*, const [ThreeDLineAttributes](#) & *a*)**

Sets the 3D line attributes of data set *column* to *la*.

Definition at line 234 of file KDChartPlotter.cpp.

References [d](#), [KDChart::AbstractDiagram::propertiesChanged\(\)](#), [KDChart::AbstractDiagram::setDataBoundariesDirty\(\)](#), and [KDChart::ThreeDLineAttributesRole](#).

```

237 {
238     setDataBoundariesDirty();
239     d->attributesModel->setHeaderData(
240         column,
241         Qt::Vertical,
242         qVariantFromValue( la ),
243         ThreeDLineAttributesRole );
244     emit propertiesChanged();
245 }
```

**9.45.4.95 void Plotter::setThreeDLineAttributes (const [ThreeDLineAttributes](#) & *a*)**

Sets the global 3D line attributes to *la*.

Definition at line 221 of file KDChartPlotter.cpp.

References [d](#), [KDChart::AbstractDiagram::propertiesChanged\(\)](#), [KDChart::AbstractDiagram::setDataBoundariesDirty\(\)](#), and [KDChart::ThreeDLineAttributesRole](#).

```

223 {
224     setDataBoundariesDirty();
225     d->attributesModel->setModelData(
226         qVariantFromValue( la ),
227         ThreeDLineAttributesRole );
228     emit propertiesChanged();
229 }
```

**9.45.4.96 void Plotter::setType (const [PlotType](#) *type*)**

Sets the plotter's type to *type*. Currently, there's no other type than [PlotType::Normal](#), but others might be added in future.

Definition at line 92 of file KDChartPlotter.cpp.

References [d](#), [KDChart::AbstractDiagram::datasetDimension\(\)](#), [KDChart::AbstractDiagram::layoutChanged\(\)](#), [Normal](#), [KDChart::AbstractDiagram::propertiesChanged\(\)](#), and [KDChart::AbstractDiagram::setDataBoundariesDirty\(\)](#).

Referenced by [clone\(\)](#).

```

93 {
94     if( d->implementor->type() == type )
95         return;
96     if( type != Plotter::Normal && datasetDimension() != 2 )
97     {
98         Q_ASSERT_X ( false, "setType()",
99                     "This line chart type can only be used with two-dimensional data." );
100         return;
101     }
102     switch( type ) {
```

```

103     case Normal:
104         d->implementor = d->normalPlotter;
105         break;
106     default:
107         Q_ASSERT_X( false, "Plotter::setType", "unknown plotter subtype" );
108     };
109
110     // d->lineType = type;
111     Q_ASSERT( d->implementor->type() == type );
112
113     setDataBoundariesDirty();
114     emit layoutChanged( this );
115     emit propertiesChanged();
116 }

```

#### 9.45.4.97 void AbstractDiagram::setUnitPrefix (const QString & *prefix*, Qt::Orientation *orientation*) [inherited]

Sets the unit prefix for all values.

##### Parameters:

*prefix* the prefix to be set

*orientation* the orientation of the axis to set

Definition at line 864 of file KDChartAbstractDiagram.cpp.

References d.

```

865 {
866     d->unitPrefix[ orientation ] = prefix;
867 }

```

#### 9.45.4.98 void AbstractDiagram::setUnitPrefix (const QString & *prefix*, int *column*, Qt::Orientation *orientation*) [inherited]

Sets the unit prefix for one value.

##### Parameters:

*prefix* the prefix to be set

*column* the value using that prefix

*orientation* the orientation of the axis to set

Definition at line 854 of file KDChartAbstractDiagram.cpp.

References d.

```

855 {
856     d->unitPrefixMap[ column ][ orientation ]= prefix;
857 }

```

**9.45.4.99 void AbstractDiagram::setUnitSuffix (const QString & *suffix*, Qt::Orientation *orientation*)** [inherited]

Sets the unit suffix for all values.

**Parameters:**

*suffix* the suffix to be set

*orientation* the orientation of the axis to set

Definition at line 885 of file KDChartAbstractDiagram.cpp.

References d.

```
886 {  
887     d->unitSuffix[ orientation ] = suffix;  
888 }
```

**9.45.4.100 void AbstractDiagram::setUnitSuffix (const QString & *suffix*, int *column*, Qt::Orientation *orientation*)** [inherited]

Sets the unit suffix for one value.

**Parameters:**

*suffix* the suffix to be set

*column* the value using that suffix

*orientation* the orientation of the axis to set

Definition at line 875 of file KDChartAbstractDiagram.cpp.

References d.

```
876 {  
877     d->unitSuffixMap[ column ][ orientation ]= suffix;  
878 }
```

**9.45.4.101 void Plotter::setValueTrackerAttributes (const QModelIndex & *index*, const ValueTrackerAttributes & *a*)**

Sets the value tracker attributes of the model index *index* to *va*.

Definition at line 311 of file KDChartPlotter.cpp.

References d, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::ValueTrackerAttributes-Role.

```
313 {  
314     d->attributesModel->setData( d->attributesModel->mapFromSource(index),  
315                                 qVariantFromValue( va ),  
316                                 KDChart::ValueTrackerAttributesRole );  
317     emit propertiesChanged();  
318 }
```

**9.45.4.102** `void AbstractCartesianDiagram::takeAxis (CartesianAxis * axis)` [virtual, inherited]

Removes the axis from the diagram, without deleting it.

The diagram no longer owns the axis, so it is the caller's responsibility to delete the axis.

**See also:**

[addAxis](#)

Definition at line 100 of file `KDChartAbstractCartesianDiagram.cpp`.

References `d`, `KDChart::AbstractAxis::deleteObserver()`, `KDChart::AbstractCartesianDiagram::layoutPlanes()`, and `KDChart::AbstractLayoutItem::setParentWidget()`.

Referenced by `KDChart::Widget::setType()`, and `KDChart::CartesianAxis::~~CartesianAxis()`.

```

101 {
102     const int idx = d->axesList.indexOf( axis );
103     if( idx != -1 )
104         d->axesList.takeAt( idx );
105     axis->deleteObserver( this );
106     axis->setParentWidget( 0 );
107     layoutPlanes();
108 }
```

**9.45.4.103** `double Plotter::threeDItemDepth (int column) const` [protected, virtual]

**Returns:**

the 3D item depth of the data set *column*

Implements [KDChart::AbstractCartesianDiagram](#).

Definition at line 299 of file `KDChartPlotter.cpp`.

References `d`, and `KDChart::ThreeDLineAttributesRole`.

```

300 {
301     return qVariantValue<ThreeDLineAttributes>(
302         d->attributesModel->headerData (
303             column,
304             Qt::Vertical,
305             KDChart::ThreeDLineAttributesRole ) ).validDepth();
306 }
```

**9.45.4.104** `double Plotter::threeDItemDepth (const QModelIndex & index) const` [protected, virtual]

**Returns:**

the 3D item depth of the model index *index*

Implements [KDChart::AbstractCartesianDiagram](#).

Definition at line 294 of file `KDChartPlotter.cpp`.

References `threeDLineAttributes()`, and `KDChart::AbstractThreeDAttributes::validDepth()`.

```
295 {  
296     return threeDLineAttributes( index ).validDepth();  
297 }
```

#### 9.45.4.105 [ThreeDLineAttributes](#) Plotter::threeDLineAttributes (const QModelIndex & *index*) const

##### Returns:

the 3D line attributes of the model index *index*

Definition at line 285 of file KDChartPlotter.cpp.

References d, and KDChart::ThreeDLineAttributesRole.

```
287 {  
288     return qVariantValue<ThreeDLineAttributes>(   
289         d->attributesModel->data(   
290             d->attributesModel->mapFromSource( index ),  
291             KDChart::ThreeDLineAttributesRole ) );  
292 }
```

#### 9.45.4.106 [ThreeDLineAttributes](#) Plotter::threeDLineAttributes (int *column*) const

##### Returns:

the 3D line attributes of data set *column*

Definition at line 274 of file KDChartPlotter.cpp.

References KDChart::AbstractDiagram::columnToIndex(), d, and KDChart::ThreeDLineAttributesRole.

```
275 {  
276     return qVariantValue<ThreeDLineAttributes>(   
277         d->attributesModel->data(   
278             d->attributesModel->mapFromSource( columnToIndex( column ) ),  
279             KDChart::ThreeDLineAttributesRole ) );  
280 }
```

#### 9.45.4.107 [ThreeDLineAttributes](#) Plotter::threeDLineAttributes () const

##### Returns:

the global 3D line attributes

Definition at line 265 of file KDChartPlotter.cpp.

References d, and KDChart::ThreeDLineAttributesRole.

Referenced by threeDItemDepth().

```
266 {  
267     return qVariantValue<ThreeDLineAttributes>(   
268         d->attributesModel->data( KDChart::ThreeDLineAttributesRole ) );  
269 }
```

**9.45.4.108 `Plotter::PlotType` `Plotter::type () const`****Returns:**

the type of the plotter

Definition at line 121 of file `KDChartPlotter.cpp`.

References `d`.

Referenced by `clone()`, and `compare()`.

```
122 {
123     return d->implementor->type();
124 }
```

**9.45.4.109 `QString AbstractDiagram::unitPrefix (Qt::Orientation orientation) const` [inherited]**

Returns the global unit prefix.

**Parameters:**

***orientation*** the orientation of the axis

**Returns:**

the unit prefix

Definition at line 908 of file `KDChartAbstractDiagram.cpp`.

References `d`.

```
909 {
910     return d->unitPrefix[ orientation ];
911 }
```

**9.45.4.110 `QString AbstractDiagram::unitPrefix (int column, Qt::Orientation orientation, bool fallback = false) const` [inherited]**

Returns the unit prefix for a special value.

**Parameters:**

***column*** the value which's prefix is requested

***orientation*** the orientation of the axis

***fallback*** if true, the global prefix is return when no specific one is set for that value

**Returns:**

the unit prefix

Definition at line 897 of file `KDChartAbstractDiagram.cpp`.

References `d`.

Referenced by `KDChart::AbstractDiagram::itemRowLabels()`, and `KDChart::CartesianAxis::paintCtx()`.



```
898 {  
899     if( !fallback || d->unitPrefixMap[ column ].contains( orientation ) )  
900         return d->unitPrefixMap[ column ][ orientation ];  
901     return d->unitPrefix[ orientation ];  
902 }
```

#### 9.45.4.111 QString AbstractDiagram::unitSuffix (Qt::Orientation *orientation*) const [inherited]

Returns the global unit suffix.

##### Parameters:

*orientation* the orientation of the axis

##### Returns:

the unit suffix

Definition at line 931 of file KDChartAbstractDiagram.cpp.

References d.

```
932 {  
933     return d->unitSuffix[ orientation ];  
934 }
```

#### 9.45.4.112 QString AbstractDiagram::unitSuffix (int *column*, Qt::Orientation *orientation*, bool *fallback* = false) const [inherited]

Returns the unit suffix for a special value.

##### Parameters:

*column* the value which's suffix is requested

*orientation* the orientation of the axis

*fallback* if true, the global suffix is return when no specific one is set for that value

##### Returns:

the unit suffix

Definition at line 920 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::itemRowLabels(), and KDChart::CartesianAxis::paintCtx().

```
921 {  
922     if( !fallback || d->unitSuffixMap[ column ].contains( orientation ) )  
923         return d->unitSuffixMap[ column ][ orientation ];  
924     return d->unitSuffix[ orientation ];  
925 }
```

**9.45.4.113 void AbstractDiagram::update () const** [inherited]

Definition at line 1091 of file KDChartAbstractDiagram.cpp.

References [d](#).

Referenced by [KDChart::AbstractDiagram::doItemsLayout\(\)](#).

```
1092 {
1093     //qDebug("KDChart::AbstractDiagram::update() called");
1094     if( d->plane )
1095         d->plane->update();
1096 }
```

**9.45.4.114 void KDChart::AbstractDiagram::useDefaultColors ()** [inherited]

Set the palette to be used, for painting datasets to the default palette.

**See also:**

[KDChart::Palette](#). FIXME: fold into one usePalette ([KDChart::Palette&](#)) method

Definition at line 974 of file KDChartAbstractDiagram.cpp.

References [d](#), and [KDChart::AttributesModel::PaletteTypeDefault](#).

```
975 {
976     d->attributesModel->setPaletteType( AttributesModel::PaletteTypeDefault );
977 }
```

**9.45.4.115 void KDChart::AbstractDiagram::useRainbowColors ()** [inherited]

Set the palette to be used, for painting datasets to the rainbow palette.

**See also:**

[KDChart::Palette](#).

Definition at line 984 of file KDChartAbstractDiagram.cpp.

References [d](#), and [KDChart::AttributesModel::PaletteTypeRainbow](#).

```
985 {
986     d->attributesModel->setPaletteType( AttributesModel::PaletteTypeRainbow );
987 }
```

**9.45.4.116 bool AbstractDiagram::usesExternalAttributesModel () const** [virtual, inherited]

Returns whether the diagram is using its own built-in attributes model or an attributes model that was set via [setAttributesModel](#).

**See also:**

[setAttributesModel](#)

Definition at line 274 of file KDChartAbstractDiagram.cpp.

References [d](#).

```
275 {  
276     return d->usesExternalAttributesModel();  
277 }
```

#### 9.45.4.117 void KDChart::AbstractDiagram::useSubduedColors() [inherited]

Set the palette to be used, for painting datasets to the subdued palette.

**See also:**

[KDChart::Palette](#).

Definition at line 979 of file KDChartAbstractDiagram.cpp.

References [d](#), and [KDChart::AttributesModel::PaletteTypeSubdued](#).

```
980 {  
981     d->attributesModel->setPaletteType( AttributesModel::PaletteTypeSubdued );  
982 }
```

#### 9.45.4.118 double AbstractDiagram::valueForCell(int row, int column) const [protected, inherited]

Helper method, retrieving the data value (DisplayRole) for a given row and column.

**Parameters:**

**row** The row to query.

**column** The column to query.

**Returns:**

The value of the display role at the given row and column as a double.

**Deprecated**

Definition at line 1085 of file KDChartAbstractDiagram.cpp.

References [KDChart::AbstractDiagram::attributesModelRootIndex\(\)](#), and [d](#).

```
1086 {  
1087     return d->attributesModel->data(  
1088         d->attributesModel->index( row, column, attributesModelRootIndex() ) ).toDouble();  
1089 }
```

#### 9.45.4.119 [ValueTrackerAttributes](#) `Plotter::valueTrackerAttributes (const QModelIndex & index) const`

Returns the value tracker attributes of the model index *index*.

Definition at line 323 of file `KDChartPlotter.cpp`.

References `d`, and `KDChart::ValueTrackerAttributesRole`.

```
325 {  
326     return qVariantValue<ValueTrackerAttributes>( d->attributesModel->data(  
327         d->attributesModel->mapFromSource( index ),  
328         KDChart::ValueTrackerAttributesRole ) );  
329 }
```

#### 9.45.4.120 `int AbstractDiagram::verticalOffset () const` [virtual, inherited]

[reimplemented]

Definition at line 953 of file `KDChartAbstractDiagram.cpp`.

```
954 { return 0; }
```

#### 9.45.4.121 `QRect AbstractDiagram::visualRect (const QModelIndex & index) const` [virtual, inherited]

[reimplemented]

Definition at line 937 of file `KDChartAbstractDiagram.cpp`.

```
938 {  
939     return QRect();  
940 }
```

#### 9.45.4.122 `QRegion AbstractDiagram::visualRegionForSelection (const QItemSelection & selection) const` [virtual, inherited]

[reimplemented]

Definition at line 970 of file `KDChartAbstractDiagram.cpp`.

```
971 { return QRegion(); }
```

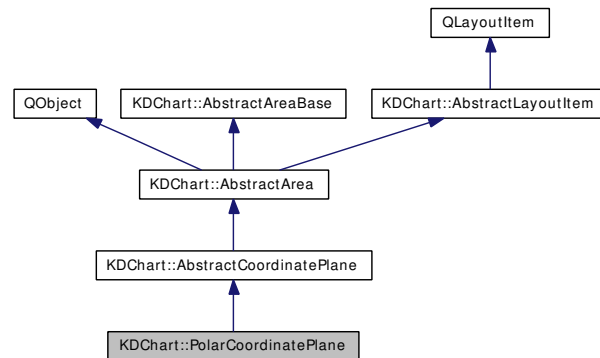
The documentation for this class was generated from the following files:

- [KDChartPlotter.h](#)
- [KDChartPlotter.cpp](#)

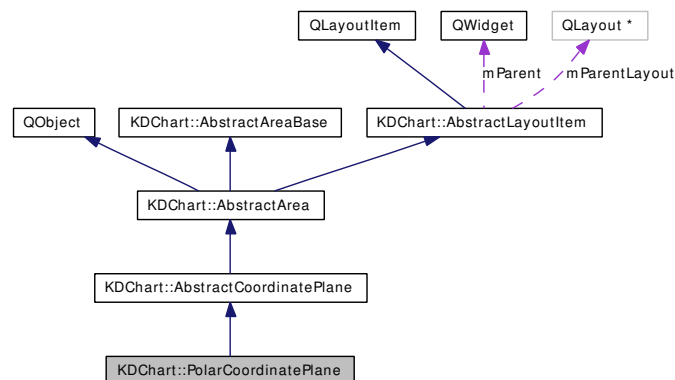
## 9.46 KDChart::PolarCoordinatePlane Class Reference

```
#include <KDChartPolarCoordinatePlane.h>
```

Inheritance diagram for KDChart::PolarCoordinatePlane:



Collaboration diagram for KDChart::PolarCoordinatePlane:



### 9.46.1 Detailed Description

Polar coordinate plane.

Definition at line 39 of file `KDChartPolarCoordinatePlane.h`.

#### Public Types

- enum [AxesCalcMode](#) {  
[Linear](#),  
[Logarithmic](#) }
- typedef `QList< CoordinateTransformation >` [CoordinateTransformationList](#)

#### Public Slots

- void [layoutPlanes](#) ()

Calling [layoutPlanes\(\)](#) on the plane triggers the global `KDChart::Chart::slotLayoutPlanes()`.

- void [relayout](#) ()  
Calling [relayout\(\)](#) on the plane triggers the global `KDChart::Chart::slotRelayout()`.
- void [setGridNeedsRecalculate](#) ()  
Used by the chart to clear the cached grid data.
- void [update](#) ()  
Calling [update\(\)](#) on the plane triggers the global `KDChart::Chart::update()`.

## Signals

- void [destroyedCoordinatePlane](#) ([AbstractCoordinatePlane](#) \*)  
Emitted when this coordinate plane is destroyed.
- void [needLayoutPlanes](#) ()  
Emitted when plane needs to trigger the Chart's layouting of the coord.
- void [needRelayout](#) ()  
Emitted when plane needs to trigger the Chart's layouting.
- void [needUpdate](#) ()  
Emitted when plane needs to update its drawings.
- void [positionChanged](#) ([AbstractArea](#) \*)
- void [propertiesChanged](#) ()  
Emitted upon change of a property of the Coordinate Plane or any of its components.

## Public Member Functions

- void [addDiagram](#) ([AbstractDiagram](#) \*diagram)  
Adds a diagram to this coordinate plane.
- void [alignToReferencePoint](#) (const [RelativePosition](#) &position)
- qreal [angleUnit](#) () const
- [BackgroundAttributes](#) [backgroundAttributes](#) () const
- virtual int [bottomOverlap](#) (bool doNotRecalculate=false) const  
This is called at layout time by `KDChart::AutoSpacerLayoutItem::sizeHint()`.
- bool [compare](#) (const [AbstractAreaBase](#) \*other) const  
Returns true if both areas have the same settings.
- [AbstractDiagram](#) \* [diagram](#) ()  
**Returns:**  
The first diagram associated with this coordinate plane.

- [ConstAbstractDiagramList diagrams \(\)](#) const  
**Returns:**  
*The list of diagrams associated with this coordinate plane.*
- [AbstractDiagramList diagrams \(\)](#)  
**Returns:**  
*The list of diagrams associated with this coordinate plane.*
- virtual Qt::Orientations [expandingDirections \(\)](#) const  
*pure virtual in [QLayoutItem](#)*
- [FrameAttributes frameAttributes \(\)](#) const
- virtual QRect [geometry \(\)](#) const  
*pure virtual in [QLayoutItem](#)*
- void [getFrameLeadings](#) (int &left, int &top, int &right, int &bottom) const
- [GridAttributes globalGridAttributes \(\)](#) const  
**Returns:**  
*The grid attributes used by this coordinate plane.*
- const [GridAttributes gridAttributes](#) (bool circular) const  
**Returns:**  
*The attributes used for grid lines drawn in circular direction (or in sagittal direction, resp.*
- [DataDimensionsList gridDimensionsList \(\)](#)  
*Returns the dimensions used for drawing the grid lines.*
- bool [hasOwnGridAttributes](#) (bool circular) const  
**Returns:**  
*Returns whether the grid attributes have been set for the respective direction via [setGridAttributes](#)(bool circular).*
- virtual bool [isEmpty \(\)](#) const  
*pure virtual in [QLayoutItem](#)*
- bool [isRubberBandZoomingEnabled \(\)](#) const  
**Returns:**  
*Whether zooming with a rubber band using the mouse is enabled.*
- const bool [isVisiblePoint](#) (const QPointF &point) const  
*Tests, if a point is visible on the coordinate plane.*
- virtual int [leftOverlap](#) (bool doNotRecalculate=false) const  
*This is called at layout time by [KDChart::AutoSpacerLayoutItem::sizeHint\(\)](#).*
- virtual QSize [maximumSize \(\)](#) const  
*pure virtual in [QLayoutItem](#)*
- virtual QSize [minimumSize \(\)](#) const

pure virtual in [QLayoutItem](#)

- virtual QSize [minimumSizeHint](#) () const  
*[reimplemented]*
- void [mouseDoubleClickEvent](#) (QMouseEvent \*event)
- void [mouseMoveEvent](#) (QMouseEvent \*event)
- void [mousePressEvent](#) (QMouseEvent \*event)
- void [mouseReleaseEvent](#) (QMouseEvent \*event)
- virtual void [paint](#) (QPainter \*)  
*reimpl*
- virtual void [paintAll](#) (QPainter &painter)  
*Call paintAll, if you want the background and the frame to be drawn before the normal [paint\(\)](#) is invoked automatically.*
- virtual void [paintBackground](#) (QPainter &painter, const QRect &rectangle)
- virtual void [paintCtx](#) ([PaintContext](#) \*context)  
*Default impl: Paint the complete item using its layouted position and size.*
- virtual void [paintFrame](#) (QPainter &painter, const QRect &rectangle)
- virtual void [paintIntoRect](#) (QPainter &painter, const QRect &rect)  
*Draws the background and frame, then calls [paint\(\)](#).*
- const [Chart](#) \* [parent](#) () const
- [Chart](#) \* [parent](#) ()
- [QLayout](#) \* [parentLayout](#) ()
- [PolarCoordinatePlane](#) ([Chart](#) \*parent=0)
- qreal [radiusUnit](#) () const
- [AbstractCoordinatePlane](#) \* [referenceCoordinatePlane](#) () const  
*There are two ways, in which planes can be caused to interact, in where they are put layouting wise: The first is the reference plane.*
- void [removeFromParentLayout](#) ()
- virtual void [replaceDiagram](#) ([AbstractDiagram](#) \*diagram, [AbstractDiagram](#) \*oldDiagram=0)  
*Replaces the old diagram, or appends the diagram, if there is none yet.*
- void [resetGridAttributes](#) (bool circular)  
*Reset the attributes to be used for grid lines drawn in circular direction (or in sagittal direction, resp.*
- virtual int [rightOverlap](#) (bool doNotRecalculate=false) const  
*This is called at layout time by [KDChart::AutoSpacerLayoutItem::sizeHint\(\)](#).*
- void [setBackgroundAttributes](#) (const [BackgroundAttributes](#) &a)
- void [setFrameAttributes](#) (const [FrameAttributes](#) &a)
- virtual void [setGeometry](#) (const QRect &r)  
*pure virtual in [QLayoutItem](#)*
- void [setGlobalGridAttributes](#) (const [GridAttributes](#) &)  
*Set the grid attributes to be used by this coordinate plane.*



- void [setGridAttributes](#) (bool circular, const [GridAttributes](#) &)  
*Set the attributes to be used for grid lines drawn in circular direction (or in sagittal direction, resp).*
- void [setParent](#) ([Chart](#) \*parent)  
*Called internally by [KDChart::Chart](#).*
- void [setParentLayout](#) ([QLayout](#) \*lay)
- virtual void [setParentWidget](#) ([QWidget](#) \*widget)  
*Inform the item about its widget: This enables the item, to trigger that widget's update, whenever the size of the item's contents has changed.*
- void [setReferenceCoordinatePlane](#) ([AbstractCoordinatePlane](#) \*plane)  
*Set another coordinate plane to be used as the reference plane for this one.*
- void [setRubberBandZoomingEnabled](#) (bool enable)  
*Enables or disables zooming with a rubber band using the mouse.*
- void [setStartPosition](#) (qreal degrees)  
*Specify the rotation of the coordinate plane.*
- virtual void [setZoomCenter](#) (const [QPointF](#) &center)  
*Set the point (in value coordinates) to be used as the center point in zoom operations.*
- virtual void [setZoomFactorX](#) (double factor)  
*Sets the zoom factor in horizontal direction, that is applied to all coordinate transformations.*
- virtual void [setZoomFactorY](#) (double factor)  
*Sets the zoom factor in vertical direction, that is applied to all coordinate transformations.*
- virtual [AbstractCoordinatePlane](#) \* [sharedAxisMasterPlane](#) ([QPainter](#) \*p=0)
- virtual [QSize](#) [sizeHint](#) () const  
*pure virtual in [QLayoutItem](#)*
- virtual void [sizeHintChanged](#) () const  
*Report changed size hint: ask the parent widget to recalculate the layout.*
- virtual [QSizePolicy](#) [sizePolicy](#) () const  
*[reimplemented]*
- qreal [startPosition](#) () const  
*Retrieve the rotation of the coordinate plane.*
- virtual void [takeDiagram](#) ([AbstractDiagram](#) \*diagram)  
*Removes the diagram from the plane, without deleting it.*
- virtual int [topOverlap](#) (bool doNotRecalculate=false) const  
*This is called at layout time by [KDChart::AutoSpacerLayoutItem::sizeHint\(\)](#).*
- const [QPointF](#) [translate](#) (const [QPointF](#) &diagramPoint) const

*Translate the given point in value space coordinates to a position in pixel space.*

- const QPointF [translatePolar](#) (const QPointF &diagramPoint) const
- virtual QPointF [zoomCenter](#) () const

**Returns:**

*The center point (in value coordinates) of the coordinate plane, that is used for zoom operations.*

- virtual double [zoomFactorX](#) () const

**Returns:**

*The zoom factor in horizontal direction, that is applied to all coordinate transformations.*

- virtual double [zoomFactorY](#) () const

**Returns:**

*The zoom factor in vertical direction, that is applied to all coordinate transformations.*

- [~PolarCoordinatePlane](#) ()

## Static Public Member Functions

- static void [paintBackgroundAttributes](#) (QPainter &painter, const QRect &rectangle, const [KDChart::BackgroundAttributes](#) &attributes)
- static void [paintFrameAttributes](#) (QPainter &painter, const QRect &rectangle, const [KDChart::FrameAttributes](#) &attributes)

## Protected Slots

- void [slotLayoutChanged](#) ([AbstractDiagram](#) \*diagram)

## Protected Member Functions

- virtual QRect [areaGeometry](#) () const
- virtual [DataDimensionsList](#) [getDataDimensionsList](#) () const
- QRect [innerRect](#) () const
- void [layoutDiagrams](#) ()

*Distribute the available space among the diagrams and axes.*

- void [paintEvent](#) (QPaintEvent \*)
- virtual void [positionHasChanged](#) ()
- void [resizeEvent](#) (QResizeEvent \*)

## Protected Attributes

- [QWidget](#) \* [mParent](#)
- [QLayout](#) \* [mParentLayout](#)

## 9.46.2 Member Typedef Documentation

### 9.46.2.1 `typedef QList<CoordinateTransformation> KDChart::PolarCoordinatePlane::CoordinateTransformationList`

Definition at line 47 of file KDChartPolarCoordinatePlane.h.

## 9.46.3 Member Enumeration Documentation

### 9.46.3.1 `enum KDChart::AbstractCoordinatePlane::AxesCalcMode` [inherited]

Enumerator:

*Linear*

*Logarithmic*

Definition at line 57 of file KDChartAbstractCoordinatePlane.h.

```
57 { Linear, Logarithmic };
```

## 9.46.4 Constructor & Destructor Documentation

### 9.46.4.1 `PolarCoordinatePlane::PolarCoordinatePlane (Chart *parent = 0)` [explicit]

Definition at line 114 of file KDChartPolarCoordinatePlane.cpp.

```
115 : AbstractCoordinatePlane ( new Private(), parent )
116 {
117     // this bloc left empty intentionally
118 }
```

### 9.46.4.2 `PolarCoordinatePlane::~~PolarCoordinatePlane ()`

Definition at line 120 of file KDChartPolarCoordinatePlane.cpp.

```
121 {
122     // this bloc left empty intentionally
123 }
```

## 9.46.5 Member Function Documentation

### 9.46.5.1 `void PolarCoordinatePlane::addDiagram (AbstractDiagram *diagram)` [virtual]

Adds a diagram to this coordinate plane.

Parameters:

*diagram* The diagram to add.

See also:

[replaceDiagram](#), [takeDiagram](#)

Reimplemented from [KDChart::AbstractCoordinatePlane](#).

Definition at line 130 of file KDChartPolarCoordinatePlane.cpp.

References [KDChart::AbstractCoordinatePlane::addDiagram\(\)](#), [KDChart::AbstractCoordinatePlane::diagram\(\)](#), and [slotLayoutChanged\(\)](#).

```

131 {
132     Q_ASSERT_X ( dynamic_cast<AbstractPolarDiagram*> ( diagram ),
133                 "PolarCoordinatePlane::addDiagram", "Only polar"
134                 "diagrams can be added to a polar coordinate plane!" );
135     AbstractCoordinatePlane::addDiagram ( diagram );
136     connect ( diagram, SIGNAL ( layoutChanged ( AbstractDiagram* ) ),
137             SLOT ( slotLayoutChanged ( AbstractDiagram* ) ) );
138
139 }
```

#### 9.46.5.2 void AbstractAreaBase::alignToReferencePoint (const [RelativePosition](#) & position) [inherited]

Definition at line 90 of file KDChartAbstractAreaBase.cpp.

```

91 {
92     Q_UNUSED( position );
93     // PENDING(kalle) FIXME
94     qWarning( "Sorry, not implemented: void AbstractAreaBase::alignToReferencePoint( const RelativePosi
95 }
```

#### 9.46.5.3 qreal PolarCoordinatePlane::angleUnit () const

Definition at line 294 of file KDChartPolarCoordinatePlane.cpp.

References [d](#).

Referenced by [layoutDiagrams\(\)](#).

```

295 {
296     Q_ASSERT_X ( d->currentTransformation != 0, "PolarCoordinatePlane::angleUnit",
297                 "Only call angleUnit() from within paint()." );
298     return d->currentTransformation->angleUnit;
299 }
```

#### 9.46.5.4 QRect AbstractArea::areaGeometry () const [protected, virtual, inherited]

Implements [KDChart::AbstractAreaBase](#).

Definition at line 150 of file KDChartAbstractArea.cpp.

Referenced by [KDChart::CartesianCoordinatePlane::drawingArea\(\)](#), [KDChart::TernaryCoordinatePlane::layoutDiagrams\(\)](#), [layoutDiagrams\(\)](#), [KDChart::TernaryCoordinatePlane::paint\(\)](#), [KDChart::CartesianAxis::paint\(\)](#), [KDChart::AbstractArea::paintAll\(\)](#), and [KDChart::CartesianAxis::paintCtx\(\)](#).

```

151 {
152     return geometry();
153 }
```

#### 9.46.5.5 [BackgroundAttributes](#) AbstractAreaBase::backgroundAttributes () const [inherited]

Definition at line 120 of file KDChartAbstractAreaBase.cpp.

References [d](#).

Referenced by KDChart::AbstractAreaBase::compare(), and updateCommonBrush().

```
121 {
122     return d->backgroundAttributes;
123 }
```

#### 9.46.5.6 `int AbstractArea::bottomOverlap (bool doNotRecalculate = false) const` [virtual, inherited]

This is called at layout time by KDChart::AutoSpacerLayoutItem::sizeHint().

The method triggers AbstractArea::sizeHint() to find out the amount of overlap at the bottom edge of the area.

##### Note:

The default implementation is not using any caching, it might make sense to implement a more sophisticated solution for derived classes that have complex work to do in sizeHint(). All we have here is a primitive flag to be set by the caller if it is sure that no sizeHint() needs to be called.

Definition at line 101 of file KDChartAbstractArea.cpp.

References [d](#).

Referenced by KDChart::CartesianAxis::maximumSize().

```
102 {
103     // Re-calculate the sizes,
104     // so we also get the amountOf..Overlap members set newly:
105     if( ! doNotRecalculate )
106         sizeHint();
107     return d->amountOfBottomOverlap;
108 }
```

#### 9.46.5.7 `bool AbstractAreaBase::compare (const AbstractAreaBase * other) const` [inherited]

Returns true if both areas have the same settings.

Definition at line 75 of file KDChartAbstractAreaBase.cpp.

References [KDChart::AbstractAreaBase::backgroundAttributes\(\)](#), [and](#) [KDChart::AbstractAreaBase::frameAttributes\(\)](#).

```
76 {
77     if( other == this ) return true;
78     if( ! other ){
79         //qDebug() << "CartesianAxis::compare() cannot compare to Null pointer";
80         return false;
81     }
```

```

82     /*
83     qDebug() << "AbstractAreaBase:" << (frameAttributes() == other->frameAttributes())
84         << (backgroundAttributes() == other->backgroundAttributes()) << "\n";
85     */
86     return (frameAttributes() == other->frameAttributes()) &&
87         (backgroundAttributes() == other->backgroundAttributes());
88 }

```

#### 9.46.5.8 void KDChart::AbstractCoordinatePlane::destroyedCoordinatePlane (AbstractCoordinatePlane \*) [signal, inherited]

Emitted when this coordinate plane is destroyed.

Referenced by KDChart::AbstractCoordinatePlane::~~AbstractCoordinatePlane().

#### 9.46.5.9 AbstractDiagram \* AbstractCoordinatePlane::diagram () [inherited]

##### Returns:

The first diagram associated with this coordinate plane.

Definition at line 117 of file KDChartAbstractCoordinatePlane.cpp.

References d.

Referenced by KDChart::TernaryCoordinatePlane::addDiagram(), addDiagram(), KDChart::CartesianCoordinatePlane::addDiagram(), KDChart::AbstractCoordinatePlane::addDiagram(), KDChart::Widget::diagram(), KDChart::CartesianCoordinatePlane::getRawDataBoundingRectFromDiagrams(), KDChart::TernaryCoordinatePlane::layoutDiagrams(), layoutDiagrams(), KDChart::AbstractCoordinatePlane::replaceDiagram(), KDChart::CartesianCoordinatePlane::setGeometry(), setStartPosition(), KDChart::CartesianCoordinatePlane::sharedAxisMasterPlane(), and KDChart::AbstractCoordinatePlane::takeDiagram().

```

118 {
119     if ( d->diagrams.isEmpty() )
120     {
121         return 0;
122     } else {
123         return d->diagrams.first();
124     }
125 }

```

#### 9.46.5.10 ConstAbstractDiagramList AbstractCoordinatePlane::diagrams () const [inherited]

##### Returns:

The list of diagrams associated with this coordinate plane.

Definition at line 132 of file KDChartAbstractCoordinatePlane.cpp.

References d.

```

133 {
134     ConstAbstractDiagramList list;
135 #ifndef QT_NO_STL

```

```

136     qCopy( d->diagrams.begin(), d->diagrams.end(), std::back_inserter( list ) );
137 #else
138     Q_FOREACH( AbstractDiagram * a, d->diagrams )
139         list.push_back( a );
140 #endif
141     return list;
142 }

```

#### 9.46.5.11 [AbstractDiagramList](#) AbstractCoordinatePlane::diagrams () [inherited]

##### Returns:

The list of diagrams associated with this coordinate plane.

Definition at line 127 of file KDChartAbstractCoordinatePlane.cpp.

References [d](#).

Referenced by [KDChart::CartesianCoordinatePlane::getDataDimensionsList\(\)](#), [KDChart::CartesianCoordinatePlane::getRawDataBoundingRectFromDiagrams\(\)](#), [KDChart::TernaryCoordinatePlane::layoutDiagrams\(\)](#), [layoutDiagrams\(\)](#), [KDChart::CartesianCoordinatePlane::layoutDiagrams\(\)](#), [KDChart::Chart::mouseDoubleClickEvent\(\)](#), [KDChart::Chart::mouseMoveEvent\(\)](#), [KDChart::Chart::mousePressEvent\(\)](#), [KDChart::Chart::mouseReleaseEvent\(\)](#), [KDChart::TernaryCoordinatePlane::paint\(\)](#), [paint\(\)](#), [KDChart::CartesianCoordinatePlane::paint\(\)](#), and [KDChart::CartesianCoordinatePlane::setGeometry\(\)](#).

```

128 {
129     return d->diagrams;
130 }

```

#### 9.46.5.12 [Qt::Orientations](#) KDChart::AbstractCoordinatePlane::expandingDirections () const [virtual, inherited]

pure virtual in [QLayoutItem](#)

Definition at line 212 of file KDChartAbstractCoordinatePlane.cpp.

```

213 {
214     return Qt::Vertical | Qt::Horizontal;
215 }

```

#### 9.46.5.13 [FrameAttributes](#) AbstractAreaBase::frameAttributes () const [inherited]

Definition at line 106 of file KDChartAbstractAreaBase.cpp.

References [d](#).

Referenced by [KDChart::Legend::clone\(\)](#), [KDChart::AbstractAreaBase::compare\(\)](#), and [updateCommonBrush\(\)](#).

```

107 {
108     return d->frameAttributes;
109 }

```

#### 9.46.5.14 **QRect KDChart::AbstractCoordinatePlane::geometry () const** [virtual, inherited]

pure virtual in [QLayoutItem](#)

Definition at line 246 of file KDChartAbstractCoordinatePlane.cpp.

References d.

Referenced by KDChart::Chart::mouseDoubleClickEvent(), KDChart::Chart::mouseMoveEvent(), KDChart::AbstractCoordinatePlane::mouseMoveEvent(), KDChart::Chart::mousePressEvent(), KDChart::Chart::mouseReleaseEvent(), KDChart::AbstractCoordinatePlane::mouseReleaseEvent(), and paint().

```
247 {
248     return d->geometry;
249 }
```

#### 9.46.5.15 **DataDimensionsList PolarCoordinatePlane::getDataDimensionsList () const** [protected, virtual]

Implements [KDChart::AbstractCoordinatePlane](#).

Definition at line 358 of file KDChartPolarCoordinatePlane.cpp.

```
359 {
360     DataDimensionsList l;
361
362     //FIXME(khz): do the real calculation
363
364     return l;
365 }
```

#### 9.46.5.16 **void AbstractAreaBase::getFrameLeadings (int & left, int & top, int & right, int & bottom) const** [inherited]

Definition at line 212 of file KDChartAbstractAreaBase.cpp.

References d.

Referenced by KDChart::AbstractAreaBase::innerRect(), and KDChart::AbstractAreaWidget::paintAll().

```
213 {
214     if( d && d->frameAttributes.isVisible() ){
215         const int padding = qMax( d->frameAttributes.padding(), 0 );
216         left  = padding;
217         top   = padding;
218         right = padding;
219         bottom = padding;
220     }else{
221         left  = 0;
222         top   = 0;
223         right = 0;
224         bottom = 0;
225     }
226 }
```



#### 9.46.5.17 [GridAttributes](#) KDChart::AbstractCoordinatePlane::globalGridAttributes () const [inherited]

**Returns:**

The grid attributes used by this coordinate plane.

**See also:**

[setGlobalGridAttributes](#)  
[CartesianCoordinatePlane::gridAttributes](#)

Definition at line 161 of file KDChartAbstractCoordinatePlane.cpp.

References [d](#).

Referenced by [gridAttributes\(\)](#), and [KDChart::CartesianCoordinatePlane::gridAttributes\(\)](#).

```
162 {  
163     return d->gridAttributes;  
164 }
```

#### 9.46.5.18 [const GridAttributes](#) KDChart::PolarCoordinatePlane::gridAttributes (bool *circular*) const

**Returns:**

The attributes used for grid lines drawn in circular direction (or in sagittal direction, resp.

).

**Note:**

This function always returns a valid set of grid attributes: If no special grid attributes were set for this direction the global attributes are returned, as returned by [AbstractCoordinatePlane::globalGridAttributes](#).

**See also:**

[setGridAttributes](#)  
[resetGridAttributes](#)  
[AbstractCoordinatePlane::globalGridAttributes](#)  
[hasOwnGridAttributes](#)

Definition at line 387 of file KDChartPolarCoordinatePlane.cpp.

References [d](#), [KDChart::AbstractCoordinatePlane::globalGridAttributes\(\)](#), and [hasOwnGridAttributes\(\)](#).

```
389 {  
390     if( hasOwnGridAttributes( circular ) ){  
391         if( circular )  
392             return d->gridAttributesCircular;  
393         else  
394             return d->gridAttributesSagittal;  
395     }else{  
396         return globalGridAttributes();  
397     }  
398 }
```

#### 9.46.5.19 [KDChart::DataDimensionsList](#) [KDChart::AbstractCoordinatePlane::gridDimensionsList\(\)](#) [inherited]

Returns the dimensions used for drawing the grid lines.

Returned data is the result of (cached) grid calculations, so - if you need that information for your own tasks - make sure to call again this function after every data modification that has changed the data range, since grid calculation is based upon the data range, thus the grid start/end might have changed if the data was changed.

##### Note:

Returned list will contain different numbers of [DataDimension](#), depending on the kind of coordinate plane used. For [CartesianCoordinatePlane](#) two [DataDimension](#) are returned: the first representing grid lines in X direction (matching the Abscissa axes) and the second indicating vertical grid lines (or Ordinate axes, resp.).

##### Returns:

The dimensions used for drawing the grid lines.

##### See also:

[DataDimension](#)

Definition at line 166 of file `KDChartAbstractCoordinatePlane.cpp`.

References d.

Referenced by `KDChart::CartesianCoordinatePlane::layoutDiagrams()`, and `KDChart::CartesianAxis::maximumSize()`.

```

167 {
168     //KDChart::DataDimensionsList l( d->grid->updateData( this ) );
169     //QDebug() << "AbstractCoordinatePlane::gridDimensionsList() Y-range:" << l.last().end - l.last().
170     //QDebug() << "AbstractCoordinatePlane::gridDimensionsList() X-range:" << l.first().end - l.first
171     return d->grid->updateData( this );
172 }
```

#### 9.46.5.20 [bool KDChart::PolarCoordinatePlane::hasOwnGridAttributes](#) ([bool circular](#)) const

##### Returns:

Returns whether the grid attributes have been set for the respective direction via `setGridAttributes( bool circular )`.

If false, the grid will use the global attributes set by [AbstractCoordinatePlane::globalGridAttributes](#) (or the default attributes, resp.)

##### See also:

[setGridAttributes](#)

[resetGridAttributes](#)

[AbstractCoordinatePlane::globalGridAttributes](#)

Definition at line 410 of file `KDChartPolarCoordinatePlane.cpp`.

References d.

Referenced by `gridAttributes()`.

```

412 {
413     return
414         ( circular )
415         ? d->hasOwnGridAttributesCircular
416         : d->hasOwnGridAttributesSagittal;
417 }

```

#### 9.46.5.21 QRect AbstractAreaBase::innerRect () const [protected, inherited]

Definition at line 228 of file KDChartAbstractAreaBase.cpp.

References KDChart::AbstractAreaBase::areaGeometry(), and KDChart::AbstractAreaBase::getFrameLeadings().

Referenced by KDChart::TextArea::paintAll(), and KDChart::AbstractArea::paintAll().

```

229 {
230     int left;
231     int top;
232     int right;
233     int bottom;
234     getFrameLeadings( left, top, right, bottom );
235     return
236         QRect( QPoint(0,0), areaGeometry().size() )
237         .adjusted( left, top, -right, -bottom );
238 }

```

#### 9.46.5.22 bool KDChart::AbstractCoordinatePlane::isEmpty () const [virtual, inherited]

pure virtual in [QLayoutItem](#)

Definition at line 205 of file KDChartAbstractCoordinatePlane.cpp.

Referenced by KDChart::CartesianCoordinatePlane::layoutDiagrams().

```

206 {
207     return false; // never empty!
208     // coordinate planes with no associated diagrams
209     // are showing a default grid of ( )1..10, 1..10 stepWidth 1
210 }

```

#### 9.46.5.23 bool KDChart::AbstractCoordinatePlane::isRubberBandZoomingEnabled () const [inherited]

##### Returns:

Whether zooming with a rubber band using the mouse is enabled.

Definition at line 280 of file KDChartAbstractCoordinatePlane.cpp.

References d.

```

281 {
282     return d->enableRubberBandZooming;
283 }

```

#### 9.46.5.24 `const bool KDChart::AbstractCoordinatePlane::isVisiblePoint (const QPointF & point)` `const` [inherited]

Tests, if a point is visible on the coordinate plane.

#### Note:

Before calling this function the point must have been translated into coordinate plane space.

Definition at line 403 of file `KDChartAbstractCoordinatePlane.cpp`.

References `d`.

```
404 {
405     return d->isVisiblePoint( this, point );
406 }
```

#### 9.46.5.25 `void PolarCoordinatePlane::layoutDiagrams ()` [protected, virtual]

Distribute the available space among the diagrams and axes.

Implements [KDChart::AbstractCoordinatePlane](#).

Definition at line 238 of file `KDChartPolarCoordinatePlane.cpp`.

References `angleUnit()`, `KDChart::AbstractArea::areaGeometry()`, `d`, `KDChart::AbstractDiagram::dataBoundaries()`, `KDChart::AbstractCoordinatePlane::diagram()`, `KDChart::AbstractCoordinatePlane::diagrams()`, `radiusUnit()`, `startPosition()`, and `KDChart::AbstractPolarDiagram::valueTotals()`.

Referenced by `resizeEvent()`, and `slotLayoutChanged()`.

```
239 {
240     // the rectangle the diagrams cover in the *plane*:
241     // (Why -3? We save 1px on each side for the antialiased drawing, and
242     // respect the way QPainter calculates the width of a painted rect (the
243     // size is the rectangle size plus the pen width). This way, most clipping
244     // for regular pens should be avoided. When pens with a penWidth or larger
245     // than 1 are used, this may not be sufficient.
246     const QRect rect( areaGeometry() );
247     d->contentRect = QRectF ( 1, 1, rect.width() - 3, rect.height() - 3 );
248
249     // FIXME distribute space according to options:
250     const qreal oldStartPosition = startPosition();
251     d->coordinateTransformations.clear();
252     Q_FOREACH( AbstractDiagram* diagram, diagrams() )
253     {
254         AbstractPolarDiagram *polarDiagram = dynamic_cast<AbstractPolarDiagram*>( diagram );
255         Q_ASSERT( polarDiagram );
256         QPair<QPointF, QPointF> dataBoundariesPair = polarDiagram->dataBoundaries();
257
258         const double angleUnit = 360 / polarDiagram->valueTotals();
259 //qDebug() << "-----";
260         const double radius = dataBoundariesPair.second.y();
261 //qDebug() << radius << "==" << dataBoundariesPair.second.y();
262         const double diagramWidth = radius * 2; // == height
263         const double planeWidth = d->contentRect.width();
264         const double planeHeight = d->contentRect.height();
265         const double radiusUnit = qMin( planeWidth, planeHeight ) / diagramWidth;
266 //qDebug() << radiusUnit << "==" << "qMin( "<< planeWidth << ", "<< planeHeight << " ) / "<< diagramWidth;
267         QPointF coordinateOrigin = QPointF ( planeWidth / 2, planeHeight / 2 );
268         coordinateOrigin += d->contentRect.topLeft();
269     }
```

```

270         CoordinateTransformation diagramTransposition;
271         diagramTransposition.originTranslation = coordinateOrigin;
272         diagramTransposition.radiusUnit = radiusUnit;
273         diagramTransposition.angleUnit = angleUnit;
274         diagramTransposition.startPosition = oldStartPosition;
275         diagramTransposition.zoom = ZoomParameters();
276         d->coordinateTransformations.append( diagramTransposition );
277     }
278 }

```

#### 9.46.5.26 void KDChart::AbstractCoordinatePlane::layoutPlanes () [slot, inherited]

Calling [layoutPlanes\(\)](#) on the plane triggers the global KDChart::Chart::slotLayoutPlanes().

Definition at line 263 of file KDChartAbstractCoordinatePlane.cpp.

References KDChart::AbstractCoordinatePlane::needLayoutPlanes().

Referenced by KDChart::AbstractCoordinatePlane::addDiagram(), KDChart::CartesianAxis::layoutPlanes(), KDChart::AbstractCartesianDiagram::layoutPlanes(), and KDChart::AbstractCoordinatePlane::replaceDiagram().

```

264 {
265     // qDebug("KDChart::AbstractCoordinatePlane::relayout() called");
266     emit needLayoutPlanes();
267 }

```

#### 9.46.5.27 int AbstractArea::leftOverlap (bool *doNotRecalculate* = false) const [virtual, inherited]

This is called at layout time by [KDChart::AutoSpacerLayoutItem::sizeHint\(\)](#).

The method triggers AbstractArea::sizeHint() to find out the amount of overlap at the left edge of the area.

##### Note:

The default implementation is not using any caching, it might make sense to implement a more sophisticated solution for derived classes that have complex work to do in sizeHint(). All we have here is a primitive flag to be set by the caller if it is sure that no sizeHint() needs to be called.

Definition at line 77 of file KDChartAbstractArea.cpp.

References d.

Referenced by KDChart::CartesianAxis::maximumSize().

```

78 {
79     // Re-calculate the sizes,
80     // so we also get the amountOf..Overlap members set newly:
81     if( ! doNotRecalculate )
82         sizeHint();
83     return d->amountOfLeftOverlap;
84 }

```

#### 9.46.5.28 QSize KDChart::AbstractCoordinatePlane::maximumSize () const [virtual, inherited]

pure virtual in [QLayoutItem](#)

Definition at line 217 of file KDChartAbstractCoordinatePlane.cpp.

Referenced by KDChart::AbstractCoordinatePlane::sizeHint().

```
218 {
219     // No maximum size set. Especially not parent()->size(), we are not layouting
220     // to the parent widget's size when using Chart::paint()!
221     return QSize(QLAYOUTSIZE_MAX, QLAYOUTSIZE_MAX);
222 }
```

#### 9.46.5.29 QSize KDChart::AbstractCoordinatePlane::minimumSize () const [virtual, inherited]

pure virtual in [QLayoutItem](#)

Definition at line 224 of file KDChartAbstractCoordinatePlane.cpp.

```
225 {
226     return QSize(60, 60); // this default can be overwritten by derived classes
227 }
```

#### 9.46.5.30 QSize KDChart::AbstractCoordinatePlane::minimumSizeHint () const [virtual, inherited]

[reimplemented]

Reimplemented in [KDChart::TernaryCoordinatePlane](#).

Definition at line 144 of file KDChartAbstractCoordinatePlane.cpp.

```
145 {
146     return QSize( 200, 200 );
147 }
```

#### 9.46.5.31 void KDChart::AbstractCoordinatePlane::mouseDoubleClickEvent (QMouseEvent \* event) [inherited]

Definition at line 325 of file KDChartAbstractCoordinatePlane.cpp.

References [d](#), and KDChart::AbstractCoordinatePlane::mousePressEvent().

Referenced by KDChart::Chart::mouseDoubleClickEvent().

```
326 {
327     if( event->button() == Qt::RightButton )
328     {
329         // otherwise the second click gets lost
330         // which is pretty annoying when zooming out fast
331         mousePressEvent( event );
332     }
333     KDAB_FOREACH( AbstractDiagram * a, d->diagrams )
```

```

334     {
335         a->mouseDoubleClickEvent( event );
336     }
337 }

```

#### 9.46.5.32 void KDChart::AbstractCoordinatePlane::mouseMoveEvent (QMouseEvent \* event) [inherited]

Definition at line 387 of file KDChartAbstractCoordinatePlane.cpp.

References `d`, and `KDChart::AbstractCoordinatePlane::geometry()`.

Referenced by `KDChart::Chart::mouseMoveEvent()`.

```

388 {
389     if( d->rubberBand != 0 )
390     {
391         const QRect normalized = QRect( d->rubberBandOrigin, event->pos() ).normalized();
392         d->rubberBand->setGeometry( normalized & geometry() );
393
394         event->accept();
395     }
396
397     KDAB_FOREACH( AbstractDiagram * a, d->diagrams )
398     {
399         a->mouseMoveEvent( event );
400     }
401 }

```

#### 9.46.5.33 void KDChart::AbstractCoordinatePlane::mousePressEvent (QMouseEvent \* event) [inherited]

Definition at line 285 of file KDChartAbstractCoordinatePlane.cpp.

References `d`, `KDChart::AbstractCoordinatePlane::parent()`, `KDChart::AbstractCoordinatePlane::setZoomCenter()`, `KDChart::AbstractCoordinatePlane::setZoomFactorX()`, and `KDChart::AbstractCoordinatePlane::setZoomFactorY()`.

Referenced by `KDChart::AbstractCoordinatePlane::mouseDoubleClickEvent()`, and `KDChart::Chart::mousePressEvent()`.

```

286 {
287     if( event->button() == Qt::LeftButton )
288     {
289         if( d->enableRubberBandZooming && d->rubberBand == 0 )
290             d->rubberBand = new QRubberBand( QRubberBand::Rectangle, qobject_cast< QWidget* >( parent() ) );
291
292         if( d->rubberBand != 0 )
293         {
294             d->rubberBandOrigin = event->pos();
295             d->rubberBand->setGeometry( QRect( event->pos(), QSize() ) );
296             d->rubberBand->show();
297
298             event->accept();
299         }
300     }
301     else if( event->button() == Qt::RightButton )
302     {
303         if( d->enableRubberBandZooming && !d->rubberBandZoomConfigHistory.isEmpty() )
304         {

```

```

305         // restore the last config from the stack
306         ZoomParameters config = d->rubberBandZoomConfigHistory.pop();
307         setZoomFactorX( config.xFactor );
308         setZoomFactorY( config.yFactor );
309         setZoomCenter( config.center() );
310
311         QWidget* const p = qobject_cast< QWidget* >( parent() );
312         if( p != 0 )
313             p->update();
314
315         event->accept();
316     }
317 }
318
319 KDAB_FOREACH( AbstractDiagram * a, d->diagrams )
320 {
321     a->mousePressEvent( event );
322 }
323 }

```

#### 9.46.5.34 void KDChart::AbstractCoordinatePlane::mouseReleaseEvent (QMouseEvent \* event) [inherited]

Definition at line 339 of file KDChartAbstractCoordinatePlane.cpp.

References `d`, `KDChart::AbstractCoordinatePlane::geometry()`, `KDChart::AbstractCoordinatePlane::setZoomCenter()`, `KDChart::AbstractCoordinatePlane::setZoomFactorX()`, `KDChart::AbstractCoordinatePlane::setZoomFactorY()`, `KDChart::AbstractCoordinatePlane::zoomCenter()`, `KDChart::AbstractCoordinatePlane::zoomFactorX()`, and `KDChart::AbstractCoordinatePlane::zoomFactorY()`.

Referenced by `KDChart::Chart::mouseReleaseEvent()`.

```

340 {
341     if( d->rubberBand != 0 )
342     {
343         // save the old config on the stack
344         d->rubberBandZoomConfigHistory.push( ZoomParameters( zoomFactorX(), zoomFactorY(), zoomCenter() ) );
345
346         // this is the height/width of the rubber band in pixel space
347         const double rubberWidth = static_cast< double >( d->rubberBand->width() );
348         const double rubberHeight = static_cast< double >( d->rubberBand->height() );
349
350         // this is the center of the rubber band in pixel space
351         const double rubberCenterX = static_cast< double >( d->rubberBand->geometry().center().x() - 0.5 );
352         const double rubberCenterY = static_cast< double >( d->rubberBand->geometry().center().y() - 0.5 );
353
354         // this is the height/width of the plane in pixel space
355         const double myWidth = static_cast< double >( geometry().width() );
356         const double myHeight = static_cast< double >( geometry().height() );
357
358         // this describes the new center of zooming, relative to the plane pixel space
359         const double newCenterX = rubberCenterX / myWidth / zoomFactorX() + zoomCenter().x() - 0.5 / zoomFactorX();
360         const double newCenterY = rubberCenterY / myHeight / zoomFactorY() + zoomCenter().y() - 0.5 / zoomFactorY();
361
362         // this will be the new zoom factor
363         const double newZoomFactorX = zoomFactorX() * myWidth / rubberWidth;
364         const double newZoomFactorY = zoomFactorY() * myHeight / rubberHeight;
365
366         // and this the new center
367         const QPointF newZoomCenter( newCenterX, newCenterY );
368
369         setZoomFactorX( newZoomFactorX );
370         setZoomFactorY( newZoomFactorY );

```



```

371         setZoomCenter( newZoomCenter );
372
373
374         d->rubberBand->parentWidget()->update();
375         delete d->rubberBand;
376         d->rubberBand = 0;
377
378         event->accept();
379     }
380
381     KDAB_FOREACH( AbstractDiagram * a, d->diagrams )
382     {
383         a->mouseReleaseEvent( event );
384     }
385 }

```

#### 9.46.5.35 void KDChart::AbstractCoordinatePlane::needLayoutPlanes () [signal, inherited]

Emitted when plane needs to trigger the Chart's layouting of the coord.  
planes.

Referenced by KDChart::AbstractCoordinatePlane::layoutPlanes().

#### 9.46.5.36 void KDChart::AbstractCoordinatePlane::needRelayout () [signal, inherited]

Emitted when plane needs to trigger the Chart's layouting.

Referenced by KDChart::AbstractCoordinatePlane::relayout().

#### 9.46.5.37 void KDChart::AbstractCoordinatePlane::needUpdate () [signal, inherited]

Emitted when plane needs to update its drawings.

Referenced by KDChart::AbstractCoordinatePlane::update().

#### 9.46.5.38 void PolarCoordinatePlane::paint (QPainter \*) [virtual]

reimpl

Implements [KDChart::AbstractLayoutItem](#).

Definition at line 141 of file KDChartPolarCoordinatePlane.cpp.

References [d](#), [KDChart::AbstractCoordinatePlane::diagrams\(\)](#), and [KDChart::AbstractCoordinatePlane::geometry\(\)](#).

```

142 {
143     AbstractDiagramList diags = diagrams();
144     if ( d->coordinateTransformations.size() == diags.size() )
145     {
146         PaintContext ctx;
147         ctx.setPainter ( painter );
148         ctx.setCoordinatePlane ( this );
149         ctx.setRectangle ( geometry() /*d->contentRect*/ );
150
151         // paint the coordinate system rulers:
152         d->currentTransformation = & ( d->coordinateTransformations[0] );

```

```

153         d->grid->drawGrid( &ctx );
154
155         // paint the diagrams:
156         for ( int i = 0; i < diags.size(); i++ )
157         {
158             d->currentTransformation = & ( d->coordinateTransformations[i] );
159             PainterSaver painterSaver( painter );
160             diags[i]->paint ( &ctx );
161         }
162         d->currentTransformation = 0;
163     } // else: diagrams have not been set up yet
164 }

```

#### 9.46.5.39 void AbstractArea::paintAll (QPainter & *painter*) [virtual, inherited]

Call `paintAll`, if you want the background and the frame to be drawn before the normal `paint()` is invoked automatically.

Reimplemented from [KDChart::AbstractLayoutItem](#).

Reimplemented in [KDChart::TernaryAxis](#).

Definition at line 123 of file `KDChartAbstractArea.cpp`.

References `KDChart::AbstractArea::areaGeometry()`, `d`, `KDChart::AbstractAreaBase::innerRect()`, `KDChart::AbstractLayoutItem::paint()`, `KDChart::AbstractAreaBase::paintBackground()`, and `KDChart::AbstractAreaBase::paintFrame()`.

Referenced by `KDChart::AbstractArea::paintIntoRect()`.

```

124 {
125     // Paint the background and frame
126     const QRect overlappingArea( geometry().adjusted(
127         -d->amountOfLeftOverlap,
128         -d->amountOfTopOverlap,
129         d->amountOfRightOverlap,
130         d->amountOfBottomOverlap ) );
131     paintBackground( painter, overlappingArea );
132     paintFrame(      painter, overlappingArea );
133
134     // temporarily adjust the widget size, to be sure all content gets calculated
135     // to fit into the inner rectangle
136     const QRect oldGeometry( areaGeometry() );
137     QRect inner( innerRect() );
138     inner.moveTo(
139         oldGeometry.left() + inner.left(),
140         oldGeometry.top()  + inner.top() );
141     const bool needAdjustGeometry = oldGeometry != inner;
142     if( needAdjustGeometry )
143         setGeometry( inner );
144     paint( &painter );
145     if( needAdjustGeometry )
146         setGeometry( oldGeometry );
147     //qDebug() << "AbstractAreaWidget::paintAll() done.";
148 }

```

#### 9.46.5.40 void AbstractAreaBase::paintBackground (QPainter & *painter*, const QRect & *rectangle*) [virtual, inherited]

Definition at line 196 of file `KDChartAbstractAreaBase.cpp`.

References `d`, and `KDCart::AbstractAreaBase::paintBackgroundAttributes()`.

Referenced by `KDCart::TextArea::paintAll()`, `KDCart::AbstractAreaWidget::paintAll()`, and `KDCart::AbstractArea::paintAll()`.

```

197 {
198     Q_ASSERT_X ( d != 0, "AbstractAreaBase::paintBackground()",
199                 "Private class was not initialized!" );
200     paintBackgroundAttributes( painter, rect, d->backgroundAttributes );
201 }
```

#### 9.46.5.41 void AbstractAreaBase::paintBackgroundAttributes (QPainter & painter, const QRect & rectangle, const KDCart::BackgroundAttributes & attributes) [static, inherited]

Definition at line 127 of file `KDCartAbstractAreaBase.cpp`.

References `KDCart::BackgroundAttributes::BackgroundPixmapModeCentered`, `KDCart::BackgroundAttributes::BackgroundPixmapModeNone`, `KDCart::BackgroundAttributes::BackgroundPixmapModeScaled`, `KDCart::BackgroundAttributes::BackgroundPixmapModeStretched`, `KDCart::BackgroundAttributes::brush()`, `KDCart::BackgroundAttributes::isVisible()`, `KDCart::BackgroundAttributes::pixmap()`, and `KDCart::BackgroundAttributes::pixmapMode()`.

Referenced by `KDCart::AbstractAreaBase::paintBackground()`.

```

129 {
130     if( !attributes.isVisible() ) return;
131
132     /* first draw the brush (may contain a pixmap)*/
133     if( Qt::NoBrush != attributes.brush().style() ) {
134         KDCart::PainterSaver painterSaver( &painter );
135         painter.setPen( Qt::NoPen );
136         const QPointF newTopLeft( painter.deviceMatrix().map( rect.topLeft() ) );
137         painter.setBrushOrigin( newTopLeft );
138         painter.setBrush( attributes.brush() );
139         painter.drawRect( rect.adjusted( 0, 0, -1, -1 ) );
140     }
141     /* next draw the backPixmap over the brush */
142     if( !attributes.pixmap().isNull() &&
143         attributes.pixmapMode() != BackgroundAttributes::BackgroundPixmapModeNone ) {
144         QPointF ol = rect.topLeft();
145         if( BackgroundAttributes::BackgroundPixmapModeCentered == attributes.pixmapMode() )
146         {
147             ol.setX( rect.center().x() - attributes.pixmap().width() / 2 );
148             ol.setY( rect.center().y() - attributes.pixmap().height() / 2 );
149             painter.drawPixmap( ol, attributes.pixmap() );
150         } else {
151             QMatrix m;
152             double zW = (double)rect.width() / (double)attributes.pixmap().width();
153             double zH = (double)rect.height() / (double)attributes.pixmap().height();
154             switch( attributes.pixmapMode() ) {
155                 case BackgroundAttributes::BackgroundPixmapModeScaled:
156                 {
157                     double z;
158                     z = qMin( zW, zH );
159                     m.scale( z, z );
160                 }
161                 break;
162                 case BackgroundAttributes::BackgroundPixmapModeStretched:
163                     m.scale( zW, zH );
164                     break;
165                 default:
```

```

166             ; // Cannot happen, previously checked
167         }
168         QPixmap pm = attributes.pixmap().transformed( m );
169         ol.setX( rect.center().x() - pm.width() / 2 );
170         ol.setY( rect.center().y() - pm.height() / 2 );
171         painter.drawPixmap( ol, pm );
172     }
173 }
174 }

```

**9.46.5.42** `void KDChart::AbstractLayoutItem::paintCtx (PaintContext * context)` [virtual, inherited]

Default impl: Paint the complete item using its layouted position and size.

Reimplemented in [KDChart::CartesianAxis](#), and [KDChart::TernaryAxis](#).

Definition at line 77 of file `KDChartLayoutItems.cpp`.

References `KDChart::AbstractLayoutItem::paint()`, and `KDChart::PaintContext::painter()`.

```

78 {
79     if( context )
80         paint( context->painter() );
81 }

```

**9.46.5.43** `void KDChart::PolarCoordinatePlane::paintEvent (QPaintEvent *)` [protected]

**9.46.5.44** `void AbstractAreaBase::paintFrame (QPainter & painter, const QRect & rectangle)` [virtual, inherited]

Definition at line 204 of file `KDChartAbstractAreaBase.cpp`.

References `d`, and `KDChart::AbstractAreaBase::paintFrameAttributes()`.

Referenced by `KDChart::TextArea::paintAll()`, `KDChart::AbstractAreaWidget::paintAll()`, and `KDChart::AbstractArea::paintAll()`.

```

205 {
206     Q_ASSERT_X ( d != 0, "AbstractAreaBase::paintFrame()",
207                 "Private class was not initialized!" );
208     paintFrameAttributes( painter, rect, d->frameAttributes );
209 }

```

**9.46.5.45** `void AbstractAreaBase::paintFrameAttributes (QPainter & painter, const QRect & rectangle, const KDChart::FrameAttributes & attributes)` [static, inherited]

Definition at line 177 of file `KDChartAbstractAreaBase.cpp`.

References `KDChart::FrameAttributes::isVisible()`, and `KDChart::FrameAttributes::pen()`.

Referenced by `KDChart::AbstractAreaBase::paintFrame()`.

```

179 {
180
181     if( !attributes.isVisible() ) return;

```

```

182
183     // Note: We set the brush to NoBrush explicitly here.
184     //       Otherwise we might get a filled rectangle, so any
185     //       previously drawn background would be overwritten by that area.
186
187     const QPen   oldPen( painter.pen() );
188     const QBrush oldBrush( painter.brush() );
189     painter.setPen( attributes.pen() );
190     painter.setBrush( Qt::NoBrush );
191     painter.drawRect( rect.adjusted( 0, 0, -1, -1 ) );
192     painter.setBrush( oldBrush );
193     painter.setPen( oldPen );
194 }

```

#### 9.46.5.46 void AbstractArea::paintIntoRect (QPainter & painter, const QRect & rect)

[virtual, inherited]

Draws the background and frame, then calls [paint\(\)](#).

In most cases there is no need to overwrite this method in a derived class, but you would overwrite [AbstractLayoutItem::paint\(\)](#) instead.

Definition at line 111 of file KDChartAbstractArea.cpp.

References [KDChart::AbstractArea::paintAll\(\)](#).

```

112 {
113     const QRect oldGeometry( geometry() );
114     if( oldGeometry != rect )
115         setGeometry( rect );
116     painter.translate( rect.left(), rect.top() );
117     paintAll( painter );
118     painter.translate( -rect.left(), -rect.top() );
119     if( oldGeometry != rect )
120         setGeometry( oldGeometry );
121 }

```

#### 9.46.5.47 const KDChart::Chart \* KDChart::AbstractCoordinatePlane::parent () const

[inherited]

Definition at line 194 of file KDChartAbstractCoordinatePlane.cpp.

References [d](#).

```

195 {
196     return d->parent;
197 }

```

#### 9.46.5.48 KDChart::Chart \* KDChart::AbstractCoordinatePlane::parent ()

[inherited]

Definition at line 199 of file KDChartAbstractCoordinatePlane.cpp.

References [d](#).

Referenced by [KDChart::AbstractCoordinatePlane::AbstractCoordinatePlane\(\)](#), [KDChart::CartesianAxis::maximumSize\(\)](#), [KDChart::AbstractCoordinatePlane::mousePressEvent\(\)](#), and [KDChart::AbstractCoordinatePlane::setParent\(\)](#).

```

200 {
201     return d->parent;
202 }

```

#### 9.46.5.49 **QLayout\*** **KDChart::AbstractLayoutItem::parentLayout ()** [inherited]

Definition at line 76 of file KDChartLayoutItems.h.

```

77     {
78         return mParentLayout;
79     }

```

#### 9.46.5.50 **void** **KDChart::AbstractArea::positionChanged** ([AbstractArea \\*](#)) [signal, inherited]

Referenced by KDChart::AbstractArea::positionHasChanged().

#### 9.46.5.51 **void** **AbstractArea::positionHasChanged ()** [protected, virtual, inherited]

Reimplemented from [KDChart::AbstractAreaBase](#).

Definition at line 155 of file KDChartAbstractArea.cpp.

References KDChart::AbstractArea::positionChanged().

```

156 {
157     emit positionChanged( this );
158 }

```

#### 9.46.5.52 **void** **KDChart::AbstractCoordinatePlane::propertiesChanged ()** [signal, inherited]

Emitted upon change of a property of the Coordinate Plane or any of its components.

Referenced by KDChart::CartesianCoordinatePlane::addDiagram(), KDChart::CartesianCoordinatePlane::adjustHorizontalRangeToData(), KDChart::CartesianCoordinatePlane::adjustRangesToData(), KDChart::CartesianCoordinatePlane::adjustVerticalRangeToData(), KDChart::CartesianCoordinatePlane::setAutoAdjustGridToZoom(), KDChart::CartesianCoordinatePlane::setAutoAdjustHorizontalRangeToData(), KDChart::CartesianCoordinatePlane::setAutoAdjustVerticalRangeToData(), KDChart::CartesianCoordinatePlane::setAxesCalcModes(), KDChart::CartesianCoordinatePlane::setAxesCalcModeX(), KDChart::CartesianCoordinatePlane::setAxesCalcModeY(), setGridAttributes(), KDChart::CartesianCoordinatePlane::setGridAttributes(), KDChart::CartesianCoordinatePlane::setHorizontalRange(), KDChart::CartesianCoordinatePlane::setHorizontalRangeReversed(), KDChart::CartesianCoordinatePlane::setIsometricScaling(), KDChart::CartesianCoordinatePlane::setVerticalRange(), KDChart::CartesianCoordinatePlane::setVerticalRangeReversed(), KDChart::CartesianCoordinatePlane::setZoomCenter(), KDChart::CartesianCoordinatePlane::setZoomFactorX(), and KDChart::CartesianCoordinatePlane::setZoomFactorY().

#### 9.46.5.53 qreal PolarCoordinatePlane::radiusUnit () const

Definition at line 301 of file KDChartPolarCoordinatePlane.cpp.

References [d](#).

Referenced by [layoutDiagrams\(\)](#).

```
302 {  
303     Q_ASSERT_X ( d->currentTransformation != 0, "PolarCoordinatePlane::radiusUnit",  
304                 "Only call radiusUnit() from within paint()." );  
305     return d->currentTransformation->radiusUnit;  
306 }
```

#### 9.46.5.54 AbstractCoordinatePlane \* KDChart::AbstractCoordinatePlane::referenceCoordinatePlane () const [inherited]

There are two ways, in which planes can be caused to interact, in where they are put layouting wise: The first is the reference plane.

If such a reference plane is set, on a plane, it will use the same cell in the layout as that one. In addition to this, planes can share an axis. In that case they will be layed out in relation to each other as suggested by the position of the axis. If, for example Plane1 and Plane2 share an axis at position Left, that will result in the layout: Axis Plane1 Plane 2, vertically. If Plane1 also happens to be Plane2's reference plane, both planes are drawn over each other. The reference plane concept allows two planes to share the same space even if neither has any axis, and in case there are shared axis, it is used to decided, whether the planes should be painted on top of each other or layed out vertically or horizontally next to each other.

##### Returns:

The reference coordinate plane associated with this one.

Definition at line 184 of file KDChartAbstractCoordinatePlane.cpp.

References [d](#).

```
185 {  
186     return d->referenceCoordinatePlane;  
187 }
```

#### 9.46.5.55 void KDChart::AbstractCoordinatePlane::relayout () [slot, inherited]

Calling [relayout\(\)](#) on the plane triggers the global KDChart::Chart::slotRelayout().

Definition at line 257 of file KDChartAbstractCoordinatePlane.cpp.

References [KDChart::AbstractCoordinatePlane::needRelayout\(\)](#).

```
258 {  
259     //qDebug("KDChart::AbstractCoordinatePlane::relayout() called");  
260     emit needRelayout();  
261 }
```

**9.46.5.56 void KDChart::AbstractLayoutItem::removeFromParentLayout ()** [inherited]

Definition at line 80 of file KDChartLayoutItems.h.

Referenced by KDChart::Chart::takeCoordinatePlane().

```

81         {
82             if( mParentLayout ){
83                 if( widget() )
84                     mParentLayout->removeWidget( widget() );
85                 else
86                     mParentLayout->removeItem( this );
87             }
88         }

```

**9.46.5.57 void AbstractCoordinatePlane::replaceDiagram (AbstractDiagram \* diagram, AbstractDiagram \* oldDiagram = 0)** [virtual, inherited]

Replaces the old diagram, or appends the diagram, if there is none yet.

**Parameters:**

**diagram** The diagram to be used instead of the old diagram. This parameter must not be zero, or the method will do nothing.

**oldDiagram** The diagram to be removed by the new diagram. This diagram will be deleted automatically. If the parameter is omitted, the very first diagram will be replaced. In case, there was no diagram yet, the new diagram will just be added.

**Note:**

If you want to re-use the old diagram, call takeDiagram and addDiagram, instead of using replaceDiagram.

**See also:**

[addDiagram](#), [takeDiagram](#)

Definition at line 86 of file KDChartAbstractCoordinatePlane.cpp.

References KDChart::AbstractCoordinatePlane::addDiagram(), d, KDChart::AbstractCoordinatePlane::diagram(), KDChart::AbstractCoordinatePlane::layoutDiagrams(), KDChart::AbstractCoordinatePlane::layoutPlanes(), KDChart::AbstractCoordinatePlane::takeDiagram(), and KDChart::AbstractCoordinatePlane::update().

Referenced by KDChart::Widget::setType().

```

87 {
88     if( diagram && oldDiagram_ != diagram ){
89         AbstractDiagram* oldDiagram = oldDiagram_;
90         if( d->diagrams.count() ){
91             if( ! oldDiagram )
92                 oldDiagram = d->diagrams.first();
93             takeDiagram( oldDiagram );
94         }
95         delete oldDiagram;
96         addDiagram( diagram );
97         layoutDiagrams();
98         layoutPlanes(); // there might be new axes, etc
99         update();
100     }
101 }

```



**9.46.5.58 void KDChart::PolarCoordinatePlane::resetGridAttributes (bool *circular*)**

Reset the attributes to be used for grid lines drawn in circular direction (or in sagittal direction, resp.

). By calling this method you specify that the global attributes set by [AbstractCoordinatePlane::setGlobalGridAttributes](#) be used.

**See also:**

[setGridAttributes](#), [gridAttributes](#)  
[AbstractCoordinatePlane::globalGridAttributes](#)  
[hasOwnGridAttributes](#)

Definition at line 380 of file KDChartPolarCoordinatePlane.cpp.

References [KDChart::AbstractCoordinatePlane::update\(\)](#).

```
382 {
383     setHasOwnGridAttributes( circular, false );
384     update();
385 }
```

**9.46.5.59 void PolarCoordinatePlane::resizeEvent (QResizeEvent \*) [protected]**

Definition at line 232 of file KDChartPolarCoordinatePlane.cpp.

References [d](#), and [layoutDiagrams\(\)](#).

```
233 {
234     d->initialResizeEventReceived = true;
235     layoutDiagrams();
236 }
```

**9.46.5.60 int AbstractArea::rightOverlap (bool *doNotRecalculate* = false) const [virtual, inherited]**

This is called at layout time by [KDChart::AutoSpacerLayoutItem::sizeHint\(\)](#).

The method triggers [AbstractArea::sizeHint\(\)](#) to find out the amount of overlap at the right edge of the area.

**Note:**

The default implementation is not using any caching, it might make sense to implement a more sophisticated solution for derived classes that have complex work to do in [sizeHint\(\)](#). All we have here is a primitive flag to be set by the caller if it is sure that no [sizeHint\(\)](#) needs to be called.

Definition at line 85 of file KDChartAbstractArea.cpp.

References [d](#).

Referenced by [KDChart::CartesianAxis::maximumSize\(\)](#).

```
86 {
87     // Re-calculate the sizes,
88     // so we also get the amountOf..Overlap members set newly:
89     if( ! doNotRecalculate )
90         sizeHint();
91     return d->amountOfRightOverlap;
92 }
```

#### 9.46.5.61 void AbstractAreaBase::setBackgroundAttributes (const [BackgroundAttributes](#) & a) [inherited]

Definition at line 111 of file KDChartAbstractAreaBase.cpp.

References [d](#), and [KDChart::AbstractAreaBase::positionHasChanged\(\)](#).

```
112 {
113     if( d->backgroundAttributes == a )
114         return;
115
116     d->backgroundAttributes = a;
117     positionHasChanged();
118 }
```

#### 9.46.5.62 void AbstractAreaBase::setFrameAttributes (const [FrameAttributes](#) & a) [inherited]

Definition at line 97 of file KDChartAbstractAreaBase.cpp.

References [d](#), and [KDChart::AbstractAreaBase::positionHasChanged\(\)](#).

Referenced by [KDChart::Legend::clone\(\)](#).

```
98 {
99     if( d->frameAttributes == a )
100         return;
101
102     d->frameAttributes = a;
103     positionHasChanged();
104 }
```

#### 9.46.5.63 void KDChart::AbstractCoordinatePlane::setGeometry (const [QRect](#) & r) [virtual, inherited]

pure virtual in [QLayoutItem](#)

##### Note:

Do not call this function directly, unless you know exactly what you are doing. Geometry management is done by KD Chart's internal layouting measures.

Reimplemented in [KDChart::CartesianCoordinatePlane](#).

Definition at line 236 of file KDChartAbstractCoordinatePlane.cpp.

References [d](#).

Referenced by [KDChart::CartesianCoordinatePlane::setGeometry\(\)](#).

```
237 {
238     // qDebug() << "KDChart::AbstractCoordinatePlane::setGeometry(" << r << ") called";
239     if( d->geometry != r ){
240         d->geometry = r;
241         // Note: We do *not* call update() here
242         // because it would invoke KDChart::update() recursively.
243     }
244 }
```

**9.46.5.64 void KDChart::AbstractCoordinatePlane::setGlobalGridAttributes (const [GridAttributes](#) &) [inherited]**

Set the grid attributes to be used by this coordinate plane.

To disable grid painting, for example, your code should like this:

```
GridAttributes ga = plane->globalGridAttributes();
ga.setGlobalGridVisible( false );
plane->setGlobalGridAttributes( ga );
```

**See also:**

[globalGridAttributes](#)  
[CartesianCoordinatePlane::setGridAttributes](#)

Definition at line 155 of file KDChartAbstractCoordinatePlane.cpp.

References [d](#), and [KDChart::AbstractCoordinatePlane::update\(\)](#).

```
156 {
157     d->gridAttributes = a;
158     update();
159 }
```

**9.46.5.65 void KDChart::PolarCoordinatePlane::setGridAttributes (bool *circular*, const [GridAttributes](#) &)**

Set the attributes to be used for grid lines drawn in circular direction (or in sagittal direction, resp.).

To disable circular grid painting, for example, your code should like this:

```
GridAttributes ga = plane->gridAttributes( bool );
ga.setGridVisible( false );
plane->setGridAttributes( bool, ga );
```

**Note:**

`setGridAttributes` overwrites the global attributes that were set by [AbstractCoordinatePlane::setGlobalGridAttributes](#). To re-activate these global attributes you can call `resetGridAttributes`.

**See also:**

[resetGridAttributes](#), [gridAttributes](#)  
[AbstractCoordinatePlane::setGlobalGridAttributes](#)  
[hasOwnGridAttributes](#)

Definition at line 367 of file KDChartPolarCoordinatePlane.cpp.

References [d](#), [KDChart::AbstractCoordinatePlane::propertiesChanged\(\)](#), and [KDChart::AbstractCoordinatePlane::update\(\)](#).

```
370 {
371     if( circular )
372         d->gridAttributesCircular = a;
373     else
```

```

374         d->gridAttributesSagittal = a;
375         setHasOwnGridAttributes( circular, true );
376         update();
377         emit propertiesChanged();
378     }

```

#### 9.46.5.66 void KDChart::AbstractCoordinatePlane::setGridNeedsRecalculate () [slot, inherited]

Used by the chart to clear the cached grid data.

Definition at line 174 of file KDChartAbstractCoordinatePlane.cpp.

References d.

Referenced by KDChart::Chart::resizeEvent().

```

175 {
176     d->grid->setNeedRecalculate();
177 }

```

#### 9.46.5.67 void KDChart::AbstractCoordinatePlane::setParent (Chart \*parent) [inherited]

Called internally by [KDChart::Chart](#).

Definition at line 189 of file KDChartAbstractCoordinatePlane.cpp.

References d, and KDChart::AbstractCoordinatePlane::parent().

Referenced by KDChart::Chart::addCoordinatePlane(), and KDChart::Chart::takeCoordinatePlane().

```

190 {
191     d->parent = parent;
192 }

```

#### 9.46.5.68 void KDChart::AbstractLayoutItem::setParentLayout (QLayout \*lay) [inherited]

Definition at line 72 of file KDChartLayoutItems.h.

```

73     {
74         mParentLayout = lay;
75     }

```

#### 9.46.5.69 void KDChart::AbstractLayoutItem::setParentWidget (QWidget \*widget) [virtual, inherited]

Inform the item about its widget: This enables the item, to trigger that widget's update, whenever the size of the item's contents has changed.

Thus, you need to call setParentWidget on every item, that has a non-fixed size.

Definition at line 64 of file KDChartLayoutItems.cpp.

References KDChart::AbstractLayoutItem::mParent.

Referenced by KDChart::HeaderFooter::setParent(), and KDChart::AbstractCartesianDiagram::takeAxis().

```
65 {  
66     mParent = widget;  
67 }
```

#### 9.46.5.70 void KDChart::AbstractCoordinatePlane::setReferenceCoordinatePlane ([AbstractCoordinatePlane](#) \* *plane*) [inherited]

Set another coordinate plane to be used as the reference plane for this one.

##### Parameters:

*plane* The coordinate plane to be used the reference plane for this one.

##### See also:

[referenceCoordinatePlane](#)

Definition at line 179 of file KDChartAbstractCoordinatePlane.cpp.

References d.

```
180 {  
181     d->referenceCoordinatePlane = plane;  
182 }
```

#### 9.46.5.71 void KDChart::AbstractCoordinatePlane::setRubberBandZoomingEnabled (bool *enable*) [inherited]

Enables or disables zooming with a rubber band using the mouse.

Definition at line 269 of file KDChartAbstractCoordinatePlane.cpp.

References d.

```
270 {  
271     d->enableRubberBandZooming = enable;  
272  
273     if( !enable && d->rubberBand != 0 )  
274     {  
275         delete d->rubberBand;  
276         d->rubberBand = 0;  
277     }  
278 }
```

#### 9.46.5.72 void PolarCoordinatePlane::setStartPosition (qreal *degrees*)

Specify the rotation of the coordinate plane.

In a Pie diagram this indicates the position where the first pie starts, in a Polar diagram it specifies the Zero position of the circular axis.

##### See also:

[startPosition](#)

Definition at line 313 of file KDChartPolarCoordinatePlane.cpp.

References [d](#), and [KDChart::AbstractCoordinatePlane::diagram\(\)](#).

```
314 {  
315     Q_ASSERT_X ( diagram(), "PolarCoordinatePlane::setStartPosition",  
316                 "setStartPosition() needs a diagram to be associated to the plane." );  
317     d->coordinateTransformations[0].startPosition = degrees;  
318 }
```

#### 9.46.5.73 void PolarCoordinatePlane::setZoomCenter (const QPointF & *center*) [virtual]

Set the point (in value coordinates) to be used as the center point in zoom operations.

##### Parameters:

*center* The point to use.

Reimplemented from [KDChart::AbstractCoordinatePlane](#).

Definition at line 352 of file KDChartPolarCoordinatePlane.cpp.

References [d](#).

```
353 {  
354     d->coordinateTransformations[0].zoom.xCenter = center.x();  
355     d->coordinateTransformations[0].zoom.yCenter = center.y();  
356 }
```

#### 9.46.5.74 void PolarCoordinatePlane::setZoomFactorX (double *factor*) [virtual]

Sets the zoom factor in horizontal direction, that is applied to all coordinate transformations.

##### Parameters:

*factor* The new zoom factor

Reimplemented from [KDChart::AbstractCoordinatePlane](#).

Definition at line 337 of file KDChartPolarCoordinatePlane.cpp.

References [d](#).

```
338 {  
339     d->coordinateTransformations[0].zoom.xFactor = factor;  
340 }
```

#### 9.46.5.75 void PolarCoordinatePlane::setZoomFactorY (double *factor*) [virtual]

Sets the zoom factor in vertical direction, that is applied to all coordinate transformations.

##### Parameters:

*factor* The new zoom factor

Reimplemented from [KDChart::AbstractCoordinatePlane](#).

Definition at line 342 of file KDChartPolarCoordinatePlane.cpp.

References [d](#).

```
343 {
344     d->coordinateTransformations[0].zoom.yFactor = factor;
345 }
```

#### 9.46.5.76 [AbstractCoordinatePlane](#) \* KDChart::AbstractCoordinatePlane::sharedAxisMasterPlane (QPainter \* p = 0) [virtual, inherited]

Reimplemented in [KDChart::CartesianCoordinatePlane](#).

Definition at line 408 of file KDChartAbstractCoordinatePlane.cpp.

Referenced by [KDChart::Plotter::paint\(\)](#), [KDChart::LineDiagram::paint\(\)](#), and [KDChart::BarDiagram::paint\(\)](#).

```
409 {
410     Q_UNUSED( p );
411     return this;
412 }
```

#### 9.46.5.77 QSize KDChart::AbstractCoordinatePlane::sizeHint () const [virtual, inherited]

pure virtual in [QLayoutItem](#)

Definition at line 229 of file KDChartAbstractCoordinatePlane.cpp.

References [KDChart::AbstractCoordinatePlane::maximumSize\(\)](#).

```
230 {
231     // we return our maximum (which is the full size of the Chart)
232     // even if we know the plane will be smaller
233     return maximumSize();
234 }
```

#### 9.46.5.78 void KDChart::AbstractLayoutItem::sizeHintChanged () const [virtual, inherited]

Report changed size hint: ask the parent widget to recalculate the layout.

Definition at line 86 of file KDChartLayoutItems.cpp.

References [KDChart::AbstractLayoutItem::mParent](#).

Referenced by [KDChart::TextLayoutItem::sizeHint\(\)](#).

```
87 {
88     // This is exactly like what QWidget::updateGeometry does.
89     // qDebug("KDChart::AbstractLayoutItem::sizeHintChanged() called");
90     if( mParent ) {
91         if ( mParent->layout() )
92             mParent->layout()->invalidate();
```

```

93         else
94             QApplication::postEvent( mParent, new QEvent( QEvent::LayoutRequest ) );
95     }
96 }

```

#### 9.46.5.79 QSizePolicy KDChart::AbstractCoordinatePlane::sizePolicy () const [virtual, inherited]

[reimplemented]

Reimplemented in [KDChart::TernaryCoordinatePlane](#).

Definition at line 150 of file KDChartAbstractCoordinatePlane.cpp.

```

151 {
152     return QSizePolicy( QSizePolicy::MinimumExpanding, QSizePolicy::MinimumExpanding );
153 }

```

#### 9.46.5.80 void PolarCoordinatePlane::slotLayoutChanged ([AbstractDiagram](#) \* *diagram*) [protected, slot]

Definition at line 308 of file KDChartPolarCoordinatePlane.cpp.

References [d](#), and [layoutDiagrams\(\)](#).

Referenced by [addDiagram\(\)](#).

```

309 {
310     if ( d->initialResizeEventReceived ) layoutDiagrams();
311 }

```

#### 9.46.5.81 qreal PolarCoordinatePlane::startPosition () const

Retrieve the rotation of the coordinate plane.

See also:

[setStartPosition](#)

Definition at line 320 of file KDChartPolarCoordinatePlane.cpp.

References [d](#).

Referenced by [layoutDiagrams\(\)](#).

```

321 {
322     return d->coordinateTransformations.size()
323         ? d->coordinateTransformations[0].startPosition
324         : 0.0;
325 }

```



### 9.46.5.82 void AbstractCoordinatePlane::takeDiagram ([AbstractDiagram](#) \* *diagram*) [virtual, inherited]

Removes the diagram from the plane, without deleting it.

The plane no longer owns the diagram, so it is the caller's responsibility to delete the diagram.

See also:

[addDiagram](#), [replaceDiagram](#)

Definition at line 104 of file KDChartAbstractCoordinatePlane.cpp.

References [d](#), [KDChart::AbstractCoordinatePlane::diagram\(\)](#), [KDChart::AbstractCoordinatePlane::layoutDiagrams\(\)](#), [KDChart::AbstractDiagram::setCoordinatePlane\(\)](#), and [KDChart::AbstractCoordinatePlane::update\(\)](#).

Referenced by [KDChart::AbstractCoordinatePlane::replaceDiagram\(\)](#).

```

105 {
106     const int idx = d->diagrams.indexOf( diagram );
107     if( idx != -1 ){
108         d->diagrams.removeAt( idx );
109         diagram->setParent( 0 );
110         diagram->setCoordinatePlane( 0 );
111         layoutDiagrams();
112         update();
113     }
114 }
```

### 9.46.5.83 int AbstractArea::topOverlap (bool *doNotRecalculate* = false) const [virtual, inherited]

This is called at layout time by [KDChart::AutoSpacerLayoutItem::sizeHint\(\)](#).

The method triggers [AbstractArea::sizeHint\(\)](#) to find out the amount of overlap at the top edge of the area.

**Note:**

The default implementation is not using any caching, it might make sense to implement a more sophisticated solution for derived classes that have complex work to do in [sizeHint\(\)](#). All we have here is a primitive flag to be set by the caller if it is sure that no [sizeHint\(\)](#) needs to be called.

Definition at line 93 of file KDChartAbstractArea.cpp.

References [d](#).

Referenced by [KDChart::CartesianAxis::maximumSize\(\)](#).

```

94 {
95     // Re-calculate the sizes,
96     // so we also get the amountOf..Overlap members set newly:
97     if( ! doNotRecalculate )
98         sizeHint();
99     return d->amountOfTopOverlap;
100 }
```

#### 9.46.5.84 `const QPointF PolarCoordinatePlane::translate (const QPointF & diagramPoint) const` [virtual]

Translate the given point in value space coordinates to a position in pixel space.

##### Parameters:

*diagramPoint* The point in value coordinates.

##### Returns:

The translated point.

Implements [KDChart::AbstractCoordinatePlane](#).

Definition at line 280 of file `KDChartPolarCoordinatePlane.cpp`.

References [d](#).

Referenced by `buildReferenceRect()`.

```
281 {
282     Q_ASSERT_X ( d->currentTransformation != 0, "PolarCoordinatePlane::translate",
283                 "Only call translate() from within paint()." );
284     return d->currentTransformation->translate ( diagramPoint );
285 }
```

#### 9.46.5.85 `const QPointF PolarCoordinatePlane::translatePolar (const QPointF & diagramPoint) const`

Definition at line 287 of file `KDChartPolarCoordinatePlane.cpp`.

References [d](#).

```
288 {
289     Q_ASSERT_X ( d->currentTransformation != 0, "PolarCoordinatePlane::translate",
290                 "Only call translate() from within paint()." );
291     return d->currentTransformation->translatePolar ( diagramPoint );
292 }
```

#### 9.46.5.86 `void KDChart::AbstractCoordinatePlane::update ()` [slot, inherited]

Calling [update\(\)](#) on the plane triggers the global `KDChart::Chart::update()`.

Definition at line 251 of file `KDChartAbstractCoordinatePlane.cpp`.

References `KDChart::AbstractCoordinatePlane::needUpdate()`.

Referenced by `KDChart::AbstractCoordinatePlane::addDiagram()`, `KDChart::CartesianCoordinatePlane::layoutDiagrams()`, `KDChart::AbstractCoordinatePlane::replaceDiagram()`, `resetGridAttributes()`, `KDChart::CartesianCoordinatePlane::resetGridAttributes()`, `KDChart::AbstractCoordinatePlane::setGlobalGridAttributes()`, `setGridAttributes()`, `KDChart::CartesianCoordinatePlane::setGridAttributes()`, and `KDChart::AbstractCoordinatePlane::takeDiagram()`.

```
252 {
253     //qDebug("KDChart::AbstractCoordinatePlane::update() called");
254     emit needUpdate();
255 }
```

**9.46.5.87 QPointF PolarCoordinatePlane::zoomCenter () const** [virtual]**Returns:**

The center point (in value coordinates) of the coordinate plane, that is used for zoom operations.

Reimplemented from [KDChart::AbstractCoordinatePlane](#).

Definition at line 347 of file KDChartPolarCoordinatePlane.cpp.

References [d](#).

```
348 {  
349     return QPointF( d->coordinateTransformations[0].zoom.xCenter, d->coordinateTransformations[0].zoom  
350 }
```

**9.46.5.88 double PolarCoordinatePlane::zoomFactorX () const** [virtual]**Returns:**

The zoom factor in horizontal direction, that is applied to all coordinate transformations.

Reimplemented from [KDChart::AbstractCoordinatePlane](#).

Definition at line 327 of file KDChartPolarCoordinatePlane.cpp.

References [d](#).

```
328 {  
329     return d->coordinateTransformations[0].zoom.xFactor;  
330 }
```

**9.46.5.89 double PolarCoordinatePlane::zoomFactorY () const** [virtual]**Returns:**

The zoom factor in vertical direction, that is applied to all coordinate transformations.

Reimplemented from [KDChart::AbstractCoordinatePlane](#).

Definition at line 332 of file KDChartPolarCoordinatePlane.cpp.

References [d](#).

```
333 {  
334     return d->coordinateTransformations[0].zoom.yFactor;  
335 }
```

**9.46.6 Member Data Documentation****9.46.6.1 QWidget\* KDChart::AbstractLayoutItem::mParent** [protected, inherited]

Definition at line 90 of file KDChartLayoutItems.h.

Referenced by [KDChart::AbstractLayoutItem::setParentWidget\(\)](#), [KDChart::TextLayoutItem::setText\(\)](#), [KDChart::TextLayoutItem::setTextAttributes\(\)](#), and [KDChart::AbstractLayoutItem::sizeHintChanged\(\)](#).

**9.46.6.2** `QLayout*` [KDChart::AbstractLayoutItem::mParentLayout](#) [protected,  
inherited]

Definition at line 91 of file `KDChartLayoutItems.h`.

Referenced by `KDChart::AutoSpacerLayoutItem::paint()`.

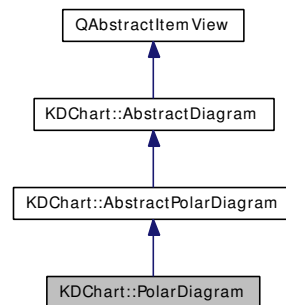
The documentation for this class was generated from the following files:

- [KDChartPolarCoordinatePlane.h](#)
- [KDChartPolarCoordinatePlane.cpp](#)

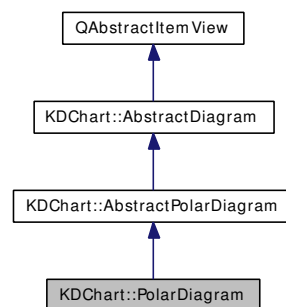
## 9.47 KDChart::PolarDiagram Class Reference

```
#include <KDChartPolarDiagram.h>
```

Inheritance diagram for KDChart::PolarDiagram:



Collaboration diagram for KDChart::PolarDiagram:



### 9.47.1 Detailed Description

**PolarDiagram** defines a common polar diagram.

Definition at line 46 of file KDChartPolarDiagram.h.

#### Signals

- void [dataHidden](#) ()  
*This signal is emitted, when the hidden status of at least one data cell was (un)set.*
- void [layoutChanged](#) ([AbstractDiagram](#) \*)  
*Diagrams are supposed to emit this signal, when the layout of one of their element changes.*
- void [modelsChanged](#) ()  
*This signal is emitted, when either the model or the [AttributesModel](#) is replaced.*
- void [propertiesChanged](#) ()  
*Emitted upon change of a property of the Diagram.*

## Public Member Functions

- bool [allowOverlappingDataValueTexts](#) () const  
*Returns:*  
*Whether data value labels are allowed to overlap.*
- bool [antiAliasing](#) () const  
*Returns:*  
*Whether anti-aliasing is to be used for rendering this diagram.*
- virtual [AttributesModel](#) \* [attributesModel](#) () const  
*Returns the [AttributesModel](#), that is used by this diagram.*
- QBrush [brush](#) (const QModelIndex &index) const  
*Retrieve the brush to be used, for painting the datapoint at the given index in the model.*
- QBrush [brush](#) (int dataset) const  
*Retrieve the brush to be used for the given dataset.*
- QBrush [brush](#) () const  
*Retrieve the brush to be used for painting datapoints globally.*
- virtual [PolarDiagram](#) \* [clone](#) () const  
*Creates an exact copy of this diagram.*
- bool [closeDatasets](#) () const
- int [columnCount](#) () const
- bool [compare](#) (const [AbstractDiagram](#) \*other) const  
*Returns true if both diagrams have the same settings.*
- [AbstractCoordinatePlane](#) \* [coordinatePlane](#) () const  
*The coordinate plane associated with the diagram.*
- const QPair< QPointF, QPointF > [dataBoundaries](#) () const  
*Return the bottom left and top right data point, that the diagram will display (unless the grid adjusts these values).*
- virtual void [dataChanged](#) (const QModelIndex &topLeft, const QModelIndex &bottomRight)  
*[reimplemented]*
- QList< QBrush > [datasetBrushes](#) () const  
*The set of dataset brushes currently used, for use in legends, etc.*
- int [datasetDimension](#) () const  
*The dataset dimension of a diagram determines, how many value dimensions it expects each datapoint to have.*
- QStringList [datasetLabels](#) () const  
*The set of dataset labels currently displayed, for use in legends, etc.*

- `QList< MarkerAttributes > datasetMarkers ()` const  
*The set of dataset markers currently used, for use in legends, etc.*
- `QList< QPen > datasetPens ()` const  
*The set of dataset pens currently used, for use in legends, etc.*
- `DataValueAttributes dataValueAttributes (const QModelIndex &index)` const  
*Retrieve the [DataValueAttributes](#) for the given index.*
- `DataValueAttributes dataValueAttributes (int column)` const  
*Retrieve the [DataValueAttributes](#) for the given dataset.*
- `DataValueAttributes dataValueAttributes ()` const  
*Retrieve the [DataValueAttributes](#) specified globally.*
- `virtual void doItemsLayout ()`  
*[reimplemented]*
- `virtual int horizontalOffset ()` const  
*[reimplemented]*
- `virtual QModelIndex indexAt (const QPoint &point)` const  
*[reimplemented]*
- `QModelIndexList indexesAt (const QPoint &point)` const  
*This method is added alongside with `indexAt` from QAIM, since in kdchart multiple indexes can be displayed at the same spot.*
- `bool isHidden (const QModelIndex &index)` const  
*Retrieve the hidden status for the given index.*
- `bool isHidden (int column)` const  
*Retrieve the hidden status for the given dataset.*
- `bool isHidden ()` const  
*Retrieve the hidden status specified globally.*
- `virtual bool isIndexHidden (const QModelIndex &index)` const  
*[reimplemented]*
- `QStringList itemRowLabels ()` const  
*The set of item row labels currently displayed, for use in Abscissa axes, etc.*
- `virtual QModelIndex moveCursor (CursorAction cursorAction, Qt::KeyboardModifiers modifiers)`  
*[reimplemented]*
- `virtual double numberOfGridRings ()` const  
*[reimplemented]*
- `virtual double numberOfValuesPerDataset ()` const

*[reimplemented]*

- void [paintDataValueText](#) (QPainter \*painter, const QModelIndex &index, const QPointF &pos, double value)
- void [paintMarker](#) (QPainter \*painter, const QModelIndex &index, const QPointF &pos)
- virtual void [paintMarker](#) (QPainter \*painter, const [MarkerAttributes](#) &markerAttributes, const QBrush &brush, const QPen &, const QPointF &point, const QSizeF &size)
- QPen [pen](#) (const QModelIndex &index) const

*Retrieve the pen to be used, for painting the datapoint at the given index in the model.*

- QPen [pen](#) (int dataset) const

*Retrieve the pen to be used for the given dataset.*

- QPen [pen](#) () const

*Retrieve the pen to be used for painting datapoints globally.*

- bool [percentMode](#) () const
- const [PolarCoordinatePlane](#) \* [polarCoordinatePlane](#) () const
- [PolarDiagram](#) (QWidget \*parent=0, [PolarCoordinatePlane](#) \*plane=0)
- virtual void [resize](#) (const QSizeF &area)

*[reimplemented]*

- bool [rotateCircularLabels](#) () const
- virtual void [scrollTo](#) (const QModelIndex &index, ScrollHint hint=EnsureVisible)

*[reimplemented]*

- void [setAllowOverlappingDataValueTexts](#) (bool allow)

*Set whether data value labels are allowed to overlap.*

- void [setAntiAliasing](#) (bool enabled)

*Set whether anti-aliasing is to be used while rendering this diagram.*

- virtual void [setAttributesModel](#) ([AttributesModel](#) \*model)

*Associate an [AttributesModel](#) with this diagram.*

- void [setBrush](#) (const QBrush &brush)

*Set the brush to be used, for painting all datasets in the model.*

- void [setBrush](#) (int dataset, const QBrush &brush)

*Set the brush to be used, for painting the given dataset.*

- void [setBrush](#) (const QModelIndex &index, const QBrush &brush)

*Set the brush to be used, for painting the datapoint at the given index.*

- void [setCloseDatasets](#) (bool closeDatasets)

*Close each of the data series by connecting the last point to its respective start point.*

- virtual void [setCoordinatePlane](#) ([AbstractCoordinatePlane](#) \*plane)

*Set the coordinate plane associated with the diagram.*



- void [setDatasetDimension](#) (int dimension)  
*Sets the dataset dimension of the diagram.*
- void [setDataValueAttributes](#) (const [DataValueAttributes](#) &a)  
*Set the [DataValueAttributes](#) for all datapoints in the model.*
- void [setDataValueAttributes](#) (int dataset, const [DataValueAttributes](#) &a)  
*Set the [DataValueAttributes](#) for the given dataset.*
- void [setDataValueAttributes](#) (const QModelIndex &index, const [DataValueAttributes](#) &a)  
*Set the [DataValueAttributes](#) for the given index.*
- void [setHidden](#) (bool hidden)  
*Hide (or unhide, resp.)*
- void [setHidden](#) (int column, bool hidden)  
*Hide (or unhide, resp.)*
- void [setHidden](#) (const QModelIndex &index, bool hidden)  
*Hide (or unhide, resp.)*
- virtual void [setModel](#) (QAbstractItemModel \*model)  
*Associate a model with the diagram.*
- void [setPen](#) (const QPen &pen)  
*Set the pen to be used, for painting all datasets in the model.*
- void [setPen](#) (int dataset, const QPen &pen)  
*Set the pen to be used, for painting the given dataset.*
- void [setPen](#) (const QModelIndex &index, const QPen &pen)  
*Set the pen to be used, for painting the datapoint at the given index.*
- void [setPercentMode](#) (bool percent)
- virtual void [setRootIndex](#) (const QModelIndex &idx)  
*Set the root index in the model, where the diagram starts referencing data for display.*
- void [setRotateCircularLabels](#) (bool rotateCircularLabels)
- virtual void [setSelection](#) (const QRect &rect, QItemSelectionModel::SelectionFlags command)  
*[reimplemented]*
- void [setShowDelimitersAtPosition](#) ([Position](#) position, bool showDelimiters)
- void [setShowLabelsAtPosition](#) ([Position](#) position, bool showLabels)
- void [setUnitPrefix](#) (const QString &prefix, Qt::Orientation orientation)  
*Sets the unit prefix for all values.*
- void [setUnitPrefix](#) (const QString &prefix, int column, Qt::Orientation orientation)  
*Sets the unit prefix for one value.*
- void [setUnitSuffix](#) (const QString &suffix, Qt::Orientation orientation)

*Sets the unit suffix for all values.*

- void [setUnitSuffix](#) (const QString &suffix, int column, Qt::Orientation orientation)

*Sets the unit suffix for one value.*

- void [setZeroDegreePosition](#) (int degrees)
- bool [showDelimitersAtPosition](#) ([Position](#) position) const
- bool [showLabelsAtPosition](#) ([Position](#) position) const
- QString [unitPrefix](#) (Qt::Orientation orientation) const

*Returns the global unit prefix.*

- QString [unitPrefix](#) (int column, Qt::Orientation orientation, bool fallback=false) const

*Returns the unit prefix for a special value.*

- QString [unitSuffix](#) (Qt::Orientation orientation) const

*Returns the global unit suffix.*

- QString [unitSuffix](#) (int column, Qt::Orientation orientation, bool fallback=false) const

*Returns the unit suffix for a special value.*

- void [update](#) () const

- void [useDefaultColors](#) ()

*Set the palette to be used, for painting datasets to the default palette.*

- void [useRainbowColors](#) ()

*Set the palette to be used, for painting datasets to the rainbow palette.*

- virtual bool [usesExternalAttributesModel](#) () const

*Returns whether the diagram is using its own built-in attributes model or an attributes model that was set via [setAttributesModel](#).*

- void [useSubduedColors](#) ()

*Set the palette to be used, for painting datasets to the subdued palette.*

- virtual double [valueTotals](#) () const

*[reimplemented]*

- virtual int [verticalOffset](#) () const

*[reimplemented]*

- virtual QRect [visualRect](#) (const QModelIndex &index) const

*[reimplemented]*

- virtual QRegion [visualRegionForSelection](#) (const QItemSelection &selection) const

*[reimplemented]*

- int [zeroDegreePosition](#) () const

- virtual [~PolarDiagram](#) ()

## Protected Member Functions

- QModelIndex [attributesModelRootIndex](#) () const
- virtual const QPair< QPointF, QPointF > [calculateDataBoundaries](#) () const  
[reimplemented]
- virtual bool [checkInvariants](#) (bool justReturnTheStatus=false) const
- QModelIndex [columnToIndex](#) (int column) const
- virtual void [paint](#) (PaintContext \*paintContext)  
[reimplemented]
- virtual void [paintDataValueTexts](#) (QPainter \*painter)
- void [paintEvent](#) (QPaintEvent \*)
- virtual void [paintMarkers](#) (QPainter \*painter)
- virtual void [paintPolarMarkers](#) (PaintContext \*ctx, const QPolygonF &polygon)
- void [resizeEvent](#) (QResizeEvent \*)
- void [setAttributesModelRootIndex](#) (const QModelIndex &)
- void [setDataBoundariesDirty](#) () const
- double [valueForCell](#) (int row, int column) const  
*Helper method, retrieving the data value (DisplayRole) for a given row and column.*

## 9.47.2 Constructor & Destructor Documentation

### 9.47.2.1 PolarDiagram::PolarDiagram (QWidget \*parent = 0, PolarCoordinatePlane \*plane = 0) [explicit]

Definition at line 49 of file KDChartPolarDiagram.cpp.

Referenced by clone().

```

49                                     :
50     AbstractPolarDiagram( new Private( ), parent, plane )
51 {
52 }
```

### 9.47.2.2 PolarDiagram::~~PolarDiagram () [virtual]

Definition at line 54 of file KDChartPolarDiagram.cpp.

```

55 {
56 }
```

## 9.47.3 Member Function Documentation

### 9.47.3.1 bool AbstractDiagram::allowOverlappingDataValueTexts () const [inherited]

**Returns:**

Whether data value labels are allowed to overlap.

Definition at line 440 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::compare().

```
441 {
442     return d->allowOverlappingDataValueTexts;
443 }
```

### 9.47.3.2 bool AbstractDiagram::antiAliasing () const [inherited]

#### Returns:

Whether anti-aliasing is to be used for rendering this diagram.

Definition at line 451 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::compare().

```
452 {
453     return d->antiAliasing;
454 }
```

### 9.47.3.3 [AttributesModel](#) \* AbstractDiagram::attributesModel () const [virtual, inherited]

Returns the [AttributesModel](#), that is used by this diagram.

By default each diagram owns its own [AttributesModel](#), which should never be deleted. Only if a user-supplied [AttributesModel](#) has been set does the pointer returned here not belong to the diagram.

#### Returns:

The [AttributesModel](#) associated with the diagram.

#### See also:

[setAttributesModel](#)

Definition at line 280 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::brush(), KDChart::AbstractDiagram::compare(), KDChart::AbstractDiagram::datasetBrushes(), KDChart::AbstractDiagram::datasetLabels(), KDChart::AbstractDiagram::datasetMarkers(), KDChart::AbstractDiagram::datasetPens(), KDChart::AbstractDiagram::dataValueAttributes(), KDChart::AbstractDiagram::isHidden(), KDChart::AbstractDiagram::itemRowLabels(), KDChart::RingDiagram::paint(), paint(), KDChart::AbstractDiagram::pen(), KDChart::AbstractCartesianDiagram::setAttributesModel(), KDChart::BarDiagram::setBarAttributes(), KDChart::AbstractDiagram::setBrush(), KDChart::AbstractCartesianDiagram::setModel(), KDChart::AbstractDiagram::setPen(), and KDChart::TernaryLineDiagram::TernaryLineDiagram().

```
281 {
282     return d->attributesModel;
283 }
```

### 9.47.3.4 QModelIndex AbstractDiagram::attributesModelRootIndex () const [protected, inherited]

returns a QModelIndex pointing into the [AttributesModel](#) that corresponds to the root index of the diagram.

Definition at line 302 of file KDChartAbstractDiagram.cpp.

References [d](#).

Referenced by [KDChart::AbstractDiagram::datasetBrushes\(\)](#), [KDChart::AbstractDiagram::datasetLabels\(\)](#), [KDChart::AbstractDiagram::datasetMarkers\(\)](#), [KDChart::AbstractDiagram::datasetPens\(\)](#), [KDChart::AbstractDiagram::itemRowLabels\(\)](#), [KDChart::Plotter::numberOfAbscissaSegments\(\)](#), [KDChart::LineDiagram::numberOfAbscissaSegments\(\)](#), [KDChart::BarDiagram::numberOfAbscissaSegments\(\)](#), [KDChart::Plotter::numberOfOrdinateSegments\(\)](#), [KDChart::LineDiagram::numberOfOrdinateSegments\(\)](#), [KDChart::BarDiagram::numberOfOrdinateSegments\(\)](#), [KDChart::AbstractDiagram::valueForCell\(\)](#), and [KDChart::LineDiagram::valueForCellTesting\(\)](#).

```

303 {
304     if ( !d->attributesModelRootIndex.isValid() )
305         d->attributesModelRootIndex = d->attributesModel->mapFromSource( rootIndex() );
306     return d->attributesModelRootIndex;
307 }
```

### 9.47.3.5 QBrush AbstractDiagram::brush (const QModelIndex & index) const [inherited]

Retrieve the brush to be used, for painting the datapoint at the given index in the model.

#### Parameters:

**index** The index of the datapoint in the model.

#### Returns:

The brush to use for painting.

Definition at line 838 of file KDChartAbstractDiagram.cpp.

References [KDChart::AbstractDiagram::attributesModel\(\)](#), [KDChart::AbstractDiagram::brush\(\)](#), [KDChart::AttributesModel::data\(\)](#), [KDChart::DatasetBrushRole](#), and [KDChart::AbstractDiagram::datasetDimension\(\)](#).

```

839 {
840     if( datasetDimension() > 1 )
841         return brush( index.column() );
842     return QVariantValue<QBrush>(
843         attributesModel()->data(
844             attributesModel()->mapFromSource( index ),
845             DatasetBrushRole ) );
846 }
```

### 9.47.3.6 QBrush AbstractDiagram::brush (int dataset) const [inherited]

Retrieve the brush to be used for the given dataset.

This will fall back automatically to what was set at model level, if there are no dataset specific settings.

**Parameters:**

*dataset* The dataset to retrieve the brush for.

**Returns:**

The brush to use for painting.

Definition at line 828 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::columnToIndex(), KDChart::AttributesModel::data(), KDChart::DatasetBrushRole, and KDChart::AbstractDiagram::datasetDimension().

```

829 {
830     if( datasetDimension() > 1 )
831         dataset /= datasetDimension();
832     return qVariantValue<QBrush>(
833         attributesModel()->data(
834             attributesModel()->mapFromSource( columnToIndex( dataset ) ),
835             DatasetBrushRole ) );
836 }
```

**9.47.3.7 QBrush AbstractDiagram::brush () const** [inherited]

Retrieve the brush to be used for painting datapoints globally.

This will fall back automatically to the default settings, if there are no specific settings.

**Returns:**

The brush to use for painting.

Definition at line 822 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AttributesModel::data(), and KDChart::DatasetBrushRole.

Referenced by KDChart::AbstractDiagram::brush(), KDChart::TernaryPointDiagram::paint(), KDChart::TernaryLineDiagram::paint(), KDChart::RingDiagram::paint(), paint(), and KDChart::AbstractDiagram::paintMarker().

```

823 {
824     return qVariantValue<QBrush>(
825         attributesModel()->data( DatasetBrushRole ) );
826 }
```

**9.47.3.8 const QPair< QPointF, QPointF > PolarDiagram::calculateDataBoundaries () const**  
[protected, virtual]

[reimplemented]

Implements [KDChart::AbstractDiagram](#).

Definition at line 100 of file KDChartPolarDiagram.cpp.

References KDChart::AbstractDiagram::checkInvariants().

```

101 {
102     if ( !checkInvariants(true) ) return QPair<QPointF, QPointF>( QPointF( 0, 0 ), QPointF( 0, 0 ) );
103     const int rowCount = model()->rowCount(rootIndex());
104     const int colCount = model()->columnCount(rootIndex());
105     double xMin = 0.0;
106     double xMax = colCount;
107     double yMin = 0, yMax = 0;
108     for ( int j=0; j<colCount; ++j ) {
109         for ( int i=0; i<rowCount; ++i ) {
110             double value = model()->data( model()->index( i, j, rootIndex() ) ).toDouble();
111             yMax = qMax( yMax, value );
112         }
113     }
114     QPointF bottomLeft ( QPointF( xMin, yMin ) );
115     QPointF topRight ( QPointF( xMax, yMax ) );
116     return QPair<QPointF, QPointF> ( bottomLeft, topRight );
117 }

```

### 9.47.3.9 bool AbstractDiagram::checkInvariants (bool *justReturnTheStatus* = false) const [protected, virtual, inherited]

Definition at line 1060 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::coordinatePlane().

Referenced by KDChart::RingDiagram::calculateDataBoundaries(), calculateDataBoundaries(), KDChart::Plotter::calculateDataBoundaries(), KDChart::PieDiagram::calculateDataBoundaries(), KDChart::LineDiagram::calculateDataBoundaries(), KDChart::BarDiagram::calculateDataBoundaries(), KDChart::RingDiagram::paint(), paint(), KDChart::Plotter::paint(), KDChart::PieDiagram::paint(), KDChart::LineDiagram::paint(), KDChart::BarDiagram::paint(), KDChart::AbstractDiagram::paint-DataValueTexts(), KDChart::AbstractDiagram::paintMarker(), and KDChart::AbstractDiagram::paint-Markers().

```

1061 {
1062     if( ! justReturnTheStatus ){
1063         Q_ASSERT_X ( model(), "AbstractDiagram::checkInvariants()",
1064                     "There is no usable model set, for the diagram." );
1065
1066         Q_ASSERT_X ( coordinatePlane(), "AbstractDiagram::checkInvariants()",
1067                     "There is no usable coordinate plane set, for the diagram." );
1068     }
1069     return model() && coordinatePlane();
1070 }

```

### 9.47.3.10 PolarDiagram \* PolarDiagram::clone () const [virtual]

Creates an exact copy of this diagram.

Definition at line 89 of file KDChartPolarDiagram.cpp.

References closeDatasets(), d, PolarDiagram(), rotateCircularLabels(), showDelimitersAtPosition(), and showLabelsAtPosition().

```

90 {
91     PolarDiagram* newDiagram = new PolarDiagram( new Private( *d ) );
92     // This needs to be copied after the fact
93     newDiagram->d->showDelimitersAtPosition = d->showDelimitersAtPosition;
94     newDiagram->d->showLabelsAtPosition = d->showLabelsAtPosition;
95     newDiagram->d->rotateCircularLabels = d->rotateCircularLabels;

```

```

96     newDiagram->d->closeDatasets = d->closeDatasets;
97     return newDiagram;
98 }

```

#### 9.47.3.11 bool PolarDiagram::closeDatasets () const

Definition at line 240 of file KDChartPolarDiagram.cpp.

References `d`.

Referenced by `clone()`, and `paint()`.

```

241 {
242     return d->closeDatasets;
243 }

```

#### 9.47.3.12 int AbstractPolarDiagram::columnCount () const [inherited]

Definition at line 60 of file KDChartAbstractPolarDiagram.cpp.

References `KDChart::AbstractPolarDiagram::numberOfValuesPerDataset()`.

Referenced by `KDChart::PieDiagram::calculateDataBoundaries()`, `KDChart::PieDiagram::paint()`, and `KDChart::PieDiagram::valueTotals()`.

```

61 {
62     return static_cast<int>( numberOfValuesPerDataset() );
63 }

```

#### 9.47.3.13 QModelIndex AbstractDiagram::columnToIndex (int column) const [protected, inherited]

Definition at line 309 of file KDChartAbstractDiagram.cpp.

Referenced by `KDChart::BarDiagram::barAttributes()`, `KDChart::AbstractDiagram::brush()`, `KDChart::AbstractDiagram::dataValueAttributes()`, `KDChart::AbstractDiagram::isHidden()`, `KDChart::Plotter::lineAttributes()`, `KDChart::LineDiagram::lineAttributes()`, `KDChart::AbstractDiagram::pen()`, `KDChart::AbstractPieDiagram::pieAttributes()`, `KDChart::BarDiagram::threeDBarAttributes()`, `KDChart::Plotter::threeDLineAttributes()`, `KDChart::LineDiagram::threeDLineAttributes()`, and `KDChart::AbstractPieDiagram::threeDPieAttributes()`.

```

310 { // FIXME (Mirko): shouldn't this be headerData? instead of the index for the first row?
311     if( model() )
312         return QModelIndex( model()->index( 0, column, rootIndex() ) );
313     return QModelIndex();
314 }

```

#### 9.47.3.14 bool AbstractDiagram::compare (const AbstractDiagram \* other) const [inherited]

Returns true if both diagrams have the same settings.

Definition at line 129 of file KDChartAbstractDiagram.cpp.



References KDChart::AbstractDiagram::allowOverlappingDataValueTexts(), KDChart::AbstractDiagram::antiAliasing(), KDChart::AbstractDiagram::attributesModel(), KDChart::AttributesModel::compare(), KDChart::AbstractDiagram::datasetDimension(), and KDChart::AbstractDiagram::percentMode().

```

130 {
131     if( other == this ) return true;
132     if( ! other ){
133         //qDebug() << "AbstractDiagram::compare() cannot compare to Null pointer";
134         return false;
135     }
136     /*
137     qDebug() << "\n                AbstractDiagram::compare() QAbstractScrollArea:";
138     // compare QAbstractScrollArea properties
139     qDebug() <<
140         ((horizontalScrollBarPolicy() == other->horizontalScrollBarPolicy()) &&
141         (verticalScrollBarPolicy() == other->verticalScrollBarPolicy()));
142     qDebug() << "AbstractDiagram::compare() QFrame:";
143     // compare QFrame properties
144     qDebug() <<
145         ((frameShadow() == other->frameShadow()) &&
146         (frameShape() == other->frameShape()) &&
147         (frameWidth() == other->frameWidth()) &&
148         (lineWidth() == other->lineWidth()) &&
149         (midLineWidth() == other->midLineWidth()));
150     qDebug() << "AbstractDiagram::compare() QAbstractItemView:";
151     // compare QAbstractItemView properties
152     qDebug() <<
153         ((alternatingRowColors() == other->alternatingRowColors()) &&
154         (hasAutoScroll() == other->hasAutoScroll()) &&
155 #if QT_VERSION > 0x040199
156         (dragDropMode() == other->dragDropMode()) &&
157         (dragDropOverwriteMode() == other->dragDropOverwriteMode()) &&
158         (horizontalScrollMode() == other->horizontalScrollMode()) &&
159         (verticalScrollMode() == other->verticalScrollMode()) &&
160 #endif
161         (dragEnabled() == other->dragEnabled()) &&
162         (editTriggers() == other->editTriggers()) &&
163         (iconSize() == other->iconSize()) &&
164         (selectionBehavior() == other->selectionBehavior()) &&
165         (selectionMode() == other->selectionMode()) &&
166         (showDropIndicator() == other->showDropIndicator()) &&
167         (tabKeyNavigation() == other->tabKeyNavigation()) &&
168         (textElideMode() == other->textElideMode()));
169     qDebug() << "AbstractDiagram::compare() AttributesModel: ";
170     // compare all of the properties stored in the attributes model
171     qDebug() << attributesModel()->compare( other->attributesModel() );
172     qDebug() << "AbstractDiagram::compare() own:";
173     // compare own properties
174     qDebug() <<
175         ((rootIndex().column() == other->rootIndex().column()) &&
176         (rootIndex().row() == other->rootIndex().row()) &&
177         (allowOverlappingDataValueTexts() == other->allowOverlappingDataValueTexts()) &&
178         (antiAliasing() == other->antiAliasing()) &&
179         (percentMode() == other->percentMode()) &&
180         (datasetDimension() == other->datasetDimension()));
181     */
182     return // compare QAbstractScrollArea properties
183         (horizontalScrollBarPolicy() == other->horizontalScrollBarPolicy()) &&
184         (verticalScrollBarPolicy() == other->verticalScrollBarPolicy()) &&
185         // compare QFrame properties
186         (frameShadow() == other->frameShadow()) &&
187         (frameShape() == other->frameShape()) &&
188         // frameWidth is a read-only property defined by the style, it should not be in here:
189         // (frameWidth() == other->frameWidth()) &&
190         (lineWidth() == other->lineWidth()) &&
191         (midLineWidth() == other->midLineWidth()) &&

```

```

192         // compare QAbstractItemView properties
193         (alternatingRowColors() == other->alternatingRowColors()) &&
194         (hasAutoScroll() == other->hasAutoScroll()) &&
195 #if QT_VERSION > 0x040199
196         (dragDropMode() == other->dragDropMode()) &&
197         (dragDropOverwriteMode() == other->dragDropOverwriteMode()) &&
198         (horizontalScrollMode() == other->horizontalScrollMode()) &&
199         (verticalScrollMode() == other->verticalScrollMode()) &&
200 #endif
201         (dragEnabled() == other->dragEnabled()) &&
202         (editTriggers() == other->editTriggers()) &&
203         (iconSize() == other->iconSize()) &&
204         (selectionBehavior() == other->selectionBehavior()) &&
205         (selectionMode() == other->selectionMode()) &&
206         (showDropIndicator() == other->showDropIndicator()) &&
207         (tabKeyNavigation() == other->tabKeyNavigation()) &&
208         (textElideMode() == other->textElideMode()) &&
209         // compare all of the properties stored in the attributes model
210         attributesModel()->compare( other->attributesModel() ) &&
211         // compare own properties
212         (rootIndex().column() == other->rootIndex().column()) &&
213         (rootIndex().row() == other->rootIndex().row()) &&
214         (allowOverlappingDataValueTexts() == other->allowOverlappingDataValueTexts()) &&
215         (antiAliasing() == other->antiAliasing()) &&
216         (percentMode() == other->percentMode()) &&
217         (datasetDimension() == other->datasetDimension());
218 }

```

#### 9.47.3.15 **AbstractCoordinatePlane** \* **AbstractDiagram::coordinatePlane () const** [inherited]

The coordinate plane associated with the diagram.

This determines how coordinates in value space are mapped into pixel space. By default this is a [Cartesian-CoordinatePlane](#).

##### Returns:

The coordinate plane associated with the diagram.

Definition at line 220 of file `KDChartAbstractDiagram.cpp`.

References d.

Referenced by `KDChart::AbstractDiagram::checkInvariants()`, `KDChart::AbstractCartesian-Diagram::layoutPlanes()`, `paint()`, `KDChart::AbstractDiagram::paintData ValueTexts()`, `KDChart::Abstract-Diagram::paintMarkers()`, `KDChart::AbstractPolarDiagram::polarCoordinatePlane()`, and `KDChart::AbstractCartesianDiagram::setCoordinatePlane()`.

```

221 {
222     return d->plane;
223 }

```

#### 9.47.3.16 **const QPair< QPointF, QPointF > AbstractDiagram::dataBoundaries () const** [inherited]

Return the bottom left and top right data point, that the diagram will display (unless the grid adjusts these values).

This method returns a cached result of calculations done by `calculateDataBoundaries`. Classes derived from [AbstractDiagram](#) must implement the `calculateDataBoundaries` function, to specify their own way of calculating the data boundaries. If derived classes want to force recalculation of the data boundaries, they can call [setDataBoundariesDirty\(\)](#)

Returned value is in diagram coordinates.

Definition at line 225 of file `KDChartAbstractDiagram.cpp`.

References `KDChart::AbstractDiagram::calculateDataBoundaries()`, and `d`.

Referenced by `KDChart::CartesianCoordinatePlane::getRawDataBoundingRectFromDiagrams()`, `KDChart::PolarCoordinatePlane::layoutDiagrams()`, `KDChart::Plotter::paint()`, `KDChart::LineDiagram::paint()`, and `KDChart::BarDiagram::paint()`.

```

226 {
227     if( d->databoundariesDirty ){
228         d->databoundaries = calculateDataBoundaries ();
229         d->databoundariesDirty = false;
230     }
231     return d->databoundaries;
232 }
```

#### 9.47.3.17 void AbstractDiagram::dataChanged (const QModelIndex & *topLeft*, const QModelIndex & *bottomRight*) [virtual, inherited]

[reimplemented]

Definition at line 330 of file `KDChartAbstractDiagram.cpp`.

References `KDChart::AbstractDiagram::setDataBoundariesDirty()`.

```

332 {
333     Q_UNUSED( topLeft );
334     Q_UNUSED( bottomRight );
335     // We are still too dumb to do intelligent updates...
336     setDataBoundariesDirty();
337     scheduleDelayedItemsLayout();
338 }
```

#### 9.47.3.18 void KDChart::AbstractDiagram::dataHidden () [signal, inherited]

This signal is emitted, when the hidden status of at least one data cell was (un)set.

Referenced by `KDChart::AbstractDiagram::setHidden()`.

#### 9.47.3.19 QList< QBrush > AbstractDiagram::datasetBrushes () const [inherited]

The set of dataset brushes currently used, for use in legends, etc.

##### Note:

Cell-level override brushes, if set, take precedence over the dataset values, so you might need to check these too, in order to find the brush, that is used for a single cell.

##### Returns:

The current set of dataset brushes.

Definition at line 1018 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::attributesModelRootIndex(), KDChart::AttributesModel::columnCount(), KDChart::DatasetBrushRole, and KDChart::AbstractDiagram::datasetDimension().

Referenced by KDChart::Legend::setBrushesFromDiagram().

```

1019 {
1020     QList<QBrush> ret;
1021     if( model() == 0 )
1022         return ret;
1023
1024     const int columnCount = attributesModel()->columnCount(attributesModelRootIndex());
1025     for( int i = 0; i < columnCount / datasetDimension(); ++i )
1026         ret << QVariantValue<QBrush>( attributesModel()->headerData( i, Qt::Vertical, DatasetBrushRole ));
1027
1028     return ret;
1029 }
```

#### 9.47.3.20 int AbstractDiagram::datasetDimension () const [inherited]

The dataset dimension of a diagram determines, how many value dimensions it expects each datapoint to have.

For each dimension it will expect one column of values in the model. If the dimensionality is 1, automatic values will be used for the abscissa.

For example a diagram with the default dimension of 1, will have one column per datapoint (the y values) and will use automatic values for the x axis (1, 2, 3, ... n). If the dimension is 2, the diagram will use the first, (and the third, fifth, etc) columns as X values, and the second, (and the fourth, sixth, etc) column as Y values.

##### Returns:

The dataset dimension of the diagram.

Definition at line 1072 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::brush(), KDChart::AbstractDiagram::compare(), KDChart::AbstractDiagram::datasetBrushes(), KDChart::AbstractDiagram::datasetLabels(), KDChart::AbstractDiagram::datasetMarkers(), KDChart::AbstractDiagram::datasetPens(), KDChart::LineDiagram::getCellValues(), KDChart::CartesianCoordinatePlane::getDataDimensionsList(), KDChart::TernaryPointDiagram::paint(), KDChart::TernaryLineDiagram::paint(), KDChart::AbstractDiagram::paintDataValueTexts(), KDChart::AbstractDiagram::paintMarkers(), KDChart::AbstractDiagram::pen(), KDChart::AbstractDiagram::setPen(), KDChart::Plotter::setType(), and KDChart::LineDiagram::setType().

```

1073 {
1074     return d->datasetDimension;
1075 }
```

#### 9.47.3.21 QStringList AbstractDiagram::datasetLabels () const [inherited]

The set of dataset labels currently displayed, for use in legends, etc.

**Returns:**

The set of dataset labels currently displayed.

Definition at line 1005 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::attributesModelRootIndex(), KDChart::AttributesModel::columnCount(), KDChart::AbstractDiagram::datasetDimension(), and KDChart::AttributesModel::headerData().

```

1006 {
1007     QStringList ret;
1008     if( model() == 0 )
1009         return ret;
1010
1011     const int columnCount = attributesModel()->columnCount(attributesModelRootIndex());
1012     for( int i = 0; i < columnCount / datasetDimension(); ++i )
1013         ret << attributesModel()->headerData( i, Qt::Horizontal, Qt::DisplayRole ).toString();
1014
1015     return ret;
1016 }
```

### 9.47.3.22 QList< [MarkerAttributes](#) > AbstractDiagram::datasetMarkers () const

[inherited]

The set of dataset markers currently used, for use in legends, etc.

**Note:**

Cell-level override markers, if set, take precedence over the dataset values, so you might need to check these too, in order to find the marker, that is shown for a single cell.

**Returns:**

The current set of dataset brushes.

Definition at line 1044 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::attributesModelRootIndex(), KDChart::AttributesModel::columnCount(), KDChart::AbstractDiagram::datasetDimension(), and KDChart::DataValueLabelAttributesRole.

```

1045 {
1046     QList<MarkerAttributes> ret;
1047     if( model() == 0 )
1048         return ret;
1049
1050     const int columnCount = attributesModel()->columnCount(attributesModelRootIndex());
1051     for( int i = 0; i < columnCount / datasetDimension(); ++i )
1052     {
1053         const DataValueAttributes a =
1054             qVariantValue<DataValueAttributes>( attributesModel()->headerData( i, Qt::Vertical, DataValueLabelAttributesRole ) );
1055         ret << a.markerAttributes();
1056     }
1057     return ret;
1058 }
```

**9.47.3.23** `QList< QPen > AbstractDiagram::datasetPens () const` [inherited]

The set of dataset pens currently used, for use in legends, etc.

**Note:**

Cell-level override pens, if set, take precedence over the dataset values, so you might need to check these too, in order to find the pens, that is used for a single cell.

**Returns:**

The current set of dataset pens.

Definition at line 1031 of file `KDChartAbstractDiagram.cpp`.

References `KDChart::AbstractDiagram::attributesModel()`, `KDChart::AbstractDiagram::attributesModelRootIndex()`, `KDChart::AttributesModel::columnCount()`, `KDChart::AbstractDiagram::datasetDimension()`, and `KDChart::DatasetPenRole`.

```

1032 {
1033     QList<QPen> ret;
1034     if( model() == 0 )
1035         return ret;
1036
1037     const int columnCount = attributesModel()->columnCount(attributesModelRootIndex());
1038     for( int i = 0; i < columnCount / datasetDimension(); ++i )
1039         ret << qVariantValue<QPen>( attributesModel()->headerData( i, Qt::Vertical, DatasetPenRole ) );
1040
1041     return ret;
1042 }
```

**9.47.3.24** `DataValueAttributes AbstractDiagram::dataValueAttributes (const QModelIndex & index) const` [inherited]

Retrieve the [DataValueAttributes](#) for the given index.

This will fall back automatically to what was set at dataset or model level, if there are no datapoint specific settings.

**Parameters:**

*index* The datapoint to retrieve the attributes for.

**Returns:**

The [DataValueAttributes](#) for the given index.

Definition at line 421 of file `KDChartAbstractDiagram.cpp`.

References `KDChart::AbstractDiagram::attributesModel()`, `KDChart::AttributesModel::data()`, and `KDChart::DataValueLabelAttributesRole`.

```

422 {
423     return qVariantValue<DataValueAttributes>(
424         attributesModel()->data( attributesModel()->mapFromSource(index),
425                                 KDChart::DataValueLabelAttributesRole ) );
426 }
```

### 9.47.3.25 [DataValueAttributes](#) AbstractDiagram::dataValueAttributes (int *column*) const [inherited]

Retrieve the [DataValueAttributes](#) for the given dataset.

This will fall back automatically to what was set at model level, if there are no dataset specific settings.

#### Parameters:

*column* The dataset to retrieve the attributes for.

#### Returns:

The [DataValueAttributes](#) for the given dataset.

Definition at line 414 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::columnToIndex(), KDChart::AttributesModel::data(), and KDChart::DataValueLabelAttributesRole.

```
415 {
416     return qVariantValue<DataValueAttributes>(
417         attributesModel()->data( attributesModel()->mapFromSource(columnToIndex( column )),
418         KDChart::DataValueLabelAttributesRole ) );
419 }
```

### 9.47.3.26 [DataValueAttributes](#) AbstractDiagram::dataValueAttributes () const [inherited]

Retrieve the [DataValueAttributes](#) specified globally.

This will fall back automatically to the default settings, if there are no specific settings.

#### Returns:

The global [DataValueAttributes](#).

Definition at line 408 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DataValueLabelAttributesRole, and KDChart::AttributesModel::modelData().

Referenced by KDChart::AbstractDiagram::paintDataValueText(), KDChart::AbstractDiagram::paintMarker(), and KDChart::TernaryLineDiagram::TernaryLineDiagram().

```
409 {
410     return qVariantValue<DataValueAttributes>(
411         attributesModel()->modelData( KDChart::DataValueLabelAttributesRole ) );
412 }
```

### 9.47.3.27 void AbstractDiagram::doItemsLayout () [virtual, inherited]

[reimplemented]

Definition at line 321 of file KDChartAbstractDiagram.cpp.

References d, and KDChart::AbstractDiagram::update().

```

322 {
323     if ( d->plane ) {
324         d->plane->layoutDiagrams();
325         update();
326     }
327     QAbstractItemView::doItemsLayout();
328 }

```

#### 9.47.3.28 `int AbstractDiagram::horizontalOffset () const` [virtual, inherited]

[reimplemented]

Definition at line 950 of file KDChartAbstractDiagram.cpp.

```

951 { return 0; }

```

#### 9.47.3.29 `QModelIndex AbstractDiagram::indexAt (const QPoint & point) const` [virtual, inherited]

[reimplemented]

Definition at line 1098 of file KDChartAbstractDiagram.cpp.

References d.

```

1099 {
1100     return d->indexAt( point );
1101 }

```

#### 9.47.3.30 `QModelIndexList AbstractDiagram::indexesAt (const QPoint & point) const` [inherited]

This method is added alongside with `indexAt` from QAIM, since in kdchart multiple indexes can be displayed at the same spot.

Definition at line 1103 of file KDChartAbstractDiagram.cpp.

References d.

```

1104 {
1105     return d->indexesAt( point );
1106 }

```

#### 9.47.3.31 `bool AbstractDiagram::isHidden (const QModelIndex & index) const` [inherited]

Retrieve the hidden status for the given index.

This will fall back automatically to what was set at dataset or diagram level, if there are no datapoint specific settings.

##### Parameters:

*index* The datapoint to retrieve the hidden status for.



**Returns:**

The hidden status for the given index.

Definition at line 380 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AttributesModel::data(), and KDChart::DataHiddenRole.

```

381 {
382     return qVariantValue<bool>(
383         attributesModel()->data(
384             attributesModel()->mapFromSource(index),
385             DataHiddenRole ) );
386 }
```

**9.47.3.32 bool AbstractDiagram::isHidden (int *column*) const** [inherited]

Retrieve the hidden status for the given dataset.

This will fall back automatically to what was set at diagram level, if there are no dataset specific settings.

**Parameters:**

***column*** The dataset to retrieve the hidden status for.

**Returns:**

The hidden status for the given dataset.

Definition at line 373 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::columnToIndex(), KDChart::AttributesModel::data(), and KDChart::DataHiddenRole.

```

374 {
375     return qVariantValue<bool>(
376         attributesModel()->data(
377             attributesModel()->mapFromSource(columnToIndex( column )),
378             DataHiddenRole ) );
379 }
```

**9.47.3.33 bool AbstractDiagram::isHidden () const** [inherited]

Retrieve the hidden status specified globally.

This will fall back automatically to the default settings (= not hidden), if there are no specific settings.

**Returns:**

The global hidden status.

Definition at line 367 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DataHiddenRole, and KDChart::AttributesModel::modelData().

Referenced by KDChart::LineDiagram::valueForCellTesting().

```

368 {
369     return qVariantValue<bool>(
370         attributesModel()->modelData( DataHiddenRole ) );
371 }

```

#### 9.47.3.34 bool AbstractDiagram::isIndexHidden (const QModelIndex & *index*) const

[virtual, inherited]

[reimplemented]

Definition at line 956 of file KDChartAbstractDiagram.cpp.

```

957 { return true; }

```

#### 9.47.3.35 QStringList AbstractDiagram::itemRowLabels () const

[inherited]

The set of item row labels currently displayed, for use in Abscissa axes, etc.

##### Returns:

The set of item row labels currently displayed.

Definition at line 989 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::attributesModelRootIndex(), KDChart::AttributesModel::rowCount(), KDChart::AbstractDiagram::unitPrefix(), and KDChart::AbstractDiagram::unitSuffix().

```

990 {
991     QStringList ret;
992     if( model() ){
993         //qDebug() << "AbstractDiagram::itemRowLabels(): " << attributesModel()->rowCount(attributesModelRootIndex());
994         const int rowCount = attributesModel()->rowCount(attributesModelRootIndex());
995         for( int i = 0; i < rowCount; ++i ){
996             //qDebug() << "item row label: " << attributesModel()->headerData( i, Qt::Vertical, Qt::DisplayRole );
997             ret << unitPrefix( i, Qt::Horizontal, true ) +
998                 attributesModel()->headerData( i, Qt::Vertical, Qt::DisplayRole ).toString() +
999                 unitSuffix( i, Qt::Horizontal, true );
1000         }
1001     }
1002     return ret;
1003 }

```

#### 9.47.3.36 void KDChart::AbstractDiagram::layoutChanged ([AbstractDiagram \\*](#))

[signal, inherited]

Diagrams are supposed to emit this signal, when the layout of one of their element changes.

Layouts can change, for example, when axes are added or removed, or when the configuration was changed in a way that the axes or the diagram itself are displayed in a different geometry. Changes in the diagrams coordinate system also result in the [layoutChanged\(\)](#) signal being emitted.

Referenced by KDChart::AbstractDiagram::setDatasetDimension(), KDChart::AbstractPieDiagram::setPieAttributes(), KDChart::AbstractPieDiagram::setThreeDPieAttributes(), KDChart::Plotter::setType(), KDChart::LineDiagram::setType(), and KDChart::BarDiagram::setType().

**9.47.3.37 void KDChart::AbstractDiagram::modelsChanged ()** [signal, inherited]

This signal is emitted, when either the model or the [AttributesModel](#) is replaced.

Referenced by `KDChart::AbstractDiagram::setAttributesModel()`, and `KDChart::AbstractDiagram::setModel()`.

**9.47.3.38 QModelIndex AbstractDiagram::moveCursor (CursorAction *cursorAction*, Qt::KeyboardModifiers *modifiers*)** [virtual, inherited]

[reimplemented]

Definition at line 947 of file `KDChartAbstractDiagram.cpp`.

```
948 { return QModelIndex(); }
```

**9.47.3.39 double PolarDiagram::numberOfGridRings () const** [virtual]

[reimplemented]

Implements [KDChart::AbstractPolarDiagram](#).

Definition at line 208 of file `KDChartPolarDiagram.cpp`.

```
209 {
210     return 5; // FIXME
211 }
```

**9.47.3.40 double PolarDiagram::numberOfValuesPerDataset () const** [virtual]

[reimplemented]

Implements [KDChart::AbstractPolarDiagram](#).

Definition at line 202 of file `KDChartPolarDiagram.cpp`.

```
203 {
204     return model() ? model()->rowCount(rootIndex()) : 0.0;
205 }
```

**9.47.3.41 void PolarDiagram::paint (PaintContext \* *paintContext*)** [protected, virtual]

[reimplemented]

Implements [KDChart::AbstractDiagram](#).

Definition at line 145 of file `KDChartPolarDiagram.cpp`.

References `KDChart::AbstractDiagram::attributesModel()`, `KDChart::AbstractDiagram::brush()`, `KDChart::AbstractDiagram::checkInvariants()`, `closeDatasets()`, `KDChart::AbstractDiagram::coordinatePlane()`, `KDChart::DatasetBrushRole`, `KDChart::AbstractDiagram::paintDataValueText()`, `KDChart::PaintContext::painter()`, `paintPolarMarkers()`, `KDChart::PaintContext::rectangle()`, and `KDChart::AbstractCoordinatePlane::translate()`.

Referenced by `paintEvent()`.

```

146 {
147     // note: Not having any data model assigned is no bug
148     //         but we can not draw a diagram then either.
149     if ( !checkInvariants(true) )
150         return;
151
152     const int rowCount = model()->rowCount( rootIndex() );
153     const int colCount = model()->columnCount( rootIndex() );
154     DataValueTextInfoList list;
155
156     for ( int j=0; j < colCount; ++j ) {
157         QBrush brush = qVariantValue<QBrush>( attributesModel()->headerData( j, Qt::Vertical, KDChart:
158         QPolygonF polygon;
159         QPointF point0;
160         for ( int i=0; i < rowCount; ++i ) {
161             QModelIndex index = model()->index( i, j, rootIndex() );
162             const double value = model()->data( index ).toDouble();
163             QPointF point = coordinatePlane()->translate( QPointF( value, i ) );
164             polygon.append( point );
165             const DataValueTextInfo info( index, point, point, value );
166             list.append( info );
167             if( ! i )
168                 point0= point;
169         }
170         if( closeDatasets() && rowCount )
171             polygon.append( point0 );
172
173         PainterSaver painterSaver( ctx->painter() );
174         ctx->painter()->setRenderHint ( QPainter::Antialiasing );
175         ctx->painter()->setBrush( brush );
176         QPen p( ctx->painter()->pen() );
177         p.setColor( brush.color() ); // FIXME use DatasetPenRole
178         p.setWidth( 2 ); // FIXME properties
179         ctx->painter()->setPen( p );
180         polygon.translate( ctx->rectangle().topLeft() );
181         ctx->painter()->drawPolyline( polygon );
182         paintPolarMarkers( ctx, polygon );
183     }
184     DataValueTextInfoListIterator it( list );
185     while ( it.hasNext() ) {
186         const DataValueTextInfo& info = it.next();
187         paintDataValueText( ctx->painter(), info.index, info.pos, info.value );
188     }
189 }

```

#### 9.47.3.42 void AbstractDiagram::paintDataValueText (QPainter \*painter, const QModelIndex & index, const QPointF & pos, double value) [inherited]

Definition at line 468 of file KDChartAbstractDiagram.cpp.

References `d`, `KDChart::DataValueAttributes::dataLabel()`, `KDChart::AbstractDiagram::dataValueAttributes()`, `KDChart::DataValueAttributes::decimalDigits()`, `KDChart::DataValueAttributes::isVisible()`, `KDChartEnums::MeasureOrientationMinimum`, `KDChart::DataValueAttributes::position()`, `KDChart::DataValueAttributes::prefix()`, `KDChart::DataValueAttributes::suffix()`, and `KDChart::DataValueAttributes::textAttributes()`.

Referenced by `KDChart::RingDiagram::paint()`, `paint()`, and `KDChart::AbstractDiagram::paintDataValueTexts()`.

```

472 {
473     // paint one data series
474     const DataValueAttributes a( dataValueAttributes(index) );
475     if ( !a.isVisible() ) return;

```

```

476
477 // handle decimal digits
478 int decimalDigits = a.decimalDigits();
479 int decimalPos = QString::number( value ).indexOf( QLatin1Char( '.' ) );
480 QString roundedValue;
481 if ( a.dataLabel().isNull() ) {
482     if ( decimalPos > 0 && value != 0 )
483         roundedValue = roundValues ( value, decimalPos, decimalDigits );
484     else
485         roundedValue = QString::number( value );
486 } else
487     roundedValue = a.dataLabel();
488 // handle prefix and suffix
489 if ( !a.prefix().isNull() )
490     roundedValue.prepend( a.prefix() );
491
492 if ( !a.suffix().isNull() )
493     roundedValue.append( a.suffix() );
494
495 const TextAttributes ta( a.textAttributes() );
496 // FIXME draw the non-text bits, background, etc
497 if ( ta.isVisible() ) {
498
499     QPointF pt( pos );
500     /* for debugging:
501     PainterSaver painterSaver( painter );
502     painter->setPen( Qt::black );
503     painter->drawLine( pos - QPointF( 1,1), pos + QPointF( 1,1) );
504     painter->drawLine( pos - QPointF(-1,1), pos + QPointF(-1,1) );
505     */
506
507     QTextDocument doc;
508     if( Qt::mightBeRichText( roundedValue ) )
509         doc.setHtml( roundedValue );
510     else
511         doc.setPlainText( roundedValue );
512
513     const RelativePosition relPos( a.position( value >= 0.0 ) );
514     const Qt::Alignment alignBottomLeft = Qt::AlignBottom | Qt::AlignLeft;
515     const QFont calculatedFont( ta.calculatedFont( d->plane, KDChartEnums::MeasureOrientationMinimum ) );
516     const QRectF boundRect( d->cachedFontMetrics( calculatedFont, painter->device() )->boundingRect( roundedValue, calculatedFont ) );
517
518     // To place correctly
519     pt.ry() -= boundRect.height();
520
521     // qDebug() << "calculatedFont's point size:" << calculatedFont.pointSizeF();
522     // adjust the text start point position, if alignment is not Bottom/Left
523     if( (relPos.alignment() & alignBottomLeft) != alignBottomLeft ) {
524         if( relPos.alignment() & Qt::AlignRight )
525             pt.rx() -= boundRect.width();
526         else if( relPos.alignment() & Qt::AlignHCenter )
527             pt.rx() -= 0.5 * boundRect.width();
528
529         if( relPos.alignment() & Qt::AlignTop )
530             pt.ry() += boundRect.height();
531         else if( relPos.alignment() & Qt::AlignVCenter )
532             pt.ry() += 0.5 * boundRect.height();
533     }
534
535     // FIXME draw the non-text bits, background, etc
536
537     if ( a.showRepetitiveDataLabels() ||
538         pos.x() <= d->lastX ||
539         d->lastRoundedValue != roundedValue ) {
540         d->lastRoundedValue = roundedValue;
541         d->lastX = pos.x();
542         PainterSaver painterSaver( painter );

```

```

543
544         doc.setDefaultFont( calculatedFont );
545         QAbstractTextDocumentLayout::PaintContext context;
546         context.palette = palette();
547         context.palette.setColor(QPalette::Text, ta.pen().color() );
548
549         painter->translate( pt );
550         painter->rotate( ta.rotation() );
551
552         QAbstractTextDocumentLayout* layout = doc.documentLayout();
553         layout->draw( painter, context );
554     }
555 }
556 }

```

**9.47.3.43 void AbstractDiagram::paintDataValueTexts (QPainter \* *painter*)** [protected, virtual, inherited]

Definition at line 584 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::checkInvariants(), KDChart::AbstractDiagram::coordinatePlane(), KDChart::AbstractDiagram::datasetDimension(), KDChart::AbstractDiagram::paintDataValueText(), and KDChart::AbstractCoordinatePlane::translate().

```

585 {
586     if ( !checkInvariants() ) return;
587     const int rowCount = model()->rowCount(rootIndex());
588     const int columnCount = model()->columnCount(rootIndex());
589     for ( int i=datasetDimension()-1; i<columnCount; i += datasetDimension() ) {
590         for ( int j=0; j< rowCount; ++j ) {
591             const QModelIndex index = model()->index( j, i, rootIndex() );
592             double value = model()->data( index ).toDouble();
593             const QPointF pos = coordinatePlane()->translate( QPointF( j, value ) );
594             paintDataValueText( painter, index, pos, value );
595         }
596     }
597 }

```

**9.47.3.44 void PolarDiagram::paintEvent (QPaintEvent \*)** [protected]

Definition at line 121 of file KDChartPolarDiagram.cpp.

References paint(), KDChart::PaintContext::setPainter(), and KDChart::PaintContext::setRectangle().

```

122 {
123     QPainter painter ( viewport() );
124     PaintContext ctx;
125     ctx.setPainter ( &painter );
126     ctx.setRectangle( QRectF ( 0, 0, width(), height() ) );
127     paint ( &ctx );
128 }

```

**9.47.3.45 void AbstractDiagram::paintMarker (QPainter \* *painter*, const QModelIndex & *index*, const QPointF & *pos*)** [inherited]

Definition at line 600 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::brush(), KDChart::AbstractDiagram::checkInvariants(), d, KDChart::AbstractDiagram::dataValueAttributes(), KDChart::MarkerAttributes::isVisible(), KDChart::DataValueAttributes::isVisible(), KDChart::DataValueAttributes::markerAttributes(), KDChart::MarkerAttributes::markerColor(), KDChart::MarkerAttributes::markerSize(), KDChart::AbstractDiagram::paintMarker(), and KDChart::MarkerAttributes::pen().

```

603 {
604     if ( !checkInvariants() ) return;
605     DataValueAttributes a = dataValueAttributes(index);
606     if ( !a.isVisible() ) return;
607     const MarkerAttributes &ma = a.markerAttributes();
608     if ( !ma.isVisible() ) return;
609
610     PainterSaver painterSaver( painter );
611     QSizeF maSize( ma.markerSize() );
612     QBrush indexBrush( brush( index ) );
613     QPen indexPen( ma.pen() );
614     if ( ma.markerColor().isValid() )
615         indexBrush.setColor( ma.markerColor() );
616
617     paintMarker( painter, ma, indexBrush, indexPen, pos, maSize );
618
619     // workaround: BC cannot be changed, otherwise we would pass the
620     // index down to next-lower paintMarker function. So far, we
621     // basically save a circle of radius maSize at pos in the
622     // reverseMapper. This means that ^^ this version of paintMarker
623     // needs to be called to reverse-map the marker.
624     d->reverseMapper.addCircle( index.row(), index.column(), pos, 2 * maSize );
625 }
```

**9.47.3.46 void AbstractDiagram::paintMarker** (QPainter \**painter*, const [MarkerAttributes](#) &*markerAttributes*, const QBrush &*brush*, const QPen &, const QPointF &*point*, const QSizeF &*size*) [virtual, inherited]

Definition at line 627 of file KDChartAbstractDiagram.cpp.

References KDChart::MarkerAttributes::Marker1Pixel, KDChart::MarkerAttributes::Marker4Pixels, KDChart::MarkerAttributes::MarkerCircle, KDChart::MarkerAttributes::MarkerCross, KDChart::MarkerAttributes::MarkerDiamond, KDChart::MarkerAttributes::MarkerFastCross, KDChart::MarkerAttributes::MarkerRing, KDChart::MarkerAttributes::MarkerSquare, and KDChart::MarkerAttributes::markerStyle().

Referenced by KDChart::TernaryPointDiagram::paint(), KDChart::TernaryLineDiagram::paint(), KDChart::MarkerLayoutItem::paintIntoRect(), KDChart::AbstractDiagram::paintMarker(), and KDChart::AbstractDiagram::paintMarkers().

```

633 {
634     const QPen oldPen( painter->pen() );
635     // Pen is used to paint 4Pixels - 1 Pixel - Ring and FastCross types.
636     // make sure to use the brush color - see above in those cases.
637     const bool isFourPixels = (markerAttributes.markerStyle() == MarkerAttributes::Marker4Pixels);
638     if( isFourPixels || (markerAttributes.markerStyle() == MarkerAttributes::Marker1Pixel) ){
639         // for high-performance point charts with tiny point markers:
640         painter->setPen( QPen( brush.color().light() ) );
641         if( isFourPixels ){
642             const qreal x = pos.x();
643             const qreal y = pos.y();
644             painter->drawLine( QPointF(x-1.0,y-1.0),
645                             QPointF(x+1.0,y-1.0) );
646             painter->drawLine( QPointF(x-1.0,y),
647                             QPointF(x+1.0,y) );

```

```

648         painter->drawLine( QPointF(x-1.0,y+1.0),
649                             QPointF(x+1.0,y+1.0) );
650     }
651     painter->drawPoint( pos );
652 }else{
653     PainterSaver painterSaver( painter );
654     // we only a solid line surrounding the markers
655     QPen painterPen( pen );
656     painterPen.setStyle( Qt::SolidLine );
657     painter->setPen( painterPen );
658     painter->setBrush( brush );
659     painter->setRenderHint ( QPainter::Antialiasing );
660     painter->translate( pos );
661     switch ( markerAttributes.markerStyle() ) {
662     case MarkerAttributes::MarkerCircle:
663         painter->drawEllipse( QRectF( 0 - maSize.height()/2, 0 - maSize.width()/2,
664                                     maSize.height(), maSize.width() ) );
665         break;
666     case MarkerAttributes::MarkerSquare:
667     {
668         QRectF rect( 0 - maSize.width()/2, 0 - maSize.height()/2,
669                     maSize.width(), maSize.height() );
670         painter->drawRect( rect );
671         break;
672     }
673     case MarkerAttributes::MarkerDiamond:
674     {
675         QVector <QPointF > diamondPoints;
676         QPointF top, left, bottom, right;
677         top = QPointF( 0, 0 - maSize.height()/2 );
678         left = QPointF( 0 - maSize.width()/2, 0 );
679         bottom = QPointF( 0, maSize.height()/2 );
680         right = QPointF( maSize.width()/2, 0 );
681         diamondPoints << top << left << bottom << right;
682         painter->drawPolygon( diamondPoints );
683         break;
684     }
685     // both handled on top of the method:
686     case MarkerAttributes::Marker1Pixel:
687     case MarkerAttributes::Marker4Pixels:
688         break;
689     case MarkerAttributes::MarkerRing:
690     {
691         painter->setPen( QPen( brush.color() ) );
692         painter->setBrush( Qt::NoBrush );
693         painter->drawEllipse( QRectF( 0 - maSize.height()/2, 0 - maSize.width()/2,
694                                     maSize.height(), maSize.width() ) );
695         break;
696     }
697     case MarkerAttributes::MarkerCross:
698     {
699         QRectF rect( maSize.width()*-0.5, maSize.height()*-0.2,
700                     maSize.width(), maSize.height()*0.4 );
701         painter->drawRect( rect );
702         rect.setTopLeft( QPointF( maSize.width()*-0.2, maSize.height()*-0.5 ) );
703         rect.setSize( QSizeF( maSize.width()*0.4, maSize.height() ) );
704         painter->drawRect( rect );
705         break;
706     }
707     case MarkerAttributes::MarkerFastCross:
708     {
709         QPointF left, right, top, bottom;
710         left = QPointF( -maSize.width()/2, 0 );
711         right = QPointF( maSize.width()/2, 0 );
712         top = QPointF( 0, -maSize.height()/2 );
713         bottom= QPointF( 0, maSize.height()/2 );
714         painter->setPen( QPen( brush.color() ) );

```



```

715             painter->drawLine( left, right );
716             painter->drawLine( top, bottom );
717             break;
718         }
719         default:
720             Q_ASSERT_X ( false, "paintMarkers()",
721                         "Type item does not match a defined Marker Type." );
722     }
723 }
724 painter->setPen( oldPen );
725 }

```

#### 9.47.3.47 void AbstractDiagram::paintMarkers (QPainter \* *painter*) [protected, virtual, inherited]

Definition at line 727 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::checkInvariants(), KDChart::AbstractDiagram::coordinatePlane(), KDChart::AbstractDiagram::datasetDimension(), KDChart::AbstractDiagram::paintMarker(), and KDChart::AbstractCoordinatePlane::translate().

```

728 {
729     if ( !checkInvariants() ) return;
730     const int rowCount = model()->rowCount( rootIndex() );
731     const int columnCount = model()->columnCount( rootIndex() );
732     for ( int i=datasetDimension()-1; i<columnCount; i += datasetDimension() ) {
733         for ( int j=0; j<rowCount; ++j ) {
734             const QModelIndex index = model()->index( j, i, rootIndex() );
735             double value = model()->data( index ).toDouble();
736             const QPointF pos = coordinatePlane()->translate( QPointF( j, value ) );
737             paintMarker( painter, index, pos );
738         }
739     }
740 }

```

#### 9.47.3.48 void PolarDiagram::paintPolarMarkers (PaintContext \* *ctx*, const QPolygonF & *polygon*) [protected, virtual]

Definition at line 134 of file KDChartPolarDiagram.cpp.

References KDChart::PaintContext::painter().

Referenced by paint().

```

135 {
136     const double markerSize = 4; // FIXME use real markers
137     for ( int i=0; i<polygon.size(); ++i ) {
138         QPointF p = polygon.at( i );
139         p.setX( p.x() - markerSize/2 );
140         p.setY( p.y() - markerSize/2 );
141         ctx->painter()->drawRect( QRectF( p, QSizeF( markerSize, markerSize ) ) );
142     }
143 }

```

#### 9.47.3.49 QPen AbstractDiagram::pen (const QModelIndex & *index*) const [inherited]

Retrieve the pen to be used, for painting the datapoint at the given index in the model.

**Parameters:**

*index* The index of the datapoint in the model.

**Returns:**

The pen to use for painting.

Definition at line 788 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AttributesModel::data(), KDChart::AbstractDiagram::datasetDimension(), KDChart::DatasetPenRole, and KDChart::AbstractDiagram::pen().

```

789 {
790     if( datasetDimension() > 1 )
791         return pen( index.column() );
792     return qVariantValue<QPen>(
793         attributesModel()->data(
794             attributesModel()->mapFromSource( index ),
795             DatasetPenRole ) );
796 }
```

**9.47.3.50 QPen AbstractDiagram::pen (int *dataset*) const** [inherited]

Retrieve the pen to be used for the given dataset.

This will fall back automatically to what was set at model level, if there are no dataset specific settings.

**Parameters:**

*dataset* The dataset to retrieve the pen for.

**Returns:**

The pen to use for painting.

Definition at line 777 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::columnToIndex(), KDChart::AttributesModel::data(), KDChart::AbstractDiagram::datasetDimension(), and KDChart::DatasetPenRole.

```

778 {
779     if( datasetDimension() > 1 )
780         dataset /= datasetDimension();
781
782     return qVariantValue<QPen>(
783         attributesModel()->data(
784             attributesModel()->mapFromSource( columnToIndex( dataset ) ),
785             DatasetPenRole ) );
786 }
```

**9.47.3.51 QPen AbstractDiagram::pen () const** [inherited]

Retrieve the pen to be used for painting datapoints globally.

This will fall back automatically to the default settings, if there are no specific settings.

**Returns:**

The pen to use for painting.

Definition at line 771 of file KDCartAbstractDiagram.cpp.

References KDCart::AbstractDiagram::attributesModel(), KDCart::AttributesModel::data(), and KDCart::DatasetPenRole.

Referenced by KDCart::TernaryPointDiagram::paint(), KDCart::TernaryLineDiagram::paint(), and KDCart::AbstractDiagram::pen().

```
772 {
773     return qVariantValue<QPen>(
774         attributesModel()->data( DatasetPenRole ) );
775 }
```

**9.47.3.52 bool AbstractDiagram::percentMode () const** [inherited]

Definition at line 462 of file KDCartAbstractDiagram.cpp.

References d.

Referenced by KDCart::AbstractDiagram::compare(), and KDCart::CartesianCoordinatePlane::getDataDimensionsList().

```
463 {
464     return d->percent;
465 }
```

**9.47.3.53 const PolarCoordinatePlane \* AbstractPolarDiagram::polarCoordinatePlane () const** [inherited]

Definition at line 55 of file KDCartAbstractPolarDiagram.cpp.

References KDCart::AbstractDiagram::coordinatePlane().

Referenced by KDCart::PieDiagram::paint().

```
56 {
57     return dynamic_cast<const PolarCoordinatePlane*>( coordinatePlane() );
58 }
```

**9.47.3.54 void KDCart::AbstractDiagram::propertiesChanged ()** [signal, inherited]

Emitted upon change of a property of the Diagram.

Referenced by KDCart::Plotter::resetLineAttributes(), KDCart::LineDiagram::resetLineAttributes(), KDCart::AbstractDiagram::setAllowOverlappingDataValueTexts(), KDCart::AbstractDiagram::setAntiAliasing(), KDCart::BarDiagram::setBarAttributes(), KDCart::AbstractDiagram::setBrush(), KDCart::AbstractDiagram::setDataValueAttributes(), KDCart::Plotter::setLineAttributes(), KDCart::LineDiagram::setLineAttributes(), KDCart::AbstractDiagram::setPen(), KDCart::AbstractDiagram::setPercentMode(), KDCart::BarDiagram::setThreeDBarAttributes(), KDCart::Plotter::setThreeDLineAttributes(), KDCart::LineDiagram::setThreeDLineAttributes(), KDCart::Plotter::setType(), KDCart::LineDiagram::setType(), KDCart::BarDiagram::setType(), KDCart::Plotter::setValueTrackerAttributes(), and KDCart::LineDiagram::setValueTrackerAttributes().

**9.47.3.55 void PolarDiagram::resize (const QSizeF & *area*)** [virtual]

[reimplemented]

Implements [KDChart::AbstractDiagram](#).

Definition at line 191 of file KDChartPolarDiagram.cpp.

```
192 {  
193 }
```

**9.47.3.56 void PolarDiagram::resizeEvent (QResizeEvent \*)** [protected]

Definition at line 130 of file KDChartPolarDiagram.cpp.

```
131 {  
132 }
```

**9.47.3.57 bool PolarDiagram::rotateCircularLabels ()** const

Definition at line 230 of file KDChartPolarDiagram.cpp.

References [d](#).

Referenced by [clone\(\)](#).

```
231 {  
232     return d->rotateCircularLabels;  
233 }
```

**9.47.3.58 void AbstractDiagram::scrollTo (const QModelIndex & *index*, ScrollHint *hint* = EnsureVisible)** [virtual, inherited]

[reimplemented]

Definition at line 942 of file KDChartAbstractDiagram.cpp.

```
943 {}
```

**9.47.3.59 void AbstractDiagram::setAllowOverlappingDataValueTexts (bool *allow*)** [inherited]

Set whether data value labels are allowed to overlap.

**Parameters:**

*allow* True means that overlapping labels are allowed.

Definition at line 434 of file KDChartAbstractDiagram.cpp.

References [d](#), and [KDChart::AbstractDiagram::propertiesChanged\(\)](#).

```
435 {  
436     d->allowOverlappingDataValueTexts = allow;  
437     emit propertiesChanged();  
438 }
```

**9.47.3.60 void AbstractDiagram::setAntiAliasing (bool *enabled*)** [inherited]

Set whether anti-aliasing is to be used while rendering this diagram.

**Parameters:**

*enabled* True means that AA is enabled.

Definition at line 445 of file KDChartAbstractDiagram.cpp.

References `d`, and `KDChart::AbstractDiagram::propertiesChanged()`.

```
446 {  
447     d->antiAliasing = enabled;  
448     emit propertiesChanged();  
449 }
```

**9.47.3.61 void AbstractDiagram::setAttributesModel ([AttributesModel](#) \* *model*)** [virtual, inherited]

Associate an [AttributesModel](#) with this diagram.

Note that the diagram does `_not_` take ownership of the [AttributesModel](#). This should thus only be used with [AttributesModels](#) that have been explicitly created by the user, and are owned by her. Setting an [AttributesModel](#) that is internal to another diagram is an error.

Correct:

```
AttributesModel *am = new AttributesModel( model, 0 );  
diagram1->setAttributesModel( am );  
diagram2->setAttributesModel( am );
```

Wrong:

```
diagram1->setAttributesModel( diagram2->attributesModel() );
```

**Parameters:**

*model* The [AttributesModel](#) to use for this diagram.

See also:

[AttributesModel](#), [usesExternalAttributesModel](#)

Reimplemented in [KDChart::AbstractCartesianDiagram](#).

Definition at line 255 of file KDChartAbstractDiagram.cpp.

References `d`, `KDChart::AbstractDiagram::modelsChanged()`, and `KDChart::AbstractDiagram::setDataBoundariesDirty()`.

Referenced by `KDChart::AbstractCartesianDiagram::setAttributesModel()`.

```
256 {  
257     if( amodel->sourceModel() != model() ) {  
258         qWarning("KDChart::AbstractDiagram::setAttributesModel() failed: "  
259                 "Trying to set an attributesmodel which works on a different "  
260                 "model than the diagram.");
```

```

261         return;
262     }
263     if( qobject_cast<PrivateAttributesModel*>(amodel) ) {
264         qWarning("KDChart::AbstractDiagram::setAttributesModel() failed: "
265             "Trying to set an attributesmodel that is private to another diagram.");
266         return;
267     }
268     d->setAttributesModel(amodel);
269     scheduleDelayedItemsLayout();
270     setDataBoundariesDirty();
271     emit modelsChanged();
272 }

```

### 9.47.3.62 void AbstractDiagram::setAttributesModelRootIndex (const QModelIndex &) [protected, inherited]

Definition at line 293 of file KDChartAbstractDiagram.cpp.

References `d`, and `KDChart::AbstractDiagram::setDataBoundariesDirty()`.

Referenced by `KDChart::AbstractDiagram::setRootIndex()`.

```

294 {
295     d->attributesModelRootIndex=idx;
296     setDataBoundariesDirty();
297     scheduleDelayedItemsLayout();
298 }

```

### 9.47.3.63 void AbstractDiagram::setBrush (const QBrush & *brush*) [inherited]

Set the brush to be used, for painting all datasets in the model.

#### Parameters:

***brush*** The brush to use.

Definition at line 806 of file KDChartAbstractDiagram.cpp.

References `KDChart::AbstractDiagram::attributesModel()`, `KDChart::DatasetBrushRole`, `KDChart::AbstractDiagram::propertiesChanged()`, and `KDChart::AttributesModel::setModelData()`.

```

807 {
808     attributesModel()->setModelData(
809         qVariantFromValue( brush ), DatasetBrushRole );
810     emit propertiesChanged();
811 }

```

### 9.47.3.64 void AbstractDiagram::setBrush (int *dataset*, const QBrush & *brush*) [inherited]

Set the brush to be used, for painting the given dataset.

#### Parameters:

***dataset*** The dataset's column in the model.

***brush*** The brush to use.

Definition at line 813 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DatasetBrushRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setHeaderData().

```
814 {
815     attributesModel()->setHeaderData(
816         column, Qt::Vertical,
817         QVariantFromValue( brush ),
818         DatasetBrushRole );
819     emit propertiesChanged();
820 }
```

### 9.47.3.65 void AbstractDiagram::setBrush (const QModelIndex & *index*, const QBrush & *brush*) [inherited]

Set the brush to be used, for painting the datapoint at the given index.

#### Parameters:

***index*** The datapoint's index in the model.

***brush*** The brush to use.

Definition at line 798 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DatasetBrushRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setData().

```
799 {
800     attributesModel()->setData(
801         attributesModel()->mapFromSource( index ),
802         QVariantFromValue( brush ), DatasetBrushRole );
803     emit propertiesChanged();
804 }
```

### 9.47.3.66 void PolarDiagram::setCloseDatasets (bool *closeDatasets*)

Close each of the data series by connecting the last point to its respective start point.

Definition at line 235 of file KDChartPolarDiagram.cpp.

References d.

```
236 {
237     d->closeDatasets = closeDatasets;
238 }
```

### 9.47.3.67 void AbstractDiagram::setCoordinatePlane ([AbstractCoordinatePlane](#) \* *plane*) [virtual, inherited]

Set the coordinate plane associated with the diagram.

This determines how coordinates in value space are mapped into pixel space. The chart takes ownership.

**Returns:**

The coordinate plane associated with the diagram.

Reimplemented in [KDChart::AbstractCartesianDiagram](#).

Definition at line 316 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractCoordinatePlane::addDiagram(), KDChart::AbstractCartesianDiagram::setCoordinatePlane(), and KDChart::AbstractCoordinatePlane::takeDiagram().

```
317 {
318     d->plane = parent;
319 }
```

#### 9.47.3.68 void AbstractDiagram::setDataBoundariesDirty () const [protected, inherited]

Definition at line 234 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::dataChanged(), KDChart::Plotter::resize(), KDChart::LineDiagram::resize(), KDChart::BarDiagram::resize(), KDChart::AbstractDiagram::setAttributesModel(), KDChart::AbstractDiagram::setAttributesModelRootIndex(), KDChart::AbstractDiagram::setDatasetDimension(), KDChart::AbstractDiagram::setModel(), KDChart::BarDiagram::setThreeDBarAttributes(), KDChart::Plotter::setThreeDLineAttributes(), KDChart::LineDiagram::setThreeDLineAttributes(), KDChart::Plotter::setType(), KDChart::LineDiagram::setType(), and KDChart::BarDiagram::setType().

```
235 {
236     d->databoundariesDirty = true;
237 }
```

#### 9.47.3.69 void AbstractDiagram::setDatasetDimension (int *dimension*) [inherited]

Sets the dataset dimension of the diagram.

**See also:**

[datasetDimension](#).

**Parameters:**

*dimension*

Definition at line 1077 of file KDChartAbstractDiagram.cpp.

References d, KDChart::AbstractDiagram::layoutChanged(), and KDChart::AbstractDiagram::setDataBoundariesDirty().

Referenced by KDChart::Widget::setType(), KDChart::TernaryLineDiagram::TernaryLineDiagram(), and KDChart::TernaryPointDiagram::TernaryPointDiagram().

```
1078 {
1079     if ( d->datasetDimension == dimension ) return;
```



```

1080     d->datasetDimension = dimension;
1081     setDataBoundariesDirty();
1082     emit layoutChanged( this );
1083 }

```

#### 9.47.3.70 void AbstractDiagram::setDataValueAttributes (const [DataValueAttributes](#) & *a*) [inherited]

Set the [DataValueAttributes](#) for all datapoints in the model.

##### Parameters:

*a* The attributes to set.

Definition at line 428 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::DataValueLabelAttributesRole](#), and [KDChart::AbstractDiagram::propertiesChanged\(\)](#).

```

429 {
430     d->attributesModel->setModelData( qVariantFromValue( a ), DataValueLabelAttributesRole );
431     emit propertiesChanged();
432 }

```

#### 9.47.3.71 void AbstractDiagram::setDataValueAttributes (int *dataset*, const [DataValueAttributes](#) & *a*) [inherited]

Set the [DataValueAttributes](#) for the given dataset.

##### Parameters:

*dataset* The dataset to set the attributes for.

*a* The attributes to set.

Definition at line 400 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::DataValueLabelAttributesRole](#), and [KDChart::AbstractDiagram::propertiesChanged\(\)](#).

```

401 {
402     d->attributesModel->setHeaderData(
403         column, Qt::Vertical,
404         qVariantFromValue( a ), DataValueLabelAttributesRole );
405     emit propertiesChanged();
406 }

```

#### 9.47.3.72 void AbstractDiagram::setDataValueAttributes (const [QModelIndex](#) & *index*, const [DataValueAttributes](#) & *a*) [inherited]

Set the [DataValueAttributes](#) for the given index.

##### Parameters:

*index* The datapoint to set the attributes for.

*a* The attributes to set.

Definition at line 389 of file KDChartAbstractDiagram.cpp.

References `d`, `KDChart::DataValueLabelAttributesRole`, and `KDChart::AbstractDiagram::propertiesChanged()`.

```

391 {
392     d->attributesModel->setData(
393         d->attributesModel->mapFromSource( index ),
394         qVariantFromValue( a ),
395         DataValueLabelAttributesRole );
396     emit propertiesChanged();
397 }
```

### 9.47.3.73 void AbstractDiagram::setHidden (bool *hidden*) [inherited]

Hide (or unhide, resp.

) all datapoints in the model.

#### Note:

Hidden data are still taken into account by the coordinate plane, so neither the grid nor your axes' ranges will change, when you hide data. For totally removing data from KD Chart's view you can use another approach: e.g. you could define a proxy model on top of your data model, and register the proxy model calling [setModel\(\)](#) instead of registering your real data model.

#### Parameters:

*hidden* The hidden status to set.

Definition at line 359 of file KDChartAbstractDiagram.cpp.

References `d`, `KDChart::AbstractDiagram::dataHidden()`, and `KDChart::DataHiddenRole`.

```

360 {
361     d->attributesModel->setModelData(
362         qVariantFromValue( hidden ),
363         DataHiddenRole );
364     emit dataHidden();
365 }
```

### 9.47.3.74 void AbstractDiagram::setHidden (int *column*, bool *hidden*) [inherited]

Hide (or unhide, resp.

) a dataset.

#### Note:

Hidden data are still taken into account by the coordinate plane, so neither the grid nor your axes' ranges will change, when you hide data. For totally removing data from KD Chart's view you can use another approach: e.g. you could define a proxy model on top of your data model, and register the proxy model calling [setModel\(\)](#) instead of registering your real data model.

**Parameters:**

***column*** The dataset to set the hidden status for.

***hidden*** The hidden status to set.

Definition at line 350 of file KDChartAbstractDiagram.cpp.

References `d`, `KDChart::AbstractDiagram::dataHidden()`, and `KDChart::DataHiddenRole`.

```

351 {
352     d->attributesModel->setHeaderData(
353         column, Qt::Vertical,
354         QVariantFromValue( hidden ),
355         DataHiddenRole );
356     emit dataHidden();
357 }
```

#### 9.47.3.75 void AbstractDiagram::setHidden (const QModelIndex & *index*, bool *hidden*) [inherited]

Hide (or unhide, resp.

) a data cell.

**Note:**

Hidden data are still taken into account by the coordinate plane, so neither the grid nor your axes' ranges will change, when you hide data. For totally removing data from KD Chart's view you can use another approach: e.g. you could define a proxy model on top of your data model, and register the proxy model calling [setModel\(\)](#) instead of registering your real data model.

**Parameters:**

***index*** The datapoint to set the hidden status for.

***hidden*** The hidden status to set.

Definition at line 341 of file KDChartAbstractDiagram.cpp.

References `d`, `KDChart::AbstractDiagram::dataHidden()`, and `KDChart::DataHiddenRole`.

```

342 {
343     d->attributesModel->setData(
344         d->attributesModel->mapFromSource( index ),
345         QVariantFromValue( hidden ),
346         DataHiddenRole );
347     emit dataHidden();
348 }
```

#### 9.47.3.76 void AbstractDiagram::setModel (QAbstractItemModel \* *model*) [virtual, inherited]

Associate a model with the diagram.

Reimplemented in [KDChart::AbstractCartesianDiagram](#).

Definition at line 239 of file KDChartAbstractDiagram.cpp.

References `d`, `KDChart::AttributesModel::initFrom()`, `KDChart::AbstractDiagram::modelsChanged()`, and `KDChart::AbstractDiagram::setDataBoundariesDirty()`.

Referenced by `KDChart::AbstractCartesianDiagram::setModel()`, and `KDChart::Widget::setType()`.

```

240 {
241     QAbstractItemView::setModel( newModel );
242     AttributesModel* amodel = new PrivateAttributesModel( newModel, this );
243     amodel->initFrom( d->attributesModel );
244     d->setAttributesModel(amodel);
245     scheduleDelayedItemsLayout();
246     setDataBoundariesDirty();
247     emit modelsChanged();
248 }
```

#### 9.47.3.77 void AbstractDiagram::setPen (const QPen & *pen*) [inherited]

Set the pen to be used, for painting all datasets in the model.

##### Parameters:

***pen*** The pen to use.

Definition at line 753 of file `KDChartAbstractDiagram.cpp`.

References `KDChart::AbstractDiagram::attributesModel()`, `KDChart::DatasetPenRole`, `KDChart::AbstractDiagram::propertiesChanged()`, and `KDChart::AttributesModel::setModelData()`.

```

754 {
755     attributesModel()->setModelData(
756         qVariantFromValue( pen ), DatasetPenRole );
757     emit propertiesChanged();
758 }
```

#### 9.47.3.78 void AbstractDiagram::setPen (int *dataset*, const QPen & *pen*) [inherited]

Set the pen to be used, for painting the given dataset.

##### Parameters:

***dataset*** The dataset's row in the model.

***pen*** The pen to use.

Definition at line 760 of file `KDChartAbstractDiagram.cpp`.

References `KDChart::AbstractDiagram::attributesModel()`, `KDChart::AbstractDiagram::datasetDimension()`, `KDChart::DatasetPenRole`, `KDChart::AbstractDiagram::propertiesChanged()`, and `KDChart::AttributesModel::setHeaderData()`.

```

761 {
762     if( datasetDimension() > 1 )
763         column *= datasetDimension();
764     attributesModel()->setHeaderData(
765         column, Qt::Vertical,
766         qVariantFromValue( pen ),
767         DatasetPenRole );
768     emit propertiesChanged();
769 }
```

### 9.47.3.79 void AbstractDiagram::setPen (const QModelIndex & *index*, const QPen & *pen*) [inherited]

Set the pen to be used, for painting the datapoint at the given index.

#### Parameters:

***index*** The datapoint's index in the model.

***pen*** The pen to use.

Definition at line 743 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::datasetDimension(), KDChart::DatasetPenRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setData().

```
744 {
745     if( datasetDimension() > 1 )
746         return setPen( index.column(), pen );
747     attributesModel()->setData(
748         attributesModel()->mapFromSource( index ),
749         qVariantFromValue( pen ), DatasetPenRole );
750     emit propertiesChanged();
751 }
```

### 9.47.3.80 void AbstractDiagram::setPercentMode (bool *percent*) [inherited]

Definition at line 456 of file KDChartAbstractDiagram.cpp.

References d, and KDChart::AbstractDiagram::propertiesChanged().

Referenced by KDChart::LineDiagram::setType(), and KDChart::BarDiagram::setType().

```
457 {
458     d->percent = percent;
459     emit propertiesChanged();
460 }
```

### 9.47.3.81 void AbstractDiagram::setRootIndex (const QModelIndex & *idx*) [virtual, inherited]

Set the root index in the model, where the diagram starts referencing data for display.

[reimplemented]

Reimplemented in [KDChart::AbstractCartesianDiagram](#).

Definition at line 286 of file KDChartAbstractDiagram.cpp.

References d, and KDChart::AbstractDiagram::setAttributesModelRootIndex().

Referenced by KDChart::AbstractCartesianDiagram::setRootIndex().

```
287 {
288     QAbstractItemView::setRootIndex(idx);
289     setAttributesModelRootIndex( d->attributesModel->mapFromSource(idx) );
290 }
```

**9.47.3.82 void PolarDiagram::setRotateCircularLabels (bool *rotateCircularLabels*)**

Definition at line 225 of file KDChartPolarDiagram.cpp.

References d.

```

226 {
227     d->rotateCircularLabels = rotateCircularLabels;
228 }
```

**9.47.3.83 void AbstractDiagram::setSelection (const QRect & *rect*, QItemSelectionModel::SelectionFlags *command*)** [virtual, inherited]

[reimplemented]

Definition at line 959 of file KDChartAbstractDiagram.cpp.

References d.

```

960 {
961     const QModelIndexList indexes = d->indexesIn( rect );
962     QItemSelection selection;
963     KDAB_FOREACH( const QModelIndex& index, indexes )
964     {
965         selection.append( QItemSelectionRange( index ) );
966     }
967     selectionModel()->select( selection, command );
968 }
```

**9.47.3.84 void PolarDiagram::setShowDelimitersAtPosition ([Position](#) *position*, bool *showDelimiters*)**

Definition at line 245 of file KDChartPolarDiagram.cpp.

References d, and KDChart::Position::value().

```

247 {
248     d->showDelimitersAtPosition[position.value()] = showDelimiters;
249 }
```

**9.47.3.85 void PolarDiagram::setShowLabelsAtPosition ([Position](#) *position*, bool *showLabels*)**

Definition at line 251 of file KDChartPolarDiagram.cpp.

References d, and KDChart::Position::value().

```

253 {
254     d->showLabelsAtPosition[position.value()] = showLabels;
255 }
```

**9.47.3.86 void AbstractDiagram::setUnitPrefix (const QString & *prefix*, Qt::Orientation *orientation*)** [inherited]

Sets the unit prefix for all values.

**Parameters:**

*prefix* the prefix to be set

*orientation* the orientation of the axis to set

Definition at line 864 of file KDChartAbstractDiagram.cpp.

References d.

```
865 {  
866     d->unitPrefix[ orientation ] = prefix;  
867 }
```

**9.47.3.87 void AbstractDiagram::setUnitPrefix (const QString & *prefix*, int *column*, Qt::Orientation *orientation*)** [inherited]

Sets the unit prefix for one value.

**Parameters:**

*prefix* the prefix to be set

*column* the value using that prefix

*orientation* the orientation of the axis to set

Definition at line 854 of file KDChartAbstractDiagram.cpp.

References d.

```
855 {  
856     d->unitPrefixMap[ column ][ orientation ] = prefix;  
857 }
```

**9.47.3.88 void AbstractDiagram::setUnitSuffix (const QString & *suffix*, Qt::Orientation *orientation*)** [inherited]

Sets the unit suffix for all values.

**Parameters:**

*suffix* the suffix to be set

*orientation* the orientation of the axis to set

Definition at line 885 of file KDChartAbstractDiagram.cpp.

References d.

```
886 {  
887     d->unitSuffix[ orientation ] = suffix;  
888 }
```

### 9.47.3.89 void AbstractDiagram::setUnitSuffix (const QString & *suffix*, int *column*, Qt::Orientation *orientation*) [inherited]

Sets the unit suffix for one value.

#### Parameters:

*suffix* the suffix to be set  
*column* the value using that suffix  
*orientation* the orientation of the axis to set

Definition at line 875 of file KDChartAbstractDiagram.cpp.

References d.

```
876 {
877     d->unitSuffixMap[ column ][ orientation ]= suffix;
878 }
```

### 9.47.3.90 void PolarDiagram::setZeroDegreePosition (int *degrees*)

#### Deprecated

Use [PolarCoordinatePlane::setStartPosition\( qreal degrees \)](#) instead.

Definition at line 213 of file KDChartPolarDiagram.cpp.

```
214 {
215     Q_UNUSED( degrees );
216     qWarning() << "Deprecated PolarDiagram::setZeroDegreePosition() called, setting ignored.";
217 }
```

### 9.47.3.91 bool PolarDiagram::showDelimitersAtPosition ([Position](#) *position*) const

Definition at line 257 of file KDChartPolarDiagram.cpp.

References d, and KDChart::Position::value().

Referenced by clone().

```
258 {
259     return d->showDelimitersAtPosition[position.value()];
260 }
```

### 9.47.3.92 bool PolarDiagram::showLabelsAtPosition ([Position](#) *position*) const

Definition at line 262 of file KDChartPolarDiagram.cpp.

References d, and KDChart::Position::value().

Referenced by clone().

```
263 {
264     return d->showLabelsAtPosition[position.value()];
265 }
```



**9.47.3.93 QString AbstractDiagram::unitPrefix (Qt::Orientation *orientation*) const**  
[inherited]

Returns the global unit prefix.

**Parameters:**

*orientation* the orientation of the axis

**Returns:**

the unit prefix

Definition at line 908 of file KDChartAbstractDiagram.cpp.

References d.

```
909 {  
910     return d->unitPrefix[ orientation ];  
911 }
```

**9.47.3.94 QString AbstractDiagram::unitPrefix (int *column*, Qt::Orientation *orientation*, bool *fallback* = false) const**  
[inherited]

Returns the unit prefix for a special value.

**Parameters:**

*column* the value which's prefix is requested

*orientation* the orientation of the axis

*fallback* if true, the global prefix is return when no specific one is set for that value

**Returns:**

the unit prefix

Definition at line 897 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::itemRowLabels(), and KDChart::CartesianAxis::paintCtx().

```
898 {  
899     if( !fallback || d->unitPrefixMap[ column ].contains( orientation ) )  
900         return d->unitPrefixMap[ column ][ orientation ];  
901     return d->unitPrefix[ orientation ];  
902 }
```

**9.47.3.95 QString AbstractDiagram::unitSuffix (Qt::Orientation *orientation*) const**  
[inherited]

Returns the global unit suffix.

**Parameters:**

*orientation* the orientation of the axis

**Returns:**

the unit suffix

Definition at line 931 of file KDChartAbstractDiagram.cpp.

References d.

```

932 {
933     return d->unitSuffix[ orientation ];
934 }
```

### 9.47.3.96 QString AbstractDiagram::unitSuffix (int *column*, Qt::Orientation *orientation*, bool *fallback* = false) const [inherited]

Returns the unit suffix for a special value.

**Parameters:**

*column* the value which's suffix is requested

*orientation* the orientation of the axis

*fallback* if true, the global suffix is return when no specific one is set for that value

**Returns:**

the unit suffix

Definition at line 920 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::itemRowLabels(), and KDChart::CartesianAxis::paintCtx().

```

921 {
922     if( !fallback || d->unitSuffixMap[ column ].contains( orientation ) )
923         return d->unitSuffixMap[ column ][ orientation ];
924     return d->unitSuffix[ orientation ];
925 }
```

### 9.47.3.97 void AbstractDiagram::update () const [inherited]

Definition at line 1091 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::doItemsLayout().

```

1092 {
1093     //QDebug("KDChart::AbstractDiagram::update() called");
1094     if( d->plane )
1095         d->plane->update();
1096 }
```

**9.47.3.98 void KDChart::AbstractDiagram::useDefaultColors ()** [inherited]

Set the palette to be used, for painting datasets to the default palette.

**See also:**

[KDChart::Palette](#). FIXME: fold into one usePalette ([KDChart::Palette&](#)) method

Definition at line 974 of file KDChartAbstractDiagram.cpp.

References [d](#), and [KDChart::AttributesModel::PaletteTypeDefault](#).

```
975 {  
976     d->attributesModel->setPaletteType( AttributesModel::PaletteTypeDefault );  
977 }
```

**9.47.3.99 void KDChart::AbstractDiagram::useRainbowColors ()** [inherited]

Set the palette to be used, for painting datasets to the rainbow palette.

**See also:**

[KDChart::Palette](#).

Definition at line 984 of file KDChartAbstractDiagram.cpp.

References [d](#), and [KDChart::AttributesModel::PaletteTypeRainbow](#).

```
985 {  
986     d->attributesModel->setPaletteType( AttributesModel::PaletteTypeRainbow );  
987 }
```

**9.47.3.100 bool AbstractDiagram::usesExternalAttributesModel () const** [virtual, inherited]

Returns whether the diagram is using its own built-in attributes model or an attributes model that was set via [setAttributesModel](#).

**See also:**

[setAttributesModel](#)

Definition at line 274 of file KDChartAbstractDiagram.cpp.

References [d](#).

```
275 {  
276     return d->usesExternalAttributesModel();  
277 }
```

**9.47.3.101 void KDChart::AbstractDiagram::useSubduedColors ()** [inherited]

Set the palette to be used, for painting datasets to the subdued palette.

**See also:**

[KDChart::Palette](#).

Definition at line 979 of file KDChartAbstractDiagram.cpp.

References [d](#), and [KDChart::AttributesModel::PaletteTypeSubdued](#).

```

980 {
981     d->attributesModel->setPaletteType( AttributesModel::PaletteTypeSubdued );
982 }
```

**9.47.3.102 double AbstractDiagram::valueForCell (int row, int column) const** [protected, inherited]

Helper method, retrieving the data value (DisplayRole) for a given row and column.

**Parameters:**

*row* The row to query.

*column* The column to query.

**Returns:**

The value of the display role at the given row and column as a double.

**Deprecated**

Definition at line 1085 of file KDChartAbstractDiagram.cpp.

References [KDChart::AbstractDiagram::attributesModelRootIndex\(\)](#), and [d](#).

```

1086 {
1087     return d->attributesModel->data(
1088         d->attributesModel->index( row, column, attributesModelRootIndex() ) ).toDouble();
1089 }
```

**9.47.3.103 double PolarDiagram::valueTotals () const** [virtual]

[reimplemented]

Implements [KDChart::AbstractPolarDiagram](#).

Definition at line 196 of file KDChartPolarDiagram.cpp.

```

197 {
198     return model()->rowCount(rootIndex());
199 }
```

**9.47.3.104 int AbstractDiagram::verticalOffset () const** [virtual, inherited]

[reimplemented]

Definition at line 953 of file KDChartAbstractDiagram.cpp.

```
954 { return 0; }
```

**9.47.3.105 QRect AbstractDiagram::visualRect (const QModelIndex & *index*) const** [virtual, inherited]

[reimplemented]

Definition at line 937 of file KDChartAbstractDiagram.cpp.

```
938 {  
939     return QRect();  
940 }
```

**9.47.3.106 QRegion AbstractDiagram::visualRegionForSelection (const QItemSelection & *selection*) const** [virtual, inherited]

[reimplemented]

Definition at line 970 of file KDChartAbstractDiagram.cpp.

```
971 { return QRegion(); }
```

**9.47.3.107 int PolarDiagram::zeroDegreePosition () const****Deprecated**

Use qreal [PolarCoordinatePlane::startPosition](#) instead.

Definition at line 219 of file KDChartPolarDiagram.cpp.

```
220 {  
221     qWarning() << "Deprecated PolarDiagram::zeroDegreePosition() called.";  
222     return 0;  
223 }
```

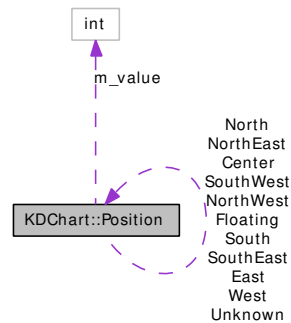
The documentation for this class was generated from the following files:

- [KDChartPolarDiagram.h](#)
- [KDChartPolarDiagram.cpp](#)

## 9.48 KDChart::Position Class Reference

```
#include <KDChartPosition.h>
```

Collaboration diagram for KDChart::Position:



### 9.48.1 Detailed Description

Defines a position, using compass terminology.

Using KDChartPosition you can specify one of nine pre-defined, logical points (see the `static const` getter methods below), in a similar way, as you would use a compass to navigate on a map.

#### Note:

Often you will declare a [Position](#) together with the [RelativePosition](#) class, to specify a logical point, which then will be used to layout your chart at runtime, e.g. for specifying the location of a floating [Legend](#) box.

For comparing a Position's value with a `switch()` statement, you can use numeric values defined in [KDChartEnums](#), like this:

```

switch( yourPosition().value() ) {
    case KDChartEnums::PositionNorthWest:
        // your code ...
        break;
    case KDChartEnums::PositionNorth:
        // your code ...
        break;
}

```

#### See also:

[RelativePosition](#), [KDChartEnums::PositionValue](#)

Definition at line 75 of file KDChartPosition.h.

### Public Types

- enum [Option](#) {  
[IncludeCenter](#) = 0,  
[ExcludeCenter](#) = 1 }

## Public Member Functions

- bool [isCorner](#) () const
- bool [isEastSide](#) () const
- bool [isFloating](#) () const
- bool [isNorthSide](#) () const
- bool [isPole](#) () const
- bool [isSouthSide](#) () const
- bool [isUnknown](#) () const
- bool [isWestSide](#) () const
- const char \* [name](#) () const

*Returns a non-translated string in English language, corresponding to this [Position](#).*

- bool [operator!=](#) (int) const
- bool [operator!=](#) (const [Position](#) &) const
- bool [operator==](#) (int) const
- bool [operator==](#) (const [Position](#) &) const
- [Position](#) ([KDChartEnums::PositionValue](#) value)

*Constructor.*

- [Position](#) ()

*Default constructor.*

- QString [printableName](#) () const

*Returns a translated string, corresponding to this [Position](#).*

- [KDChartEnums::PositionValue](#) [value](#) () const

*Returns an integer value corresponding to this [Position](#).*

## Static Public Member Functions

- static [Position](#) [fromName](#) (const QByteArray &name)
- static [Position](#) [fromName](#) (const char \*name)
- static QList< QByteArray > [names](#) (Options options=IncludeCenter)

*Returns a list of all string, corresponding to the pre-defined positions.*

- static QStringList [printableNames](#) (Options options=IncludeCenter)

*Returns a list of all translated string, corresponding to the pre-defined positions.*

## Static Public Attributes

- static const [Position](#) & [Center](#)
- static const [Position](#) & [East](#)
- static const [Position](#) & [Floating](#)
- static const [Position](#) & [North](#)
- static const [Position](#) & [NorthEast](#)
- static const [Position](#) & [NorthWest](#)

- static const [Position](#) & [South](#)
- static const [Position](#) & [SouthEast](#)
- static const [Position](#) & [SouthWest](#)
- static const [Position](#) & [Unknown](#)
- static const [Position](#) & [West](#)

## 9.48.2 Member Enumeration Documentation

### 9.48.2.1 enum [KDChart::Position::Option](#)

Enumerator:

*IncludeCenter*

*ExcludeCenter*

Definition at line 113 of file KDChartPosition.h.

```
113 { IncludeCenter=0, ExcludeCenter=1 };
```

## 9.48.3 Constructor & Destructor Documentation

### 9.48.3.1 [Position::Position \(\)](#)

Default constructor.

Creates a new [Position](#), defaulting it to [Position::Unknown](#).

Definition at line 100 of file KDChartPosition.cpp.

Referenced by [fromName\(\)](#).

```
101      : m_value( KDChartEnums::PositionUnknown )
102  {
103
104  }
```

### 9.48.3.2 [Position::Position \(KDChartEnums::PositionValue value\)](#)

Constructor.

Creates a new [Position](#), defaulting it to the respective value.

Valid values ranging from zero (unknown value) to 10. If invalid value is passed, a [Position::Unknown](#) is created.

**Note:**

Normally there is no need to call this constructor, but you would rather use one of the nine pre-defined, static values, e.g. like this:

```
* const KDChart::Position myPosition = KDChart::Position::NorthEast;
*
```

Definition at line 124 of file KDChartPosition.cpp.



```
125     : m_value( value )
126 {
127
128 }
```

## 9.48.4 Member Function Documentation

### 9.48.4.1 [Position](#) Position::fromName (const QByteArray & *name*) [static]

Definition at line 243 of file KDChartPosition.cpp.

References fromName().

```
243                                     {
244     return fromName( name.data() );
245 }
```

### 9.48.4.2 [Position](#) Position::fromName (const char \* *name*) [static]

Definition at line 235 of file KDChartPosition.cpp.

References maxPositionValue, Position(), and staticPositionNames.

Referenced by fromName().

```
236 {
237     for( int i=1; i<=maxPositionValue; ++i)
238         if ( !qstrcmp( name, staticPositionNames[i] ) )
239             return Position(i);
240     return Position(0);
241 }
```

### 9.48.4.3 [bool](#) Position::isCorner () const

Definition at line 168 of file KDChartPosition.cpp.

References NorthEast, NorthWest, SouthEast, SouthWest, and value().

```
169 {
170     return m_value == Position::NorthWest.value() ||
171            m_value == Position::NorthEast.value() ||
172            m_value == Position::SouthEast.value() ||
173            m_value == Position::SouthWest.value();
174 }
```

### 9.48.4.4 [bool](#) Position::isEastSide () const

Definition at line 155 of file KDChartPosition.cpp.

References East, NorthEast, SouthEast, and value().

```
156 {
157     return m_value == Position::NorthEast.value() ||
158            m_value == Position::East.value() ||
159            m_value == Position::SouthEast.value();
160 }
```

#### 9.48.4.5 bool Position::isFloating () const

Definition at line 181 of file KDChartPosition.cpp.

References Floating, and value().

Referenced by KDChart::Chart::reLayoutFloatingLegends().

```
182 {  
183     return m_value == Position::Floating.value();  
184 }
```

#### 9.48.4.6 bool Position::isNorthSide () const

Definition at line 149 of file KDChartPosition.cpp.

References North, NorthEast, NorthWest, and value().

```
150 {  
151     return m_value == Position::NorthWest.value() ||  
152           m_value == Position::North.value() ||  
153           m_value == Position::NorthEast.value();  
154 }
```

#### 9.48.4.7 bool Position::isPole () const

Definition at line 175 of file KDChartPosition.cpp.

References North, South, and value().

```
176 {  
177     return m_value == Position::North.value() ||  
178           m_value == Position::South.value();  
179 }
```

#### 9.48.4.8 bool Position::isSouthSide () const

Definition at line 161 of file KDChartPosition.cpp.

References South, SouthEast, SouthWest, and value().

```
162 {  
163     return m_value == Position::SouthWest.value() ||  
164           m_value == Position::South.value() ||  
165           m_value == Position::SouthEast.value();  
166 }
```

#### 9.48.4.9 bool Position::isUnknown () const

Definition at line 138 of file KDChartPosition.cpp.

References Unknown, and value().

```
139 {  
140     return m_value == Position::Unknown.value();  
141 }
```

#### 9.48.4.10 bool Position::isWestSide () const

Definition at line 143 of file KDChartPosition.cpp.

References NorthWest, SouthWest, value(), and West.

```
144 {  
145     return  m_value == Position::SouthWest.value() ||  
146            m_value == Position::West.value() ||  
147            m_value == Position::NorthWest.value();  
148 }
```

#### 9.48.4.11 const char \* Position::name () const

Returns a non-translated string in English language, corresponding to this [Position](#).

Definition at line 189 of file KDChartPosition.cpp.

References staticPositionNames.

Referenced by operator<<().

```
190 {  
191     return staticPositionNames[m_value];  
192 }
```

#### 9.48.4.12 QList< QByteArray > Position::names (Options *options* = IncludeCenter) [static]

Returns a list of all string, corresponding to the pre-defined positions.

##### Parameters:

*options* if set to ExcludeCenter, the returned list does not contain the Center position.

Definition at line 210 of file KDChartPosition.cpp.

References IncludeCenter, maxPositionValue, and staticPositionNames.

```
211 {  
212     QList<QByteArray> list;  
213     const int start = ( options & IncludeCenter ) ? 1 : 2;  
214     for( int i=start; i<=maxPositionValue; ++i)  
215         list.append( staticPositionNames[i] );  
216     return list;  
217 }
```

#### 9.48.4.13 bool KDChart::Position::operator!=(int) const

Definition at line 132 of file KDChartPosition.h.

References operator==( ).

```
132 { return !operator==( other ); }
```

**9.48.4.14** `bool KDChart::Position::operator!=(const Position &) const`

Definition at line 131 of file KDChartPosition.h.

```
131 { return !operator==( other ); }
```

**9.48.4.15** `bool Position::operator==(int) const`

Definition at line 253 of file KDChartPosition.cpp.

References `value()`.

```
254 {
255     return ( value() == value_ );
256 }
```

**9.48.4.16** `bool Position::operator==(const Position &) const`

Definition at line 247 of file KDChartPosition.cpp.

References `value()`.

Referenced by `operator!==( )`.

```
248 {
249     return ( value() == r.value() );
250 }
```

**9.48.4.17** `QString Position::printableName () const`

Returns a translated string, corresponding to this [Position](#).

Definition at line 197 of file KDChartPosition.cpp.

References `staticPositionNames`.

Referenced by `printableNames()`.

```
198 {
199     return tr(staticPositionNames[m_value]);
200 }
```

**9.48.4.18** `QStringList Position::printableNames (Options options = IncludeCenter)  
[static]`

Returns a list of all translated string, corresponding to the pre-defined positions.

**Parameters:**

*options* if set to `ExcludeCenter`, the returned list does not contain the Center position.

Definition at line 226 of file KDChartPosition.cpp.

References `IncludeCenter`, `maxPositionValue`, and `printableName()`.

```

227 {
228     QStringList list;
229     const int start = ( options & IncludeCenter ) ? 1 : 2;
230     for( int i=start; i<=maxPositionValue; ++i)
231         list.append( Position(i).printableName() );
232     return list;
233 }

```

#### 9.48.4.19 KDCartEnums::PositionValue Position::value () const

Returns an integer value corresponding to this [Position](#).

Definition at line 133 of file KDCartPosition.cpp.

Referenced by [isCorner\(\)](#), [isEastSide\(\)](#), [isFloating\(\)](#), [isNorthSide\(\)](#), [isPole\(\)](#), [isSouthSide\(\)](#), [isUnknown\(\)](#), [isWestSide\(\)](#), [operator==\(\)](#), [KDCart::TernaryAxis::setPosition\(\)](#), [KDCart::PolarDiagram::setShowDelimitersAtPosition\(\)](#), [KDCart::PolarDiagram::setShowLabelsAtPosition\(\)](#), [KDCart::PolarDiagram::showDelimitersAtPosition\(\)](#), and [KDCart::PolarDiagram::showLabelsAtPosition\(\)](#).

```

134 {
135     return static_cast<KDCartEnums::PositionValue>( m_value );
136 }

```

### 9.48.5 Member Data Documentation

#### 9.48.5.1 const Position & Position::Center [static]

Definition at line 101 of file KDCartPosition.h.

Referenced by [KDCart::PositionPoints::point\(\)](#).

#### 9.48.5.2 const Position & Position::East [static]

Definition at line 105 of file KDCartPosition.h.

Referenced by [isEastSide\(\)](#), and [KDCart::PositionPoints::point\(\)](#).

#### 9.48.5.3 const Position & Position::Floating [static]

Definition at line 111 of file KDCartPosition.h.

Referenced by [isFloating\(\)](#), and [KDCart::Legend::setFloatingPosition\(\)](#).

#### 9.48.5.4 const Position & Position::North [static]

Definition at line 103 of file KDCartPosition.h.

Referenced by [isNorthSide\(\)](#), [isPole\(\)](#), and [KDCart::PositionPoints::point\(\)](#).

#### 9.48.5.5 const Position & Position::NorthEast [static]

Definition at line 104 of file KDCartPosition.h.

Referenced by [isCorner\(\)](#), [isEastSide\(\)](#), [isNorthSide\(\)](#), and [KDCart::PositionPoints::point\(\)](#).

**9.48.5.6** `const Position & Position::NorthWest` `[static]`

Definition at line 102 of file `KDChartPosition.h`.

Referenced by `isCorner()`, `isNorthSide()`, `isWestSide()`, and `KDChart::PositionPoints::point()`.

**9.48.5.7** `const Position & Position::South` `[static]`

Definition at line 107 of file `KDChartPosition.h`.

Referenced by `isPole()`, `isSouthSide()`, and `KDChart::PositionPoints::point()`.

**9.48.5.8** `const Position & Position::SouthEast` `[static]`

Definition at line 106 of file `KDChartPosition.h`.

Referenced by `isCorner()`, `isEastSide()`, `isSouthSide()`, and `KDChart::PositionPoints::point()`.

**9.48.5.9** `const Position & Position::SouthWest` `[static]`

Definition at line 108 of file `KDChartPosition.h`.

Referenced by `isCorner()`, `isSouthSide()`, `isWestSide()`, and `KDChart::PositionPoints::point()`.

**9.48.5.10** `const Position & Position::Unknown` `[static]`

Definition at line 100 of file `KDChartPosition.h`.

Referenced by `isUnknown()`, and `KDChart::RelativePosition::resetReferencePosition()`.

**9.48.5.11** `const Position & Position::West` `[static]`

Definition at line 109 of file `KDChartPosition.h`.

Referenced by `isWestSide()`, and `KDChart::PositionPoints::point()`.

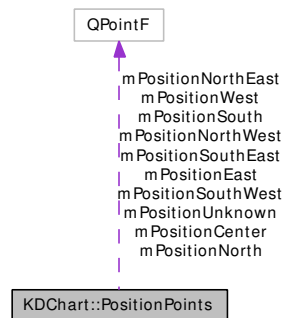
The documentation for this class was generated from the following files:

- [KDChartPosition.h](#)
- [KDChartPosition.cpp](#)

## 9.49 KDChart::PositionPoints Class Reference

```
#include <KDChartPosition.h>
```

Collaboration diagram for KDChart::PositionPoints:



### 9.49.1 Detailed Description

Stores the absolute target points of a [Position](#).

Definition at line 138 of file KDChartPosition.h.

### Public Member Functions

- bool [isNull](#) () const
- const QPointF [point](#) ([Position](#) position) const
- [PositionPoints](#) (QPointF northWest, QPointF northEast, QPointF southEast, QPointF southWest)
- [PositionPoints](#) (const QRectF &rect)
- [PositionPoints](#) (const QPointF &onePointForAllPositions)
- [PositionPoints](#) (QPointF center, QPointF northWest, QPointF north, QPointF northEast, QPointF east, QPointF southEast, QPointF south, QPointF southWest, QPointF west)
- [PositionPoints](#) ()

### Public Attributes

- QPointF [mPositionCenter](#)
- QPointF [mPositionEast](#)
- QPointF [mPositionNorth](#)
- QPointF [mPositionNorthEast](#)
- QPointF [mPositionNorthWest](#)
- QPointF [mPositionSouth](#)
- QPointF [mPositionSouthEast](#)
- QPointF [mPositionSouthWest](#)
- QPointF [mPositionUnknown](#)
- QPointF [mPositionWest](#)

## 9.49.2 Constructor & Destructor Documentation

### 9.49.2.1 KDChart::PositionPoints::PositionPoints ()

Definition at line 141 of file KDChartPosition.h.

```
141 {} // all points get initialized with the default automatically
```

### 9.49.2.2 KDChart::PositionPoints::PositionPoints (QPointF *center*, QPointF *northWest*, QPointF *north*, QPointF *northEast*, QPointF *east*, QPointF *southEast*, QPointF *south*, QPointF *southWest*, QPointF *west*)

Definition at line 143 of file KDChartPosition.h.

```
153      : mPositionCenter(      center )
154      , mPositionNorthWest(  northWest )
155      , mPositionNorth(      north )
156      , mPositionNorthEast(  northEast )
157      , mPositionEast(        east )
158      , mPositionSouthEast(  southEast )
159      , mPositionSouth(       south )
160      , mPositionSouthWest(  southWest )
161      , mPositionWest(        west )
162      {}
```

### 9.49.2.3 KDChart::PositionPoints::PositionPoints (const QPointF & *onePointForAllPositions*)

Definition at line 163 of file KDChartPosition.h.

```
165      : mPositionCenter(      onePointForAllPositions )
166      , mPositionNorthWest(  onePointForAllPositions )
167      , mPositionNorth(      onePointForAllPositions )
168      , mPositionNorthEast(  onePointForAllPositions )
169      , mPositionEast(        onePointForAllPositions )
170      , mPositionSouthEast(  onePointForAllPositions )
171      , mPositionSouth(       onePointForAllPositions )
172      , mPositionSouthWest(  onePointForAllPositions )
173      , mPositionWest(        onePointForAllPositions )
174      {}
```

### 9.49.2.4 KDChart::PositionPoints::PositionPoints (const QRectF & *rect*)

Definition at line 175 of file KDChartPosition.h.

```
177      {
178          const QRectF r( rect.normalized() );
179          mPositionCenter      = r.center();
180          mPositionNorthWest   = r.topLeft();
181          mPositionNorth       = QPointF(r.center().x(), r.top());
182          mPositionNorthEast   = r.topRight();
183          mPositionEast        = QPointF(r.right(), r.center().y());
184          mPositionSouthEast   = r.bottomRight();
185          mPositionSouth       = QPointF(r.center().x(), r.bottom());
186          mPositionSouthWest   = r.bottomLeft();
187          mPositionWest        = QPointF(r.left(), r.center().y());
188      }
```



### 9.49.2.5 KDChart::PositionPoints::PositionPoints (QPointF *northWest*, QPointF *northEast*, QPointF *southEast*, QPointF *southWest*)

Definition at line 189 of file KDChartPosition.h.

```

194      : mPositionCenter(      (northWest + southEast) / 2.0 )
195      , mPositionNorthWest( northWest )
196      , mPositionNorth(      (northWest + northEast) / 2.0 )
197      , mPositionNorthEast( northEast )
198      , mPositionEast(       (northEast + southEast) / 2.0 )
199      , mPositionSouthEast( southEast )
200      , mPositionSouth(      (southWest + southEast) / 2.0 )
201      , mPositionSouthWest( southWest )
202      , mPositionWest(       (northWest + southWest) / 2.0 )
203      {}

```

## 9.49.3 Member Function Documentation

### 9.49.3.1 bool KDChart::PositionPoints::isNull () const

Definition at line 229 of file KDChartPosition.h.

Referenced by KDChart::RelativePosition::setReferencePoints().

```

230      {
231          return
232              mPositionUnknown.isNull() &&
233              mPositionCenter.isNull() &&
234              mPositionNorthWest.isNull() &&
235              mPositionNorth.isNull() &&
236              mPositionNorthEast.isNull() &&
237              mPositionEast.isNull() &&
238              mPositionSouthEast.isNull() &&
239              mPositionSouth.isNull() &&
240              mPositionSouthWest.isNull() &&
241              mPositionWest.isNull();
242      }

```

### 9.49.3.2 const QPointF KDChart::PositionPoints::point (Position *position*) const

Definition at line 205 of file KDChartPosition.h.

References KDChart::Position::Center, KDChart::Position::East, KDChart::Position::North, KDChart::Position::NorthEast, KDChart::Position::NorthWest, KDChart::Position::South, KDChart::Position::SouthEast, KDChart::Position::SouthWest, and KDChart::Position::West.

```

206      {
207          //qDebug() << "point( " << position.name() << " )";
208          if( position == Position::Center)
209              return mPositionCenter;
210          if( position == Position::NorthWest)
211              return mPositionNorthWest;
212          if( position == Position::North)
213              return mPositionNorth;
214          if( position == Position::NorthEast)
215              return mPositionNorthEast;
216          if( position == Position::East)
217              return mPositionEast;
218          if( position == Position::SouthEast)

```

```
219         return mPositionSouthEast;
220     if( position == Position::South)
221         return mPositionSouth;
222     if( position == Position::SouthWest)
223         return mPositionSouthWest;
224     if( position == Position::West)
225         return mPositionWest;
226     return mPositionUnknown;
227 }
```

## 9.49.4 Member Data Documentation

### 9.49.4.1 **QPointF** [KDChart::PositionPoints::mPositionCenter](#)

Definition at line 245 of file KDChartPosition.h.

### 9.49.4.2 **QPointF** [KDChart::PositionPoints::mPositionEast](#)

Definition at line 249 of file KDChartPosition.h.

### 9.49.4.3 **QPointF** [KDChart::PositionPoints::mPositionNorth](#)

Definition at line 247 of file KDChartPosition.h.

### 9.49.4.4 **QPointF** [KDChart::PositionPoints::mPositionNorthEast](#)

Definition at line 248 of file KDChartPosition.h.

### 9.49.4.5 **QPointF** [KDChart::PositionPoints::mPositionNorthWest](#)

Definition at line 246 of file KDChartPosition.h.

### 9.49.4.6 **QPointF** [KDChart::PositionPoints::mPositionSouth](#)

Definition at line 251 of file KDChartPosition.h.

### 9.49.4.7 **QPointF** [KDChart::PositionPoints::mPositionSouthEast](#)

Definition at line 250 of file KDChartPosition.h.

### 9.49.4.8 **QPointF** [KDChart::PositionPoints::mPositionSouthWest](#)

Definition at line 252 of file KDChartPosition.h.

### 9.49.4.9 **QPointF** [KDChart::PositionPoints::mPositionUnknown](#)

Definition at line 244 of file KDChartPosition.h.

#### 9.49.4.10 QPointF [KDChart::PositionPoints::mPositionWest](#)

Definition at line 253 of file KDChartPosition.h.

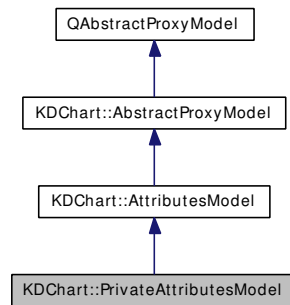
The documentation for this class was generated from the following file:

- [KDChartPosition.h](#)

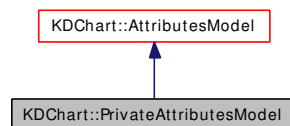
## 9.50 KDChart::PrivateAttributesModel Class Reference

```
#include <KDChartAbstractDiagram.h>
```

Inheritance diagram for KDChart::PrivateAttributesModel:



Collaboration diagram for KDChart::PrivateAttributesModel:



### 9.50.1 Detailed Description

Internally used class just adding a special constructor used by [AbstractDiagram](#).

Definition at line 656 of file `KDChartAbstractDiagram.h`.

### Public Types

- enum `PaletteType` {  
     `PaletteTypeDefault` = 0,  
     `PaletteTypeRainbow` = 1,  
     `PaletteTypeSubdued` = 2 }

### Signals

- void `attributesChanged` (const `QModelIndex` &, const `QModelIndex` &)

### Public Member Functions

- int `columnCount` (const `QModelIndex` &) const  
     *[reimplemented]*
- bool `compare` (const `AttributesModel` \*other) const
- bool `compareAttributes` (int role, const `QVariant` &a, const `QVariant` &b) const

- QVariant [data](#) (const QModelIndex &, int role=Qt::DisplayRole) const  
*[reimplemented]*
- QVariant [data](#) (int column, int role) const  
*Returns the data that were specified at per column level, or the globally set data, or the default data, or QVariant().*
- QVariant [data](#) (int role) const  
*Returns the data that were specified at global level, or the default data, or QVariant().*
- QVariant [headerData](#) (int section, Qt::Orientation orientation, int role=Qt::DisplayRole) const  
*[reimplemented]*
- QModelIndex [index](#) (int row, int col, const QModelIndex &index) const
- void [initFrom](#) (const [AttributesModel](#) \*other)
- bool [isKnownAttributesRole](#) (int role) const  
*Returns whether the given role corresponds to one of the known internally used ones.*
- QModelIndex [mapFromSource](#) (const QModelIndex &sourceIndex) const
- QModelIndex [mapToSource](#) (const QModelIndex &proxyIndex) const
- QVariant [modelData](#) (int role) const
- [PaletteType](#) [paletteType](#) () const
- QModelIndex [parent](#) (const QModelIndex &index) const
- [PrivateAttributesModel](#) (QAbstractItemModel \*model, [QObject](#) \*parent=0)
- bool [resetData](#) (const QModelIndex &index, int role=Qt::DisplayRole)  
*Remove any explicit attributes settings that might have been specified before.*
- bool [resetHeaderData](#) (int section, Qt::Orientation orientation, int role=Qt::DisplayRole)  
*Remove any explicit attributes settings that might have been specified before.*
- int [rowCount](#) (const QModelIndex &) const  
*[reimplemented]*
- bool [setData](#) (const QModelIndex &index, const QVariant &value, int role=Qt::DisplayRole)  
*[reimplemented]*
- void [setDefaultForRole](#) (int role, const QVariant &value)  
*Define the default value for a certain role.*
- bool [setHeaderData](#) (int section, Qt::Orientation orientation, const QVariant &value, int role=Qt::DisplayRole)  
*[reimplemented]*
- bool [setModelData](#) (const QVariant value, int role)
- void [setPaletteType](#) ([PaletteType](#) type)  
*Sets the palettetype used by this attributesmodel.*
- void [setSourceModel](#) (QAbstractItemModel \*sourceModel)  
*[reimplemented]*

## Protected Member Functions

- `const QMap< int, QMap< int, QMap< int, QVariant > > > dataMap () const`  
*needed for serialization*
- `const QMap< int, QMap< int, QVariant > > horizontalHeaderDataMap () const`  
*needed for serialization*
- `const QMap< int, QVariant > modelDataMap () const`  
*needed for serialization*
- `void setDataMap (const QMap< int, QMap< int, QMap< int, QVariant > > > map)`  
*needed for serialization*
- `void setHorizontalHeaderDataMap (const QMap< int, QMap< int, QVariant > > map)`  
*needed for serialization*
- `void setModelDataMap (const QMap< int, QVariant > map)`  
*needed for serialization*
- `void setVerticalHeaderDataMap (const QMap< int, QMap< int, QVariant > > map)`  
*needed for serialization*
- `const QMap< int, QMap< int, QVariant > > verticalHeaderDataMap () const`  
*needed for serialization*

## 9.50.2 Member Enumeration Documentation

### 9.50.2.1 enum [KDChart::AttributesModel::PaletteType](#) [inherited]

Enumerator:

*[PaletteTypeDefault](#)*  
*[PaletteTypeRainbow](#)*  
*[PaletteTypeSubdued](#)*

Definition at line 50 of file `KDChartAttributesModel.h`.

```

50         {
51             PaletteTypeDefault = 0,
52             PaletteTypeRainbow = 1,
53             PaletteTypeSubdued = 2
54     };

```

## 9.50.3 Constructor & Destructor Documentation

### 9.50.3.1 [KDChart::PrivateAttributesModel::PrivateAttributesModel](#) ([QAbstractItemModel](#) \* *model*, [QObject](#) \* *parent* = 0) [explicit]

Definition at line 659 of file `KDChartAbstractDiagram.h`.

```

660         : AttributesModel(model,parent) {}

```

## 9.50.4 Member Function Documentation

### 9.50.4.1 void KDChart::AttributesModel::attributesChanged (const QModelIndex &, const QModelIndex &) [signal, inherited]

Referenced by KDChart::AttributesModel::setData(), KDChart::AttributesModel::setHeaderData(), and KDChart::AttributesModel::setModelData().

### 9.50.4.2 int AttributesModel::columnCount (const QModelIndex &) const [inherited]

[reimplemented]

Definition at line 523 of file KDChartAttributesModel.cpp.

References KDChart::AbstractProxyModel::mapToSource().

Referenced by KDChart::AbstractDiagram::datasetBrushes(), KDChart::AbstractDiagram::datasetLabels(), KDChart::AbstractDiagram::datasetMarkers(), KDChart::AbstractDiagram::datasetPens(), and KDChart::AttributesModel::setModelData().

```

524 {
525     if ( sourceModel() ) {
526         return sourceModel()->columnCount( mapToSource(index) );
527     } else {
528         return 0;
529     }
530 }
```

### 9.50.4.3 bool AttributesModel::compare (const AttributesModel \* other) const [inherited]

Definition at line 83 of file KDChartAttributesModel.cpp.

References KDChart::AttributesModel::compareAttributes(), KDChart::AttributesModel::mDataMap, KDChart::AttributesModel::mHorizontalHeaderDataMap, KDChart::AttributesModel::mModelDataMap, KDChart::AttributesModel::mVerticalHeaderDataMap, and KDChart::AttributesModel::paletteType().

Referenced by KDChart::AbstractDiagram::compare().

```

84 {
85     if( other == this ) return true;
86     if( ! other ){
87         //qDebug() << "AttributesModel::compare() cannot compare to Null pointer";
88         return false;
89     }
90
91     {
92         if (mDataMap.count() != other->mDataMap.count()){
93             //qDebug() << "AttributesModel::compare() dataMap have different sizes";
94             return false;
95         }
96         QMap<int, QMap<int, QMap<int, QVariant>>>>::const_iterator itA = mDataMap.constBegin();
97         QMap<int, QMap<int, QMap<int, QVariant>>>>::const_iterator itB = other->mDataMap.constBegin();
98         while (itA != mDataMap.constEnd()) {
99             if ((*itA).count() != (*itB).count()){
100                 //qDebug() << "AttributesModel::compare() dataMap/map have different sizes";
101                 return false;
102             }
103             QMap<int, QMap<int, QVariant>>>>::const_iterator it2A = (*itA).constBegin();
104             QMap<int, QMap<int, QVariant>>>>::const_iterator it2B = (*itB).constBegin();
105             while (it2A != itA->constEnd()) {
```

```

106         if ((*it2A).count() != (*it2B).count()){
107             //qDebug() << "AttributesModel::compare() dataMap/map/map have different sizes:"
108             //             << (*it2A).count() << (*it2B).count();
109             return false;
110         }
111         QMap<int, QVariant>::const_iterator it3A = (*it2A).constBegin();
112         QMap<int, QVariant>::const_iterator it3B = (*it2B).constBegin();
113         while (it3A != it2A->constEnd()) {
114             if ( it3A.key() != it3B.key() ){
115                 //qDebug( "AttributesModel::compare()\n"
116                 //             "    dataMap[%i, %i] values have different types.  A: %x  B: %x",
117                 //             itA.key(), it2A.key(), it3A.key(), it3B.key());
118                 return false;
119             }
120             if ( ! compareAttributes( it3A.key(), it3A.value(), it3B.value() ) ){
121                 //qDebug( "AttributesModel::compare()\n"
122                 //             "    dataMap[%i, %i] values are different. Role: %x", itA.key(), it2A
123                 return false;
124             }
125             ++it3A;
126             ++it3B;
127         }
128         ++it2A;
129         ++it2B;
130     }
131     ++itA;
132     ++itB;
133 }
134 }
135 {
136     if (mHorizontalHeaderDataMap.count() != other->mHorizontalHeaderDataMap.count()){
137         //qDebug() << "AttributesModel::compare() horizontalHeaderDataMap have different sizes";
138         return false;
139     }
140     QMap<int, QMap<int, QVariant> >::const_iterator itA = mHorizontalHeaderDataMap.constBegin();
141     QMap<int, QMap<int, QVariant> >::const_iterator itB = other->mHorizontalHeaderDataMap.constBeg
142     while (itA != mHorizontalHeaderDataMap.constEnd()) {
143         if ((*itA).count() != (*itB).count()){
144             //qDebug() << "AttributesModel::compare() horizontalHeaderDataMap/map have different s
145             return false;
146         }
147         QMap<int, QVariant>::const_iterator it2A = (*itA).constBegin();
148         QMap<int, QVariant>::const_iterator it2B = (*itB).constBegin();
149         while (it2A != itA->constEnd()) {
150             if ( it2A.key() != it2B.key() ){
151                 //qDebug( "AttributesModel::compare()\n"
152                 //             "    horizontalHeaderDataMap[ %i ] values have different types.  A: %x  B
153                 //             itA.key(), it2A.key(), it2B.key());
154                 return false;
155             }
156             if ( ! compareAttributes( it2A.key(), it2A.value(), it2B.value() ) ){
157                 //qDebug( "AttributesModel::compare()\n"
158                 //             "    horizontalHeaderDataMap[ %i ] values are different. Role: %x", itA.k
159                 return false;
160             }
161             ++it2A;
162             ++it2B;
163         }
164         ++itA;
165         ++itB;
166     }
167 }
168 {
169     if (mVerticalHeaderDataMap.count() != other->mVerticalHeaderDataMap.count()){
170         //qDebug() << "AttributesModel::compare() verticalHeaderDataMap have different sizes";
171         return false;
172     }

```



```

173     QMap<int, QMap<int, QVariant>>::const_iterator itA = mVerticalHeaderDataMap.constBegin();
174     QMap<int, QMap<int, QVariant>>::const_iterator itB = other->mVerticalHeaderDataMap.constBegin();
175     while (itA != mVerticalHeaderDataMap.constEnd()) {
176         if ((*itA).count() != (*itB).count()){
177             //qDebug() << "AttributesModel::compare() verticalHeaderDataMap/map have different sizes";
178             return false;
179         }
180         QMap<int, QVariant>::const_iterator it2A = (*itA).constBegin();
181         QMap<int, QVariant>::const_iterator it2B = (*itB).constBegin();
182         while (it2A != itA->constEnd()) {
183             if ( it2A.key() != it2B.key() ){
184                 //qDebug( "AttributesModel::compare()\n"
185                 //      "    verticalHeaderDataMap[ %i ] values have different types.  A: %x  B: %x",
186                 //      itA.key(), it2A.key(), it2B.key());
187                 return false;
188             }
189             if ( ! compareAttributes( it2A.key(), it2A.value(), it2B.value() ) ){
190                 //qDebug( "AttributesModel::compare()\n"
191                 //      "    verticalHeaderDataMap[ %i ] values are different. Role: %x", itA.key(), it2A.value());
192                 return false;
193             }
194             ++it2A;
195             ++it2B;
196         }
197         ++itA;
198         ++itB;
199     }
200 }
201 {
202     if (mModelDataMap.count() != other->mModelDataMap.count()){
203         //qDebug() << "AttributesModel::compare() modelDataMap have different sizes:" << mModelDataMap.count();
204         return false;
205     }
206     QMap<int, QVariant>::const_iterator itA = mModelDataMap.constBegin();
207     QMap<int, QVariant>::const_iterator itB = other->mModelDataMap.constBegin();
208     while (itA != mModelDataMap.constEnd()) {
209         if ( itA.key() != itB.key() ){
210             //qDebug( "AttributesModel::compare()\n"
211             //      "    modelDataMap values have different types.  A: %x  B: %x",
212             //      itA.key(), itB.key());
213             return false;
214         }
215         if ( ! compareAttributes( itA.key(), itA.value(), itB.value() ) ){
216             //qDebug( "AttributesModel::compare()\n"
217             //      "    modelDataMap values are different. Role: %x", itA.key() );
218             return false;
219         }
220         ++itA;
221         ++itB;
222     }
223 }
224 if (paletteType() != other->paletteType()){
225     //qDebug() << "AttributesModel::compare() palette types are different";
226     return false;
227 }
228 return true;
229 }

```

#### 9.50.4.4 bool AttributesModel::compareAttributes (int *role*, const QVariant & *a*, const QVariant & *b*) const [inherited]

Definition at line 231 of file KDChartAttributesModel.cpp.

References KDChart::BarAttributesRole, KDChart::DataHiddenRole, KDChart::DatasetBrushRole, KDChart::DatasetPenRole, KDChart::DataValueLabelAttributesRole, KDChart::Attributes-

Model::isKnownAttributesRole(), KDChart::LineAttributesRole, KDChart::PieAttributesRole, KDChart::ThreeDAttributesRole, KDChart::ThreeDBarAttributesRole, KDChart::ThreeDLineAttributesRole, KDChart::ThreeDPieAttributesRole, and KDChart::ValueTrackerAttributesRole.

Referenced by KDChart::AttributesModel::compare().

```

233 {
234     if( isKnownAttributesRole( role ) ){
235         switch( role ) {
236             case DataValueLabelAttributesRole:
237                 return (qVariantValue<DataValueAttributes>( a ) ==
238                     qVariantValue<DataValueAttributes>( b ));
239             case DatasetBrushRole:
240                 return (qVariantValue<QBrush>( a ) ==
241                     qVariantValue<QBrush>( b ));
242             case DatasetPenRole:
243                 return (qVariantValue<QPen>( a ) ==
244                     qVariantValue<QPen>( b ));
245             case ThreeDAttributesRole:
246                 // As of yet there is no ThreeDAttributes class,
247                 // and the AbstractThreeDAttributes class is pure virtual,
248                 // so we ignore this role for now.
249                 // (khz, 04.04.2007)
250                 /*
251                 return (qVariantValue<ThreeDAttributes>( a ) ==
252                     qVariantValue<ThreeDAttributes>( b ));
253                 */
254                 break;
255             case LineAttributesRole:
256                 return (qVariantValue<LineAttributes>( a ) ==
257                     qVariantValue<LineAttributes>( b ));
258             case ThreeDLineAttributesRole:
259                 return (qVariantValue<ThreeDLineAttributes>( a ) ==
260                     qVariantValue<ThreeDLineAttributes>( b ));
261             case BarAttributesRole:
262                 return (qVariantValue<BarAttributes>( a ) ==
263                     qVariantValue<BarAttributes>( b ));
264             case ThreeDBarAttributesRole:
265                 return (qVariantValue<ThreeDBarAttributes>( a ) ==
266                     qVariantValue<ThreeDBarAttributes>( b ));
267             case PieAttributesRole:
268                 return (qVariantValue<PieAttributes>( a ) ==
269                     qVariantValue<PieAttributes>( b ));
270             case ThreeDPieAttributesRole:
271                 return (qVariantValue<ThreeDPieAttributes>( a ) ==
272                     qVariantValue<ThreeDPieAttributes>( b ));
273             case ValueTrackerAttributesRole:
274                 return (qVariantValue<ValueTrackerAttributes>( a ) ==
275                     qVariantValue<ValueTrackerAttributes>( b ));
276             case DataHiddenRole:
277                 return (qVariantValue<bool>( a ) ==
278                     qVariantValue<bool>( b ));
279             default:
280                 Q_ASSERT( false ); // all of our own roles need to be handled
281                 break;
282         }
283     }else{
284         return (a == b);
285     }
286     return true;
287 }

```

#### 9.50.4.5 QVariant AttributesModel::data (const QModelIndex &, int role = Qt::DisplayRole) const [inherited]

[reimplemented]

Definition at line 373 of file KDChartAttributesModel.cpp.

References KDChart::AttributesModel::data(), KDChart::AttributesModel::dataMap(), and KDChart::AbstractProxyModel::mapToSource().

```

374 {
375     //qDebug() << "AttributesModel::data(" << index << role << ")";
376     if( index.isValid() ) {
377         Q_ASSERT( index.model() == this );
378     }
379
380     if( sourceModel() == 0 )
381         return QVariant();
382
383     if( index.isValid() )
384     {
385         const QVariant sourceData = sourceModel()->data( mapToSource(index), role );
386         if( sourceData.isValid() )
387             return sourceData;
388     }
389
390     // check if we are storing a value for this role at this cell index
391     if( mDataMap.contains( index.column() ) )
392     {
393         const QMap< int, QMap< int, QVariant > >& colDataMap = mDataMap[ index.column() ];
394         if( colDataMap.contains( index.row() ) )
395         {
396             const QMap< int, QVariant >& dataMap = colDataMap[ index.row() ];
397             if( dataMap.contains( role ) )
398             {
399                 const QVariant v = dataMap[ role ];
400                 if( v.isValid() )
401                     return v;
402             }
403         }
404     }
405     // check if there is something set for the column (dataset), or at global level
406     if( index.isValid() )
407         return data( index.column(), role ); // includes automatic fallback to default
408
409     return QVariant();
410 }
```

#### 9.50.4.6 QVariant AttributesModel::data (int column, int role) const [inherited]

Returns the data that were specified at per column level, or the globally set data, or the default data, or QVariant().

Definition at line 357 of file KDChartAttributesModel.cpp.

References KDChart::AttributesModel::data(), KDChart::AttributesModel::headerData(), and KDChart::AttributesModel::isKnownAttributesRole().

```

358 {
359     if ( isKnownAttributesRole( role ) ) {
360         // check if there is something set for the column (dataset)
361         QVariant v;
362         v = headerData( column, Qt::Vertical, role );
```

```

363
364     // check if there is something set at global level
365     if ( !v.isValid() )
366         v = data( role ); // includes automatic fallback to default
367     return v;
368 }
369 return QVariant();
370 }

```

#### 9.50.4.7 QVariant AttributesModel::data( int role ) const [inherited]

Returns the data that were specified at global level, or the default data, or QVariant().

Definition at line 339 of file KDChartAttributesModel.cpp.

References KDChart::AttributesModel::isKnownAttributesRole(), and KDChart::AttributesModel::modelData().

Referenced by KDChart::AbstractDiagram::brush(), KDChart::AttributesModel::data(), KDChart::AbstractDiagram::dataValueAttributes(), KDChart::AbstractDiagram::isHidden(), and KDChart::AbstractDiagram::pen().

```

340 {
341     if ( isKnownAttributesRole( role ) ) {
342         // check if there is something set at global level
343         QVariant v = modelData( role );
344
345         // else return the default setting, if any
346         if ( !v.isValid() )
347             v = defaultsForRole( role );
348         return v;
349     }
350     return QVariant();
351 }

```

#### 9.50.4.8 const QMap< int, QMap< int, QMap< int, QVariant > > > AttributesModel::dataMap() const [protected, inherited]

needed for serialization

Definition at line 564 of file KDChartAttributesModel.cpp.

Referenced by KDChart::AttributesModel::data(), KDChart::AttributesModel::headerData(), KDChart::AttributesModel::setData(), and KDChart::AttributesModel::setHeaderData().

```

565 {
566     return mDataMap;
567 }

```

#### 9.50.4.9 QVariant AttributesModel::headerData( int section, Qt::Orientation orientation, int role = Qt::DisplayRole ) const [inherited]

[reimplemented]

Definition at line 290 of file KDChartAttributesModel.cpp.

References KDChart::AttributesModel::dataMap(), KDChart::DatasetBrushRole, KDChart::DatasetPenRole, KDChart::Palette::defaultPalette(), KDChart::Palette::getBrush(), KDChart::AttributesModel::modelData(), KDChart::AttributesModel::paletteType(), KDChart::AttributesModel::PaletteTypeDefault, KDChart::AttributesModel::PaletteTypeRainbow, KDChart::AttributesModel::PaletteTypeSubdued, KDChart::Palette::rainbowPalette(), and KDChart::Palette::subduedPalette().

Referenced by KDChart::AttributesModel::data(), KDChart::AbstractDiagram::datasetLabels(), and KDChart::AttributesModel::setHeaderData().

```

293 {
294     QVariant sourceData = sourceModel()->headerData( section, orientation, role );
295     if ( sourceData.isValid() ) return sourceData;
296     // the source model didn't have data set, let's use our stored values
297     const QMap<int, QMap<int, QVariant> >& map = orientation == Qt::Horizontal ? mHorizontalHeaderDataMa
298     if ( map.contains( section ) ) {
299         const QMap<int, QVariant> &dataMap = map[ section ];
300         if ( dataMap.contains( role ) ) {
301             return dataMap[ role ];
302         }
303     }
304
305     // Default values if nothing else matches
306     switch ( role ) {
307     case Qt::DisplayRole:
308         return QLatin1String( orientation == Qt::Vertical ? "Series " : "Item " ) + QString::number( se
309
310     case KDChart::DatasetBrushRole: {
311         if ( paletteType() == PaletteTypeSubdued )
312             return Palette::subduedPalette().getBrush( section );
313         else if ( paletteType() == PaletteTypeRainbow )
314             return Palette::rainbowPalette().getBrush( section );
315         else if ( paletteType() == PaletteTypeDefault )
316             return Palette::defaultPalette().getBrush( section );
317         else
318             qWarning("Unknown type of fallback palette!");
319     }
320     case KDChart::DatasetPenRole: {
321         // default to the color set for the brush (or it's defaults)
322         // but only if no per model override was set
323         if ( !modelData( role ).isValid() ) {
324             QBrush brush = qVariantValue<QBrush>( headerData( section, orientation, DatasetBrushRole ) )
325             return QPen( brush.color() );
326         }
327     }
328     default:
329         break;
330     }
331
332     return QVariant();
333 }

```

#### 9.50.4.10 const QMap< int, QMap< int, QVariant > > AttributesModel::horizontalHeaderData-Map () const [protected, inherited]

needed for serialization

Definition at line 569 of file KDChartAttributesModel.cpp.

```

570 {
571     return mHorizontalHeaderDataMap;
572 }

```

#### 9.50.4.11 QModelIndex KDChart::AbstractProxyModel::index (int row, int col, const QModelIndex & index) const [inherited]

[reimplemented]

Definition at line 84 of file KDChartAbstractProxyModel.cpp.

References KDChart::AbstractProxyModel::mapFromSource(), and KDChart::AbstractProxyModel::mapToSource().

Referenced by KDChart::AttributesModel::setHeaderData(), and KDChart::AttributesModel::setModelData().

```
85 {
86     Q_ASSERT(sourceModel());
87     return mapFromSource(sourceModel()->index( row, col, mapToSource(index) ));
88 }
```

#### 9.50.4.12 void AttributesModel::initFrom (const AttributesModel \* other) [inherited]

Definition at line 70 of file KDChartAttributesModel.cpp.

References KDChart::AttributesModel::mDataMap, KDChart::AttributesModel::mDefaultsMap, KDChart::AttributesModel::mHorizontalHeaderDataMap, KDChart::AttributesModel::mModelDataMap, KDChart::AttributesModel::mVerticalHeaderDataMap, KDChart::AttributesModel::paletteType(), and KDChart::AttributesModel::setPaletteType().

Referenced by KDChart::AbstractDiagram::setModel().

```
71 {
72     if( other == this || ! other ) return;
73
74     mDataMap = other->mDataMap;
75     mHorizontalHeaderDataMap = other->mHorizontalHeaderDataMap;
76     mVerticalHeaderDataMap = other->mVerticalHeaderDataMap;
77     mModelDataMap = other->mModelDataMap;
78     mDefaultsMap = other->mDefaultsMap;
79
80     setPaletteType( other->paletteType() );
81 }
```

#### 9.50.4.13 bool AttributesModel::isKnownAttributesRole (int role) const [inherited]

Returns whether the given role corresponds to one of the known internally used ones.

Definition at line 413 of file KDChartAttributesModel.cpp.

References KDChart::BarAttributesRole, KDChart::DataHiddenRole, KDChart::DatasetBrushRole, KDChart::DatasetPenRole, KDChart::DataValueLabelAttributesRole, KDChart::LineAttributesRole, KDChart::PieAttributesRole, KDChart::ThreeDAttributesRole, KDChart::ThreeDBarAttributesRole, KDChart::ThreeDLineAttributesRole, KDChart::ThreeDPieAttributesRole, and KDChart::ValueTrackerAttributesRole.

Referenced by KDChart::AttributesModel::compareAttributes(), KDChart::AttributesModel::data(), KDChart::AttributesModel::setData(), and KDChart::AttributesModel::setHeaderData().

```
414 {
415     bool oneOfOurs = false;
```

```

416     switch( role ) {
417         // fallthrough intended
418         case DataValueLabelAttributesRole:
419         case DatasetBrushRole:
420         case DatasetPenRole:
421         case ThreeDAttributesRole:
422         case LineAttributesRole:
423         case ThreeDLineAttributesRole:
424         case BarAttributesRole:
425         case ThreeDBarAttributesRole:
426         case PieAttributesRole:
427         case ThreeDPieAttributesRole:
428         case ValueTrackerAttributesRole:
429         case DataHiddenRole:
430             oneOfOurs = true;
431         default:
432             break;
433     }
434     return oneOfOurs;
435 }

```

#### 9.50.4.14 QModelIndex KDChart::AbstractProxyModel::mapFromSource (const QModelIndex & sourceIndex) const [inherited]

[reimplemented]

Definition at line 52 of file KDChartAbstractProxyModel.cpp.

Referenced by KDChart::AbstractProxyModel::index(), and KDChart::AbstractProxyModel::parent().

```

53 {
54     if ( !sourceIndex.isValid() )
55         return QModelIndex();
56     //qDebug() << "sourceIndex.model()="<<sourceIndex.model();
57     //qDebug() << "model()="<<sourceModel();
58     Q_ASSERT( sourceIndex.model() == sourceModel() );
59
60     // Create an index that preserves the internal pointer from the source;
61     // this way AbstractProxyModel preserves the structure of the source model
62     return createIndex( sourceIndex.row(), sourceIndex.column(), sourceIndex.internalPointer() );
63 }

```

#### 9.50.4.15 QModelIndex KDChart::AbstractProxyModel::mapToSource (const QModelIndex & proxyIndex) const [inherited]

[reimplemented]

Definition at line 65 of file KDChartAbstractProxyModel.cpp.

Referenced by KDChart::AttributesModel::columnCount(), KDChart::AttributesModel::data(), KDChart::AbstractProxyModel::index(), KDChart::AbstractProxyModel::parent(), KDChart::AttributesModel::rowCount(), and KDChart::AttributesModel::setData().

```

66 {
67     if ( !proxyIndex.isValid() )
68         return QModelIndex();
69     if( proxyIndex.model() != this )
70         qDebug() << proxyIndex.model() << this;
71     Q_ASSERT( proxyIndex.model() == this );
72     // So here we need to create a source index which holds that internal pointer.

```

```

73 // No way to pass it to sourceModel()->index... so we have to do the ugly way:
74 QModelIndex sourceIndex;
75 KDPrivateModelIndex* hack = reinterpret_cast<KDPrivateModelIndex*>(&sourceIndex);
76 hack->r = proxyIndex.row();
77 hack->c = proxyIndex.column();
78 hack->p = proxyIndex.internalPointer();
79 hack->m = sourceModel();
80 Q_ASSERT( sourceIndex.isValid() );
81 return sourceIndex;
82 }

```

#### 9.50.4.16 QVariant KDChart::AttributesModel::modelData (int role) const [inherited]

Definition at line 509 of file KDChartAttributesModel.cpp.

Referenced by KDChart::AttributesModel::data(), KDChart::AbstractDiagram::dataValueAttributes(), KDChart::AttributesModel::headerData(), and KDChart::AbstractDiagram::isHidden().

```

510 {
511     return mModelDataMap.value( role, QVariant() );
512 }

```

#### 9.50.4.17 const QMap< int, QVariant > AttributesModel::modelDataMap () const [protected, inherited]

needed for serialization

Definition at line 579 of file KDChartAttributesModel.cpp.

```

580 {
581     return mModelDataMap;
582 }

```

#### 9.50.4.18 AttributesModel::PaletteType AttributesModel::paletteType () const [inherited]

Definition at line 493 of file KDChartAttributesModel.cpp.

Referenced by KDChart::AttributesModel::compare(), KDChart::AttributesModel::headerData(), and KDChart::AttributesModel::initFrom().

```

494 {
495     return mPaletteType;
496 }

```

#### 9.50.4.19 QModelIndex KDChart::AbstractProxyModel::parent (const QModelIndex & index) const [inherited]

[reimplemented]

Definition at line 90 of file KDChartAbstractProxyModel.cpp.

References KDChart::AbstractProxyModel::mapFromSource(), and KDChart::AbstractProxyModel::mapToSource().



```
91 {
92     Q_ASSERT(sourceModel());
93     return mapFromSource(sourceModel()->parent( mapToSource(index) ));
94 }
```

#### 9.50.4.20 bool AttributesModel::resetData (const QModelIndex & *index*, int *role* = Qt::DisplayRole) [inherited]

Remove any explicit attributes settings that might have been specified before.

Definition at line 457 of file KDChartAttributesModel.cpp.

References KDChart::AttributesModel::setData().

```
458 {
459     return setData ( index, QVariant(), role );
460 }
```

#### 9.50.4.21 bool AttributesModel::resetHeaderData (int *section*, Qt::Orientation *orientation*, int *role* = Qt::DisplayRole) [inherited]

Remove any explicit attributes settings that might have been specified before.

Definition at line 483 of file KDChartAttributesModel.cpp.

References KDChart::AttributesModel::setHeaderData().

```
484 {
485     return setHeaderData ( section, orientation, QVariant(), role );
486 }
```

#### 9.50.4.22 int AttributesModel::rowCount (const QModelIndex &) const [inherited]

[reimplemented]

Definition at line 514 of file KDChartAttributesModel.cpp.

References KDChart::AbstractProxyModel::mapToSource().

Referenced by KDChart::AbstractDiagram::itemRowLabels(), KDChart::AttributesModel::setHeaderData(), and KDChart::AttributesModel::setModelData().

```
515 {
516     if ( sourceModel() ) {
517         return sourceModel()->rowCount( mapToSource(index) );
518     } else {
519         return 0;
520     }
521 }
```

#### 9.50.4.23 bool AttributesModel::setData (const QModelIndex & *index*, const QVariant & *value*, int *role* = Qt::DisplayRole) [inherited]

[reimplemented]

Definition at line 443 of file KDChartAttributesModel.cpp.

References KDChart::AttributesModel::attributesChanged(), KDChart::AttributesModel::dataMap(), KDChart::AttributesModel::isKnownAttributesRole(), and KDChart::AbstractProxyModel::mapToSource().

Referenced by KDChart::AttributesModel::resetData(), KDChart::BarDiagram::setBarAttributes(), KDChart::AbstractDiagram::setBrush(), and KDChart::AbstractDiagram::setPen().

```

444 {
445     if ( !isKnownAttributesRole( role ) ) {
446         return sourceModel()->setData( mapToSource(index), value, role );
447     } else {
448         QMap< int, QMap< int, QVariant> > &colDataMap = mDataMap[ index.column() ];
449         QMap<int, QVariant> &dataMap = colDataMap[ index.row() ];
450         //qDebug() << "AttributesModel::setData" <<"role" << role << "value" << value;
451         dataMap.insert( role, value );
452         emit attributesChanged( index, index );
453         return true;
454     }
455 }
```

#### 9.50.4.24 void AttributesModel::setDataMap (const QMap< int, QMap< int, QMap< int, QVariant > > > *map*) [protected, inherited]

needed for serialization

Definition at line 585 of file KDChartAttributesModel.cpp.

```

586 {
587     mDataMap = map;
588 }
```

#### 9.50.4.25 void AttributesModel::setDefaultForRole (int *role*, const QVariant & *value*) [inherited]

Define the default value for a certain role.

Passing a default-constructed QVariant is equivalent to removing the default.

Definition at line 605 of file KDChartAttributesModel.cpp.

Referenced by KDChart::AttributesModel::AttributesModel(), and KDChart::TernaryLineDiagram::TernaryLineDiagram().

```

606 {
607     if ( value.isValid() ) {
608         mDefaultsMap.insert( role, value );
609     } else {
610         // erase the possibly existing value to not let the map grow:
611         QMap<int, QVariant>::iterator it = mDefaultsMap.find( role );
612         if ( it != mDefaultsMap.end() ) {
613             mDefaultsMap.erase( it );
614         }
615     }
616
617     Q_ASSERT( defaultsForRole( role ) == value );
618 }
```

#### 9.50.4.26 bool AttributesModel::setHeaderData (int *section*, Qt::Orientation *orientation*, const QVariant & *value*, int *role* = Qt::DisplayRole) [inherited]

[reimplemented]

Definition at line 462 of file KDChartAttributesModel.cpp.

References KDChart::AttributesModel::attributesChanged(), KDChart::AttributesModel::dataMap(), KDChart::AttributesModel::headerData(), KDChart::AbstractProxyModel::index(), KDChart::AttributesModel::isKnownAttributesRole(), and KDChart::AttributesModel::rowCount().

Referenced by KDChart::AttributesModel::resetHeaderData(), KDChart::AbstractDiagram::setBrush(), and KDChart::AbstractDiagram::setPen().

```

464 {
465     if( sourceModel() != 0 && headerData( section, orientation, role ) == value )
466         return true;
467     if ( !isKnownAttributesRole( role ) ) {
468         return sourceModel()->setHeaderData( section, orientation, value, role );
469     } else {
470         QMap<int, QMap<int, QVariant> > &sectionDataMap
471             = orientation == Qt::Horizontal ? mHorizontalHeaderDataMap : mVerticalHeaderDataMap;
472         QMap<int, QVariant> &dataMap = sectionDataMap[ section ];
473         dataMap.insert( role, value );
474         if( sourceModel() ){
475             emit attributesChanged( index( 0, section, QModelIndex() ),
476                                     index( rowCount( QModelIndex() ), section, QModelIndex() ) );
477             emit headerDataChanged( orientation, section, section );
478         }
479         return true;
480     }
481 }
```

#### 9.50.4.27 void AttributesModel::setHorizontalHeaderDataMap (const QMap< int, QMap< int, QVariant > > *map*) [protected, inherited]

needed for serialization

Definition at line 590 of file KDChartAttributesModel.cpp.

```

591 {
592     mHorizontalHeaderDataMap = map;
593 }
```

#### 9.50.4.28 bool KDChart::AttributesModel::setModelData (const QVariant *value*, int *role*) [inherited]

Definition at line 498 of file KDChartAttributesModel.cpp.

References KDChart::AttributesModel::attributesChanged(), KDChart::AttributesModel::columnCount(), KDChart::AbstractProxyModel::index(), and KDChart::AttributesModel::rowCount().

Referenced by KDChart::AbstractDiagram::setBrush(), and KDChart::AbstractDiagram::setPen().

```

499 {
500     mModelDataMap.insert( role, value );
501     if( sourceModel() ){
502         emit attributesChanged( index( 0, 0, QModelIndex() ),
```

```

503             index( rowCount( QModelIndex() ),
504                   columnCount( QModelIndex() ), QModelIndex() ) );
505     }
506     return true;
507 }

```

#### 9.50.4.29 void AttributesModel::setModelDataMap (const QMap< int, QVariant > *map*) [protected, inherited]

needed for serialization

Definition at line 600 of file KDChartAttributesModel.cpp.

```

601 {
602     mModelDataMap = map;
603 }

```

#### 9.50.4.30 void AttributesModel::setPaletteType ([PaletteType](#) *type*) [inherited]

Sets the palettetype used by this attributesmodel.

Definition at line 488 of file KDChartAttributesModel.cpp.

Referenced by KDChart::AttributesModel::initFrom().

```

489 {
490     mPaletteType = type;
491 }

```

#### 9.50.4.31 void AttributesModel::setSourceModel (QAbstractItemModel \* *sourceModel*) [inherited]

[reimplemented]

Definition at line 532 of file KDChartAttributesModel.cpp.

Referenced by KDChart::AttributesModel::AttributesModel().

```

533 {
534     if( this->sourceModel() != 0 )
535     {
536         disconnect( this->sourceModel(), SIGNAL( dataChanged( const QModelIndex&, const QModelIndex&))
537                  this, SIGNAL( dataChanged( const QModelIndex&, const QModelIndex&)) );
538         disconnect( this->sourceModel(), SIGNAL( rowsInserted( const QModelIndex&, int, int ) ),
539                  this, SIGNAL( rowsInserted( const QModelIndex&, int, int ) ) );
540         disconnect( this->sourceModel(), SIGNAL( rowsRemoved( const QModelIndex&, int, int ) ),
541                  this, SIGNAL( rowsRemoved( const QModelIndex&, int, int ) ) );
542         disconnect( this->sourceModel(), SIGNAL( columnsInserted( const QModelIndex&, int, int ) ),
543                  this, SIGNAL( columnsInserted( const QModelIndex&, int, int ) ) );
544         disconnect( this->sourceModel(), SIGNAL( columnsRemoved( const QModelIndex&, int, int ) ),
545                  this, SIGNAL( columnsRemoved( const QModelIndex&, int, int ) ) );
546     }
547     QAbstractProxyModel::setSourceModel( sourceModel );
548     if( this->sourceModel() != NULL )
549     {
550         connect( this->sourceModel(), SIGNAL( dataChanged( const QModelIndex&, const QModelIndex&)),
551                this, SIGNAL( dataChanged( const QModelIndex&, const QModelIndex&)) );

```

```

552         connect( this->sourceModel(), SIGNAL( rowsInserted( const QModelIndex&, int, int ) ),
553                 this, SIGNAL( rowsInserted( const QModelIndex&, int, int ) ) );
554         connect( this->sourceModel(), SIGNAL( rowsRemoved( const QModelIndex&, int, int ) ),
555                 this, SIGNAL( rowsRemoved( const QModelIndex&, int, int ) ) );
556         connect( this->sourceModel(), SIGNAL( columnsInserted( const QModelIndex&, int, int ) ),
557                 this, SIGNAL( columnsInserted( const QModelIndex&, int, int ) ) );
558         connect( this->sourceModel(), SIGNAL( columnsRemoved( const QModelIndex&, int, int ) ),
559                 this, SIGNAL( columnsRemoved( const QModelIndex&, int, int ) ) );
560     }
561 }

```

#### 9.50.4.32 void AttributesModel::setVerticalHeaderDataMap (const QMap< int, QMap< int, QVariant > > map) [protected, inherited]

needed for serialization

Definition at line 595 of file KDChartAttributesModel.cpp.

```

596 {
597     mVerticalHeaderDataMap = map;
598 }

```

#### 9.50.4.33 const QMap< int, QMap< int, QVariant > > AttributesModel::verticalHeaderDataMap () const [protected, inherited]

needed for serialization

Definition at line 574 of file KDChartAttributesModel.cpp.

```

575 {
576     return mVerticalHeaderDataMap;
577 }

```

The documentation for this class was generated from the following file:

- [KDChartAbstractDiagram.h](#)

## 9.51 KDChart::RelativePosition Class Reference

```
#include <KDChartRelativePosition.h>
```

### 9.51.1 Detailed Description

Defines relative position information: reference area, position in this area, horizontal / vertical padding, and rotating.

Using [RelativePosition](#) you can specify the relative parts of some position information, and you can specify the absolute parts: the reference area, and the position in this area.

#### Note:

To get an absolute position, you have three options:

- either you declare both, the relative and the absolute parts, using `setReferenceArea` for the later,
- or you specify a set of points, using `setReferencePoints`,
- or you refrain from using either, but leave it to KD [Chart](#) to find a matching reference area for you.

Definition at line 62 of file `KDChartRelativePosition.h`.

### Public Member Functions

- Qt::Alignment [alignment](#) () const
- const QPointF [calculatedPoint](#) (const QSizeF &autoSize) const  
*Calculate a point, according to the reference area/position and horiz/vert padding.*
- [Measure](#) [horizontalPadding](#) () const
- bool [operator!=](#) (const [RelativePosition](#) &other) const
- [RelativePosition](#) & [operator=](#) (const [RelativePosition](#) &other)
- bool [operator==](#) (const [RelativePosition](#) &) const
- QObject \* [referenceArea](#) () const
- const QPointF [referencePoint](#) () const  
*Return the reference point, according to the reference area/position, but ignoring horiz/vert padding.*
- const [PositionPoints](#) [referencePoints](#) () const
- [Position](#) [referencePosition](#) () const
- [RelativePosition](#) (const [RelativePosition](#) &)
- [RelativePosition](#) ()
- void [resetReferencePosition](#) ()  
*Resets the position of the anchor point to the built-in default.*
- qreal [rotation](#) () const
- void [setAlignment](#) (Qt::Alignment flags)  
*Specifies the location of the content, that is to be positioned by this [RelativePosition](#).*
- void [setHorizontalPadding](#) (const [Measure](#) &padding)  
*Specifies the horizontal width of the gap between the anchor point and the content, that is to be positioned by this [RelativePosition](#).*

- void [setReferenceArea](#) (QObject \*area)  
*Specifies the reference area to be used to find the anchor point.*
- void [setReferencePoints](#) (const [PositionPoints](#) &points)  
*Specifies a set of points from which the anchor point will be selected.*
- void [setReferencePosition](#) ([Position](#) position)  
*Specifies the position of the anchor point.*
- void [setRotation](#) (qreal rot)
- void [setVerticalPadding](#) (const [Measure](#) &padding)  
*Specifies the vertical width of the gap between the anchor point and the content, that is to be positioned by this [RelativePosition](#).*
- [Measure verticalPadding](#) () const
- [~RelativePosition](#) ()

## 9.51.2 Constructor & Destructor Documentation

### 9.51.2.1 RelativePosition::RelativePosition ()

Definition at line 70 of file KDChartRelativePosition.cpp.

```
71      : _d( new Private )
72  {
73
74  }
```

### 9.51.2.2 RelativePosition::RelativePosition (const [RelativePosition](#) &)

Definition at line 76 of file KDChartRelativePosition.cpp.

```
77      : _d( new Private( *r._d ) )
78  {
79
80  }
```

### 9.51.2.3 RelativePosition::~~RelativePosition ()

Definition at line 88 of file KDChartRelativePosition.cpp.

```
89  {
90      delete _d;
91  }
```

### 9.51.3 Member Function Documentation

#### 9.51.3.1 Qt::Alignment RelativePosition::alignment () const

Definition at line 130 of file KDChartRelativePosition.cpp.

References d.

Referenced by operator<<(), and operator==().

```

130                                     {
131     return d->alignment;
132 }
```

#### 9.51.3.2 const QPointF RelativePosition::calculatedPoint (const QSizeF & autoSize) const

Calculate a point, according to the reference area/position and horiz/vert padding.

This method is called at drawing time: The returned point is used as anchor point. Note that calculatedPoint ignores the alignment setting, it just returns the point, so the calling code needs to take alignment into account explicitly.

**See also:**

[referencePoint](#), [setReferenceArea](#), [setReferencePosition](#), [setHorizontalPadding](#), [setVerticalPadding](#)

Definition at line 185 of file KDChartRelativePosition.cpp.

References KDChart::Measure::calculatedValue(), horizontalPadding(), KDChartEnums::MeasureOrientationHorizontal, KDChartEnums::MeasureOrientationVertical, referencePoint(), and verticalPadding().

```

186 {
187     const QPointF pt( referencePoint() );
188     const qreal dx = horizontalPadding().calculatedValue( autoSize, KDChartEnums::MeasureOrientationHorizontal );
189     const qreal dy = verticalPadding().calculatedValue( autoSize, KDChartEnums::MeasureOrientationVertical );
190     //qDebug() << "rect.center() " << rect.center();
191     //qDebug() << "pt.x() " << pt.x() << " pt.y() " << pt.y();
192     return QPointF( pt.x() + dx, pt.y() + dy );
193 }
```

#### 9.51.3.3 Measure RelativePosition::horizontalPadding () const

Definition at line 138 of file KDChartRelativePosition.cpp.

References d.

Referenced by calculatedPoint(), operator<<(), and operator==().

```

138                                     {
139     return d->horizontalPadding;
140 }
```



**9.51.3.4 bool KDChart::RelativePosition::operator!=(const RelativePosition & other) const**

Definition at line 198 of file KDChartRelativePosition.h.

References operator==( ).

```
198 { return !operator==( other ); }
```

**9.51.3.5 RelativePosition & RelativePosition::operator=(const RelativePosition & other)**

Definition at line 82 of file KDChartRelativePosition.cpp.

```
82                                     {
83     RelativePosition copy( other );
84     copy.swap( *this );
85     return *this;
86 }
```

**9.51.3.6 bool RelativePosition::operator==(const RelativePosition &) const**

Definition at line 196 of file KDChartRelativePosition.cpp.

References alignment(), d, horizontalPadding(), referenceArea(), referencePosition(), rotation(), and verticalPadding().

Referenced by operator!=( ).

```
197 {
198     return d->area          == r.referenceArea() &&
199            d->position       == r.referencePosition() &&
200            d->alignment      == r.alignment() &&
201            d->horizontalPadding == r.horizontalPadding() &&
202            d->verticalPadding == r.verticalPadding() &&
203            d->rotation       == r.rotation() ;
204 }
```

**9.51.3.7 QObject \* RelativePosition::referenceArea () const**

Definition at line 101 of file KDChartRelativePosition.cpp.

References d.

Referenced by operator<<(), and operator==( ).

```
101                                     {
102     return d->area;
103 }
```

**9.51.3.8 const QPointF RelativePosition::referencePoint () const**

Return the reference point, according to the reference area/position, but ignoring horiz/vert padding.

This method is called at drawing time. The returned point is used to test if the label of a data value is to be printed: labels are printed only, if their reference points are either inside or touching the coordinate plane.

See also:

[calculatedPoint](#), [setReferenceArea](#), [setReferencePosition](#), [setHorizontalPadding](#), [setVerticalPadding](#)

Definition at line 159 of file KDChartRelativePosition.cpp.

References d.

Referenced by [calculatedPoint\(\)](#).

```

160 {
161     bool useRect = (d->area != 0);
162     QRect rect;
163     if( useRect ){
164         const QWidget* widget = dynamic_cast<const QWidget*>(d->area);
165         if( widget ){
166             const QLayout * layout = widget->layout();
167             rect = layout ? layout->geometry() : widget->geometry();
168         }else{
169             const AbstractArea* kdcArea = dynamic_cast<const AbstractArea*>(d->area);
170             if( kdcArea )
171                 rect = kdcArea->geometry();
172             else
173                 useRect = false;
174         }
175     }
176     QPointF pt;
177     if ( useRect )
178         pt = PositionPoints( rect ).point( d->position );
179     else
180         pt = d->points.point( d->position );
181     return pt;
182 }
```

#### 9.51.3.9 [const PositionPoints](#) RelativePosition::referencePoints () const

Definition at line 110 of file KDChartRelativePosition.cpp.

References d.

```

110                                     {
111     return d->points;
112 }
```

#### 9.51.3.10 [Position](#) RelativePosition::referencePosition () const

Definition at line 122 of file KDChartRelativePosition.cpp.

References d.

Referenced by [operator<<\(\)](#), and [operator==\(\)](#).

```

122                                     {
123     return d->position;
124 }
```

### 9.51.3.11 void RelativePosition::resetReferencePosition ()

Resets the position of the anchor point to the built-in default.

If the anchor point of a [RelativePosition](#) is reset (or never changed from the default setting, resp.) KD Chart will choose an appropriate [Position](#) at run-time.

e.g. BarDiagrams will use [Position::NorthWest](#) / [Position::SouthEast](#) for positive / negative values.

**See also:**

[setReferencePosition](#), [setReferenceArea](#), [setAlignment](#), [setHorizontalPadding](#), [setVerticalPadding](#), [KDChart::Position](#)

Definition at line 118 of file KDChartRelativePosition.cpp.

References [d](#), and [KDChart::Position::Unknown](#).

```
118                                     {
119     d->position = Position::Unknown;
120 }
```

### 9.51.3.12 qreal RelativePosition::rotation () const

Definition at line 154 of file KDChartRelativePosition.cpp.

References [d](#).

Referenced by [operator<<\(\)](#), and [operator==\(\)](#).

```
154                                     {
155     return d->rotation;
156 }
```

### 9.51.3.13 void RelativePosition::setAlignment (Qt::Alignment *flags*)

Specifies the location of the content, that is to be positioned by this [RelativePosition](#).

Aligning is applied, after horiz./vert. padding was retrieved to calculate the real reference point, so aligning is seen as relative to that point.

**See also:**

[setReferencePosition](#), [setReferenceArea](#), [setHorizontalPadding](#), [setVerticalPadding](#)

Definition at line 126 of file KDChartRelativePosition.cpp.

References [d](#).

```
126                                     {
127     d->alignment = align;
128 }
```

#### 9.51.3.14 void RelativePosition::setHorizontalPadding (const [Measure](#) & *padding*)

Specifies the horizontal width of the gap between the anchor point and the content, that is to be positioned by this [RelativePosition](#).

##### Note:

When printing data value texts the font height is used as reference size for both, horizontal and vertical padding, if the respective padding's [Measure](#) is using automatic reference area detection.

##### See also:

[setVerticalPadding](#), [setReferencePosition](#), [setReferenceArea](#)

Definition at line 134 of file `KDChartRelativePosition.cpp`.

References d.

```
134                                     {  
135     d->horizontalPadding = pad;  
136 }
```

#### 9.51.3.15 void RelativePosition::setReferenceArea ([QObject](#) \* *area*)

Specifies the reference area to be used to find the anchor point.

The reference area's type can be either [QWidget](#), or be derived from [KDChart::AbstractArea](#).

##### Note:

Usage of reference area and reference points works mutually exclusively: Only one setting can be valid, so any former specification of reference points is reset when you call `setReferenceArea`.

Also note: In a few cases [KD Chart](#) will ignore your area (or points, resp.) settings! Relative positioning of data value texts is an example: For these the reference area is the respective data area taking precedence over your settings.

##### See also:

[setReferencePosition](#), [setAlignment](#), [setHorizontalPadding](#), [setVerticalPadding](#)

Definition at line 95 of file `KDChartRelativePosition.cpp`.

References d, and `setReferencePoints()`.

Referenced by `setReferencePoints()`.

```
95                                     {  
96     d->area = area;  
97     if( area )  
98         setReferencePoints( PositionPoints() );  
99 }
```

**9.51.3.16 void RelativePosition::setReferencePoints (const [PositionPoints](#) & *points*)**

Specifies a set of points from which the anchor point will be selected.

**Note:**

Usage of reference area and reference points works mutually exclusively: Only one setting can be valid, so any former specification of reference area is reset when you call setReferencePoints.

Also note: In a few cases KD [Chart](#) will ignore your points (or area, resp.) settings! Relative positioning of data value texts is an example: For these the reference area is the respective data area taking precedence over your settings.

**See also:**

[setReferenceArea](#), [setReferencePosition](#), [setAlignment](#), [setHorizontalPadding](#), [setVerticalPadding](#)

Definition at line 105 of file KDChartRelativePosition.cpp.

References [d](#), [KDChart::PositionPoints::isNull\(\)](#), and [setReferenceArea\(\)](#).

Referenced by [setReferenceArea\(\)](#).

```
105                                     {
106     d->points = points;
107     if( !points.isNull() )
108         setReferenceArea( 0 );
109 }
```

**9.51.3.17 void RelativePosition::setReferencePosition ([Position](#) *position*)**

Specifies the position of the anchor point.

The anchor point of a [RelativePosition](#) may be one of the pre-defined points of it's reference area - for details see [KDChart::Position](#).

**See also:**

[resetReferencePosition](#), [setReferenceArea](#), [setAlignment](#), [setHorizontalPadding](#), [setVerticalPadding](#), [KDChart::Position](#)

Definition at line 114 of file KDChartRelativePosition.cpp.

References [d](#).

```
114                                     {
115     d->position = pos;
116 }
```

**9.51.3.18 void RelativePosition::setRotation (qreal *rot*)**

Definition at line 150 of file KDChartRelativePosition.cpp.

References [d](#).

```
150                                     {
151     d->rotation = rot;
152 }
```

**9.51.3.19 void RelativePosition::setVerticalPadding (const [Measure](#) & *padding*)**

Specifies the vertical width of the gap between the anchor point and the content, that is to be positioned by this [RelativePosition](#).

**Note:**

When printing data value texts the font height is used as reference size for both, horizontal and vertical padding, if the respective padding's [Measure](#) is using automatic reference area detection.

**See also:**

[setHorizontalPadding](#), [setReferencePosition](#), [setReferenceArea](#)

Definition at line 142 of file `KDChartRelativePosition.cpp`.

References d.

```
142                                     {
143     d->verticalPadding = pad;
144 }
```

**9.51.3.20 [Measure](#) RelativePosition::verticalPadding () const**

Definition at line 146 of file `KDChartRelativePosition.cpp`.

References d.

Referenced by `calculatedPoint()`, `operator<<()`, and `operator==()`.

```
146                                     {
147     return d->verticalPadding;
148 }
```

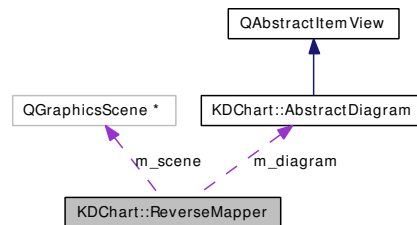
The documentation for this class was generated from the following files:

- [KDChartRelativePosition.h](#)
- [KDChartRelativePosition.cpp](#)

## 9.52 KDChart::ReverseMapper Class Reference

```
#include <ReverseMapper.h>
```

Collaboration diagram for KDChart::ReverseMapper:



### 9.52.1 Detailed Description

The [ReverseMapper](#) stores information about objects on a chart and their respective model indexes.

Definition at line 47 of file ReverseMapper.h.

### Public Member Functions

- void [addCircle](#) (int row, int column, const QPointF &location, const QSizeF &diameter)
- void [addItem](#) ([ChartGraphicsItem](#) \*item)
- void [addLine](#) (int row, int column, const QPointF &from, const QPointF &to)
- void [addPolygon](#) (int row, int column, const QPolygonF &polygon)
- void [addRect](#) (int row, int column, const QRectF &rect)
- void [clear](#) ()
- QModelIndexList [indexesAt](#) (const QPointF &point) const
- QModelIndexList [indexesIn](#) (const QRect &rect) const
- [ReverseMapper](#) ([AbstractDiagram](#) \*diagram)
- [ReverseMapper](#) ()
- void [setDiagram](#) ([AbstractDiagram](#) \*diagram)
- [~ReverseMapper](#) ()

### 9.52.2 Constructor & Destructor Documentation

#### 9.52.2.1 ReverseMapper::ReverseMapper ()

Definition at line 44 of file ReverseMapper.cpp.

```

45     : m_scene( 0 )
46     , m_diagram( 0 )
47 {
48 }
```

### 9.52.2.2 ReverseMapper::ReverseMapper ([AbstractDiagram](#) \* *diagram*) [explicit]

Definition at line 50 of file ReverseMapper.cpp.

```
51     : m_scene( 0 )
52     , m_diagram( diagram )
53 {
54 }
```

### 9.52.2.3 ReverseMapper::~ReverseMapper ()

Definition at line 56 of file ReverseMapper.cpp.

```
57 {
58     delete m_scene; m_scene = 0;
59 }
```

## 9.52.3 Member Function Documentation

### 9.52.3.1 void ReverseMapper::addCircle (int *row*, int *column*, const QPointF & *location*, const QSizeF & *diameter*)

Definition at line 129 of file ReverseMapper.cpp.

References addPolygon().

```
130 {
131     QPainterPath path;
132     QPointF ossfet( -0.5*diameter.width(), -0.5*diameter.height() );
133     path.addEllipse( QRectF( location + ossfet, diameter ) );
134     addPolygon( row, column, QPolygonF( path.toFillPolygon() ) );
135 }
```

### 9.52.3.2 void ReverseMapper::addItem ([ChartGraphicsItem](#) \* *item*)

Definition at line 111 of file ReverseMapper.cpp.

Referenced by addPolygon().

```
112 {
113     Q_ASSERT( m_scene );
114     m_scene->addItem( item );
115 }
```

### 9.52.3.3 void ReverseMapper::addLine (int *row*, int *column*, const QPointF & *from*, const QPointF & *to*)

Definition at line 137 of file ReverseMapper.cpp.

References addPolygon().



```

138 {
139     // lines do not make good polygons to click on. we calculate a 2
140     // pixel wide rectangle, where the original line is exactly
141     // centered in.
142     // make a 3 pixel wide polygon from the line:
143     static const QPointF pixel( 1.0, 1.0 );
144     QPointF left, right;
145     if ( from.x() < to.x() ) {
146         left = from;
147         right = to;
148     } else {
149         right = from;
150         left = to;
151     }
152     const QPointF lineVector( right - left );
153     const qreal lineVectorLength = sqrt( lineVector.x() * lineVector.x() + lineVector.y() * lineVector.y() );
154     const QPointF lineVectorUnit( lineVector / lineVectorLength );
155     const QPointF normOfLineVectorUnit( -lineVectorUnit.y(), lineVectorUnit.x() );
156     // now the four polygon end points:
157     const QPointF one( left - lineVectorUnit + normOfLineVectorUnit );
158     const QPointF two( left + lineVectorUnit - normOfLineVectorUnit );
159     const QPointF three( right + lineVectorUnit - normOfLineVectorUnit );
160     const QPointF four( right + lineVectorUnit + normOfLineVectorUnit );
161     addPolygon( row, column, QPolygonF() << one << two << three << four );
162 }

```

#### 9.52.3.4 void ReverseMapper::addPolygon (int row, int column, const QPolygonF & polygon)

Definition at line 122 of file ReverseMapper.cpp.

References addItem().

Referenced by addCircle(), addLine(), and addRect().

```

123 {
124     ChartGraphicsItem* item = new ChartGraphicsItem( row, column );
125     item->setPolygon( polygon );
126     addItem( item );
127 }

```

#### 9.52.3.5 void ReverseMapper::addRect (int row, int column, const QRectF & rect)

Definition at line 117 of file ReverseMapper.cpp.

References addPolygon().

```

118 {
119     addPolygon( row, column, QPolygonF( rect ) );
120 }

```

#### 9.52.3.6 void ReverseMapper::clear ()

Definition at line 67 of file ReverseMapper.cpp.

```

68 {
69     delete m_scene;
70     m_scene = new QGraphicsScene();
71 }

```

### 9.52.3.7 QModelIndexList ReverseMapper::indexesAt (const QPointF & *point*) const

Definition at line 92 of file ReverseMapper.cpp.

References KDChart::ChartGraphicsItem::column(), and KDChart::ChartGraphicsItem::row().

```

93 {
94     Q_ASSERT( m_diagram );
95     if ( m_scene && m_scene->sceneRect().contains( point ) ) {
96         QList<QGraphicsItem *> items = m_scene->items( point );
97         QModelIndexList indexes;
98         Q_FOREACH( QGraphicsItem* item, items ) {
99             ChartGraphicsItem* i = qgraphicsitem_cast<ChartGraphicsItem*>( item );
100             if ( i ) {
101                 QModelIndex index ( m_diagram->model()->index( i->row(), i->column() ) );
102                 indexes << index;
103             }
104         }
105         return indexes;
106     } else {
107         return QModelIndexList();
108     }
109 }
```

### 9.52.3.8 QModelIndexList ReverseMapper::indexesIn (const QRect & *rect*) const

Definition at line 73 of file ReverseMapper.cpp.

References KDChart::ChartGraphicsItem::column(), and KDChart::ChartGraphicsItem::row().

```

74 {
75     Q_ASSERT( m_diagram );
76     if ( m_scene && m_scene->sceneRect().intersects( rect ) ) {
77         QList<QGraphicsItem *> items = m_scene->items( rect );
78         QModelIndexList indexes;
79         Q_FOREACH( QGraphicsItem* item, items ) {
80             ChartGraphicsItem* i = qgraphicsitem_cast<ChartGraphicsItem*>( item );
81             if ( i ) {
82                 QModelIndex index ( m_diagram->model()->index( i->row(), i->column() ) );
83                 indexes << index;
84             }
85         }
86         return indexes;
87     } else {
88         return QModelIndexList();
89     }
90 }
```

### 9.52.3.9 void ReverseMapper::setDiagram (AbstractDiagram \* *diagram*)

Definition at line 61 of file ReverseMapper.cpp.

```

62 {
63
64     m_diagram = diagram;
65 }
```

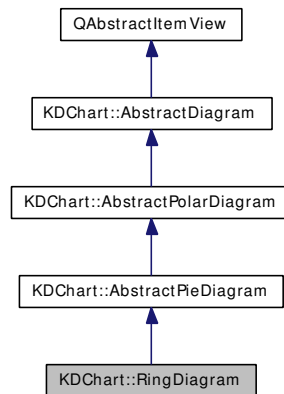
The documentation for this class was generated from the following files:

- [ReverseMapper.h](#)
- [ReverseMapper.cpp](#)

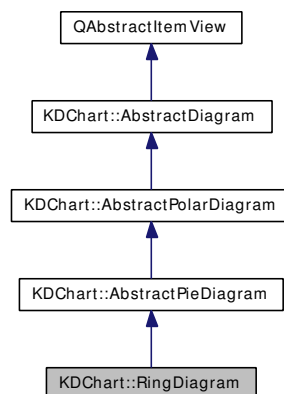
## 9.53 KDChart::RingDiagram Class Reference

```
#include <KDChartRingDiagram.h>
```

Inheritance diagram for KDChart::RingDiagram:



Collaboration diagram for KDChart::RingDiagram:



### 9.53.1 Detailed Description

[RingDiagram](#) defines a common ring diagram.

Definition at line 40 of file KDChartRingDiagram.h.

#### Signals

- void [dataHidden](#) ()  
*This signal is emitted, when the hidden status of at least one data cell was (un)set.*
- void [layoutChanged](#) ([AbstractDiagram](#) \*)  
*Diagrams are supposed to emit this signal, when the layout of one of their element changes.*
- void [modelsChanged](#) ()

*This signal is emitted, when either the model or the [AttributesModel](#) is replaced.*

- void [propertiesChanged](#) ()

*Emitted upon change of a property of the Diagram.*

## Public Member Functions

- bool [allowOverlappingDataValueTexts](#) () const

**Returns:**

*Whether data value labels are allowed to overlap.*

- bool [antiAliasing](#) () const

**Returns:**

*Whether anti-aliasing is to be used for rendering this diagram.*

- virtual [AttributesModel](#) \* [attributesModel](#) () const

*Returns the [AttributesModel](#), that is used by this diagram.*

- QBrush [brush](#) (const QModelIndex &index) const

*Retrieve the brush to be used, for painting the datapoint at the given index in the model.*

- QBrush [brush](#) (int dataset) const

*Retrieve the brush to be used for the given dataset.*

- QBrush [brush](#) () const

*Retrieve the brush to be used for painting datapoints globally.*

- virtual [RingDiagram](#) \* [clone](#) () const

*Creates an exact copy of this diagram.*

- int [columnCount](#) () const

- bool [compare](#) (const [AbstractDiagram](#) \*other) const

*Returns true if both diagrams have the same settings.*

- [AbstractCoordinatePlane](#) \* [coordinatePlane](#) () const

*The coordinate plane associated with the diagram.*

- const QPair< QPointF, QPointF > [dataBoundaries](#) () const

*Return the bottom left and top right data point, that the diagram will display (unless the grid adjusts these values).*

- virtual void [dataChanged](#) (const QModelIndex &topLeft, const QModelIndex &bottomRight)

*[reimplemented]*

- QList< QBrush > [datasetBrushes](#) () const

*The set of dataset brushes currently used, for use in legends, etc.*

- int [datasetDimension](#) () const

*The dataset dimension of a diagram determines, how many value dimensions it expects each datapoint to have.*

- QStringList [datasetLabels](#) () const  
*The set of dataset labels currently displayed, for use in legends, etc.*
- QList< [MarkerAttributes](#) > [datasetMarkers](#) () const  
*The set of dataset markers currently used, for use in legends, etc.*
- QList< QPen > [datasetPens](#) () const  
*The set of dataset pens currently used, for use in legends, etc.*
- [DataValueAttributes](#) [dataValueAttributes](#) (const QModelIndex &index) const  
*Retrieve the [DataValueAttributes](#) for the given index.*
- [DataValueAttributes](#) [dataValueAttributes](#) (int column) const  
*Retrieve the [DataValueAttributes](#) for the given dataset.*
- [DataValueAttributes](#) [dataValueAttributes](#) () const  
*Retrieve the [DataValueAttributes](#) specified globally.*
- virtual void [doItemsLayout](#) ()  
*[reimplemented]*
- qreal [granularity](#) () const  
**Returns:**  
*the granularity.*
- virtual int [horizontalOffset](#) () const  
*[reimplemented]*
- virtual QModelIndex [indexAt](#) (const QPoint &point) const  
*[reimplemented]*
- QModelIndexList [indexesAt](#) (const QPoint &point) const  
*This method is added alongside with [indexAt](#) from QAIM, since in kdchart multiple indexes can be displayed at the same spot.*
- bool [isHidden](#) (const QModelIndex &index) const  
*Retrieve the hidden status for the given index.*
- bool [isHidden](#) (int column) const  
*Retrieve the hidden status for the given dataset.*
- bool [isHidden](#) () const  
*Retrieve the hidden status specified globally.*
- virtual bool [isIndexHidden](#) (const QModelIndex &index) const  
*[reimplemented]*

- QStringList [itemRowLabels](#) () const  
*The set of item row labels currently displayed, for use in Abscissa axes, etc.*
- virtual QModelIndex [moveCursor](#) (CursorAction cursorAction, Qt::KeyboardModifiers modifiers)  
*[reimplemented]*
- virtual double [numberOfGridRings](#) () const  
*[reimplemented]*
- virtual double [numberOfValuesPerDataset](#) () const  
*[reimplemented]*
- void [paintDataValueText](#) (QPainter \*painter, const QModelIndex &index, const QPointF &pos, double value)
- void [paintMarker](#) (QPainter \*painter, const QModelIndex &index, const QPointF &pos)
- virtual void [paintMarker](#) (QPainter \*painter, const [MarkerAttributes](#) &markerAttributes, const QBrush &brush, const QPen &, const QPointF &point, const QSizeF &size)
- QPen [pen](#) (const QModelIndex &index) const  
*Retrieve the pen to be used, for painting the datapoint at the given index in the model.*
- QPen [pen](#) (int dataset) const  
*Retrieve the pen to be used for the given dataset.*
- QPen [pen](#) () const  
*Retrieve the pen to be used for painting datapoints globally.*
- bool [percentMode](#) () const
- [PieAttributes](#) [pieAttributes](#) (const QModelIndex &index) const
- [PieAttributes](#) [pieAttributes](#) (int column) const
- [PieAttributes](#) [pieAttributes](#) () const
- const [PolarCoordinatePlane](#) \* [polarCoordinatePlane](#) () const
- bool [relativeThickness](#) () const
- virtual void [resize](#) (const QSizeF &area)  
*[reimplemented]*
- [RingDiagram](#) (QWidget \*parent=0, [PolarCoordinatePlane](#) \*plane=0)
- virtual void [scrollTo](#) (const QModelIndex &index, ScrollHint hint=EnsureVisible)  
*[reimplemented]*
- void [setAllowOverlappingDataValueTexts](#) (bool allow)  
*Set whether data value labels are allowed to overlap.*
- void [setAntiAliasing](#) (bool enabled)  
*Set whether anti-aliasing is to be used while rendering this diagram.*
- virtual void [setAttributesModel](#) ([AttributesModel](#) \*model)  
*Associate an [AttributesModel](#) with this diagram.*
- void [setBrush](#) (const QBrush &brush)  
*Set the brush to be used, for painting all datasets in the model.*

- void [setBrush](#) (int dataset, const QBrush &brush)  
*Set the brush to be used, for painting the given dataset.*
- void [setBrush](#) (const QModelIndex &index, const QBrush &brush)  
*Set the brush to be used, for painting the datapoint at the given index.*
- virtual void [setCoordinatePlane](#) ([AbstractCoordinatePlane](#) \*plane)  
*Set the coordinate plane associated with the diagram.*
- void [setDatasetDimension](#) (int dimension)  
*Sets the dataset dimension of the diagram.*
- void [setDataValueAttributes](#) (const [DataValueAttributes](#) &a)  
*Set the [DataValueAttributes](#) for all datapoints in the model.*
- void [setDataValueAttributes](#) (int dataset, const [DataValueAttributes](#) &a)  
*Set the [DataValueAttributes](#) for the given dataset.*
- void [setDataValueAttributes](#) (const QModelIndex &index, const [DataValueAttributes](#) &a)  
*Set the [DataValueAttributes](#) for the given index.*
- void [setGranularity](#) (qreal value)  
*Set the granularity: the smaller the granularity the more your diagram segments will show facettes instead of rounded segments.*
- void [setHidden](#) (bool hidden)  
*Hide (or unhide, resp.*
- void [setHidden](#) (int column, bool hidden)  
*Hide (or unhide, resp.*
- void [setHidden](#) (const QModelIndex &index, bool hidden)  
*Hide (or unhide, resp.*
- virtual void [setModel](#) (QAbstractItemModel \*model)  
*Associate a model with the diagram.*
- void [setPen](#) (const QPen &pen)  
*Set the pen to be used, for painting all datasets in the model.*
- void [setPen](#) (int dataset, const QPen &pen)  
*Set the pen to be used, for painting the given dataset.*
- void [setPen](#) (const QModelIndex &index, const QPen &pen)  
*Set the pen to be used, for painting the datapoint at the given index.*
- void [setPercentMode](#) (bool percent)
- void [setPieAttributes](#) (int column, const [PieAttributes](#) &a)
- void [setPieAttributes](#) (const [PieAttributes](#) &a)



- void [setRelativeThickness](#) (bool relativeThickness)
- virtual void [setRootIndex](#) (const QModelIndex &idx)  
*Set the root index in the model, where the diagram starts referencing data for display.*
- virtual void [setSelection](#) (const QRect &rect, QItemSelectionModel::SelectionFlags command)  
*[reimplemented]*
- void [setStartPosition](#) (int degrees)
- void [setThreeDPieAttributes](#) (const QModelIndex &index, const [ThreeDPieAttributes](#) &a)
- void [setThreeDPieAttributes](#) (int column, const [ThreeDPieAttributes](#) &a)
- void [setThreeDPieAttributes](#) (const [ThreeDPieAttributes](#) &a)
- void [setUnitPrefix](#) (const QString &prefix, Qt::Orientation orientation)  
*Sets the unit prefix for all values.*
- void [setUnitPrefix](#) (const QString &prefix, int column, Qt::Orientation orientation)  
*Sets the unit prefix for one value.*
- void [setUnitSuffix](#) (const QString &suffix, Qt::Orientation orientation)  
*Sets the unit suffix for all values.*
- void [setUnitSuffix](#) (const QString &suffix, int column, Qt::Orientation orientation)  
*Sets the unit suffix for one value.*
- int [startPosition](#) () const
- [ThreeDPieAttributes](#) [threeDPieAttributes](#) (const QModelIndex &index) const
- [ThreeDPieAttributes](#) [threeDPieAttributes](#) (int column) const
- [ThreeDPieAttributes](#) [threeDPieAttributes](#) () const
- QString [unitPrefix](#) (Qt::Orientation orientation) const  
*Returns the global unit prefix.*
- QString [unitPrefix](#) (int column, Qt::Orientation orientation, bool fallback=false) const  
*Returns the unit prefix for a special value.*
- QString [unitSuffix](#) (Qt::Orientation orientation) const  
*Returns the global unit suffix.*
- QString [unitSuffix](#) (int column, Qt::Orientation orientation, bool fallback=false) const  
*Returns the unit suffix for a special value.*
- void [update](#) () const
- void [useDefaultColors](#) ()  
*Set the palette to be used, for painting datasets to the default palette.*
- void [useRainbowColors](#) ()  
*Set the palette to be used, for painting datasets to the rainbow palette.*
- virtual bool [usesExternalAttributesModel](#) () const  
*Returns whether the diagram is using its own built-in attributes model or an attributes model that was set via [setAttributesModel](#).*

- void [useSubduedColors](#) ()  
*Set the palette to be used, for painting datasets to the subdued palette.*
- virtual double [valueTotals](#) () const  
*[reimplemented]*
- virtual int [verticalOffset](#) () const  
*[reimplemented]*
- virtual QRect [visualRect](#) (const QModelIndex &index) const  
*[reimplemented]*
- virtual QRegion [visualRegionForSelection](#) (const QItemSelection &selection) const  
*[reimplemented]*
- virtual [~RingDiagram](#) ()

## Protected Member Functions

- QModelIndex [attributesModelRootIndex](#) () const
- virtual const QPair< QPointF, QPointF > [calculateDataBoundaries](#) () const  
*[reimplemented]*
- virtual bool [checkInvariants](#) (bool justReturnTheStatus=false) const
- QModelIndex [columnToIndex](#) (int column) const
- virtual void [paint](#) (PaintContext \*paintContext)  
*[reimplemented]*
- virtual void [paintDataValueTexts](#) (QPainter \*painter)
- void [paintEvent](#) (QPaintEvent \*)
- virtual void [paintMarkers](#) (QPainter \*painter)
- void [resizeEvent](#) (QResizeEvent \*)
- void [setAttributesModelRootIndex](#) (const QModelIndex &)
- void [setDataBoundariesDirty](#) () const
- double [valueForCell](#) (int row, int column) const  
*Helper method, retrieving the data value (DisplayRole) for a given row and column.*

## 9.53.2 Constructor & Destructor Documentation

### 9.53.2.1 RingDiagram::RingDiagram (QWidget \*parent = 0, PolarCoordinatePlane \*plane = 0) *[explicit]*

Definition at line 50 of file KDChartRingDiagram.cpp.

Referenced by clone().

```

50                                     :
51     AbstractPieDiagram( new Private(), parent, plane )
52 {
53     init();
54 }
```

### 9.53.2.2 RingDiagram::~~RingDiagram () [virtual]

Definition at line 56 of file KDChartRingDiagram.cpp.

```
57 {  
58 }
```

## 9.53.3 Member Function Documentation

### 9.53.3.1 bool AbstractDiagram::allowOverlappingDataValueTexts () const [inherited]

#### Returns:

Whether data value labels are allowed to overlap.

Definition at line 440 of file KDChartAbstractDiagram.cpp.

References [d](#).

Referenced by [KDChart::AbstractDiagram::compare\(\)](#).

```
441 {  
442     return d->allowOverlappingDataValueTexts;  
443 }
```

### 9.53.3.2 bool AbstractDiagram::antiAliasing () const [inherited]

#### Returns:

Whether anti-aliasing is to be used for rendering this diagram.

Definition at line 451 of file KDChartAbstractDiagram.cpp.

References [d](#).

Referenced by [KDChart::AbstractDiagram::compare\(\)](#).

```
452 {  
453     return d->antiAliasing;  
454 }
```

### 9.53.3.3 [AttributesModel](#) \* AbstractDiagram::attributesModel () const [virtual, inherited]

Returns the [AttributesModel](#), that is used by this diagram.

By default each diagram owns its own [AttributesModel](#), which should never be deleted. Only if a user-supplied [AttributesModel](#) has been set does the pointer returned here not belong to the diagram.

#### Returns:

The [AttributesModel](#) associated with the diagram.

#### See also:

[setAttributesModel](#)

Definition at line 280 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::brush(), KDChart::AbstractDiagram::compare(), KDChart::AbstractDiagram::datasetBrushes(), KDChart::AbstractDiagram::datasetLabels(), KDChart::AbstractDiagram::datasetMarkers(), KDChart::AbstractDiagram::datasetPens(), KDChart::AbstractDiagram::dataValueAttributes(), KDChart::AbstractDiagram::isHidden(), KDChart::AbstractDiagram::itemRowLabels(), paint(), KDChart::PolarDiagram::paint(), KDChart::AbstractDiagram::pen(), KDChart::AbstractCartesianDiagram::setAttributesModel(), KDChart::BarDiagram::setBarAttributes(), KDChart::AbstractDiagram::setBrush(), KDChart::AbstractCartesianDiagram::setModel(), KDChart::AbstractDiagram::setPen(), and KDChart::TernaryLineDiagram::TernaryLineDiagram().

```
281 {
282     return d->attributesModel;
283 }
```

#### 9.53.3.4 QModelIndex AbstractDiagram::attributesModelRootIndex () const [protected, inherited]

returns a QModelIndex pointing into the [AttributesModel](#) that corresponds to the root index of the diagram.

Definition at line 302 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::datasetBrushes(), KDChart::AbstractDiagram::datasetLabels(), KDChart::AbstractDiagram::datasetMarkers(), KDChart::AbstractDiagram::datasetPens(), KDChart::AbstractDiagram::itemRowLabels(), KDChart::Plotter::numberOfAbscissaSegments(), KDChart::LineDiagram::numberOfAbscissaSegments(), KDChart::BarDiagram::numberOfAbscissaSegments(), KDChart::Plotter::numberOfOrdinateSegments(), KDChart::LineDiagram::numberOfOrdinateSegments(), KDChart::BarDiagram::numberOfOrdinateSegments(), KDChart::AbstractDiagram::valueForCell(), and KDChart::LineDiagram::valueForCellTesting().

```
303 {
304     if ( !d->attributesModelRootIndex.isValid() )
305         d->attributesModelRootIndex = d->attributesModel->mapFromSource( rootIndex() );
306     return d->attributesModelRootIndex;
307 }
```

#### 9.53.3.5 QBrush AbstractDiagram::brush (const QModelIndex & index) const [inherited]

Retrieve the brush to be used, for painting the datapoint at the given index in the model.

##### Parameters:

*index* The index of the datapoint in the model.

##### Returns:

The brush to use for painting.

Definition at line 838 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::brush(), KDChart::AttributesModel::data(), KDChart::DatasetBrushRole, and KDChart::AbstractDiagram::datasetDimension().

```

839 {
840     if( datasetDimension() > 1 )
841         return brush( index.column() );
842     return QVariantValue<QBrush>(
843         attributesModel()->data(
844             attributesModel()->mapFromSource( index ),
845             DatasetBrushRole ) );
846 }
```

### 9.53.3.6 QBrush AbstractDiagram::brush (int *dataset*) const [inherited]

Retrieve the brush to be used for the given dataset.

This will fall back automatically to what was set at model level, if there are no dataset specific settings.

#### Parameters:

*dataset* The dataset to retrieve the brush for.

#### Returns:

The brush to use for painting.

Definition at line 828 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::columnToIndex(), KDChart::AttributesModel::data(), KDChart::DatasetBrushRole, and KDChart::AbstractDiagram::datasetDimension().

```

829 {
830     if( datasetDimension() > 1 )
831         dataset /= datasetDimension();
832     return QVariantValue<QBrush>(
833         attributesModel()->data(
834             attributesModel()->mapFromSource( columnToIndex( dataset ) ),
835             DatasetBrushRole ) );
836 }
```

### 9.53.3.7 QBrush AbstractDiagram::brush () const [inherited]

Retrieve the brush to be used for painting datapoints globally.

This will fall back automatically to the default settings, if there are no specific settings.

#### Returns:

The brush to use for painting.

Definition at line 822 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AttributesModel::data(), and KDChart::DatasetBrushRole.

Referenced by KDChart::AbstractDiagram::brush(), KDChart::TernaryPointDiagram::paint(), KDChart::TernaryLineDiagram::paint(), paint(), KDChart::PolarDiagram::paint(), and KDChart::AbstractDiagram::paintMarker().

```

823 {
824     return qVariantValue<QBrush>(
825         attributesModel()->data( DatasetBrushRole ) );
826 }

```

#### 9.53.3.8 **const QPair< QPointF, QPointF > RingDiagram::calculateDataBoundaries () const** [protected, virtual]

[reimplemented]

Implements [KDChart::AbstractDiagram](#).

Definition at line 82 of file KDChartRingDiagram.cpp.

References [KDChart::AbstractDiagram::checkInvariants\(\)](#).

```

83 {
84     if ( !checkInvariants(true) ) return QPair<QPointF, QPointF>( QPointF( 0, 0 ), QPointF( 0, 0 ) );
85
86     QPointF bottomLeft ( QPointF( 0, 0 ) );
87     QPointF topRight ( QPointF( 1, 1 ) );
88     return QPair<QPointF, QPointF> ( bottomLeft, topRight );
89 }

```

#### 9.53.3.9 **bool AbstractDiagram::checkInvariants (bool *justReturnTheStatus* = false) const** [protected, virtual, inherited]

Definition at line 1060 of file KDChartAbstractDiagram.cpp.

References [KDChart::AbstractDiagram::coordinatePlane\(\)](#).

Referenced by [calculateDataBoundaries\(\)](#), [KDChart::PolarDiagram::calculateDataBoundaries\(\)](#), [KDChart::Plotter::calculateDataBoundaries\(\)](#), [KDChart::PieDiagram::calculateDataBoundaries\(\)](#), [KDChart::LineDiagram::calculateDataBoundaries\(\)](#), [KDChart::BarDiagram::calculateDataBoundaries\(\)](#), [paint\(\)](#), [KDChart::PolarDiagram::paint\(\)](#), [KDChart::Plotter::paint\(\)](#), [KDChart::PieDiagram::paint\(\)](#), [KDChart::LineDiagram::paint\(\)](#), [KDChart::BarDiagram::paint\(\)](#), [KDChart::AbstractDiagram::paintDataValueTexts\(\)](#), [KDChart::AbstractDiagram::paintMarker\(\)](#), and [KDChart::AbstractDiagram::paintMarkers\(\)](#).

```

1061 {
1062     if( ! justReturnTheStatus ){
1063         Q_ASSERT_X ( model(), "AbstractDiagram::checkInvariants()",
1064             "There is no usable model set, for the diagram." );
1065
1066         Q_ASSERT_X ( coordinatePlane(), "AbstractDiagram::checkInvariants()",
1067             "There is no usable coordinate plane set, for the diagram." );
1068     }
1069     return model() && coordinatePlane();
1070 }

```

#### 9.53.3.10 **[RingDiagram](#) \* RingDiagram::clone () const** [virtual]

Creates an exact copy of this diagram.

Definition at line 67 of file KDChartRingDiagram.cpp.

References [d](#), and [RingDiagram\(\)](#).

```

68 {
69     return new RingDiagram( new Private( *d ) );
70 }

```

### 9.53.3.11 int AbstractPolarDiagram::columnCount () const [inherited]

Definition at line 60 of file KDCartAbstractPolarDiagram.cpp.

References KDCart::AbstractPolarDiagram::numberOfValuesPerDataset().

Referenced by KDCart::PieDiagram::calculateDataBoundaries(), KDCart::PieDiagram::paint(), and KDCart::PieDiagram::valueTotals().

```

61 {
62     return static_cast<int>( numberOfValuesPerDataset() );
63 }

```

### 9.53.3.12 QModelIndex AbstractDiagram::columnToIndex (int column) const [protected, inherited]

Definition at line 309 of file KDCartAbstractDiagram.cpp.

Referenced by KDCart::BarDiagram::barAttributes(), KDCart::AbstractDiagram::brush(), KDCart::AbstractDiagram::dataValueAttributes(), KDCart::AbstractDiagram::isHidden(), KDCart::Plotter::lineAttributes(), KDCart::LineDiagram::lineAttributes(), KDCart::AbstractDiagram::pen(), KDCart::AbstractPieDiagram::pieAttributes(), KDCart::BarDiagram::threeDBarAttributes(), KDCart::Plotter::threeDLineAttributes(), KDCart::LineDiagram::threeDLineAttributes(), and KDCart::AbstractPieDiagram::threeDPieAttributes().

```

310 { // FIXME (Mirko): shouldn't this be headerData? instead of the index for the first row?
311     if( model() )
312         return QModelIndex( model()->index( 0, column, rootIndex() ) );
313     return QModelIndex();
314 }

```

### 9.53.3.13 bool AbstractDiagram::compare (const AbstractDiagram \* other) const [inherited]

Returns true if both diagrams have the same settings.

Definition at line 129 of file KDCartAbstractDiagram.cpp.

References KDCart::AbstractDiagram::allowOverlappingDataValueTexts(), KDCart::AbstractDiagram::antiAliasing(), KDCart::AbstractDiagram::attributesModel(), KDCart::AttributesModel::compare(), KDCart::AbstractDiagram::datasetDimension(), and KDCart::AbstractDiagram::percentMode().

```

130 {
131     if( other == this ) return true;
132     if( ! other ){
133         //qDebug() << "AbstractDiagram::compare() cannot compare to Null pointer";
134         return false;
135     }
136     /*
137     qDebug() << "\n                AbstractDiagram::compare() QAbstractScrollArea:";

```

```

138         // compare QAbstractScrollArea properties
139     qDebug() <<
140         ((horizontalScrollBarPolicy() == other->horizontalScrollBarPolicy()) &&
141         (verticalScrollBarPolicy() == other->verticalScrollBarPolicy()));
142     qDebug() << "AbstractDiagram::compare() QFrame:";
143     // compare QFrame properties
144     qDebug() <<
145         ((frameShadow() == other->frameShadow()) &&
146         (frameShape() == other->frameShape()) &&
147         (frameWidth() == other->frameWidth()) &&
148         (lineWidth() == other->lineWidth()) &&
149         (midLineWidth() == other->midLineWidth()));
150     qDebug() << "AbstractDiagram::compare() QAbstractItemView:";
151     // compare QAbstractItemView properties
152     qDebug() <<
153         ((alternatingRowColors() == other->alternatingRowColors()) &&
154         (hasAutoScroll() == other->hasAutoScroll()) &&
155     #if QT_VERSION > 0x040199
156         (dragDropMode() == other->dragDropMode()) &&
157         (dragDropOverwriteMode() == other->dragDropOverwriteMode()) &&
158         (horizontalScrollMode() == other->horizontalScrollMode()) &&
159         (verticalScrollMode() == other->verticalScrollMode()) &&
160     #endif
161         (dragEnabled() == other->dragEnabled()) &&
162         (editTriggers() == other->editTriggers()) &&
163         (iconSize() == other->iconSize()) &&
164         (selectionBehavior() == other->selectionBehavior()) &&
165         (selectionMode() == other->selectionMode()) &&
166         (showDropIndicator() == other->showDropIndicator()) &&
167         (tabKeyNavigation() == other->tabKeyNavigation()) &&
168         (textElideMode() == other->textElideMode()));
169     qDebug() << "AbstractDiagram::compare() AttributesModel: ";
170     // compare all of the properties stored in the attributes model
171     qDebug() << attributesModel()->compare( other->attributesModel() );
172     qDebug() << "AbstractDiagram::compare() own:";
173     // compare own properties
174     qDebug() <<
175         ((rootIndex().column() == other->rootIndex().column()) &&
176         (rootIndex().row() == other->rootIndex().row()) &&
177         (allowOverlappingDataValueTexts() == other->allowOverlappingDataValueTexts()) &&
178         (antiAliasing() == other->antiAliasing()) &&
179         (percentMode() == other->percentMode()) &&
180         (datasetDimension() == other->datasetDimension()));
181     */
182     return // compare QAbstractScrollArea properties
183         (horizontalScrollBarPolicy() == other->horizontalScrollBarPolicy()) &&
184         (verticalScrollBarPolicy() == other->verticalScrollBarPolicy()) &&
185         // compare QFrame properties
186         (frameShadow() == other->frameShadow()) &&
187         (frameShape() == other->frameShape()) &&
188         // frameWidth is a read-only property defined by the style, it should not be in here:
189         // (frameWidth() == other->frameWidth()) &&
190         (lineWidth() == other->lineWidth()) &&
191         (midLineWidth() == other->midLineWidth()) &&
192         // compare QAbstractItemView properties
193         (alternatingRowColors() == other->alternatingRowColors()) &&
194         (hasAutoScroll() == other->hasAutoScroll()) &&
195     #if QT_VERSION > 0x040199
196         (dragDropMode() == other->dragDropMode()) &&
197         (dragDropOverwriteMode() == other->dragDropOverwriteMode()) &&
198         (horizontalScrollMode() == other->horizontalScrollMode()) &&
199         (verticalScrollMode() == other->verticalScrollMode()) &&
200     #endif
201         (dragEnabled() == other->dragEnabled()) &&
202         (editTriggers() == other->editTriggers()) &&
203         (iconSize() == other->iconSize()) &&
204         (selectionBehavior() == other->selectionBehavior()) &&

```



```

205         (selectionMode() == other->selectionMode()) &&
206         (showDropIndicator() == other->showDropIndicator()) &&
207         (tabKeyNavigation() == other->tabKeyNavigation()) &&
208         (textElideMode() == other->textElideMode()) &&
209         // compare all of the properties stored in the attributes model
210         attributesModel()->compare( other->attributesModel() ) &&
211         // compare own properties
212         (rootIndex().column() == other->rootIndex().column()) &&
213         (rootIndex().row() == other->rootIndex().row()) &&
214         (allowOverlappingDataValueTexts() == other->allowOverlappingDataValueTexts()) &&
215         (antiAliasing() == other->antiAliasing()) &&
216         (percentMode() == other->percentMode()) &&
217         (datasetDimension() == other->datasetDimension());
218     }

```

#### 9.53.3.14 [AbstractCoordinatePlane](#) \* [AbstractDiagram::coordinatePlane](#) () const [inherited]

The coordinate plane associated with the diagram.

This determines how coordinates in value space are mapped into pixel space. By default this is a [Cartesian-CoordinatePlane](#).

##### Returns:

The coordinate plane associated with the diagram.

Definition at line 220 of file `KDChartAbstractDiagram.cpp`.

References `d`.

Referenced by `KDChart::AbstractDiagram::checkInvariants()`, `KDChart::AbstractCartesianDiagram::layoutPlanes()`, `KDChart::PolarDiagram::paint()`, `KDChart::AbstractDiagram::paintDataValueTexts()`, `KDChart::AbstractDiagram::paintMarkers()`, `KDChart::AbstractPolarDiagram::polarCoordinatePlane()`, and `KDChart::AbstractCartesianDiagram::setCoordinatePlane()`.

```

221 {
222     return d->plane;
223 }

```

#### 9.53.3.15 `const QPair< QPointF, QPointF > AbstractDiagram::dataBoundaries` () const [inherited]

Return the bottom left and top right data point, that the diagram will display (unless the grid adjusts these values).

This method returns a cached result of calculations done by `calculateDataBoundaries`. Classes derived from [AbstractDiagram](#) must implement the `calculateDataBoundaries` function, to specify their own way of calculating the data boundaries. If derived classes want to force recalculation of the data boundaries, they can call `setDataBoundariesDirty()`

Returned value is in diagram coordinates.

Definition at line 225 of file `KDChartAbstractDiagram.cpp`.

References `KDChart::AbstractDiagram::calculateDataBoundaries()`, and `d`.

Referenced by `KDChart::CartesianCoordinatePlane::getRawDataBoundingRectFromDiagrams()`, `KDChart::PolarCoordinatePlane::layoutDiagrams()`, `KDChart::Plotter::paint()`, `KDChart::LineDiagram::paint()`, and `KDChart::BarDiagram::paint()`.

```

226 {
227     if( d->databoundariesDirty ){
228         d->databoundaries = calculateDataBoundaries ();
229         d->databoundariesDirty = false;
230     }
231     return d->databoundaries;
232 }

```

### 9.53.3.16 void AbstractDiagram::dataChanged (const QModelIndex & *topLeft*, const QModelIndex & *bottomRight*) [virtual, inherited]

[reimplemented]

Definition at line 330 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::setDataBoundariesDirty().

```

332 {
333     Q_UNUSED( topLeft );
334     Q_UNUSED( bottomRight );
335     // We are still too dumb to do intelligent updates...
336     setDataBoundariesDirty();
337     scheduleDelayedItemsLayout();
338 }

```

### 9.53.3.17 void KDChart::AbstractDiagram::dataHidden () [signal, inherited]

This signal is emitted, when the hidden status of at least one data cell was (un)set.

Referenced by KDChart::AbstractDiagram::setHidden().

### 9.53.3.18 QList< QBrush > AbstractDiagram::datasetBrushes () const [inherited]

The set of dataset brushes currently used, for use in legends, etc.

#### Note:

Cell-level override brushes, if set, take precedence over the dataset values, so you might need to check these too, in order to find the brush, that is used for a single cell.

#### Returns:

The current set of dataset brushes.

Definition at line 1018 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::attributesModelRootIndex(), KDChart::AttributesModel::columnCount(), KDChart::DatasetBrushRole, and KDChart::AbstractDiagram::datasetDimension().

Referenced by KDChart::Legend::setBrushesFromDiagram().

```

1019 {
1020     QList<QBrush> ret;
1021     if( model() == 0 )
1022         return ret;

```

```

1023
1024     const int columnCount = attributesModel()->columnCount(attributesModelRootIndex());
1025     for( int i = 0; i < columnCount / datasetDimension(); ++i )
1026         ret << qVariantValue<QBrush>( attributesModel()->headerData( i, Qt::Vertical, DatasetBrushRol
1027
1028     return ret;
1029 }

```

#### 9.53.3.19 int AbstractDiagram::datasetDimension () const [inherited]

The dataset dimension of a diagram determines, how many value dimensions it expects each datapoint to have.

For each dimension it will expect one column of values in the model. If the dimensionality is 1, automatic values will be used for the abscissa.

For example a diagram with the default dimension of 1, will have one column per datapoint (the y values) and will use automatic values for the x axis (1, 2, 3, ... n). If the dimension is 2, the diagram will use the first, (and the third, fifth, etc) columns as X values, and the second, (and the fourth, sixth, etc) column as Y values.

#### Returns:

The dataset dimension of the diagram.

Definition at line 1072 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::brush(), KDChart::AbstractDiagram::compare(), KDChart::AbstractDiagram::datasetBrushes(), KDChart::AbstractDiagram::datasetLabels(), KDChart::AbstractDiagram::datasetMarkers(), KDChart::AbstractDiagram::datasetPens(), KDChart::LineDiagram::getCellValues(), KDChart::CartesianCoordinatePlane::getDataDimensionsList(), KDChart::TernaryPointDiagram::paint(), KDChart::TernaryLineDiagram::paint(), KDChart::AbstractDiagram::paintDataValueTexts(), KDChart::AbstractDiagram::paintMarkers(), KDChart::AbstractDiagram::pen(), KDChart::AbstractDiagram::setPen(), KDChart::Plotter::setType(), and KDChart::LineDiagram::setType().

```

1073 {
1074     return d->datasetDimension;
1075 }

```

#### 9.53.3.20 QStringList AbstractDiagram::datasetLabels () const [inherited]

The set of dataset labels currently displayed, for use in legends, etc.

#### Returns:

The set of dataset labels currently displayed.

Definition at line 1005 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::attributesModelRootIndex(), KDChart::AttributesModel::columnCount(), KDChart::AbstractDiagram::datasetDimension(), and KDChart::AttributesModel::headerData().

```

1006 {
1007     QStringList ret;
1008     if( model() == 0 )
1009         return ret;
1010
1011     const int columnCount = attributesModel()->columnCount(attributesModelRootIndex());
1012     for( int i = 0; i < columnCount / datasetDimension(); ++i )
1013         ret << attributesModel()->headerData( i, Qt::Horizontal, Qt::DisplayRole ).toString();
1014
1015     return ret;
1016 }

```

### 9.53.3.21 QList< [MarkerAttributes](#) > AbstractDiagram::datasetMarkers () const

[inherited]

The set of dataset markers currently used, for use in legends, etc.

#### Note:

Cell-level override markers, if set, take precedence over the dataset values, so you might need to check these too, in order to find the marker, that is shown for a single cell.

#### Returns:

The current set of dataset brushes.

Definition at line 1044 of file KDChartAbstractDiagram.cpp.

References [KDChart::AbstractDiagram::attributesModel\(\)](#), [KDChart::AbstractDiagram::attributesModelRootIndex\(\)](#), [KDChart::AttributesModel::columnCount\(\)](#), [KDChart::AbstractDiagram::datasetDimension\(\)](#), and [KDChart::DataValueLabelAttributesRole](#).

```

1045 {
1046     QList<MarkerAttributes> ret;
1047     if( model() == 0 )
1048         return ret;
1049
1050     const int columnCount = attributesModel()->columnCount(attributesModelRootIndex());
1051     for( int i = 0; i < columnCount / datasetDimension(); ++i )
1052     {
1053         const DataValueAttributes a =
1054             qVariantValue<DataValueAttributes>( attributesModel()->headerData( i, Qt::Vertical, DataValueLabelAttributesRole ));
1055         ret << a.markerAttributes();
1056     }
1057     return ret;
1058 }

```

### 9.53.3.22 QList< QPen > AbstractDiagram::datasetPens () const

[inherited]

The set of dataset pens currently used, for use in legends, etc.

#### Note:

Cell-level override pens, if set, take precedence over the dataset values, so you might need to check these too, in order to find the pens, that is used for a single cell.

#### Returns:

The current set of dataset pens.

Definition at line 1031 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::attributesModelRootIndex(), KDChart::AttributesModel::columnCount(), KDChart::AbstractDiagram::datasetDimension(), and KDChart::DatasetPenRole.

```

1032 {
1033     QList<QPen> ret;
1034     if( model() == 0 )
1035         return ret;
1036
1037     const int columnCount = attributesModel()->columnCount(attributesModelRootIndex());
1038     for( int i = 0; i < columnCount / datasetDimension(); ++i )
1039         ret << qVariantValue<QPen>( attributesModel()->headerData( i, Qt::Vertical, DatasetPenRole ) );
1040
1041     return ret;
1042 }
```

### 9.53.3.23 [DataValueAttributes](#) AbstractDiagram::dataValueAttributes (const QModelIndex & index) const [inherited]

Retrieve the [DataValueAttributes](#) for the given index.

This will fall back automatically to what was set at dataset or model level, if there are no datapoint specific settings.

#### Parameters:

*index* The datapoint to retrieve the attributes for.

#### Returns:

The [DataValueAttributes](#) for the given index.

Definition at line 421 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AttributesModel::data(), and KDChart::DataValueLabelAttributesRole.

```

422 {
423     return qVariantValue<DataValueAttributes>(
424         attributesModel()->data( attributesModel()->mapFromSource(index),
425                                 KDChart::DataValueLabelAttributesRole ) );
426 }
```

### 9.53.3.24 [DataValueAttributes](#) AbstractDiagram::dataValueAttributes (int column) const [inherited]

Retrieve the [DataValueAttributes](#) for the given dataset.

This will fall back automatically to what was set at model level, if there are no dataset specific settings.

#### Parameters:

*column* The dataset to retrieve the attributes for.

#### Returns:

The [DataValueAttributes](#) for the given dataset.

Definition at line 414 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::columnToIndex(), KDChart::AttributesModel::data(), and KDChart::DataValueLabelAttributesRole.

```

415 {
416     return qVariantValue<DataValueAttributes>(
417         attributesModel()->data( attributesModel()->mapFromSource(columnToIndex( column )),
418         KDChart::DataValueLabelAttributesRole ) );
419 }
```

### 9.53.3.25 [DataValueAttributes](#) AbstractDiagram::dataValueAttributes () const [inherited]

Retrieve the [DataValueAttributes](#) specified globally.

This will fall back automatically to the default settings, if there are no specific settings.

#### Returns:

The global [DataValueAttributes](#).

Definition at line 408 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DataValueLabelAttributesRole, and KDChart::AttributesModel::modelData().

Referenced by KDChart::AbstractDiagram::paintDataValueText(), KDChart::AbstractDiagram::paintMarker(), and KDChart::TernaryLineDiagram::TernaryLineDiagram().

```

409 {
410     return qVariantValue<DataValueAttributes>(
411         attributesModel()->modelData( KDChart::DataValueLabelAttributesRole ) );
412 }
```

### 9.53.3.26 void AbstractDiagram::doItemsLayout () [virtual, inherited]

[reimplemented]

Definition at line 321 of file KDChartAbstractDiagram.cpp.

References d, and KDChart::AbstractDiagram::update().

```

322 {
323     if ( d->plane ) {
324         d->plane->layoutDiagrams();
325         update();
326     }
327     QAbstractItemView::doItemsLayout();
328 }
```

### 9.53.3.27 qreal AbstractPieDiagram::granularity () const [inherited]

#### Returns:

the granularity.

Definition at line 69 of file KDChartAbstractPieDiagram.cpp.

References d.

Referenced by KDChart::PieDiagram::paint().

```
70 {
71     return (d->granularity < 0.05 || d->granularity > 36.0)
72           ? 1.0
73           : d->granularity;
74 }
```

### 9.53.3.28 int AbstractDiagram::horizontalOffset () const [virtual, inherited]

[reimplemented]

Definition at line 950 of file KDChartAbstractDiagram.cpp.

```
951 { return 0; }
```

### 9.53.3.29 QModelIndex AbstractDiagram::indexAt (const QPoint & point) const [virtual, inherited]

[reimplemented]

Definition at line 1098 of file KDChartAbstractDiagram.cpp.

References d.

```
1099 {
1100     return d->indexAt( point );
1101 }
```

### 9.53.3.30 QModelIndexList AbstractDiagram::indexesAt (const QPoint & point) const [inherited]

This method is added alongside with indexAt from QAIM, since in kdchart multiple indexes can be displayed at the same spot.

Definition at line 1103 of file KDChartAbstractDiagram.cpp.

References d.

```
1104 {
1105     return d->indexesAt( point );
1106 }
```

### 9.53.3.31 bool AbstractDiagram::isHidden (const QModelIndex & index) const [inherited]

Retrieve the hidden status for the given index.

This will fall back automatically to what was set at dataset or diagram level, if there are no datapoint specific settings.

**Parameters:**

*index* The datapoint to retrieve the hidden status for.

**Returns:**

The hidden status for the given index.

Definition at line 380 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AttributesModel::data(), and KDChart::DataHiddenRole.

```

381 {
382     return qVariantValue<bool>(
383         attributesModel()->data(
384             attributesModel()->mapFromSource(index),
385             DataHiddenRole ) );
386 }
```

### 9.53.3.32 bool AbstractDiagram::isHidden (int *column*) const [inherited]

Retrieve the hidden status for the given dataset.

This will fall back automatically to what was set at diagram level, if there are no dataset specific settings.

**Parameters:**

*column* The dataset to retrieve the hidden status for.

**Returns:**

The hidden status for the given dataset.

Definition at line 373 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::columnToIndex(), KDChart::AttributesModel::data(), and KDChart::DataHiddenRole.

```

374 {
375     return qVariantValue<bool>(
376         attributesModel()->data(
377             attributesModel()->mapFromSource(columnToIndex( column ) ),
378             DataHiddenRole ) );
379 }
```

### 9.53.3.33 bool AbstractDiagram::isHidden () const [inherited]

Retrieve the hidden status specified globally.

This will fall back automatically to the default settings ( = not hidden), if there are no specific settings.

**Returns:**

The global hidden status.



Definition at line 367 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DataHiddenRole, and KDChart::AttributesModel::modelData().

Referenced by KDChart::LineDiagram::valueForCellTesting().

```

368 {
369     return qVariantValue<bool>(
370         attributesModel()->modelData( DataHiddenRole ) );
371 }
```

### 9.53.3.34 bool AbstractDiagram::isIndexHidden (const QModelIndex & *index*) const [virtual, inherited]

[reimplemented]

Definition at line 956 of file KDChartAbstractDiagram.cpp.

```

957 { return true; }
```

### 9.53.3.35 QStringList AbstractDiagram::itemRowLabels () const [inherited]

The set of item row labels currently displayed, for use in Abscissa axes, etc.

#### Returns:

The set of item row labels currently displayed.

Definition at line 989 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::attributesModelRootIndex(), KDChart::AttributesModel::rowCount(), KDChart::AbstractDiagram::unitPrefix(), and KDChart::AbstractDiagram::unitSuffix().

```

990 {
991     QStringList ret;
992     if( model() ){
993         //qDebug() << "AbstractDiagram::itemRowLabels(): " << attributesModel()->rowCount(attributesModelRootIndex());
994         const int rowCount = attributesModel()->rowCount(attributesModelRootIndex());
995         for( int i = 0; i < rowCount; ++i ){
996             //qDebug() << "item row label: " << attributesModel()->headerData( i, Qt::Vertical, Qt::DisplayRole ).toString();
997             ret << unitPrefix( i, Qt::Horizontal, true ) +
998                 attributesModel()->headerData( i, Qt::Vertical, Qt::DisplayRole ).toString() +
999                 unitSuffix( i, Qt::Horizontal, true );
1000         }
1001     }
1002     return ret;
1003 }
```

### 9.53.3.36 void KDChart::AbstractDiagram::layoutChanged ([AbstractDiagram \\*](#)) [signal, inherited]

Diagrams are supposed to emit this signal, when the layout of one of their element changes.

Layouts can change, for example, when axes are added or removed, or when the configuration was changed in a way that the axes or the diagram itself are displayed in a different geometry. Changes in the diagrams coordinate system also result in the [layoutChanged\(\)](#) signal being emitted.

Referenced by [KDChart::AbstractDiagram::setDatasetDimension\(\)](#), [KDChart::AbstractPieDiagram::setPieAttributes\(\)](#), [KDChart::AbstractPieDiagram::setThreeDPieAttributes\(\)](#), [KDChart::Plotter::setType\(\)](#), [KDChart::LineDiagram::setType\(\)](#), and [KDChart::BarDiagram::setType\(\)](#).

### 9.53.3.37 void KDChart::AbstractDiagram::modelsChanged () [signal, inherited]

This signal is emitted, when either the model or the [AttributesModel](#) is replaced.

Referenced by [KDChart::AbstractDiagram::setAttributesModel\(\)](#), and [KDChart::AbstractDiagram::setModel\(\)](#).

### 9.53.3.38 QModelIndex AbstractDiagram::moveCursor (CursorAction *cursorAction*, Qt::KeyboardModifiers *modifiers*) [virtual, inherited]

[reimplemented]

Definition at line 947 of file [KDChartAbstractDiagram.cpp](#).

```
948 { return QModelIndex(); }
```

### 9.53.3.39 double RingDiagram::numberOfGridRings () const [virtual]

[reimplemented]

Implements [KDChart::AbstractPolarDiagram](#).

Definition at line 153 of file [KDChartRingDiagram.cpp](#).

```
154 {
155     return 1;
156 }
```

### 9.53.3.40 double RingDiagram::numberOfValuesPerDataset () const [virtual]

[reimplemented]

Implements [KDChart::AbstractPolarDiagram](#).

Definition at line 147 of file [KDChartRingDiagram.cpp](#).

```
148 {
149     return model() ? model()->columnCount(rootIndex()) : 0.0;
150 }
```

### 9.53.3.41 void RingDiagram::paint (PaintContext \* *paintContext*) [protected, virtual]

[reimplemented]

Implements [KDChart::AbstractDiagram](#).

Definition at line 104 of file KDChartRingDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::brush(), KDChart::AbstractDiagram::checkInvariants(), KDChart::DatasetBrushRole, KDChart::AbstractDiagram::paintDataValueText(), and KDChart::PaintContext::painter().

Referenced by paintEvent().

```

105 {
106     // note: Not having any data model assigned is no bug
107     //       but we can not draw a diagram then either.
108     if ( !checkInvariants(true) )
109         return;
110
111     const int colCount = model()->columnCount(rootIndex());
112     DataValueTextInfoList list;
113     for ( int j=0; j<colCount; ++j ) {
114         QBrush brush = qVariantValue<QBrush>( attributesModel()->headerData( j, Qt::Vertical, KDChart::
115         PainterSaver painterSaver( ctx->painter() );
116         ctx->painter()->setRenderHint ( QPainter::Antialiasing );
117         ctx->painter()->setBrush( brush );
118         QPen p( ctx->painter()->pen() );
119         p.setColor( brush.color() );
120         p.setWidth( 2 );// FIXME properties, use DatasetPenRole
121         ctx->painter()->setPen( p );
122         //ctx->painter()->drawPolyline( polygon );
123     }
124     DataValueTextInfoListIterator it( list );
125     while ( it.hasNext() ) {
126         const DataValueTextInfo& info = it.next();
127         paintDataValueText( ctx->painter(), info.index, info.pos, info.value );
128     }
129 }
```

#### 9.53.3.42 void AbstractDiagram::paintDataValueText (QPainter \*painter, const QModelIndex &index, const QPointF &pos, double value) [inherited]

Definition at line 468 of file KDChartAbstractDiagram.cpp.

References d, KDChart::DataValueAttributes::dataLabel(), KDChart::AbstractDiagram::dataValueAttributes(), KDChart::DataValueAttributes::decimalDigits(), KDChart::DataValueAttributes::isVisible(), KDChartEnums::MeasureOrientationMinimum, KDChart::DataValueAttributes::position(), KDChart::DataValueAttributes::prefix(), KDChart::DataValueAttributes::suffix(), and KDChart::DataValueAttributes::textAttributes().

Referenced by paint(), KDChart::PolarDiagram::paint(), and KDChart::AbstractDiagram::paintDataValueTexts().

```

472 {
473     // paint one data series
474     const DataValueAttributes a( dataValueAttributes(index) );
475     if ( !a.isVisible() ) return;
476
477     // handle decimal digits
478     int decimalDigits = a.decimalDigits();
479     int decimalPos = QString::number( value ).indexOf( QLatin1Char( '.' ) );
480     QString roundedValue;
481     if ( a.dataLabel().isNull() ) {
482         if ( decimalPos > 0 && value != 0 )
483             roundedValue = roundValues ( value, decimalPos, decimalDigits );
484         else
485             roundedValue = QString::number( value );

```

```

486     } else
487         roundedValue = a.dataLabel();
488         // handle prefix and suffix
489     if ( !a.prefix().isNull() )
490         roundedValue.prepend( a.prefix() );
491
492     if ( !a.suffix().isNull() )
493         roundedValue.append( a.suffix() );
494
495     const TextAttributes ta( a.textAttributes() );
496     // FIXME draw the non-text bits, background, etc
497     if ( ta.isVisible() ) {
498
499         QPointF pt( pos );
500         /* for debugging:
501         PainterSaver painterSaver( painter );
502         painter->setPen( Qt::black );
503         painter->drawLine( pos - QPointF( 1,1), pos + QPointF( 1,1) );
504         painter->drawLine( pos - QPointF(-1,1), pos + QPointF(-1,1) );
505         */
506
507         QTextDocument doc;
508         if( Qt::mightBeRichText( roundedValue ) )
509             doc.setHtml( roundedValue );
510         else
511             doc.setPlainText( roundedValue );
512
513         const RelativePosition relPos( a.position( value >= 0.0 ) );
514         const Qt::Alignment alignBottomLeft = Qt::AlignBottom | Qt::AlignLeft;
515         const QFont calculatedFont( ta.calculatedFont( d->plane, KDChartEnums::MeasureOrientationMinimum ) );
516         const QRectF boundRect( d->cachedFontMetrics( calculatedFont, painter->device() )->boundingRect( roundedValue ) );
517
518         // To place correctly
519         pt.ry() -= boundRect.height();
520
521         // qDebug() << "calculatedFont's point size:" << calculatedFont.pointSizeF();
522         // adjust the text start point position, if alignment is not Bottom/Left
523         if( (relPos.alignment() & alignBottomLeft) != alignBottomLeft ) {
524             if( relPos.alignment() & Qt::AlignRight )
525                 pt.rx() -= boundRect.width();
526             else if( relPos.alignment() & Qt::AlignHCenter )
527                 pt.rx() -= 0.5 * boundRect.width();
528
529             if( relPos.alignment() & Qt::AlignTop )
530                 pt.ry() += boundRect.height();
531             else if( relPos.alignment() & Qt::AlignVCenter )
532                 pt.ry() += 0.5 * boundRect.height();
533         }
534
535         // FIXME draw the non-text bits, background, etc
536
537         if ( a.showRepetitiveDataLabels() ||
538             pos.x() <= d->lastX ||
539             d->lastRoundedValue != roundedValue ) {
540             d->lastRoundedValue = roundedValue;
541             d->lastX = pos.x();
542             PainterSaver painterSaver( painter );
543
544             doc.setDefaultFont( calculatedFont );
545             QAbstractTextDocumentLayout::PaintContext context;
546             context.palette = palette();
547             context.palette.setColor(QPalette::Text, ta.pen().color() );
548
549             painter->translate( pt );
550             painter->rotate( ta.rotation() );
551
552             QAbstractTextDocumentLayout* layout = doc.documentLayout();

```

```

553         layout->draw( painter, context );
554     }
555 }
556 }

```

### 9.53.3.43 void AbstractDiagram::paintDataValueTexts (QPainter \* *painter*) [protected, virtual, inherited]

Definition at line 584 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::checkInvariants(), KDChart::AbstractDiagram::coordinatePlane(), KDChart::AbstractDiagram::datasetDimension(), KDChart::AbstractDiagram::paintDataValueText(), and KDChart::AbstractCoordinatePlane::translate().

```

585 {
586     if ( !checkInvariants() ) return;
587     const int rowCount = model()->rowCount( rootIndex() );
588     const int columnCount = model()->columnCount( rootIndex() );
589     for ( int i=datasetDimension()-1; i<columnCount; i += datasetDimension() ) {
590         for ( int j=0; j< rowCount; ++j ) {
591             const QModelIndex index = model()->index( j, i, rootIndex() );
592             double value = model()->data( index ).toDouble();
593             const QPointF pos = coordinatePlane()->translate( QPointF( j, value ) );
594             paintDataValueText( painter, index, pos, value );
595         }
596     }
597 }

```

### 9.53.3.44 void RingDiagram::paintEvent (QPaintEvent \*) [protected]

Definition at line 91 of file KDChartRingDiagram.cpp.

References paint(), KDChart::PaintContext::setPainter(), and KDChart::PaintContext::setRectangle().

```

92 {
93     QPainter painter ( viewport() );
94     PaintContext ctx;
95     ctx.setPainter ( &painter );
96     ctx.setRectangle( QRectF ( 0, 0, width(), height() ) );
97     paint ( &ctx );
98 }

```

### 9.53.3.45 void AbstractDiagram::paintMarker (QPainter \* *painter*, const QModelIndex & *index*, const QPointF & *pos*) [inherited]

Definition at line 600 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::brush(), KDChart::AbstractDiagram::checkInvariants(), d, KDChart::AbstractDiagram::dataValueAttributes(), KDChart::MarkerAttributes::isVisible(), KDChart::DataValueAttributes::isVisible(), KDChart::DataValueAttributes::markerAttributes(), KDChart::MarkerAttributes::markerColor(), KDChart::MarkerAttributes::markerSize(), KDChart::AbstractDiagram::paintMarker(), and KDChart::MarkerAttributes::pen().

```

603 {
604     if ( !checkInvariants() ) return;

```

```

605     DataValueAttributes a = dataValueAttributes(index);
606     if ( !a.isVisible() ) return;
607     const MarkerAttributes &ma = a.markerAttributes();
608     if ( !ma.isVisible() ) return;
609
610     PainterSaver painterSaver( painter );
611     QSizeF maSize( ma.markerSize() );
612     QBrush indexBrush( brush( index ) );
613     QPen indexPen( ma.pen() );
614     if ( ma.markerColor().isValid() )
615         indexBrush.setColor( ma.markerColor() );
616
617     paintMarker( painter, ma, indexBrush, indexPen, pos, maSize );
618
619     // workaround: BC cannot be changed, otherwise we would pass the
620     // index down to next-lower paintMarker function. So far, we
621     // basically save a circle of radius maSize at pos in the
622     // reverseMapper. This means that ^^^ this version of paintMarker
623     // needs to be called to reverse-map the marker.
624     d->reverseMapper.addCircle( index.row(), index.column(), pos, 2 * maSize );
625 }

```

**9.53.3.46 void AbstractDiagram::paintMarker (QPainter \* *painter*, const [MarkerAttributes](#) & *markerAttributes*, const QBrush & *brush*, const QPen &, const QPointF & *point*, const QSizeF & *size*)** [virtual, inherited]

Definition at line 627 of file KDChartAbstractDiagram.cpp.

References KDChart::MarkerAttributes::Marker1Pixel, KDChart::MarkerAttributes::Marker4Pixels, KDChart::MarkerAttributes::MarkerCircle, KDChart::MarkerAttributes::MarkerCross, KDChart::MarkerAttributes::MarkerDiamond, KDChart::MarkerAttributes::MarkerFastCross, KDChart::MarkerAttributes::MarkerRing, KDChart::MarkerAttributes::MarkerSquare, and KDChart::MarkerAttributes::markerStyle().

Referenced by KDChart::TernaryPointDiagram::paint(), KDChart::TernaryLineDiagram::paint(), KDChart::MarkerLayoutItem::paintIntoRect(), KDChart::AbstractDiagram::paintMarker(), and KDChart::AbstractDiagram::paintMarkers().

```

633 {
634     const QPen oldPen( painter->pen() );
635     // Pen is used to paint 4Pixels - 1 Pixel - Ring and FastCross types.
636     // make sure to use the brush color - see above in those cases.
637     const bool isFourPixels = (markerAttributes.markerStyle() == MarkerAttributes::Marker4Pixels);
638     if( isFourPixels || (markerAttributes.markerStyle() == MarkerAttributes::Marker1Pixel) ){
639         // for high-performance point charts with tiny point markers:
640         painter->setPen( QPen( brush.color().light() ) );
641         if( isFourPixels ){
642             const qreal x = pos.x();
643             const qreal y = pos.y();
644             painter->drawLine( QPointF(x-1.0,y-1.0),
645                             QPointF(x+1.0,y-1.0) );
646             painter->drawLine( QPointF(x-1.0,y),
647                             QPointF(x+1.0,y) );
648             painter->drawLine( QPointF(x-1.0,y+1.0),
649                             QPointF(x+1.0,y+1.0) );
650         }
651         painter->drawPoint( pos );
652     }else{
653         PainterSaver painterSaver( painter );
654         // we only a solid line surrounding the markers
655         QPen painterPen( pen );
656         painterPen.setStyle( Qt::SolidLine );

```

```

657     painter->setPen( painterPen );
658     painter->setBrush( brush );
659     painter->setRenderHint ( QPainter::Antialiasing );
660     painter->translate( pos );
661     switch ( markerAttributes.markerStyle() ) {
662     case MarkerAttributes::MarkerCircle:
663         painter->drawEllipse( QRectF( 0 - maSize.height()/2, 0 - maSize.width()/2,
664             maSize.height(), maSize.width() ) );
665         break;
666     case MarkerAttributes::MarkerSquare:
667     {
668         QRectF rect( 0 - maSize.width()/2, 0 - maSize.height()/2,
669             maSize.width(), maSize.height() );
670         painter->drawRect( rect );
671         break;
672     }
673     case MarkerAttributes::MarkerDiamond:
674     {
675         QVector <QPointF > diamondPoints;
676         QPointF top, left, bottom, right;
677         top = QPointF( 0, 0 - maSize.height()/2 );
678         left = QPointF( 0 - maSize.width()/2, 0 );
679         bottom = QPointF( 0, maSize.height()/2 );
680         right = QPointF( maSize.width()/2, 0 );
681         diamondPoints << top << left << bottom << right;
682         painter->drawPolygon( diamondPoints );
683         break;
684     }
685     // both handled on top of the method:
686     case MarkerAttributes::Marker1Pixel:
687     case MarkerAttributes::Marker4Pixels:
688         break;
689     case MarkerAttributes::MarkerRing:
690     {
691         painter->setPen( QPen( brush.color() ) );
692         painter->setBrush( Qt::NoBrush );
693         painter->drawEllipse( QRectF( 0 - maSize.height()/2, 0 - maSize.width()/2,
694             maSize.height(), maSize.width() ) );
695         break;
696     }
697     case MarkerAttributes::MarkerCross:
698     {
699         QRectF rect( maSize.width()*-0.5, maSize.height()*-0.2,
700             maSize.width(), maSize.height()*0.4 );
701         painter->drawRect( rect );
702         rect.setTopLeft( QPointF( maSize.width()*-0.2, maSize.height()*-0.5 ) );
703         rect.setSize( QSizeF( maSize.width()*0.4, maSize.height() ) );
704         painter->drawRect( rect );
705         break;
706     }
707     case MarkerAttributes::MarkerFastCross:
708     {
709         QPointF left, right, top, bottom;
710         left = QPointF( -maSize.width()/2, 0 );
711         right = QPointF( maSize.width()/2, 0 );
712         top = QPointF( 0, -maSize.height()/2 );
713         bottom = QPointF( 0, maSize.height()/2 );
714         painter->setPen( QPen( brush.color() ) );
715         painter->drawLine( left, right );
716         painter->drawLine( top, bottom );
717         break;
718     }
719     default:
720         Q_ASSERT_X ( false, "paintMarkers()",
721             "Type item does not match a defined Marker Type." );
722 }
723 }

```

```

724     painter->setPen( oldPen );
725 }

```

### 9.53.3.47 void AbstractDiagram::paintMarkers (QPainter \* *painter*) [protected, virtual, inherited]

Definition at line 727 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::checkInvariants(), KDChart::AbstractDiagram::coordinatePlane(), KDChart::AbstractDiagram::datasetDimension(), KDChart::AbstractDiagram::paintMarker(), and KDChart::AbstractCoordinatePlane::translate().

```

728 {
729     if ( !checkInvariants() ) return;
730     const int rowCount = model()->rowCount(rootIndex());
731     const int columnCount = model()->columnCount(rootIndex());
732     for ( int i=datasetDimension()-1; i<columnCount; i += datasetDimension() ) {
733         for ( int j=0; j< rowCount; ++j ) {
734             const QModelIndex index = model()->index( j, i, rootIndex() );
735             double value = model()->data( index ).toDouble();
736             const QPointF pos = coordinatePlane()->translate( QPointF( j, value ) );
737             paintMarker( painter, index, pos );
738         }
739     }
740 }

```

### 9.53.3.48 QPen AbstractDiagram::pen (const QModelIndex & *index*) const [inherited]

Retrieve the pen to be used, for painting the datapoint at the given index in the model.

#### Parameters:

***index*** The index of the datapoint in the model.

#### Returns:

The pen to use for painting.

Definition at line 788 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AttributesModel::data(), KDChart::AbstractDiagram::datasetDimension(), KDChart::DatasetPenRole, and KDChart::AbstractDiagram::pen().

```

789 {
790     if( datasetDimension() > 1 )
791         return pen( index.column() );
792     return qVariantValue<QPen>(
793         attributesModel()->data(
794             attributesModel()->mapFromSource( index ),
795             DatasetPenRole ) );
796 }

```



**9.53.3.49 QPen AbstractDiagram::pen (int *dataset*) const** [inherited]

Retrieve the pen to be used for the given dataset.

This will fall back automatically to what was set at model level, if there are no dataset specific settings.

**Parameters:**

*dataset* The dataset to retrieve the pen for.

**Returns:**

The pen to use for painting.

Definition at line 777 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::columnToIndex(), KDChart::AttributesModel::data(), KDChart::AbstractDiagram::datasetDimension(), and KDChart::DatasetPenRole.

```

778 {
779     if( datasetDimension() > 1 )
780         dataset /= datasetDimension();
781
782     return qVariantValue<QPen>(
783         attributesModel()->data(
784             attributesModel()->mapFromSource( columnToIndex( dataset ) ),
785             DatasetPenRole ) );
786 }
```

**9.53.3.50 QPen AbstractDiagram::pen () const** [inherited]

Retrieve the pen to be used for painting datapoints globally.

This will fall back automatically to the default settings, if there are no specific settings.

**Returns:**

The pen to use for painting.

Definition at line 771 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AttributesModel::data(), and KDChart::DatasetPenRole.

Referenced by KDChart::TernaryPointDiagram::paint(), KDChart::TernaryLineDiagram::paint(), and KDChart::AbstractDiagram::pen().

```

772 {
773     return qVariantValue<QPen>(
774         attributesModel()->data( DatasetPenRole ) );
775 }
```

**9.53.3.51 bool AbstractDiagram::percentMode () const** [inherited]

Definition at line 462 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::compare(), and KDChart::CartesianCoordinatePlane::getDataDimensionsList().

```
463 {
464     return d->percent;
465 }
```

### 9.53.3.52 **PieAttributes** AbstractPieDiagram::pieAttributes (const QModelIndex & *index*) const [inherited]

Definition at line 122 of file KDChartAbstractPieDiagram.cpp.

References d, and KDChart::PieAttributesRole.

```
123 {
124     return qVariantValue<PieAttributes>(
125         d->attributesModel->data(
126             d->attributesModel->mapFromSource( index ),
127             PieAttributesRole ) );
128 }
```

### 9.53.3.53 **PieAttributes** AbstractPieDiagram::pieAttributes (int *column*) const [inherited]

Definition at line 114 of file KDChartAbstractPieDiagram.cpp.

References KDChart::AbstractDiagram::columnToIndex(), d, and KDChart::PieAttributesRole.

```
115 {
116     return qVariantValue<PieAttributes>(
117         d->attributesModel->data(
118             d->attributesModel->mapFromSource( columnToIndex( column ) ).column(),
119             PieAttributesRole ) );
120 }
```

### 9.53.3.54 **PieAttributes** AbstractPieDiagram::pieAttributes () const [inherited]

Definition at line 105 of file KDChartAbstractPieDiagram.cpp.

References d, and KDChart::PieAttributesRole.

Referenced by KDChart::PieDiagram::calculateDataBoundaries(), and KDChart::PieDiagram::paint().

```
106 {
107     return qVariantValue<PieAttributes>(
108         d->attributesModel->data( PieAttributesRole ) );
109 }
```

### 9.53.3.55 **const PolarCoordinatePlane \*** AbstractPolarDiagram::polarCoordinatePlane () const [inherited]

Definition at line 55 of file KDChartAbstractPolarDiagram.cpp.

References KDChart::AbstractDiagram::coordinatePlane().

Referenced by KDChart::PieDiagram::paint().

```

56 {
57     return dynamic_cast<const PolarCoordinatePlane*>( coordinatePlane() );
58 }

```

### 9.53.3.56 void KDCart::AbstractDiagram::propertiesChanged () [signal, inherited]

Emitted upon change of a property of the Diagram.

Referenced by KDCart::Plotter::resetLineAttributes(), KDCart::LineDiagram::resetLineAttributes(), KDCart::AbstractDiagram::setAllowOverlappingDataValueTexts(), KDCart::AbstractDiagram::setAntiAliasing(), KDCart::BarDiagram::setBarAttributes(), KDCart::AbstractDiagram::setBrush(), KDCart::AbstractDiagram::setDataValueAttributes(), KDCart::Plotter::setLineAttributes(), KDCart::LineDiagram::setLineAttributes(), KDCart::AbstractDiagram::setPen(), KDCart::AbstractDiagram::setPercentMode(), KDCart::BarDiagram::setThreeDBarAttributes(), KDCart::Plotter::setThreeDLineAttributes(), KDCart::LineDiagram::setThreeDLineAttributes(), KDCart::Plotter::setType(), KDCart::LineDiagram::setType(), KDCart::BarDiagram::setType(), KDCart::Plotter::setValueTrackerAttributes(), and KDCart::LineDiagram::setValueTrackerAttributes().

### 9.53.3.57 bool RingDiagram::relativeThickness () const

Definition at line 77 of file KDCartRingDiagram.cpp.

References d.

```

78 {
79     return d->relativeThickness;
80 }

```

### 9.53.3.58 void RingDiagram::resize (const QSizeF & area) [virtual]

[reimplemented]

Implements [KDCart::AbstractDiagram](#).

Definition at line 131 of file KDCartRingDiagram.cpp.

```

132 {
133 }

```

### 9.53.3.59 void RingDiagram::resizeEvent (QResizeEvent \*) [protected]

Definition at line 100 of file KDCartRingDiagram.cpp.

```

101 {
102 }

```

### 9.53.3.60 void AbstractDiagram::scrollTo (const QModelIndex & index, ScrollHint hint = EnsureVisible) [virtual, inherited]

[reimplemented]

Definition at line 942 of file KDCartAbstractDiagram.cpp.

```
943 {}
```

### 9.53.3.61 void AbstractDiagram::setAllowOverlappingDataValueTexts (bool *allow*) [inherited]

Set whether data value labels are allowed to overlap.

#### Parameters:

*allow* True means that overlapping labels are allowed.

Definition at line 434 of file KDChartAbstractDiagram.cpp.

References `d`, and `KDChart::AbstractDiagram::propertiesChanged()`.

```
435 {  
436     d->allowOverlappingDataValueTexts = allow;  
437     emit propertiesChanged();  
438 }
```

### 9.53.3.62 void AbstractDiagram::setAntiAliasing (bool *enabled*) [inherited]

Set whether anti-aliasing is to be used while rendering this diagram.

#### Parameters:

*enabled* True means that AA is enabled.

Definition at line 445 of file KDChartAbstractDiagram.cpp.

References `d`, and `KDChart::AbstractDiagram::propertiesChanged()`.

```
446 {  
447     d->antiAliasing = enabled;  
448     emit propertiesChanged();  
449 }
```

### 9.53.3.63 void AbstractDiagram::setAttributesModel ([AttributesModel](#) \* *model*) [virtual, inherited]

Associate an [AttributesModel](#) with this diagram.

Note that the diagram does `_not_` take ownership of the [AttributesModel](#). This should thus only be used with [AttributesModels](#) that have been explicitly created by the user, and are owned by her. Setting an [AttributesModel](#) that is internal to another diagram is an error.

Correct:

```
AttributesModel *am = new AttributesModel( model, 0 );  
diagram1->setAttributesModel( am );  
diagram2->setAttributesModel( am );
```

Wrong:

```
diagram1->setAttributesModel( diagram2->attributesModel() );
```

**Parameters:**

*model* The [AttributesModel](#) to use for this diagram.

**See also:**

[AttributesModel](#), [usesExternalAttributesModel](#)

Reimplemented in [KDChart::AbstractCartesianDiagram](#).

Definition at line 255 of file `KDChartAbstractDiagram.cpp`.

References `d`, `KDChart::AbstractDiagram::modelsChanged()`, and `KDChart::AbstractDiagram::setDataBoundariesDirty()`.

Referenced by `KDChart::AbstractCartesianDiagram::setAttributesModel()`.

```
256 {
257     if( amodel->sourceModel() != model() ) {
258         qWarning("KDChart::AbstractDiagram::setAttributesModel() failed: "
259                 "Trying to set an attributesmodel which works on a different "
260                 "model than the diagram.");
261         return;
262     }
263     if( qobject_cast<PrivateAttributesModel*>(amodel) ) {
264         qWarning("KDChart::AbstractDiagram::setAttributesModel() failed: "
265                 "Trying to set an attributesmodel that is private to another diagram.");
266         return;
267     }
268     d->setAttributesModel(amodel);
269     scheduleDelayedItemsLayout();
270     setDataBoundariesDirty();
271     emit modelsChanged();
272 }
```

#### 9.53.3.64 void AbstractDiagram::setAttributesModelRootIndex (const QModelIndex &) [protected, inherited]

Definition at line 293 of file `KDChartAbstractDiagram.cpp`.

References `d`, and `KDChart::AbstractDiagram::setDataBoundariesDirty()`.

Referenced by `KDChart::AbstractDiagram::setRootIndex()`.

```
294 {
295     d->attributesModelRootIndex=idx;
296     setDataBoundariesDirty();
297     scheduleDelayedItemsLayout();
298 }
```

#### 9.53.3.65 void AbstractDiagram::setBrush (const QBrush & brush) [inherited]

Set the brush to be used, for painting all datasets in the model.

**Parameters:**

*brush* The brush to use.

Definition at line 806 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DatasetBrushRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setModelData().

```
807 {
808     attributesModel()->setModelData(
809         qVariantFromValue( brush ), DatasetBrushRole );
810     emit propertiesChanged();
811 }
```

### 9.53.3.66 void AbstractDiagram::setBrush (int *dataset*, const QBrush & *brush*) [inherited]

Set the brush to be used, for painting the given dataset.

#### Parameters:

*dataset* The dataset's column in the model.

*brush* The brush to use.

Definition at line 813 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DatasetBrushRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setHeaderData().

```
814 {
815     attributesModel()->setHeaderData(
816         column, Qt::Vertical,
817         qVariantFromValue( brush ),
818         DatasetBrushRole );
819     emit propertiesChanged();
820 }
```

### 9.53.3.67 void AbstractDiagram::setBrush (const QModelIndex & *index*, const QBrush & *brush*) [inherited]

Set the brush to be used, for painting the datapoint at the given index.

#### Parameters:

*index* The datapoint's index in the model.

*brush* The brush to use.

Definition at line 798 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DatasetBrushRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setData().

```
799 {
800     attributesModel()->setData(
801         attributesModel()->mapFromSource( index ),
802         qVariantFromValue( brush ), DatasetBrushRole );
803     emit propertiesChanged();
804 }
```

### 9.53.3.68 void AbstractDiagram::setCoordinatePlane ([AbstractCoordinatePlane](#) \* *plane*) [virtual, inherited]

Set the coordinate plane associated with the diagram.

This determines how coordinates in value space are mapped into pixel space. The chart takes ownership.

#### Returns:

The coordinate plane associated with the diagram.

Reimplemented in [KdChart::AbstractCartesianDiagram](#).

Definition at line 316 of file KdChartAbstractDiagram.cpp.

References [d](#).

Referenced by [KdChart::AbstractCoordinatePlane::addDiagram\(\)](#), [KdChart::AbstractCartesianDiagram::setCoordinatePlane\(\)](#), and [KdChart::AbstractCoordinatePlane::takeDiagram\(\)](#).

```
317 {
318     d->plane = parent;
319 }
```

### 9.53.3.69 void AbstractDiagram::setDataBoundariesDirty () const [protected, inherited]

Definition at line 234 of file KdChartAbstractDiagram.cpp.

References [d](#).

Referenced by [KdChart::AbstractDiagram::dataChanged\(\)](#), [KdChart::Plotter::resize\(\)](#), [KdChart::LineDiagram::resize\(\)](#), [KdChart::BarDiagram::resize\(\)](#), [KdChart::AbstractDiagram::setAttributesModel\(\)](#), [KdChart::AbstractDiagram::setAttributesModelRootIndex\(\)](#), [KdChart::AbstractDiagram::setDatasetDimension\(\)](#), [KdChart::AbstractDiagram::setModel\(\)](#), [KdChart::BarDiagram::setThreeDBarAttributes\(\)](#), [KdChart::Plotter::setThreeDLineAttributes\(\)](#), [KdChart::LineDiagram::setThreeDLineAttributes\(\)](#), [KdChart::Plotter::setType\(\)](#), [KdChart::LineDiagram::setType\(\)](#), and [KdChart::BarDiagram::setType\(\)](#).

```
235 {
236     d->databoundariesDirty = true;
237 }
```

### 9.53.3.70 void AbstractDiagram::setDatasetDimension (int *dimension*) [inherited]

Sets the dataset dimension of the diagram.

#### See also:

[datasetDimension](#).

#### Parameters:

*dimension*

Definition at line 1077 of file KdChartAbstractDiagram.cpp.

References `d`, `KDChart::AbstractDiagram::layoutChanged()`, and `KDChart::AbstractDiagram::setDataBoundariesDirty()`.

Referenced by `KDChart::Widget::setType()`, `KDChart::TernaryLineDiagram::TernaryLineDiagram()`, and `KDChart::TernaryPointDiagram::TernaryPointDiagram()`.

```
1078 {
1079     if ( d->datasetDimension == dimension ) return;
1080     d->datasetDimension = dimension;
1081     setDataBoundariesDirty();
1082     emit layoutChanged( this );
1083 }
```

### 9.53.3.71 void AbstractDiagram::setDataValueAttributes (const [DataValueAttributes](#) & *a*) [inherited]

Set the [DataValueAttributes](#) for all datapoints in the model.

#### Parameters:

*a* The attributes to set.

Definition at line 428 of file `KDChartAbstractDiagram.cpp`.

References `d`, `KDChart::DataValueLabelAttributesRole`, and `KDChart::AbstractDiagram::propertiesChanged()`.

```
429 {
430     d->attributesModel->setModelData( qVariantFromValue( a ), DataValueLabelAttributesRole );
431     emit propertiesChanged();
432 }
```

### 9.53.3.72 void AbstractDiagram::setDataValueAttributes (int *dataset*, const [DataValueAttributes](#) & *a*) [inherited]

Set the [DataValueAttributes](#) for the given dataset.

#### Parameters:

*dataset* The dataset to set the attributes for.

*a* The attributes to set.

Definition at line 400 of file `KDChartAbstractDiagram.cpp`.

References `d`, `KDChart::DataValueLabelAttributesRole`, and `KDChart::AbstractDiagram::propertiesChanged()`.

```
401 {
402     d->attributesModel->setHeaderData(
403         column, Qt::Vertical,
404         qVariantFromValue( a ), DataValueLabelAttributesRole );
405     emit propertiesChanged();
406 }
```



### 9.53.3.73 void AbstractDiagram::setDataValueAttributes (const QModelIndex & *index*, const [DataValueAttributes](#) & *a*) [inherited]

Set the [DataValueAttributes](#) for the given index.

#### Parameters:

*index* The datapoint to set the attributes for.

*a* The attributes to set.

Definition at line 389 of file KDCartAbstractDiagram.cpp.

References [d](#), [KDCart::DataValueLabelAttributesRole](#), and [KDCart::AbstractDiagram::propertiesChanged\(\)](#).

```

391 {
392     d->attributesModel->setData(
393         d->attributesModel->mapFromSource( index ),
394         QVariantFromValue( a ),
395         DataValueLabelAttributesRole );
396     emit propertiesChanged();
397 }
```

### 9.53.3.74 void AbstractPieDiagram::setGranularity (qreal *value*) [inherited]

Set the granularity: the smaller the granularity the more your diagram segments will show facettes instead of rounded segments.

#### Parameters:

*value* the granularity value between 0.05 (one twentieth of a degree) and 36.0 (one tenth of a full circle), other values will be interpreted as 1.0.

Definition at line 64 of file KDCartAbstractPieDiagram.cpp.

References [d](#).

```

65 {
66     d->granularity = value;
67 }
```

### 9.53.3.75 void AbstractDiagram::setHidden (bool *hidden*) [inherited]

Hide (or unhide, resp.

) all datapoints in the model.

#### Note:

Hidden data are still taken into account by the coordinate plane, so neither the grid nor your axes' ranges will change, when you hide data. For totally removing data from KD Chart's view you can use another approach: e.g. you could define a proxy model on top of your data model, and register the proxy model calling [setModel\(\)](#) instead of registering your real data model.

**Parameters:**

***hidden*** The hidden status to set.

Definition at line 359 of file KDChartAbstractDiagram.cpp.

References `d`, `KDChart::AbstractDiagram::dataHidden()`, and `KDChart::DataHiddenRole`.

```

360 {
361     d->attributesModel->setModelData(
362         qVariantFromValue( hidden ),
363         DataHiddenRole );
364     emit dataHidden();
365 }
```

### 9.53.3.76 void AbstractDiagram::setHidden (int *column*, bool *hidden*) [inherited]

Hide (or unhide, resp.

) a dataset.

**Note:**

Hidden data are still taken into account by the coordinate plane, so neither the grid nor your axes' ranges will change, when you hide data. For totally removing data from KD Chart's view you can use another approach: e.g. you could define a proxy model on top of your data model, and register the proxy model calling [setModel\(\)](#) instead of registering your real data model.

**Parameters:**

***column*** The dataset to set the hidden status for.

***hidden*** The hidden status to set.

Definition at line 350 of file KDChartAbstractDiagram.cpp.

References `d`, `KDChart::AbstractDiagram::dataHidden()`, and `KDChart::DataHiddenRole`.

```

351 {
352     d->attributesModel->setHeaderData(
353         column, Qt::Vertical,
354         qVariantFromValue( hidden ),
355         DataHiddenRole );
356     emit dataHidden();
357 }
```

### 9.53.3.77 void AbstractDiagram::setHidden (const QModelIndex & *index*, bool *hidden*) [inherited]

Hide (or unhide, resp.

) a data cell.

**Note:**

Hidden data are still taken into account by the coordinate plane, so neither the grid nor your axes' ranges will change, when you hide data. For totally removing data from KD Chart's view you can use another approach: e.g. you could define a proxy model on top of your data model, and register the proxy model calling [setModel\(\)](#) instead of registering your real data model.

**Parameters:**

***index*** The datapoint to set the hidden status for.

***hidden*** The hidden status to set.

Definition at line 341 of file KDChartAbstractDiagram.cpp.

References `d`, `KDChart::AbstractDiagram::dataHidden()`, and `KDChart::DataHiddenRole`.

```

342 {
343     d->attributesModel->setData(
344         d->attributesModel->mapFromSource( index ),
345         qVariantFromValue( hidden ),
346         DataHiddenRole );
347     emit dataHidden();
348 }
```

### 9.53.3.78 void AbstractDiagram::setModel (QAbstractItemModel \* *model*) [virtual, inherited]

Associate a model with the diagram.

Reimplemented in [KDChart::AbstractCartesianDiagram](#).

Definition at line 239 of file KDChartAbstractDiagram.cpp.

References `d`, `KDChart::AttributesModel::initFrom()`, `KDChart::AbstractDiagram::modelsChanged()`, and `KDChart::AbstractDiagram::setDataBoundariesDirty()`.

Referenced by `KDChart::AbstractCartesianDiagram::setModel()`, and `KDChart::Widget::setType()`.

```

240 {
241     QAbstractItemView::setModel( newModel );
242     AttributesModel* amodel = new PrivateAttributesModel( newModel, this );
243     amodel->initFrom( d->attributesModel );
244     d->setAttributesModel( amodel );
245     scheduleDelayedItemsLayout();
246     setDataBoundariesDirty();
247     emit modelsChanged();
248 }
```

### 9.53.3.79 void AbstractDiagram::setPen (const QPen & *pen*) [inherited]

Set the pen to be used, for painting all datasets in the model.

**Parameters:**

***pen*** The pen to use.

Definition at line 753 of file KDChartAbstractDiagram.cpp.

References `KDChart::AbstractDiagram::attributesModel()`, `KDChart::DatasetPenRole`, `KDChart::AbstractDiagram::propertiesChanged()`, and `KDChart::AttributesModel::setModelData()`.

```

754 {
755     attributesModel()->setModelData(
756         qVariantFromValue( pen ), DatasetPenRole );
757     emit propertiesChanged();
758 }
```

**9.53.3.80 void AbstractDiagram::setPen (int *dataset*, const QPen & *pen*)** [inherited]

Set the pen to be used, for painting the given dataset.

**Parameters:**

*dataset* The dataset's row in the model.

*pen* The pen to use.

Definition at line 760 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::datasetDimension(), KDChart::DatasetPenRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setHeaderData().

```

761 {
762     if( datasetDimension() > 1 )
763         column *= datasetDimension();
764     attributesModel()->setHeaderData(
765         column, Qt::Vertical,
766         qVariantFromValue( pen ),
767         DatasetPenRole );
768     emit propertiesChanged();
769 }
```

**9.53.3.81 void AbstractDiagram::setPen (const QModelIndex & *index*, const QPen & *pen*)** [inherited]

Set the pen to be used, for painting the datapoint at the given index.

**Parameters:**

*index* The datapoint's index in the model.

*pen* The pen to use.

Definition at line 743 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::datasetDimension(), KDChart::DatasetPenRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setData().

```

744 {
745     if( datasetDimension() > 1 )
746         return setPen( index.column(), pen );
747     attributesModel()->setData(
748         attributesModel()->mapFromSource( index ),
749         qVariantFromValue( pen ), DatasetPenRole );
750     emit propertiesChanged();
751 }
```

**9.53.3.82 void AbstractDiagram::setPercentMode (bool *percent*)** [inherited]

Definition at line 456 of file KDChartAbstractDiagram.cpp.

References d, and KDChart::AbstractDiagram::propertiesChanged().

Referenced by KDChart::LineDiagram::setType(), and KDChart::BarDiagram::setType().

```

457 {
458     d->percent = percent;
459     emit propertiesChanged();
460 }

```

### 9.53.3.83 void AbstractPieDiagram::setPieAttributes (int *column*, const [PieAttributes](#) & *a*) [inherited]

Definition at line 95 of file KDChartAbstractPieDiagram.cpp.

References [d](#), [KDChart::AbstractDiagram::layoutChanged\(\)](#), and [KDChart::PieAttributesRole](#).

```

96 {
97     d->attributesModel->setHeaderData(
98         column, Qt::Vertical, qVariantFromValue( attrs ), PieAttributesRole );
99     emit layoutChanged( this );
100 }

```

### 9.53.3.84 void AbstractPieDiagram::setPieAttributes (const [PieAttributes](#) & *a*) [inherited]

Definition at line 89 of file KDChartAbstractPieDiagram.cpp.

References [d](#), [KDChart::AbstractDiagram::layoutChanged\(\)](#), and [KDChart::PieAttributesRole](#).

```

90 {
91     d->attributesModel->setModelData( qVariantFromValue( attrs ), PieAttributesRole );
92     emit layoutChanged( this );
93 }

```

### 9.53.3.85 void RingDiagram::setRelativeThickness (bool *relativeThickness*)

Definition at line 72 of file KDChartRingDiagram.cpp.

References [d](#).

```

73 {
74     d->relativeThickness = relativeThickness;
75 }

```

### 9.53.3.86 void AbstractDiagram::setRootIndex (const QModelIndex & *idx*) [virtual, inherited]

Set the root index in the model, where the diagram starts referencing data for display.

[reimplemented]

Reimplemented in [KDChart::AbstractCartesianDiagram](#).

Definition at line 286 of file KDChartAbstractDiagram.cpp.

References [d](#), and [KDChart::AbstractDiagram::setAttributesModelRootIndex\(\)](#).

Referenced by [KDChart::AbstractCartesianDiagram::setRootIndex\(\)](#).

```

287 {
288     QAbstractItemView::setRootIndex(idx);
289     setAttributesModelRootIndex( d->attributesModel->mapFromSource(idx) );
290 }

```

### 9.53.3.87 void AbstractDiagram::setSelection (const QRect & rect, QItemSelectionModel::SelectionFlags command) [virtual, inherited]

[reimplemented]

Definition at line 959 of file KDChartAbstractDiagram.cpp.

References d.

```

960 {
961     const QModelIndexList indexes = d->indexesIn( rect );
962     QItemSelection selection;
963     KDAB_FOREACH( const QModelIndex& index, indexes )
964     {
965         selection.append( QItemSelectionRange( index ) );
966     }
967     selectionModel()->select( selection, command );
968 }

```

### 9.53.3.88 void AbstractPieDiagram::setStartPosition (int degrees) [inherited]

**Deprecated**

Use [PolarCoordinatePlane::setStartPosition\( qreal degrees \)](#) instead.

Definition at line 77 of file KDChartAbstractPieDiagram.cpp.

```

78 {
79     Q_UNUSED( degrees );
80     qWarning() << "Deprecated AbstractPieDiagram::setStartPosition() called, setting ignored.";
81 }

```

### 9.53.3.89 void AbstractPieDiagram::setThreeDPieAttributes (const QModelIndex & index, const ThreeDPieAttributes & a) [inherited]

Definition at line 144 of file KDChartAbstractPieDiagram.cpp.

References KDChart::AbstractDiagram::layoutChanged(), and KDChart::ThreeDPieAttributesRole.

```

145 {
146     model()->setData( index, QVariantFromValue( tda ), ThreeDPieAttributesRole );
147     emit layoutChanged( this );
148 }

```

### 9.53.3.90 void AbstractPieDiagram::setThreeDPieAttributes (int column, const ThreeDPieAttributes & a) [inherited]

Definition at line 137 of file KDChartAbstractPieDiagram.cpp.

References d, KDChart::AbstractDiagram::layoutChanged(), and KDChart::ThreeDPieAttributesRole.

```

138 {
139     d->attributesModel->setHeaderData(
140         column, Qt::Vertical, qVariantFromValue( tda ), ThreeDPieAttributesRole );
141     emit layoutChanged( this );
142 }

```

#### 9.53.3.91 void AbstractPieDiagram::setThreeDPieAttributes (const [ThreeDPieAttributes](#) & a) [inherited]

Definition at line 131 of file KDChartAbstractPieDiagram.cpp.

References [d](#), [KDChart::AbstractDiagram::layoutChanged\(\)](#), and [KDChart::ThreeDPieAttributesRole](#).

```

132 {
133     d->attributesModel->setModelData( qVariantFromValue( tda ), ThreeDPieAttributesRole );
134     emit layoutChanged( this );
135 }

```

#### 9.53.3.92 void AbstractDiagram::setUnitPrefix (const QString & *prefix*, Qt::Orientation *orientation*) [inherited]

Sets the unit prefix for all values.

##### Parameters:

*prefix* the prefix to be set  
*orientation* the orientation of the axis to set

Definition at line 864 of file KDChartAbstractDiagram.cpp.

References [d](#).

```

865 {
866     d->unitPrefix[ orientation ] = prefix;
867 }

```

#### 9.53.3.93 void AbstractDiagram::setUnitPrefix (const QString & *prefix*, int *column*, Qt::Orientation *orientation*) [inherited]

Sets the unit prefix for one value.

##### Parameters:

*prefix* the prefix to be set  
*column* the value using that prefix  
*orientation* the orientation of the axis to set

Definition at line 854 of file KDChartAbstractDiagram.cpp.

References [d](#).

```

855 {
856     d->unitPrefixMap[ column ][ orientation ]= prefix;
857 }

```

### 9.53.3.94 void AbstractDiagram::setUnitSuffix (const QString & *suffix*, Qt::Orientation *orientation*) [inherited]

Sets the unit suffix for all values.

#### Parameters:

*suffix* the suffix to be set

*orientation* the orientation of the axis to set

Definition at line 885 of file KDChartAbstractDiagram.cpp.

References d.

```
886 {
887     d->unitSuffix[ orientation ] = suffix;
888 }
```

### 9.53.3.95 void AbstractDiagram::setUnitSuffix (const QString & *suffix*, int *column*, Qt::Orientation *orientation*) [inherited]

Sets the unit suffix for one value.

#### Parameters:

*suffix* the suffix to be set

*column* the value using that suffix

*orientation* the orientation of the axis to set

Definition at line 875 of file KDChartAbstractDiagram.cpp.

References d.

```
876 {
877     d->unitSuffixMap[ column ][ orientation ]= suffix;
878 }
```

### 9.53.3.96 int AbstractPieDiagram::startPosition () const [inherited]

#### Deprecated

Use qreal [PolarCoordinatePlane::startPosition](#) instead.

Definition at line 83 of file KDChartAbstractPieDiagram.cpp.

```
84 {
85     qWarning() << "Deprecated AbstractPieDiagram::startPosition() called.";
86     return 0;
87 }
```



**9.53.3.97 ThreeDPieAttributes AbstractPieDiagram::threeDPieAttributes (const QModelIndex & index) const** [inherited]

Definition at line 170 of file KDChartAbstractPieDiagram.cpp.

References d, and KDChart::ThreeDPieAttributesRole.

```
171 {
172     return qVariantValue<ThreeDPieAttributes>(
173         d->attributesModel->data(
174             d->attributesModel->mapFromSource( index ),
175             ThreeDPieAttributesRole ) );
176 }
```

**9.53.3.98 ThreeDPieAttributes AbstractPieDiagram::threeDPieAttributes (int column) const** [inherited]

Definition at line 162 of file KDChartAbstractPieDiagram.cpp.

References KDChart::AbstractDiagram::columnToIndex(), d, and KDChart::ThreeDPieAttributesRole.

```
163 {
164     return qVariantValue<ThreeDPieAttributes>(
165         d->attributesModel->data(
166             d->attributesModel->mapFromSource( columnToIndex( column ) ).column(),
167             ThreeDPieAttributesRole ) );
168 }
```

**9.53.3.99 ThreeDPieAttributes AbstractPieDiagram::threeDPieAttributes () const** [inherited]

Definition at line 153 of file KDChartAbstractPieDiagram.cpp.

References d, and KDChart::ThreeDPieAttributesRole.

Referenced by KDChart::PieDiagram::paint().

```
154 {
155     return qVariantValue<ThreeDPieAttributes>(
156         d->attributesModel->data( ThreeDPieAttributesRole ) );
157 }
```

**9.53.3.100 QString AbstractDiagram::unitPrefix (Qt::Orientation orientation) const** [inherited]

Returns the global unit prefix.

**Parameters:**

*orientation* the orientation of the axis

**Returns:**

the unit prefix

Definition at line 908 of file KDChartAbstractDiagram.cpp.

References d.

```
909 {  
910     return d->unitPrefix[ orientation ];  
911 }
```

#### 9.53.3.101 QString AbstractDiagram::unitPrefix (int *column*, Qt::Orientation *orientation*, bool *fallback* = false) const [inherited]

Returns the unit prefix for a special value.

##### Parameters:

*column* the value which's prefix is requested

*orientation* the orientation of the axis

*fallback* if true, the global prefix is return when no specific one is set for that value

##### Returns:

the unit prefix

Definition at line 897 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::itemRowLabels(), and KDChart::CartesianAxis::paintCtx().

```
898 {  
899     if( !fallback || d->unitPrefixMap[ column ].contains( orientation ) )  
900         return d->unitPrefixMap[ column ][ orientation ];  
901     return d->unitPrefix[ orientation ];  
902 }
```

#### 9.53.3.102 QString AbstractDiagram::unitSuffix (Qt::Orientation *orientation*) const [inherited]

Returns the global unit suffix.

##### Parameters:

*orientation* the orientation of the axis

##### Returns:

the unit siffix

Definition at line 931 of file KDChartAbstractDiagram.cpp.

References d.

```
932 {  
933     return d->unitSuffix[ orientation ];  
934 }
```

### 9.53.3.103 QString AbstractDiagram::unitSuffix (int *column*, Qt::Orientation *orientation*, bool *fallback* = false) const [inherited]

Returns the unit suffix for a special value.

#### Parameters:

***column*** the value which's suffix is requested

***orientation*** the orientation of the axis

***fallback*** if true, the global suffix is return when no specific one is set for that value

#### Returns:

the unit suffix

Definition at line 920 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::itemRowLabels(), and KDChart::CartesianAxis::paintCtx().

```

921 {
922     if( !fallback || d->unitSuffixMap[ column ].contains( orientation ) )
923         return d->unitSuffixMap[ column ][ orientation ];
924     return d->unitSuffix[ orientation ];
925 }
```

### 9.53.3.104 void AbstractDiagram::update () const [inherited]

Definition at line 1091 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::doItemsLayout().

```

1092 {
1093     //qDebug("KDChart::AbstractDiagram::update() called");
1094     if( d->plane )
1095         d->plane->update();
1096 }
```

### 9.53.3.105 void KDChart::AbstractDiagram::useDefaultColors () [inherited]

Set the palette to be used, for painting datasets to the default palette.

#### See also:

[KDChart::Palette](#). FIXME: fold into one usePalette ([KDChart::Palette&](#)) method

Definition at line 974 of file KDChartAbstractDiagram.cpp.

References d, and KDChart::AttributesModel::PaletteTypeDefault.

```

975 {
976     d->attributesModel->setPaletteType( AttributesModel::PaletteTypeDefault );
977 }
```

**9.53.3.106 void KDChart::AbstractDiagram::useRainbowColors ()** [inherited]

Set the palette to be used, for painting datasets to the rainbow palette.

**See also:**

[KDChart::Palette](#).

Definition at line 984 of file KDChartAbstractDiagram.cpp.

References [d](#), and [KDChart::AttributesModel::PaletteTypeRainbow](#).

```
985 {  
986     d->attributesModel->setPaletteType( AttributesModel::PaletteTypeRainbow );  
987 }
```

**9.53.3.107 bool AbstractDiagram::usesExternalAttributesModel () const** [virtual, inherited]

Returns whether the diagram is using its own built-in attributes model or an attributes model that was set via [setAttributesModel](#).

**See also:**

[setAttributesModel](#)

Definition at line 274 of file KDChartAbstractDiagram.cpp.

References [d](#).

```
275 {  
276     return d->usesExternalAttributesModel();  
277 }
```

**9.53.3.108 void KDChart::AbstractDiagram::useSubduedColors ()** [inherited]

Set the palette to be used, for painting datasets to the subdued palette.

**See also:**

[KDChart::Palette](#).

Definition at line 979 of file KDChartAbstractDiagram.cpp.

References [d](#), and [KDChart::AttributesModel::PaletteTypeSubdued](#).

```
980 {  
981     d->attributesModel->setPaletteType( AttributesModel::PaletteTypeSubdued );  
982 }
```

### 9.53.3.109 double AbstractDiagram::valueForCell (int *row*, int *column*) const [protected, inherited]

Helper method, retrieving the data value (DisplayRole) for a given row and column.

#### Parameters:

***row*** The row to query.

***column*** The column to query.

#### Returns:

The value of the display role at the given row and column as a double.

#### Deprecated

Definition at line 1085 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModelRootIndex(), and d.

```
1086 {
1087     return d->attributesModel->data(
1088         d->attributesModel->index( row, column, attributesModelRootIndex() ) ).toDouble();
1089 }
```

### 9.53.3.110 double RingDiagram::valueTotals () const [virtual]

[reimplemented]

Implements [KDChart::AbstractPolarDiagram](#).

Definition at line 136 of file KDChartRingDiagram.cpp.

```
137 {
138     double total = 0;
139     const int colCount = model()->columnCount(rootIndex());
140     for ( int j=0; j<colCount; ++j ) {
141         total += model()->data( model()->index( 0, j, rootIndex() ) ).toDouble();
142     }
143     return total;
144 }
```

### 9.53.3.111 int AbstractDiagram::verticalOffset () const [virtual, inherited]

[reimplemented]

Definition at line 953 of file KDChartAbstractDiagram.cpp.

```
954 { return 0; }
```

**9.53.3.112** **QRect AbstractDiagram::visualRect (const QModelIndex & *index*) const** [virtual, inherited]

[reimplemented]

Definition at line 937 of file KDChartAbstractDiagram.cpp.

```
938 {  
939     return QRect();  
940 }
```

**9.53.3.113** **QRegion AbstractDiagram::visualRegionForSelection (const QItemSelection & *selection*) const** [virtual, inherited]

[reimplemented]

Definition at line 970 of file KDChartAbstractDiagram.cpp.

```
971 { return QRegion(); }
```

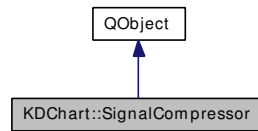
The documentation for this class was generated from the following files:

- [KDChartRingDiagram.h](#)
- [KDChartRingDiagram.cpp](#)

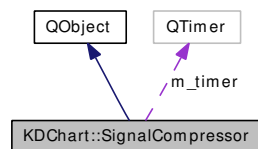
## 9.54 KDChart::SignalCompressor Class Reference

```
#include <KDChartSignalCompressor.h>
```

Inheritance diagram for KDChart::SignalCompressor:



Collaboration diagram for KDChart::SignalCompressor:



### 9.54.1 Detailed Description

[SignalCompressor](#) compresses signals where the same signal needs to be emitted by several pieces of the code, but only one of the signals should be received at the end.

Usage: create a object of [SignalCompressor](#), and give it the name and object of the signal it is supposed to manage instead of emitting the signal, call [emitSignal\(\)](#) on the compressor the signal will only be emitted once, and that is after the current call stack ends and returns to the event loop

With the current implementation, the class changes the semantics of signals to be a queued connection. If that is not wanted, another compression algorithm needs to be implemented. Also, at the moment, only nullary signals are supported, as parameters could not be compressed. A typical use of the class is to compress update notifications. This class is not part of the published [KDChart](#) API.

Definition at line 58 of file KDChartSignalCompressor.h.

### Public Slots

- void [emitSignal](#) ()

### Signals

- void [finallyEmit](#) ()

### Public Member Functions

- [SignalCompressor](#) ([QObject](#) \*receiver, const char \*signal, [QObject](#) \*parent=0)

## 9.54.2 Constructor & Destructor Documentation

### 9.54.2.1 `SignalCompressor::SignalCompressor (QObject * receiver, const char * signal, QObject * parent = 0)`

Definition at line 34 of file `KDChartSignalCompressor.cpp`.

References `finallyEmit()`.

```
36      : QObject( parent )
37  {
38      connect( this, SIGNAL( finallyEmit() ), receiver, signal );
39      connect( &m_timer, SIGNAL( timeout() ), SLOT( nowGoAlready() ) );
40      m_timer.setSingleShot( true );
41      // m_timer.setIntervall( 0 ); // default, just to know...
42  }
```

## 9.54.3 Member Function Documentation

### 9.54.3.1 `void SignalCompressor::emitSignal () [slot]`

Definition at line 44 of file `KDChartSignalCompressor.cpp`.

```
45  {
46      if ( !m_timer.isActive() ) m_timer.start();
47  }
```

### 9.54.3.2 `void KDChart::SignalCompressor::finallyEmit () [signal]`

Referenced by `SignalCompressor()`.

The documentation for this class was generated from the following files:

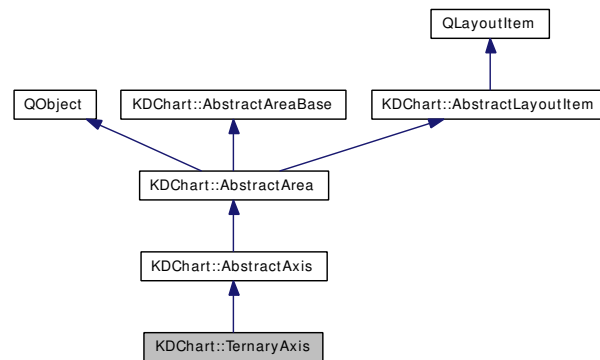
- [KDChartSignalCompressor.h](#)
- [KDChartSignalCompressor.cpp](#)



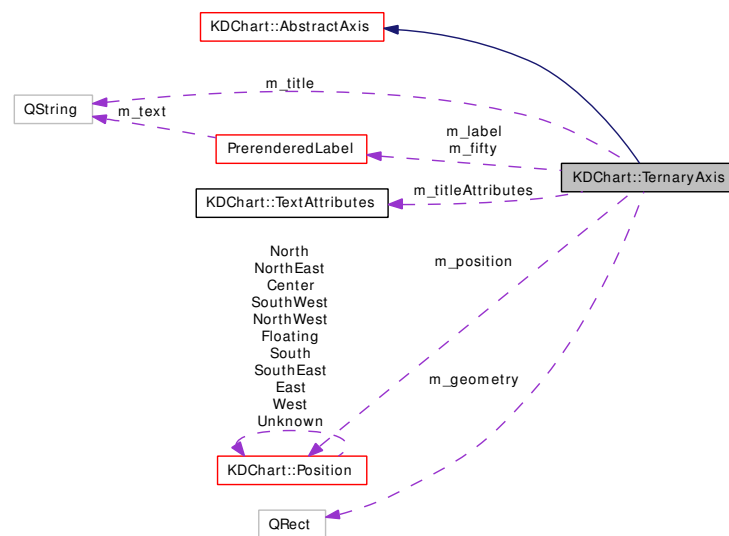
## 9.55 KDChart::TernaryAxis Class Reference

```
#include <KDChartTernaryAxis.h>
```

Inheritance diagram for KDChart::TernaryAxis:



Collaboration diagram for KDChart::TernaryAxis:



### 9.55.1 Detailed Description

The class for ternary axes.

Definition at line 47 of file KDChartTernaryAxis.h.

#### Public Slots

- void [update](#) ()

## Signals

- void [positionChanged](#) ([AbstractArea](#) \*)

## Public Member Functions

- void [alignToReferencePoint](#) (const [RelativePosition](#) &position)
- [BackgroundAttributes](#) [backgroundAttributes](#) () const
- virtual int [bottomOverlap](#) (bool doNotRecalculate=false) const  
*This is called at layout time by [KDChart::AutoSpacerLayoutItem::sizeHint\(\)](#).*
- bool [compare](#) (const [AbstractAreaBase](#) \*other) const  
*Returns true if both areas have the same settings.*
- bool [compare](#) (const [AbstractAxis](#) \*other) const  
*Returns true if both axes have the same settings.*
- virtual void [connectSignals](#) ()  
*Wiring the signal/slot connections.*
- const [AbstractCoordinatePlane](#) \* [coordinatePlane](#) () const  
*Convenience function, returns the coordinate plane, in which this axis is used.*
- void [createObserver](#) ([AbstractDiagram](#) \*diagram)
- virtual const QString [customizedLabel](#) (const QString &label) const  
*Implement this method if you want to adjust axis labels before they are printed.*
- void [deleteObserver](#) ([AbstractDiagram](#) \*diagram)
- const [AbstractDiagram](#) \* [diagram](#) () const
- virtual Qt::Orientations [expandingDirections](#) () const
- [FrameAttributes](#) [frameAttributes](#) () const
- virtual QRect [geometry](#) () const
- void [getFrameLeadings](#) (int &left, int &top, int &right, int &bottom) const
- bool [hasDefaultTitleTextAttributes](#) () const
- virtual bool [isEmpty](#) () const
- QStringList [labels](#) () const  
*Returns a list of strings, that are used as axis labels, as set via [setLabels](#).*
- virtual int [leftOverlap](#) (bool doNotRecalculate=false) const  
*This is called at layout time by [KDChart::AutoSpacerLayoutItem::sizeHint\(\)](#).*
- virtual QSize [maximumSize](#) () const
- virtual QSize [minimumSize](#) () const
- bool [observedBy](#) ([AbstractDiagram](#) \*diagram) const
- virtual void [paint](#) (QPainter \*)
- virtual void [paintAll](#) (QPainter &)  
*Call [paintAll](#), if you want the background and the frame to be drawn before the normal [paint\(\)](#) is invoked automatically.*
- virtual void [paintBackground](#) (QPainter &painter, const QRect &rectangle)

- virtual void [paintCtx](#) ([PaintContext](#) \*)  
*Default impl: Paint the complete item using its layouted position and size.*
- virtual void [paintFrame](#) (QPainter &painter, const QRect &rectangle)
- virtual void [paintIntoRect](#) (QPainter &painter, const QRect &rect)  
*Draws the background and frame, then calls [paint\(\)](#).*
- QLayout \* [parentLayout](#) ()
- virtual const [Position](#) [position](#) () const
- void [removeFromParentLayout](#) ()
- QPair< QSizeF, QSizeF > [requiredMargins](#) () const
- void [resetTitleTextAttributes](#) ()
- virtual int [rightOverlap](#) (bool doNotRecalculate=false) const  
*This is called at layout time by [KDChart::AutoSpacerLayoutItem::sizeHint\(\)](#).*
- void [setBackgroundAttributes](#) (const [BackgroundAttributes](#) &a)
- void [setFrameAttributes](#) (const [FrameAttributes](#) &a)
- virtual void [setGeometry](#) (const QRect &rect)
- void [setLabels](#) (const QStringList &list)  
*Use this to specify your own set of strings, to be used as axis labels.*
- void [setParentLayout](#) (QLayout \*lay)
- virtual void [setParentWidget](#) (QWidget \*widget)  
*Inform the item about its widget: This enables the item, to trigger that widget's update, whenever the size of the item's contents has changed.*
- virtual void [setPosition](#) ([Position](#) p)
- void [setShortLabels](#) (const QStringList &list)  
*Use this to specify your own set of strings, to be used as axis labels, in case the normal labels are too long.*
- void [setTextAttributes](#) (const [TextAttributes](#) &a)  
*Use this to specify the text attributes to be used for axis labels.*
- void [setTitleText](#) (const QString &text)
- void [setTitleTextAttributes](#) (const [TextAttributes](#) &a)
- QStringList [shortLabels](#) () const  
*Returns a list of strings, that are used as axis labels, as set via [setShortLabels](#).*
- virtual QSize [sizeHint](#) () const
- virtual void [sizeHintChanged](#) () const  
*Report changed size hint: ask the parent widget to recalculate the layout.*
- [TernaryAxis](#) ([AbstractTernaryDiagram](#) \*diagram=0)
- [TextAttributes](#) [textAttributes](#) () const  
*Returns the text attributes to be used for axis labels.*
- QString [titleText](#) () const
- [TextAttributes](#) [titleTextAttributes](#) () const
- virtual int [topOverlap](#) (bool doNotRecalculate=false) const  
*This is called at layout time by [KDChart::AutoSpacerLayoutItem::sizeHint\(\)](#).*
- [~TernaryAxis](#) ()

## Static Public Member Functions

- static void [paintBackgroundAttributes](#) (QPainter &painter, const QRect &rectangle, const [KDChart::BackgroundAttributes](#) &attributes)
- static void [paintFrameAttributes](#) (QPainter &painter, const QRect &rectangle, const [KDChart::FrameAttributes](#) &attributes)

## Protected Slots

- virtual void [delayedInit](#) ()  
*called for initializing after the c'tor has completed*

## Protected Member Functions

- virtual QRect [areaGeometry](#) () const
- QRect [innerRect](#) () const
- virtual void [positionHasChanged](#) ()

## Protected Attributes

- QWidget \* [mParent](#)
- QLayout \* [mParentLayout](#)

## 9.55.2 Constructor & Destructor Documentation

### 9.55.2.1 TernaryAxis::TernaryAxis ([AbstractTernaryDiagram](#) \* *diagram* = 0) [explicit]

Definition at line 49 of file `KDChartTernaryAxis.cpp`.

References `KDChart::AbstractAxis::diagram()`, `KDChartEnums::PositionSouth`, `resetTitleTextAttributes()`, `setPosition()`, and `PrerenderedLabel::setText()`.

```

50     : AbstractAxis( diagram )
51     , m_position( KDChartEnums::PositionUnknown )
52     , m_label( new PrerenderedLabel )
53     , m_fifty( new PrerenderedLabel )
54 {
55     resetTitleTextAttributes();
56     setPosition( KDChartEnums::PositionSouth ); // arbitrary
57     m_fifty->setText( QObject::tr( "50%" ) ); // const
58     // FIXME is this consistent with other diagram/axis/plane implementations?
59     diagram->addAxis( this );
60 }
```

### 9.55.2.2 TernaryAxis::~~TernaryAxis ()

Definition at line 62 of file `KDChartTernaryAxis.cpp`.

```

63 {
64     delete m_label; m_label = 0;
65     delete m_label; m_fifty = 0;
66 }
```

### 9.55.3 Member Function Documentation

#### 9.55.3.1 void AbstractAreaBase::alignToReferencePoint (const [RelativePosition](#) & position) [inherited]

Definition at line 90 of file KDCartAbstractAreaBase.cpp.

```

91 {
92     Q_UNUSED( position );
93     // PENDING(kalle) FIXME
94     qWarning( "Sorry, not implemented: void AbstractAreaBase::alignToReferencePoint( const RelativePosi
95 }
```

#### 9.55.3.2 QRect AbstractArea::areaGeometry () const [protected, virtual, inherited]

Implements [KDCart::AbstractAreaBase](#).

Definition at line 150 of file KDCartAbstractArea.cpp.

Referenced by KDCart::CartesianCoordinatePlane::drawingArea(), KDCart::TernaryCoordinatePlane::layoutDiagrams(), KDCart::PolarCoordinatePlane::layoutDiagrams(), KDCart::TernaryCoordinatePlane::paint(), KDCart::CartesianAxis::paint(), KDCart::AbstractArea::paintAll(), and KDCart::CartesianAxis::paintCtx().

```

151 {
152     return geometry();
153 }
```

#### 9.55.3.3 [BackgroundAttributes](#) AbstractAreaBase::backgroundAttributes () const [inherited]

Definition at line 120 of file KDCartAbstractAreaBase.cpp.

References d.

Referenced by KDCart::AbstractAreaBase::compare(), and updateCommonBrush().

```

121 {
122     return d->backgroundAttributes;
123 }
```

#### 9.55.3.4 int AbstractArea::bottomOverlap (bool *doNotRecalculate* = false) const [virtual, inherited]

This is called at layout time by KDCart::AutoSpacerLayoutItem::sizeHint().

The method triggers AbstractArea::sizeHint() to find out the amount of overlap at the bottom edge of the area.

#### Note:

The default implementation is not using any caching, it might make sense to implement a more sophisticated solution for derived classes that have complex work to do in sizeHint(). All we have here is a primitive flag to be set by the caller if it is sure that no sizeHint() needs to be called.

Definition at line 101 of file KDChartAbstractArea.cpp.

References d.

Referenced by KDChart::CartesianAxis::maximumSize().

```

102 {
103     // Re-calculate the sizes,
104     // so we also get the amountOf..Overlap members set newly:
105     if( ! doNotRecalculate )
106         sizeHint();
107     return d->amountOfBottomOverlap;
108 }
```

#### 9.55.3.5 bool AbstractAreaBase::compare (const AbstractAreaBase \* other) const [inherited]

Returns true if both areas have the same settings.

Definition at line 75 of file KDChartAbstractAreaBase.cpp.

References KDChart::AbstractAreaBase::backgroundAttributes(), and KDChart::AbstractAreaBase::frameAttributes().

```

76 {
77     if( other == this ) return true;
78     if( ! other ){
79         //qDebug() << "CartesianAxis::compare() cannot compare to Null pointer";
80         return false;
81     }
82     /*
83     qDebug() << "AbstractAreaBase:" << (frameAttributes() == other->frameAttributes())
84         << (backgroundAttributes() == other->backgroundAttributes()) << "\n";
85     */
86     return (frameAttributes() == other->frameAttributes()) &&
87         (backgroundAttributes() == other->backgroundAttributes());
88 }
```

#### 9.55.3.6 bool AbstractAxis::compare (const AbstractAxis \* other) const [inherited]

Returns true if both axes have the same settings.

Definition at line 142 of file KDChartAbstractAxis.cpp.

References KDChart::AbstractAxis::labels(), KDChart::AbstractAxis::shortLabels(), and KDChart::AbstractAxis::textAttributes().

```

143 {
144     if( other == this ) return true;
145     if( ! other ){
146         //qDebug() << "CartesianAxis::compare() cannot compare to Null pointer";
147         return false;
148     }
149     /*
150     qDebug() << (textAttributes() == other->textAttributes());
151     qDebug() << (labels() == other->labels());
152     qDebug() << (shortLabels() == other->shortLabels());
153     */
154     return ( static_cast<const AbstractAreaBase*>(this)->compare( other ) ) &&
155         (textAttributes() == other->textAttributes()) &&
```

```

156         (labels() == other->labels()) &&
157         (shortLabels() == other->shortLabels());
158     }

```

### 9.55.3.7 void AbstractAxis::connectSignals () [virtual, inherited]

Wiring the signal/slot connections.

This method gets called automatically, each time, when you assign the axis to a diagram, either by passing a diagram\* to the c'tor, or by calling the diagram's setAxis method, resp.

If overwriting this method in derived classes, make sure to call this base method [AbstractAxis::connectSignals\(\)](#), so your axis gets connected to the diagram's built-in signals.

See also:

[AbstractCartesianDiagram::addAxis\(\)](#)

Definition at line 211 of file KDChartAbstractAxis.cpp.

References [d](#), and [KDChart::AbstractAxis::update\(\)](#).

Referenced by [KDChart::AbstractAxis::createObserver\(\)](#).

```

212 {
213     if( d->observer ) {
214         connect( d->observer, SIGNAL( diagramDataChanged( AbstractDiagram *) ),
215                 this, SLOT( update() ) );
216     }
217 }

```

### 9.55.3.8 const [AbstractCoordinatePlane](#) \* AbstractAxis::coordinatePlane () const [inherited]

Convenience function, returns the coordinate plane, in which this axis is used.

If the axis is not used in a coordinate plane, the return value is Zero.

Definition at line 324 of file KDChartAbstractAxis.cpp.

References [d](#).

Referenced by [KDChart::CartesianCoordinatePlane::sharedAxisMasterPlane\(\)](#).

```

325 {
326     if( d->diagram() )
327         return d->diagram()->coordinatePlane();
328     return 0;
329 }

```

### 9.55.3.9 void AbstractAxis::createObserver ([AbstractDiagram](#) \* *diagram*) [inherited]

Definition at line 177 of file KDChartAbstractAxis.cpp.

References [KDChart::AbstractAxis::connectSignals\(\)](#), [d](#), and [KDChart::AbstractAxis::diagram\(\)](#).

```

178 {
179     if( d->setDiagram( diagram ) )
180         connectSignals();
181 }

```

#### 9.55.3.10 `const QString AbstractAxis::customizedLabel (const QString & label) const` [virtual, inherited]

Implement this method if you want to adjust axis labels before they are printed.

KD Chart is calling this method immediately before drawing the text, this means: What you return here will be drawn without further modifications.

##### Parameters:

*label* The text of the label as KD Chart has calculated it automatically (or as it was taken from a QStringList provided by you, resp.)

##### Returns:

The text to be drawn. By default this is the same as `label`.

Definition at line 161 of file `KDChartAbstractAxis.cpp`.

Referenced by `KDChart::CartesianAxis::maximumSize()`, and `KDChart::CartesianAxis::paintCtx()`.

```

162 {
163     return label;
164 }

```

#### 9.55.3.11 `void AbstractAxis::delayedInit ()` [protected, virtual, slot, inherited]

called for initializing after the c'tor has completed

Definition at line 134 of file `KDChartAbstractAxis.cpp`.

References `d`.

Referenced by `KDChart::AbstractAxis::AbstractAxis()`.

```

135 {
136     // We call setDiagram() here, because the c'tor of Private
137     // only has stored the pointers, but it did not call setDiagram().
138     if( d )
139         d->setDiagram( 0, true /* delayedInit */ );
140 }

```

#### 9.55.3.12 `void AbstractAxis::deleteObserver (AbstractDiagram * diagram)` [inherited]

Definition at line 193 of file `KDChartAbstractAxis.cpp`.

References `d`, and `KDChart::AbstractAxis::diagram()`.

Referenced by `KDChart::AbstractCartesianDiagram::takeAxis()`, and `KDChart::AbstractCartesianDiagram::~~AbstractCartesianDiagram()`.

```

194 {
195     d->unsetDiagram( diagram );
196 }

```



**9.55.3.13** `const AbstractDiagram * KDChart::AbstractAxis::diagram () const` [inherited]

Definition at line 331 of file KDChartAbstractAxis.cpp.

References [d](#).

Referenced by [KDChart::AbstractAxis::createObserver\(\)](#), [KDChart::AbstractAxis::deleteObserver\(\)](#), [KDChart::AbstractAxis::observedBy\(\)](#), [KDChart::CartesianAxis::paintCtx\(\)](#), [TernaryAxis\(\)](#), and [KDChart::CartesianAxis::~~CartesianAxis\(\)](#).

```
332 {  
333     return d->diagram();  
334 }
```

**9.55.3.14** `Qt::Orientations TernaryAxis::expandingDirections () const` [virtual]

Definition at line 130 of file KDChartTernaryAxis.cpp.

```
131 {  
132     return Qt::Vertical | Qt::Horizontal;  
133 }
```

**9.55.3.15** `FrameAttributes AbstractAreaBase::frameAttributes () const` [inherited]

Definition at line 106 of file KDChartAbstractAreaBase.cpp.

References [d](#).

Referenced by [KDChart::Legend::clone\(\)](#), [KDChart::AbstractAreaBase::compare\(\)](#), and [updateCommonBrush\(\)](#).

```
107 {  
108     return d->frameAttributes;  
109 }
```

**9.55.3.16** `QRect TernaryAxis::geometry () const` [virtual]

Implements [KDChart::AbstractAxis](#).

Definition at line 104 of file KDChartTernaryAxis.cpp.

```
105 {  
106     return m_geometry;  
107 }
```

**9.55.3.17** `void AbstractAreaBase::getFrameLeadings (int & left, int & top, int & right, int & bottom) const` [inherited]

Definition at line 212 of file KDChartAbstractAreaBase.cpp.

References [d](#).

Referenced by [KDChart::AbstractAreaBase::innerRect\(\)](#), and [KDChart::AbstractAreaWidget::paintAll\(\)](#).

```

213 {
214     if( d && d->frameAttributes.isVisible() ){
215         const int padding = qMax( d->frameAttributes.padding(), 0 );
216         left    = padding;
217         top     = padding;
218         right   = padding;
219         bottom  = padding;
220     }else{
221         left    = 0;
222         top     = 0;
223         right   = 0;
224         bottom  = 0;
225     }
226 }

```

### 9.55.3.18 bool TernaryAxis::hasDefaultTitleTextAttributes () const

Definition at line 201 of file KDChartTernaryAxis.cpp.

```

202 {
203     TextAttributes a;
204     return m_titleAttributes == a;
205 }

```

### 9.55.3.19 QRect AbstractAreaBase::innerRect () const [protected, inherited]

Definition at line 228 of file KDChartAbstractAreaBase.cpp.

References KDChart::AbstractAreaBase::areaGeometry(), and KDChart::AbstractAreaBase::getFrameLeadings().

Referenced by KDChart::TextArea::paintAll(), and KDChart::AbstractArea::paintAll().

```

229 {
230     int left;
231     int top;
232     int right;
233     int bottom;
234     getFrameLeadings( left, top, right, bottom );
235     return
236         QRect( QPoint(0,0), areaGeometry().size() )
237             .adjusted( left, top, -right, -bottom );
238 }

```

### 9.55.3.20 bool TernaryAxis::isEmpty () const [virtual]

Definition at line 98 of file KDChartTernaryAxis.cpp.

```

99 {
100     // todo: what's this method for?
101     return false;
102 }

```

**9.55.3.21 QStringList AbstractAxis::labels () const** [inherited]

Returns a list of strings, that are used as axis labels, as set via setLabels.

**See also:**

[setLabels](#)

Definition at line 281 of file KDChartAbstractAxis.cpp.

References d.

Referenced by KDChart::AbstractAxis::compare(), KDChart::CartesianAxis::maximumSize(), paintCtx(), and KDChart::CartesianAxis::paintCtx().

```
282 {  
283     return d->hardLabels;  
284 }
```

**9.55.3.22 int AbstractArea::leftOverlap (bool *doNotRecalculate* = false) const** [virtual, inherited]

This is called at layout time by [KDChart::AutoSpacerLayoutItem::sizeHint\(\)](#).

The method triggers AbstractArea::sizeHint() to find out the amount of overlap at the left edge of the area.

**Note:**

The default implementation is not using any caching, it might make sense to implement a more sophisticated solution for derived classes that have complex work to do in sizeHint(). All we have here is a primitive flag to be set by the caller if it is sure that no sizeHint() needs to be called.

Definition at line 77 of file KDChartAbstractArea.cpp.

References d.

Referenced by KDChart::CartesianAxis::maximumSize().

```
78 {  
79     // Re-calculate the sizes,  
80     // so we also get the amountOf..Overlap members set newly:  
81     if( ! doNotRecalculate )  
82         sizeHint();  
83     return d->amountOfLeftOverlap;  
84 }
```

**9.55.3.23 QSize TernaryAxis::maximumSize () const** [virtual]

Definition at line 120 of file KDChartTernaryAxis.cpp.

```
121 {  
122     return QSize( 300, 200 );  
123 }
```

**9.55.3.24** `QSize TernaryAxis::minimumSize () const` [virtual]

Definition at line 114 of file KDChartTernaryAxis.cpp.

```
115 {
116     // todo: return realistic sizes
117     return QSize( 100, 100 );
118 }
```

**9.55.3.25** `bool KDChart::AbstractAxis::observedBy (AbstractDiagram * diagram) const`  
[inherited]

Definition at line 336 of file KDChartAbstractAxis.cpp.

References [d](#), and [KDChart::AbstractAxis::diagram\(\)](#).

```
337 {
338     return d->hasDiagram( diagram );
339 }
```

**9.55.3.26** `void TernaryAxis::paint (QPainter *)` [virtual]

Implements [KDChart::AbstractLayoutItem](#).

Definition at line 73 of file KDChartTernaryAxis.cpp.

```
74 {
75     // not used
76 }
```

**9.55.3.27** `void TernaryAxis::paintAll (QPainter &)` [virtual]

Call `paintAll`, if you want the background and the frame to be drawn before the normal [paint\(\)](#) is invoked automatically.

Reimplemented from [KDChart::AbstractArea](#).

Definition at line 68 of file KDChartTernaryAxis.cpp.

```
69 {
70     // not used
71 }
```

**9.55.3.28** `void AbstractAreaBase::paintBackground (QPainter & painter, const QRect & rectangle)` [virtual, inherited]

Definition at line 196 of file KDChartAbstractAreaBase.cpp.

References [d](#), and [KDChart::AbstractAreaBase::paintBackgroundAttributes\(\)](#).

Referenced by [KDChart::TextArea::paintAll\(\)](#), [KDChart::AbstractAreaWidget::paintAll\(\)](#), and [KDChart::AbstractArea::paintAll\(\)](#).

```

197 {
198     Q_ASSERT_X ( d != 0, "AbstractAreaBase::paintBackground()",
199                 "Private class was not initialized!" );
200     paintBackgroundAttributes( painter, rect, d->backgroundAttributes );
201 }

```

**9.55.3.29 void AbstractAreaBase::paintBackgroundAttributes (QPainter & *painter*, const QRect & *rectangle*, const [KDCart::BackgroundAttributes](#) & *attributes*)** [static, inherited]

Definition at line 127 of file KDCartAbstractAreaBase.cpp.

References [KDCart::BackgroundAttributes::BackgroundPixmapModeCentered](#), [KDCart::BackgroundAttributes::BackgroundPixmapModeNone](#), [KDCart::BackgroundAttributes::BackgroundPixmapModeScaled](#), [KDCart::BackgroundAttributes::BackgroundPixmapModeStretched](#), [KDCart::BackgroundAttributes::brush\(\)](#), [KDCart::BackgroundAttributes::isVisible\(\)](#), [KDCart::BackgroundAttributes::pixmap\(\)](#), and [KDCart::BackgroundAttributes::pixmapMode\(\)](#).

Referenced by [KDCart::AbstractAreaBase::paintBackground\(\)](#).

```

129 {
130     if( !attributes.isVisible() ) return;
131
132     /* first draw the brush (may contain a pixmap)*/
133     if( Qt::NoBrush != attributes.brush().style() ) {
134         KDCart::PainterSaver painterSaver( &painter );
135         painter.setPen( Qt::NoPen );
136         const QPointF newTopLeft( painter.deviceMatrix().map( rect.topLeft() ) );
137         painter.setBrushOrigin( newTopLeft );
138         painter.setBrush( attributes.brush() );
139         painter.drawRect( rect.adjusted( 0, 0, -1, -1 ) );
140     }
141     /* next draw the backPixmap over the brush */
142     if( !attributes.pixmap().isNull() &&
143         attributes.pixmapMode() != BackgroundAttributes::BackgroundPixmapModeNone ) {
144         QPointF ol = rect.topLeft();
145         if( BackgroundAttributes::BackgroundPixmapModeCentered == attributes.pixmapMode() )
146         {
147             ol.setX( rect.center().x() - attributes.pixmap().width() / 2 );
148             ol.setY( rect.center().y() - attributes.pixmap().height() / 2 );
149             painter.drawPixmap( ol, attributes.pixmap() );
150         } else {
151             QMatrix m;
152             double zW = (double)rect.width() / (double)attributes.pixmap().width();
153             double zH = (double)rect.height() / (double)attributes.pixmap().height();
154             switch( attributes.pixmapMode() ) {
155                 case BackgroundAttributes::BackgroundPixmapModeScaled:
156                 {
157                     double z;
158                     z = qMin( zW, zH );
159                     m.scale( z, z );
160                 }
161                 break;
162                 case BackgroundAttributes::BackgroundPixmapModeStretched:
163                     m.scale( zW, zH );
164                     break;
165                 default:
166                     ; // Cannot happen, previously checked
167             }
168             QPixmap pm = attributes.pixmap().transformed( m );
169             ol.setX( rect.center().x() - pm.width() / 2 );
170             ol.setY( rect.center().y() - pm.height() / 2 );
171             painter.drawPixmap( ol, pm );

```

```

172     }
173 }
174 }

```

### 9.55.3.30 void TernaryAxis::paintCtx ([PaintContext](#) \*) [virtual]

Default impl: Paint the complete item using its layouted position and size.

Reimplemented from [KDChart::AbstractLayoutItem](#).

Definition at line 78 of file `KDChartTernaryAxis.cpp`.

References `KDChart::PaintContext::coordinatePlane()`, `KDChart::AbstractAxis::labels()`, `KDChart::PaintContext::painter()`, `PrerenderedLabel::pixmap()`, `PrerenderedElement::position()`, `KDChart::PaintContext::rectangle()`, `PrerenderedLabel::referencePointLocation()`, and `KDChart::TernaryCoordinatePlane::translate()`.

```

79 {
80     QPainter* p = paintContext->painter();
81     TernaryCoordinatePlane* plane =
82         (TernaryCoordinatePlane*) paintContext->coordinatePlane();
83     // QObject* refArea = plane->parent();
84     QRectF drawArea = paintContext->rectangle();
85     QRectF titleArea;
86
87     // paint the axis label (across the triangle, that one):
88     QList<PrerenderedLabel*> labels;
89     labels << m_label << m_fifty;
90     Q_FOREACH( PrerenderedLabel* label, labels ) {
91         const QPixmap& pixmap = label->pixmap();
92         QPointF point = plane->translate( label->position() )
93             - label->referencePointLocation();
94         p->drawPixmap( point, pixmap );
95     }
96 }

```

### 9.55.3.31 void AbstractAreaBase::paintFrame (QPainter & *painter*, const QRect & *rectangle*) [virtual, inherited]

Definition at line 204 of file `KDChartAbstractAreaBase.cpp`.

References `d`, and `KDChart::AbstractAreaBase::paintFrameAttributes()`.

Referenced by `KDChart::TextArea::paintAll()`, `KDChart::AbstractAreaWidget::paintAll()`, and `KDChart::AbstractArea::paintAll()`.

```

205 {
206     Q_ASSERT_X ( d != 0, "AbstractAreaBase::paintFrame()",
207                 "Private class was not initialized!" );
208     paintFrameAttributes( painter, rect, d->frameAttributes );
209 }

```

### 9.55.3.32 void AbstractAreaBase::paintFrameAttributes (QPainter & *painter*, const QRect & *rectangle*, const [KDChart::FrameAttributes](#) & *attributes*) [static, inherited]

Definition at line 177 of file `KDChartAbstractAreaBase.cpp`.

References KDChart::FrameAttributes::isVisible(), and KDChart::FrameAttributes::pen().

Referenced by KDChart::AbstractAreaBase::paintFrame().

```

179 {
180
181     if( !attributes.isVisible() ) return;
182
183     // Note: We set the brush to NoBrush explicitly here.
184     //       Otherwise we might get a filled rectangle, so any
185     //       previously drawn background would be overwritten by that area.
186
187     const QPen    oldPen( painter.pen() );
188     const QBrush oldBrush( painter.brush() );
189     painter.setPen( attributes.pen() );
190     painter.setBrush( Qt::NoBrush );
191     painter.drawRect( rect.adjusted( 0, 0, -1, -1 ) );
192     painter.setBrush( oldBrush );
193     painter.setPen( oldPen );
194 }
```

### 9.55.3.33 void AbstractArea::paintIntoRect (QPainter & painter, const QRect & rect) [virtual, inherited]

Draws the background and frame, then calls [paint\(\)](#).

In most cases there is no need to overwrite this method in a derived class, but you would overwrite [AbstractLayoutItem::paint\(\)](#) instead.

Definition at line 111 of file KDChartAbstractArea.cpp.

References KDChart::AbstractArea::paintAll().

```

112 {
113     const QRect oldGeometry( geometry() );
114     if( oldGeometry != rect )
115         setGeometry( rect );
116     painter.translate( rect.left(), rect.top() );
117     paintAll( painter );
118     painter.translate( -rect.left(), -rect.top() );
119     if( oldGeometry != rect )
120         setGeometry( oldGeometry );
121 }
```

### 9.55.3.34 QLayout\* KDChart::AbstractLayoutItem::parentLayout () [inherited]

Definition at line 76 of file KDChartLayoutItems.h.

```

77     {
78         return mParentLayout;
79     }
```

### 9.55.3.35 const [Position](#) TernaryAxis::position () const [virtual]

Definition at line 135 of file KDChartTernaryAxis.cpp.

Referenced by requiredMargins(), and setPosition().

```

136 {
137     return m_position;
138 }

```

**9.55.3.36** void **KDChart::AbstractArea::positionChanged** ([AbstractArea \\*](#)) [signal, inherited]

Referenced by **KDChart::AbstractArea::positionHasChanged()**.

**9.55.3.37** void **AbstractArea::positionHasChanged** () [protected, virtual, inherited]

Reimplemented from [KDChart::AbstractAreaBase](#).

Definition at line 155 of file **KDChartAbstractArea.cpp**.

References **KDChart::AbstractArea::positionChanged()**.

```

156 {
157     emit positionChanged( this );
158 }

```

**9.55.3.38** void **KDChart::AbstractLayoutItem::removeFromParentLayout** () [inherited]

Definition at line 80 of file **KDChartLayoutItems.h**.

Referenced by **KDChart::Chart::takeCoordinatePlane()**.

```

81     {
82         if( mParentLayout ){
83             if( widget() )
84                 mParentLayout->removeWidget( widget() );
85             else
86                 mParentLayout->removeItem( this );
87         }
88     }

```

**9.55.3.39** **QPair< QSizeF, QSizeF > TernaryAxis::requiredMargins** () const

Definition at line 260 of file **KDChartTernaryAxis.cpp**.

References **PrerenderedLabel::pixmap()**, **position()**, **KDChartEnums::PositionEast**, **KDChartEnums::PositionSouth**, **KDChartEnums::PositionWest**, and **PrerenderedLabel::referencePointLocation()**.

Referenced by **KDChart::TernaryCoordinatePlane::layoutDiagrams()**.

```

261 {
262     QSizeF topleft( 0.0, 0.0 );
263     QSizeF bottomRight( 0.0, 0.0 );
264
265     switch( position().value() ) {
266     case KDChartEnums::PositionSouth:
267         // the label of the south axis is, in fact, up north.
268         topleft.setHeight( m_label->pixmap().height() );
269         bottomRight.setHeight( m_fifty->pixmap().height() );

```



```

270         break;
271     case KDChartEnums::PositionWest:
272         bottomRight.setWidth( m_label->pixmap().width()
273                               - m_label->referencePointLocation().x() );
274         bottomRight.setHeight( m_label->pixmap().height()
275                                - m_label->referencePointLocation().y() );
276         break;
277     case KDChartEnums::PositionEast:
278         topleft.setWidth( m_label->pixmap().width()
279                           - ( m_label->pixmap().width()
280                               - m_label->referencePointLocation().x() ) );
281         bottomRight.setHeight( m_label->pixmap().height()
282                                - ( m_label->pixmap().height()
283                                    - m_label->referencePointLocation().y() ) );
284         break;
285     default:
286         qDebug() << "TernaryAxis::requiredMargins: unknown location";
287     }
288     // qDebug() << "TernaryAxis::requiredMargins:" << topleft << bottomRight;
289     return QPair<QSizeF, QSizeF>( topleft, bottomRight );
290 }

```

### 9.55.3.40 void TernaryAxis::resetTitleTextAttributes ()

Definition at line 194 of file KDChartTernaryAxis.cpp.

Referenced by TernaryAxis().

```

195 {
196     TextAttributes a;
197     m_titleAttributes = a;
198     updatePrerenderedLabels();
199 }

```

### 9.55.3.41 int AbstractArea::rightOverlap (bool *doNotRecalculate* = false) const [virtual, inherited]

This is called at layout time by [KDChart::AutoSpacerLayoutItem::sizeHint\(\)](#).

The method triggers AbstractArea::sizeHint() to find out the amount of overlap at the right edge of the area.

#### Note:

The default implementation is not using any caching, it might make sense to implement a more sophisticated solution for derived classes that have complex work to do in sizeHint(). All we have here is a primitive flag to be set by the caller if it is sure that no sizeHint() needs to be called.

Definition at line 85 of file KDChartAbstractArea.cpp.

References d.

Referenced by KDChart::CartesianAxis::maximumSize().

```

86 {
87     // Re-calculate the sizes,
88     // so we also get the amountOf..Overlap members set newly:
89     if( ! doNotRecalculate )
90         sizeHint();
91     return d->amountOfRightOverlap;
92 }

```

### 9.55.3.42 void AbstractAreaBase::setBackgroundAttributes (const [BackgroundAttributes](#) & a) [inherited]

Definition at line 111 of file KDChartAbstractAreaBase.cpp.

References [d](#), and [KDChart::AbstractAreaBase::positionHasChanged\(\)](#).

```
112 {  
113     if( d->backgroundAttributes == a )  
114         return;  
115  
116     d->backgroundAttributes = a;  
117     positionHasChanged();  
118 }
```

### 9.55.3.43 void AbstractAreaBase::setFrameAttributes (const [FrameAttributes](#) & a) [inherited]

Definition at line 97 of file KDChartAbstractAreaBase.cpp.

References [d](#), and [KDChart::AbstractAreaBase::positionHasChanged\(\)](#).

Referenced by [KDChart::Legend::clone\(\)](#).

```
98 {  
99     if( d->frameAttributes == a )  
100         return;  
101  
102     d->frameAttributes = a;  
103     positionHasChanged();  
104 }
```

### 9.55.3.44 void TernaryAxis::setGeometry (const QRect & rect) [virtual]

Implements [KDChart::AbstractAxis](#).

Definition at line 109 of file KDChartTernaryAxis.cpp.

```
110 {  
111     m_geometry = rect;  
112 }
```

### 9.55.3.45 void AbstractAxis::setLabels (const QStringList & list) [inherited]

Use this to specify your own set of strings, to be used as axis labels.

Labels specified via `setLabels` take precedence: If a non-empty list is passed, KD [Chart](#) will use these strings as axis labels, instead of calculating them.

If you a smaller number of strings than the number of labels drawn at this axis, KD [Chart](#) will iterate over the list, repeating the strings, until all labels are drawn. As an example you could specify the seven days of the week as abscissa labels, which would be repeatedly used then.

By passing an empty `QStringList` you can reset the default behaviour.

See also:

[labels](#), [setShortLabels](#)

Definition at line 267 of file KDChartAbstractAxis.cpp.

References [d](#), and [KDChart::AbstractAxis::update\(\)](#).

```

268 {
269     if( d->hardLabels == list )
270         return;
271
272     d->hardLabels = list;
273     update();
274 }
```

### 9.55.3.46 void KDChart::AbstractLayoutItem::setParentLayout (QLayout \* lay) [inherited]

Definition at line 72 of file KDChartLayoutItems.h.

```

73     {
74         mParentLayout = lay;
75     }
```

### 9.55.3.47 void KDChart::AbstractLayoutItem::setParentWidget (QWidget \* widget) [virtual, inherited]

Inform the item about its widget: This enables the item, to trigger that widget's update, whenever the size of the item's contents has changed.

Thus, you need to call setParentWidget on every item, that has a non-fixed size.

Definition at line 64 of file KDChartLayoutItems.cpp.

References [KDChart::AbstractLayoutItem::mParent](#).

Referenced by [KDChart::HeaderFooter::setParent\(\)](#), and [KDChart::AbstractCartesianDiagram::takeAxis\(\)](#).

```

65 {
66     mParent = widget;
67 }
```

### 9.55.3.48 void TernaryAxis::setPosition (Position p) [virtual]

Definition at line 140 of file KDChartTernaryAxis.cpp.

References [position\(\)](#), [KDChartEnums::PositionEast](#), [KDChartEnums::PositionSouth](#), [KDChartEnums::PositionWest](#), [PrerenderedLabel::setText\(\)](#), and [KDChart::Position::value\(\)](#).

Referenced by [TernaryAxis\(\)](#).

```

141 {
142     if ( p == position() ) return;
143
144     if ( p != KDChartEnums::PositionWest
```

```

145         && p != KDChartEnums::PositionEast
146         && p != KDChartEnums::PositionSouth )
147     {
148         qDebug() << "TernaryAxis::setPosition: only south, east and west are supported "
149             "positions for ternary axes.";
150         return;
151     }
152
153     if ( m_title.isEmpty() )
154         switch( p.value() ) {
155             case KDChartEnums::PositionSouth:
156                 m_label->setText( tr( "A" ) );
157                 break;
158             case KDChartEnums::PositionWest:
159                 m_label->setText( tr( "C" ) );
160                 break;
161             case KDChartEnums::PositionEast:
162                 m_label->setText( tr( "B" ) );
163                 break;
164             default:
165                 break;
166         }
167
168     m_position = p;
169     updatePrerenderedLabels(); // position has changed
170 }

```

#### 9.55.3.49 void AbstractAxis::setShortLabels (const QStringList & *list*) [inherited]

Use this to specify your own set of strings, to be used as axis labels, in case the normal labels are too long.

##### Note:

Setting done via setShortLabels will be ignored, if you did not pass a non-empty string list via setLabels too!

By passing an empty QStringList you can reset the default behaviour.

##### See also:

[shortLabels](#), [setLabels](#)

Definition at line 297 of file KDChartAbstractAxis.cpp.

References [d](#), and [KDChart::AbstractAxis::update\(\)](#).

```

298 {
299     if( d->hardShortLabels == list )
300         return;
301
302     d->hardShortLabels = list;
303     update();
304 }

```

#### 9.55.3.50 void AbstractAxis::setTextAttributes (const [TextAttributes](#) & *a*) [inherited]

Use this to specify the text attributes to be used for axis labels.

By default, the reference area will be set at painting time. It will be the then-valid coordinate plane's parent widget, so normally, it will be the [KDChart::Chart](#). Thus the labels of all of your axes in all of your diagrams within that [Chart](#) will be drawn in same font size, by default.

**See also:**

[textAttributes](#), [setLabels](#)

Definition at line 231 of file KDChartAbstractAxis.cpp.

References [d](#), and [KDChart::AbstractAxis::update\(\)](#).

```
232 {
233     if( d->textAttributes == a )
234         return;
235
236     d->textAttributes = a;
237     update();
238 }
```

#### 9.55.3.51 void TernaryAxis::setTitleText (const QString & text)

Definition at line 172 of file KDChartTernaryAxis.cpp.

References [PrerenderedLabel::setText\(\)](#).

```
173 {
174     m_title = text; // do not remove
175     m_label->setText( text );
176 }
```

#### 9.55.3.52 void TernaryAxis::setTitleTextAttributes (const [TextAttributes](#) & a)

Definition at line 183 of file KDChartTernaryAxis.cpp.

```
184 {
185     m_titleAttributes = a;
186     updatePrerenderedLabels();
187 }
```

#### 9.55.3.53 QStringList AbstractAxis::shortLabels () const [inherited]

Returns a list of strings, that are used as axis labels, as set via [setShortLabels](#).

**Note:**

Setting done via [setShortLabels](#) will be ignored, if you did not pass a non-empty string list via [setLabels](#) too!

**See also:**

[setShortLabels](#)

Definition at line 314 of file KDChartAbstractAxis.cpp.

References d.

Referenced by KDChart::AbstractAxis::compare(), and KDChart::CartesianAxis::paintCtx().

```
315 {
316     return d->hardShortLabels;
317 }
```

#### 9.55.3.54 QSize TernaryAxis::sizeHint () const [virtual]

Definition at line 125 of file KDChartTernaryAxis.cpp.

```
126 {
127     return QSize( 150, 100 );
128 }
```

#### 9.55.3.55 void KDChart::AbstractLayoutItem::sizeHintChanged () const [virtual, inherited]

Report changed size hint: ask the parent widget to recalculate the layout.

Definition at line 86 of file KDChartLayoutItems.cpp.

References KDChart::AbstractLayoutItem::mParent.

Referenced by KDChart::TextLayoutItem::sizeHint().

```
87 {
88     // This is exactly like what QWidget::updateGeometry does.
89     qDebug("KDChart::AbstractLayoutItem::sizeHintChanged() called");
90     if( mParent ) {
91         if ( mParent->layout() )
92             mParent->layout()->invalidate();
93         else
94             QApplication::postEvent( mParent, new QEvent( QEvent::LayoutRequest ) );
95     }
96 }
```

#### 9.55.3.56 [TextAttributes](#) AbstractAxis::textAttributes () const [inherited]

Returns the text attributes to be used for axis labels.

**See also:**

[setTextAttributes](#)

Definition at line 245 of file KDChartAbstractAxis.cpp.

References d.

Referenced by KDChart::AbstractAxis::compare(), KDChart::CartesianAxis::maximumSize(), KDChart::CartesianAxis::paintCtx(), and KDChart::CartesianAxis::titleTextAttributes().

```
246 {
247     return d->textAttributes;
248 }
```

**9.55.3.57** `QString TernaryAxis::titleText () const`

Definition at line 178 of file KDChartTernaryAxis.cpp.

References `PrerenderedLabel::text()`.

```
179 {
180     return m_label->text();
181 }
```

**9.55.3.58** `TextAttributes TernaryAxis::titleTextAttributes () const`

Definition at line 189 of file KDChartTernaryAxis.cpp.

```
190 {
191     return m_titleAttributes;
192 }
```

**9.55.3.59** `int AbstractArea::topOverlap (bool doNotRecalculate = false) const` `[virtual, inherited]`

This is called at layout time by `KDChart::AutoSpacerLayoutItem::sizeHint()`.

The method triggers `AbstractArea::sizeHint()` to find out the amount of overlap at the top edge of the area.

**Note:**

The default implementation is not using any caching, it might make sense to implement a more sophisticated solution for derived classes that have complex work to do in `sizeHint()`. All we have here is a primitive flag to be set by the caller if it is sure that no `sizeHint()` needs to be called.

Definition at line 93 of file KDChartAbstractArea.cpp.

References `d`.

Referenced by `KDChart::CartesianAxis::maximumSize()`.

```
94 {
95     // Re-calculate the sizes,
96     // so we also get the amountOf..Overlap members set newly:
97     if( ! doNotRecalculate )
98         sizeHint();
99     return d->amountOfTopOverlap;
100 }
```

**9.55.3.60** `void KDChart::AbstractAxis::update ()` `[slot, inherited]`

Definition at line 341 of file KDChartAbstractAxis.cpp.

References `d`.

Referenced by `KDChart::AbstractAxis::connectSignals()`, `KDChart::AbstractAxis::setLabels()`, `KDChart::AbstractAxis::setShortLabels()`, and `KDChart::AbstractAxis::setTextAttributes()`.

```
342 {
343     if( d->diagram() )
344         d->diagram()->update();
345 }
```

## 9.55.4 Member Data Documentation

### 9.55.4.1 [QWidget\\*](#) [KDChart::AbstractLayoutItem::mParent](#) [protected, inherited]

Definition at line 90 of file KDChartLayoutItems.h.

Referenced by [KDChart::AbstractLayoutItem::setParentWidget\(\)](#), [KDChart::TextLayoutItem::setText\(\)](#), [KDChart::TextLayoutItem::setTextAttributes\(\)](#), and [KDChart::AbstractLayoutItem::sizeHintChanged\(\)](#).

### 9.55.4.2 [QLayout\\*](#) [KDChart::AbstractLayoutItem::mParentLayout](#) [protected, inherited]

Definition at line 91 of file KDChartLayoutItems.h.

Referenced by [KDChart::AutoSpacerLayoutItem::paint\(\)](#).

The documentation for this class was generated from the following files:

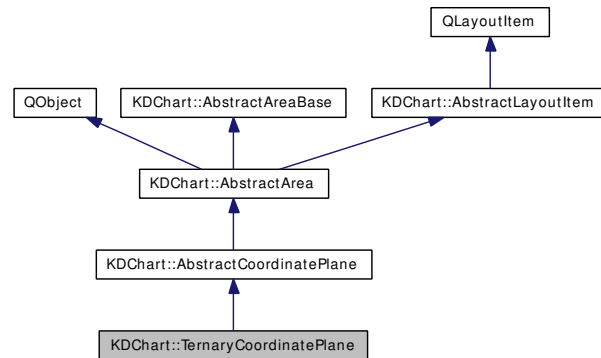
- [KDChartTernaryAxis.h](#)
- [KDChartTernaryAxis.cpp](#)



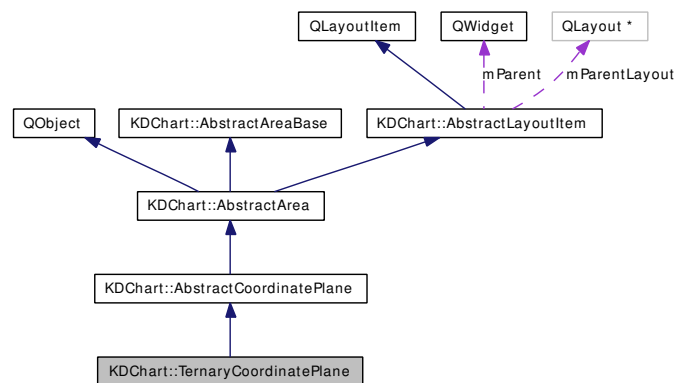
## 9.56 KDChart::TernaryCoordinatePlane Class Reference

```
#include <KDChartTernaryCoordinatePlane.h>
```

Inheritance diagram for KDChart::TernaryCoordinatePlane:



Collaboration diagram for KDChart::TernaryCoordinatePlane:



### 9.56.1 Detailed Description

Ternary coordinate plane.

Definition at line 42 of file KDChartTernaryCoordinatePlane.h.

#### Public Types

- enum [AxesCalcMode](#) {  
[Linear](#),  
[Logarithmic](#) }

#### Public Slots

- void [layoutPlanes](#) ()

Calling [layoutPlanes\(\)](#) on the plane triggers the global `KDChart::Chart::slotLayoutPlanes()`.

- void [relayout](#) ()  
Calling [relayout\(\)](#) on the plane triggers the global `KDChart::Chart::slotRelayout()`.
- void [setGridNeedsRecalculate](#) ()  
Used by the chart to clear the cached grid data.
- void [update](#) ()  
Calling [update\(\)](#) on the plane triggers the global `KDChart::Chart::update()`.

## Signals

- void [destroyedCoordinatePlane](#) ([AbstractCoordinatePlane](#) \*)  
Emitted when this coordinate plane is destroyed.
- void [needLayoutPlanes](#) ()  
Emitted when plane needs to trigger the Chart's layouting of the coord.
- void [needRelayout](#) ()  
Emitted when plane needs to trigger the Chart's layouting.
- void [needUpdate](#) ()  
Emitted when plane needs to update its drawings.
- void [positionChanged](#) ([AbstractArea](#) \*)
- void [propertiesChanged](#) ()  
Emitted upon change of a property of the Coordinate Plane or any of its components.

## Public Member Functions

- void [addDiagram](#) ([AbstractDiagram](#) \*diagram)  
Adds a diagram to this coordinate plane.
- void [alignToReferencePoint](#) (const [RelativePosition](#) &position)
- [BackgroundAttributes](#) [backgroundAttributes](#) () const
- virtual int [bottomOverlap](#) (bool doNotRecalculate=false) const  
This is called at layout time by `KDChart::AutoSpacerLayoutItem::sizeHint()`.
- bool [compare](#) (const [AbstractAreaBase](#) \*other) const  
Returns true if both areas have the same settings.
- [AbstractDiagram](#) \* [diagram](#) ()  
**Returns:**  
The first diagram associated with this coordinate plane.
- [ConstAbstractDiagramList](#) [diagrams](#) () const

**Returns:**

*The list of diagrams associated with this coordinate plane.*

- [AbstractDiagramList diagrams](#) ()

**Returns:**

*The list of diagrams associated with this coordinate plane.*

- virtual Qt::Orientations [expandingDirections](#) () const

*pure virtual in [QLayoutItem](#)*

- [FrameAttributes frameAttributes](#) () const

- virtual QRect [geometry](#) () const

*pure virtual in [QLayoutItem](#)*

- [DataDimensionsList getDataDimensionsList](#) () const
- void [getFrameLeadings](#) (int &left, int &top, int &right, int &bottom) const
- [GridAttributes globalGridAttributes](#) () const

**Returns:**

*The grid attributes used by this coordinate plane.*

- [DataDimensionsList gridDimensionsList](#) ()

*Returns the dimensions used for drawing the grid lines.*

- virtual bool [isEmpty](#) () const

*pure virtual in [QLayoutItem](#)*

- bool [isRubberBandZoomingEnabled](#) () const

**Returns:**

*Whether zooming with a rubber band using the mouse is enabled.*

- const bool [isVisiblePoint](#) (const QPointF &point) const

*Tests, if a point is visible on the coordinate plane.*

- void [layoutDiagrams](#) ()

*Distribute the available space among the diagrams and axes.*

- virtual int [leftOverlap](#) (bool doNotRecalculate=false) const

*This is called at layout time by [KDChart::AutoSpacerLayoutItem::sizeHint\(\)](#).*

- virtual QSize [maximumSize](#) () const

*pure virtual in [QLayoutItem](#)*

- virtual QSize [minimumSize](#) () const

*pure virtual in [QLayoutItem](#)*

- QSize [minimumSizeHint](#) () const

*[reimplemented]*

- void [mouseDoubleClickEvent](#) (QMouseEvent \*event)

- void [mouseMoveEvent](#) (QMouseEvent \*event)

- void [mousePressEvent](#) (QMouseEvent \*event)
- void [mouseReleaseEvent](#) (QMouseEvent \*event)
- void [paint](#) (QPainter \*)
- virtual void [paintAll](#) (QPainter &painter)
 

*Call paintAll, if you want the background and the frame to be drawn before the normal [paint\(\)](#) is invoked automatically.*
- virtual void [paintBackground](#) (QPainter &painter, const QRect &rectangle)
- virtual void [paintCtx](#) (PaintContext \*context)
 

*Default impl: Paint the complete item using its layouted position and size.*
- virtual void [paintFrame](#) (QPainter &painter, const QRect &rectangle)
- virtual void [paintIntoRect](#) (QPainter &painter, const QRect &rect)
 

*Draws the background and frame, then calls [paint\(\)](#).*
- const Chart \* [parent](#) () const
- Chart \* [parent](#) ()
- QLayout \* [parentLayout](#) ()
- AbstractCoordinatePlane \* [referenceCoordinatePlane](#) () const
 

*There are two ways, in which planes can be caused to interact, in where they are put layouting wise: The first is the reference plane.*
- void [removeFromParentLayout](#) ()
- virtual void [replaceDiagram](#) (AbstractDiagram \*diagram, AbstractDiagram \*oldDiagram=0)
 

*Replaces the old diagram, or appends the diagram, if there is none yet.*
- virtual int [rightOverlap](#) (bool doNotRecalculate=false) const
 

*This is called at layout time by [KDChart::AutoSpacerLayoutItem::sizeHint\(\)](#).*
- void [setBackgroundAttributes](#) (const BackgroundAttributes &a)
- void [setFrameAttributes](#) (const FrameAttributes &a)
- virtual void [setGeometry](#) (const QRect &r)
 

*pure virtual in [QLayoutItem](#)*
- void [setGlobalGridAttributes](#) (const GridAttributes &)
 

*Set the grid attributes to be used by this coordinate plane.*
- void [setParent](#) (Chart \*parent)
 

*Called internally by [KDChart::Chart](#).*
- void [setParentLayout](#) (QLayout \*lay)
- virtual void [setParentWidget](#) (QWidget \*widget)
 

*Inform the item about its widget: This enables the item, to trigger that widget's update, whenever the size of the item's contents has changed.*
- void [setReferenceCoordinatePlane](#) (AbstractCoordinatePlane \*plane)
 

*Set another coordinate plane to be used as the reference plane for this one.*
- void [setRubberBandZoomingEnabled](#) (bool enable)
 

*Enables or disables zooming with a rubber band using the mouse.*

- virtual void [setZoomCenter](#) (const QPointF &center)  
*Set the point (in value coordinates) to be used as the center point in zoom operations.*
- virtual void [setZoomFactorX](#) (double factor)  
*Sets the zoom factor in horizontal direction, that is applied to all coordinate transformations.*
- virtual void [setZoomFactorY](#) (double factor)  
*Sets the zoom factor in vertical direction, that is applied to all coordinate transformations.*
- virtual [AbstractCoordinatePlane](#) \* [sharedAxisMasterPlane](#) (QPainter \*p=0)
- virtual QSize [sizeHint](#) () const  
*pure virtual in [QLayoutItem](#)*
- virtual void [sizeHintChanged](#) () const  
*Report changed size hint: ask the parent widget to recalculate the layout.*
- QSizePolicy [sizePolicy](#) () const  
*[reimplemented]*
- virtual void [takeDiagram](#) ([AbstractDiagram](#) \*diagram)  
*Removes the diagram from the plane, without deleting it.*
- [TernaryCoordinatePlane](#) ([Chart](#) \*parent=0)
- virtual int [topOverlap](#) (bool doNotRecalculate=false) const  
*This is called at layout time by [KDChart::AutoSpacerLayoutItem::sizeHint\(\)](#).*
- const QPointF [translate](#) (const QPointF &diagramPoint) const  
*Translate the given point in value space coordinates to a position in pixel space.*
- virtual QPointF [zoomCenter](#) () const  
**Returns:**  
*The center point (in value coordinates) of the coordinate plane, that is used for zoom operations.*
- virtual double [zoomFactorX](#) () const  
**Returns:**  
*The zoom factor in horizontal direction, that is applied to all coordinate transformations.*
- virtual double [zoomFactorY](#) () const  
**Returns:**  
*The zoom factor in vertical direction, that is applied to all coordinate transformations.*
- [~TernaryCoordinatePlane](#) ()

## Static Public Member Functions

- static void [paintBackgroundAttributes](#) (QPainter &painter, const QRect &rectangle, const [KDChart::BackgroundAttributes](#) &attributes)
- static void [paintFrameAttributes](#) (QPainter &painter, const QRect &rectangle, const [KDChart::FrameAttributes](#) &attributes)

## Protected Member Functions

- virtual QRect [areaGeometry](#) () const
- QRect [innerRect](#) () const
- virtual void [positionHasChanged](#) ()

## Protected Attributes

- QWidget \* [mParent](#)
- QLayout \* [mParentLayout](#)

## 9.56.2 Member Enumeration Documentation

**9.56.2.1** enum [KDChart::AbstractCoordinatePlane::AxesCalcMode](#) [inherited]

Enumerator:

*Linear*

*Logarithmic*

Definition at line 57 of file KDChartAbstractCoordinatePlane.h.

```
57 { Linear, Logarithmic };
```

## 9.56.3 Constructor & Destructor Documentation

**9.56.3.1** TernaryCoordinatePlane::TernaryCoordinatePlane ([Chart](#) \* *parent* = 0) [explicit]

Definition at line 51 of file KDChartTernaryCoordinatePlane.cpp.

```
52      : AbstractCoordinatePlane( new Private(), parent )
53 {
54 }
```

**9.56.3.2** TernaryCoordinatePlane::~~TernaryCoordinatePlane ()

Definition at line 56 of file KDChartTernaryCoordinatePlane.cpp.

```
57 {
58 }
```

## 9.56.4 Member Function Documentation

**9.56.4.1** void TernaryCoordinatePlane::addDiagram ([AbstractDiagram](#) \* *diagram*) [virtual]

Adds a diagram to this coordinate plane.

Parameters:

*diagram* The diagram to add.

See also:

[replaceDiagram](#), [takeDiagram](#)

Reimplemented from [KDChart::AbstractCoordinatePlane](#).

Definition at line 64 of file KDChartTernaryCoordinatePlane.cpp.

References [KDChart::AbstractCoordinatePlane::addDiagram\(\)](#), and [KDChart::AbstractCoordinatePlane::diagram\(\)](#).

```

65 {
66     Q_ASSERT_X ( dynamic_cast<AbstractTernaryDiagram*>( diagram ),
67                 "TernaryCoordinatePlane::addDiagram", "Only ternary "
68                 "diagrams can be added to a ternary coordinate plane!" );
69     AbstractCoordinatePlane::addDiagram ( diagram );
70 //     connect ( diagram, SIGNAL ( layoutChanged ( AbstractDiagram* ) ),
71 //             SLOT ( slotLayoutChanged ( AbstractDiagram* ) ) );
72 //     connect( diagram, SIGNAL( propertiesChanged() ),this, SIGNAL( propertiesChanged() ) );
73 }
```

#### 9.56.4.2 void AbstractAreaBase::alignToReferencePoint (const [RelativePosition](#) & position) [inherited]

Definition at line 90 of file KDChartAbstractAreaBase.cpp.

```

91 {
92     Q_UNUSED( position );
93     // PENDING(kalle) FIXME
94     qWarning( "Sorry, not implemented: void AbstractAreaBase::alignToReferencePoint( const RelativePosi
95 }
```

#### 9.56.4.3 QRect AbstractArea::areaGeometry () const [protected, virtual, inherited]

Implements [KDChart::AbstractAreaBase](#).

Definition at line 150 of file KDChartAbstractArea.cpp.

Referenced by [KDChart::CartesianCoordinatePlane::drawingArea\(\)](#), [layoutDiagrams\(\)](#), [KDChart::PolarCoordinatePlane::layoutDiagrams\(\)](#), [paint\(\)](#), [KDChart::CartesianAxis::paint\(\)](#), [KDChart::AbstractArea::paintAll\(\)](#), and [KDChart::CartesianAxis::paintCtx\(\)](#).

```

151 {
152     return geometry();
153 }
```

#### 9.56.4.4 [BackgroundAttributes](#) AbstractAreaBase::backgroundAttributes () const [inherited]

Definition at line 120 of file KDChartAbstractAreaBase.cpp.

References [d](#).

Referenced by [KDChart::AbstractAreaBase::compare\(\)](#), and [updateCommonBrush\(\)](#).

```

121 {
122     return d->backgroundAttributes;
123 }

```

#### 9.56.4.5 int AbstractArea::bottomOverlap (bool *doNotRecalculate* = false) const [virtual, inherited]

This is called at layout time by KDChart::AutoSpacerLayoutItem::sizeHint().

The method triggers AbstractArea::sizeHint() to find out the amount of overlap at the bottom edge of the area.

#### Note:

The default implementation is not using any caching, it might make sense to implement a more sophisticated solution for derived classes that have complex work to do in sizeHint(). All we have here is a primitive flag to be set by the caller if it is sure that no sizeHint() needs to be called.

Definition at line 101 of file KDChartAbstractArea.cpp.

References d.

Referenced by KDChart::CartesianAxis::maximumSize().

```

102 {
103     // Re-calculate the sizes,
104     // so we also get the amountOf..Overlap members set newly:
105     if( ! doNotRecalculate )
106         sizeHint();
107     return d->amountOfBottomOverlap;
108 }

```

#### 9.56.4.6 bool AbstractAreaBase::compare (const [AbstractAreaBase](#) \* *other*) const [inherited]

Returns true if both areas have the same settings.

Definition at line 75 of file KDChartAbstractAreaBase.cpp.

References KDChart::AbstractAreaBase::backgroundAttributes(), and KDChart::AbstractAreaBase::frameAttributes().

```

76 {
77     if( other == this ) return true;
78     if( ! other ){
79         //qDebug() << "CartesianAxis::compare() cannot compare to Null pointer";
80         return false;
81     }
82     /*
83     qDebug() << "AbstractAreaBase:" << (frameAttributes() == other->frameAttributes())
84         << (backgroundAttributes() == other->backgroundAttributes()) << "\n";
85     */
86     return (frameAttributes() == other->frameAttributes()) &&
87         (backgroundAttributes() == other->backgroundAttributes());
88 }

```



#### 9.56.4.7 void KDChart::AbstractCoordinatePlane::destroyedCoordinatePlane (AbstractCoordinatePlane \*) [signal, inherited]

Emitted when this coordinate plane is destroyed.

Referenced by KDChart::AbstractCoordinatePlane::~~AbstractCoordinatePlane().

#### 9.56.4.8 AbstractDiagram \* AbstractCoordinatePlane::diagram () [inherited]

##### Returns:

The first diagram associated with this coordinate plane.

Definition at line 117 of file KDChartAbstractCoordinatePlane.cpp.

References d.

Referenced by addDiagram(), KDChart::PolarCoordinatePlane::addDiagram(), KDChart::CartesianCoordinatePlane::addDiagram(), KDChart::AbstractCoordinatePlane::addDiagram(), KDChart::Widget::diagram(), KDChart::CartesianCoordinatePlane::getRawDataBoundingRectFromDiagrams(), layoutDiagrams(), KDChart::PolarCoordinatePlane::layoutDiagrams(), KDChart::AbstractCoordinatePlane::replaceDiagram(), KDChart::CartesianCoordinatePlane::setGeometry(), KDChart::PolarCoordinatePlane::setStartPosition(), KDChart::CartesianCoordinatePlane::sharedAxisMasterPlane(), and KDChart::AbstractCoordinatePlane::takeDiagram().

```

118 {
119     if ( d->diagrams.isEmpty() )
120     {
121         return 0;
122     } else {
123         return d->diagrams.first();
124     }
125 }
```

#### 9.56.4.9 ConstAbstractDiagramList AbstractCoordinatePlane::diagrams () const [inherited]

##### Returns:

The list of diagrams associated with this coordinate plane.

Definition at line 132 of file KDChartAbstractCoordinatePlane.cpp.

References d.

```

133 {
134     ConstAbstractDiagramList list;
135 #ifndef QT_NO_STL
136     qCopy( d->diagrams.begin(), d->diagrams.end(), std::back_inserter( list ) );
137 #else
138     Q_FOREACH( AbstractDiagram * a, d->diagrams )
139         list.push_back( a );
140 #endif
141     return list;
142 }
```

**9.56.4.10 [AbstractDiagramList](#) AbstractCoordinatePlane::diagrams ()** [inherited]**Returns:**

The list of diagrams associated with this coordinate plane.

Definition at line 127 of file KDChartAbstractCoordinatePlane.cpp.

References d.

Referenced by KDChart::CartesianCoordinatePlane::getDataDimensionsList(), KDChart::CartesianCoordinatePlane::getRawDataBoundingRectFromDiagrams(), layoutDiagrams(), KDChart::PolarCoordinatePlane::layoutDiagrams(), KDChart::CartesianCoordinatePlane::layoutDiagrams(), KDChart::Chart::mouseDoubleClickEvent(), KDChart::Chart::mouseMoveEvent(), KDChart::Chart::mousePressEvent(), KDChart::Chart::mouseReleaseEvent(), paint(), KDChart::PolarCoordinatePlane::paint(), KDChart::CartesianCoordinatePlane::paint(), and KDChart::CartesianCoordinatePlane::setGeometry().

```
128 {
129     return d->diagrams;
130 }
```

**9.56.4.11 [Qt::Orientations](#) KDChart::AbstractCoordinatePlane::expandingDirections () const** [virtual, inherited]

pure virtual in [QLayoutItem](#)

Definition at line 212 of file KDChartAbstractCoordinatePlane.cpp.

```
213 {
214     return Qt::Vertical | Qt::Horizontal;
215 }
```

**9.56.4.12 [FrameAttributes](#) AbstractAreaBase::frameAttributes () const** [inherited]

Definition at line 106 of file KDChartAbstractAreaBase.cpp.

References d.

Referenced by KDChart::Legend::clone(), KDChart::AbstractAreaBase::compare(), and updateCommonBrush().

```
107 {
108     return d->frameAttributes;
109 }
```

**9.56.4.13 [QRect](#) KDChart::AbstractCoordinatePlane::geometry () const** [virtual, inherited]

pure virtual in [QLayoutItem](#)

Definition at line 246 of file KDChartAbstractCoordinatePlane.cpp.

References d.

Referenced by KDChart::Chart::mouseDoubleClickEvent(), KDChart::Chart::mouseMoveEvent(), KDChart::AbstractCoordinatePlane::mouseMoveEvent(), KDChart::Chart::mousePressEvent(), KDChart::Chart::mouseReleaseEvent(), KDChart::AbstractCoordinatePlane::mouseReleaseEvent(), and KDChart::PolarCoordinatePlane::paint().

```
247 {
248     return d->geometry;
249 }
```

#### 9.56.4.14 [DataDimensionsList](#) TernaryCoordinatePlane::getDataDimensionsList () const [virtual]

Implements [KDChart::AbstractCoordinatePlane](#).

Definition at line 199 of file KDChartTernaryCoordinatePlane.cpp.

```
200 {    // not needed
201     return DataDimensionsList();
202 }
```

#### 9.56.4.15 void AbstractAreaBase::getFrameLeadings (int & left, int & top, int & right, int & bottom) const [inherited]

Definition at line 212 of file KDChartAbstractAreaBase.cpp.

References [d](#).

Referenced by KDChart::AbstractAreaBase::innerRect(), and KDChart::AbstractAreaWidget::paintAll().

```
213 {
214     if( d && d->frameAttributes.isVisible() ){
215         const int padding = qMax( d->frameAttributes.padding(), 0 );
216         left    = padding;
217         top     = padding;
218         right   = padding;
219         bottom  = padding;
220     }else{
221         left    = 0;
222         top     = 0;
223         right   = 0;
224         bottom  = 0;
225     }
226 }
```

#### 9.56.4.16 [GridAttributes](#) KDChart::AbstractCoordinatePlane::globalGridAttributes () const [inherited]

##### Returns:

The grid attributes used by this coordinate plane.

##### See also:

[setGlobalGridAttributes](#)  
[CartesianCoordinatePlane::gridAttributes](#)

Definition at line 161 of file KDChartAbstractCoordinatePlane.cpp.

References d.

Referenced by KDChart::PolarCoordinatePlane::gridAttributes(), and KDChart::CartesianCoordinatePlane::gridAttributes().

```
162 {
163     return d->gridAttributes;
164 }
```

#### 9.56.4.17 [KDChart::DataDimensionsList](#) KDChart::AbstractCoordinatePlane::gridDimensionsList () [inherited]

Returns the dimensions used for drawing the grid lines.

Returned data is the result of (cached) grid calculations, so - if you need that information for your own tasks - make sure to call again this function after every data modification that has changed the data range, since grid calculation is based upon the data range, thus the grid start/end might have changed if the data was changed.

##### Note:

Returned list will contain different numbers of [DataDimension](#), depending on the kind of coordinate plane used. For [CartesianCoordinatePlane](#) two [DataDimension](#) are returned: the first representing grid lines in X direction (matching the Abscissa axes) and the second indicating vertical grid lines (or Ordinate axes, resp.).

##### Returns:

The dimensions used for drawing the grid lines.

##### See also:

[DataDimension](#)

Definition at line 166 of file KDChartAbstractCoordinatePlane.cpp.

References d.

Referenced by KDChart::CartesianCoordinatePlane::layoutDiagrams(), and KDChart::CartesianAxis::maximumSize().

```
167 {
168     //KDChart::DataDimensionsList l( d->grid->updateData( this ) );
169     //qDebug() << "AbstractCoordinatePlane::gridDimensionsList() Y-range:" << l.last().end - l.last().
170     //qDebug() << "AbstractCoordinatePlane::gridDimensionsList() X-range:" << l.first().end - l.first().
171     return d->grid->updateData( this );
172 }
```

#### 9.56.4.18 [QRect AbstractAreaBase::innerRect \(\) const](#) [protected, inherited]

Definition at line 228 of file KDChartAbstractAreaBase.cpp.

References KDChart::AbstractAreaBase::areaGeometry(), and KDChart::AbstractAreaBase::getFrameLeadings().

Referenced by KDChart::TextArea::paintAll(), and KDChart::AbstractArea::paintAll().

```

229 {
230     int left;
231     int top;
232     int right;
233     int bottom;
234     setFrameLeadings( left, top, right, bottom );
235     return
236         QRect( QPoint(0,0), areaGeometry().size() )
237             .adjusted( left, top, -right, -bottom );
238 }

```

#### 9.56.4.19 bool KDChart::AbstractCoordinatePlane::isEmpty () const [virtual, inherited]

pure virtual in [QLayoutItem](#)

Definition at line 205 of file KDChartAbstractCoordinatePlane.cpp.

Referenced by KDChart::CartesianCoordinatePlane::layoutDiagrams().

```

206 {
207     return false; // never empty!
208     // coordinate planes with no associated diagrams
209     // are showing a default grid of ( )1..10, 1..10) stepWidth 1
210 }

```

#### 9.56.4.20 bool KDChart::AbstractCoordinatePlane::isRubberBandZoomingEnabled () const [inherited]

##### Returns:

Whether zooming with a rubber band using the mouse is enabled.

Definition at line 280 of file KDChartAbstractCoordinatePlane.cpp.

References d.

```

281 {
282     return d->enableRubberBandZooming;
283 }

```

#### 9.56.4.21 const bool KDChart::AbstractCoordinatePlane::isVisiblePoint (const QPointF & point) const [inherited]

Tests, if a point is visible on the coordinate plane.

##### Note:

Before calling this function the point must have been translated into coordinate plane space.

Definition at line 403 of file KDChartAbstractCoordinatePlane.cpp.

References d.

```

404 {
405     return d->isVisiblePoint( this, point );
406 }

```

**9.56.4.22 void TernaryCoordinatePlane::layoutDiagrams () [virtual]**

Distribute the available space among the diagrams and axes.

Implements [KDChart::AbstractCoordinatePlane](#).

Definition at line 75 of file `KDChartTernaryCoordinatePlane.cpp`.

References [KDChart::AbstractArea::areaGeometry\(\)](#), [d](#), [KDChart::AbstractCoordinatePlane::diagram\(\)](#), [KDChart::AbstractCoordinatePlane::diagrams\(\)](#), [KDChart::TernaryAxis::requiredMargins\(\)](#), [TriangleHeight](#), and [TriangleWidth](#).

```

76 { // this is our "resize event":
77 // all diagrams always take the same space, nothing to be done here
78 // the "inner" margin (adjustments to diagram coordinates)
79 QRectF diagramNativeRectangle ( QPointF( 0.0, 0.0 ),
80                                     QSizeF( TriangleWidth, TriangleHeight ) );
81 QPair<QSizeF, QSizeF> margins = grid()->requiredMargins();
82 d->diagramRect = areaGeometry();
83 diagramNativeRectangle.adjust
84     (-margins.first.width(), -margins.first.height(),
85      margins.second.width(), margins.second.height() );
86
87 // the "outer" margin (distance between diagram contents and area,
88 // determined by axis label overlap
89 {
90     QSizeF topleft( 0.0, 0.0 );
91     QSizeF bottomRight( 0.0, 0.0 );
92     _FOREACH( AbstractDiagram* abstractDiagram, diagrams() ) {
93         AbstractTernaryDiagram* diagram =
94             qobject_cast<AbstractTernaryDiagram*>( abstractDiagram );
95         Q_ASSERT( diagram );
96         _FOREACH( TernaryAxis* axis, diagram->axes() ) {
97             QPair<QSizeF, QSizeF> margin = axis->requiredMargins();
98             topleft = topleft.expandedTo( margin.first );
99             bottomRight = bottomRight.expandedTo( margin.second );
100         }
101     }
102     d->diagramRectContainer =
103         d->diagramRect.adjusted( topleft.width(),
104                                 topleft.height(),
105                                 -bottomRight.width(),
106                                 -bottomRight.height() );
107 }
108
109 // now calculate isometric projection, x and y widget coordinate
110 // units, and location of (0.0, 0.0) in diagram coordinates
111 QPointF zeroZeroPoint = d->diagramRectContainer.bottomLeft();
112 double w = d->diagramRectContainer.width();
113 double h = d->diagramRectContainer.height();
114 double usableWidth;
115 double usableHeight;
116
117 if ( TriangleHeight * w > h ) {
118     // shorten width:
119     usableWidth = h / diagramNativeRectangle.height();
120     usableHeight = h;
121     zeroZeroPoint.setX( zeroZeroPoint.x() + ( w - usableWidth ) / 2 );
122 } else {
123     // reduce height:
124     usableWidth = w;
125     usableHeight = diagramNativeRectangle.height() * w;
126     zeroZeroPoint.setY( zeroZeroPoint.y() - ( h - usableHeight ) / 2 );
127 }
128 // the rectangle has 1 as it's width, and TriangleHeight as it's
129 // height - so this is how we translate that to widget coordinates:
130 d->xUnit = usableWidth / diagramNativeRectangle.width(); // only because we normalize the values t

```

```

131     d->yUnit = -usableHeight / diagramNativeRectangle.height();
132
133     // now move zeroZeroPoint so that it does not include the tick marks
134     {
135         double descent = diagramNativeRectangle.height() - TriangleHeight;
136         double rightShift = -diagramNativeRectangle.x();
137         zeroZeroPoint += QPointF( rightShift * d->xUnit, descent * d->yUnit );
138     }
139
140     d->diagramRect.setBottomLeft( zeroZeroPoint );
141     d->diagramRect.setTopRight( QPointF( usableWidth, -usableHeight ) + zeroZeroPoint );
142 }

```

#### 9.56.4.23 void KDChart::AbstractCoordinatePlane::layoutPlanes () [slot, inherited]

Calling [layoutPlanes\(\)](#) on the plane triggers the global [KDChart::Chart::slotLayoutPlanes\(\)](#).

Definition at line 263 of file [KDChartAbstractCoordinatePlane.cpp](#).

References [KDChart::AbstractCoordinatePlane::needLayoutPlanes\(\)](#).

Referenced by [KDChart::AbstractCoordinatePlane::addDiagram\(\)](#), [KDChart::CartesianAxis::layoutPlanes\(\)](#), [KDChart::AbstractCartesianDiagram::layoutPlanes\(\)](#), and [KDChart::AbstractCoordinatePlane::replaceDiagram\(\)](#).

```

264 {
265     //qDebug("KDChart::AbstractCoordinatePlane::relayout() called");
266     emit needLayoutPlanes();
267 }

```

#### 9.56.4.24 int AbstractArea::leftOverlap (bool *doNotRecalculate* = false) const [virtual, inherited]

This is called at layout time by [KDChart::AutoSpacerLayoutItem::sizeHint\(\)](#).

The method triggers [AbstractArea::sizeHint\(\)](#) to find out the amount of overlap at the left edge of the area.

##### Note:

The default implementation is not using any caching, it might make sense to implement a more sophisticated solution for derived classes that have complex work to do in [sizeHint\(\)](#). All we have here is a primitive flag to be set by the caller if it is sure that no [sizeHint\(\)](#) needs to be called.

Definition at line 77 of file [KDChartAbstractArea.cpp](#).

References [d](#).

Referenced by [KDChart::CartesianAxis::maximumSize\(\)](#).

```

78 {
79     // Re-calculate the sizes,
80     // so we also get the amountOf..Overlap members set newly:
81     if( ! doNotRecalculate )
82         sizeHint();
83     return d->amountOfLeftOverlap;
84 }

```

#### 9.56.4.25 **QSize KDChart::AbstractCoordinatePlane::maximumSize () const** [virtual, inherited]

pure virtual in [QLayoutItem](#)

Definition at line 217 of file KDChartAbstractCoordinatePlane.cpp.

Referenced by KDChart::AbstractCoordinatePlane::sizeHint().

```
218 {
219     // No maximum size set. Especially not parent()->size(), we are not layouting
220     // to the parent widget's size when using Chart::paint()!
221     return QSize(QLAYOUTSIZE_MAX, QLAYOUTSIZE_MAX);
222 }
```

#### 9.56.4.26 **QSize KDChart::AbstractCoordinatePlane::minimumSize () const** [virtual, inherited]

pure virtual in [QLayoutItem](#)

Definition at line 224 of file KDChartAbstractCoordinatePlane.cpp.

```
225 {
226     return QSize(60, 60); // this default can be overwritten by derived classes
227 }
```

#### 9.56.4.27 **QSize TernaryCoordinatePlane::minimumSizeHint () const** [virtual]

[reimplemented]

Reimplemented from [KDChart::AbstractCoordinatePlane](#).

Definition at line 150 of file KDChartTernaryCoordinatePlane.cpp.

```
151 {
152     // FIXME temp
153     return QSize();
154 }
```

#### 9.56.4.28 **void KDChart::AbstractCoordinatePlane::mouseDoubleClickEvent (QMouseEvent \* event)** [inherited]

Definition at line 325 of file KDChartAbstractCoordinatePlane.cpp.

References [d](#), and [KDChart::AbstractCoordinatePlane::mousePressEvent\(\)](#).

Referenced by [KDChart::Chart::mouseDoubleClickEvent\(\)](#).

```
326 {
327     if( event->button() == Qt::RightButton )
328     {
329         // otherwise the second click gets lost
330         // which is pretty annoying when zooming out fast
331         mousePressEvent( event );
332     }
333     KDAB_FOREACH( AbstractDiagram * a, d->diagrams )
```



```

334     {
335         a->mouseDoubleClickEvent( event );
336     }
337 }

```

#### 9.56.4.29 void KDChart::AbstractCoordinatePlane::mouseMoveEvent (QMouseEvent \* event) [inherited]

Definition at line 387 of file KDChartAbstractCoordinatePlane.cpp.

References `d`, and `KDChart::AbstractCoordinatePlane::geometry()`.

Referenced by `KDChart::Chart::mouseMoveEvent()`.

```

388 {
389     if( d->rubberBand != 0 )
390     {
391         const QRect normalized = QRect( d->rubberBandOrigin, event->pos() ).normalized();
392         d->rubberBand->setGeometry( normalized & geometry() );
393
394         event->accept();
395     }
396
397     KDAB_FOREACH( AbstractDiagram * a, d->diagrams )
398     {
399         a->mouseMoveEvent( event );
400     }
401 }

```

#### 9.56.4.30 void KDChart::AbstractCoordinatePlane::mousePressEvent (QMouseEvent \* event) [inherited]

Definition at line 285 of file KDChartAbstractCoordinatePlane.cpp.

References `d`, `KDChart::AbstractCoordinatePlane::parent()`, `KDChart::AbstractCoordinatePlane::setZoomCenter()`, `KDChart::AbstractCoordinatePlane::setZoomFactorX()`, and `KDChart::AbstractCoordinatePlane::setZoomFactorY()`.

Referenced by `KDChart::AbstractCoordinatePlane::mouseDoubleClickEvent()`, and `KDChart::Chart::mousePressEvent()`.

```

286 {
287     if( event->button() == Qt::LeftButton )
288     {
289         if( d->enableRubberBandZooming && d->rubberBand == 0 )
290             d->rubberBand = new QRubberBand( QRubberBand::Rectangle, qobject_cast< QWidget* >( parent() ) );
291
292         if( d->rubberBand != 0 )
293         {
294             d->rubberBandOrigin = event->pos();
295             d->rubberBand->setGeometry( QRect( event->pos(), QSize() ) );
296             d->rubberBand->show();
297
298             event->accept();
299         }
300     }
301     else if( event->button() == Qt::RightButton )
302     {
303         if( d->enableRubberBandZooming && !d->rubberBandZoomConfigHistory.isEmpty() )
304         {

```

```

305         // restore the last config from the stack
306         ZoomParameters config = d->rubberBandZoomConfigHistory.pop();
307         setZoomFactorX( config.xFactor );
308         setZoomFactorY( config.yFactor );
309         setZoomCenter( config.center() );
310
311         QWidget* const p = qobject_cast< QWidget* >( parent() );
312         if( p != 0 )
313             p->update();
314
315         event->accept();
316     }
317 }
318
319 KDAB_FOREACH( AbstractDiagram * a, d->diagrams )
320 {
321     a->mousePressEvent( event );
322 }
323 }

```

#### 9.56.4.31 void KDChart::AbstractCoordinatePlane::mouseReleaseEvent (QMouseEvent \* event) [inherited]

Definition at line 339 of file KDChartAbstractCoordinatePlane.cpp.

References `d`, `KDChart::AbstractCoordinatePlane::geometry()`, `KDChart::AbstractCoordinatePlane::setZoomCenter()`, `KDChart::AbstractCoordinatePlane::setZoomFactorX()`, `KDChart::AbstractCoordinatePlane::setZoomFactorY()`, `KDChart::AbstractCoordinatePlane::zoomCenter()`, `KDChart::AbstractCoordinatePlane::zoomFactorX()`, and `KDChart::AbstractCoordinatePlane::zoomFactorY()`.

Referenced by `KDChart::Chart::mouseReleaseEvent()`.

```

340 {
341     if( d->rubberBand != 0 )
342     {
343         // save the old config on the stack
344         d->rubberBandZoomConfigHistory.push( ZoomParameters( zoomFactorX(), zoomFactorY(), zoomCenter() ) );
345
346         // this is the height/width of the rubber band in pixel space
347         const double rubberWidth = static_cast< double >( d->rubberBand->width() );
348         const double rubberHeight = static_cast< double >( d->rubberBand->height() );
349
350         // this is the center of the rubber band in pixel space
351         const double rubberCenterX = static_cast< double >( d->rubberBand->geometry().center().x() - 0.5 );
352         const double rubberCenterY = static_cast< double >( d->rubberBand->geometry().center().y() - 0.5 );
353
354         // this is the height/width of the plane in pixel space
355         const double myWidth = static_cast< double >( geometry().width() );
356         const double myHeight = static_cast< double >( geometry().height() );
357
358         // this describes the new center of zooming, relative to the plane pixel space
359         const double newCenterX = rubberCenterX / myWidth / zoomFactorX() + zoomCenter().x() - 0.5 / zoomFactorX();
360         const double newCenterY = rubberCenterY / myHeight / zoomFactorY() + zoomCenter().y() - 0.5 / zoomFactorY();
361
362         // this will be the new zoom factor
363         const double newZoomFactorX = zoomFactorX() * myWidth / rubberWidth;
364         const double newZoomFactorY = zoomFactorY() * myHeight / rubberHeight;
365
366         // and this the new center
367         const QPointF newZoomCenter( newCenterX, newCenterY );
368
369         setZoomFactorX( newZoomFactorX );
370         setZoomFactorY( newZoomFactorY );

```

```

371         setZoomCenter( newZoomCenter );
372
373
374         d->rubberBand->parentWidget()->update();
375         delete d->rubberBand;
376         d->rubberBand = 0;
377
378         event->accept();
379     }
380
381     KDAB_FOREACH( AbstractDiagram * a, d->diagrams )
382     {
383         a->mouseReleaseEvent( event );
384     }
385 }

```

#### 9.56.4.32 void KDChart::AbstractCoordinatePlane::needLayoutPlanes () [signal, inherited]

Emitted when plane needs to trigger the Chart's layouting of the coord.  
planes.

Referenced by KDChart::AbstractCoordinatePlane::layoutPlanes().

#### 9.56.4.33 void KDChart::AbstractCoordinatePlane::needRelayout () [signal, inherited]

Emitted when plane needs to trigger the Chart's layouting.

Referenced by KDChart::AbstractCoordinatePlane::relayout().

#### 9.56.4.34 void KDChart::AbstractCoordinatePlane::needUpdate () [signal, inherited]

Emitted when plane needs to update its drawings.

Referenced by KDChart::AbstractCoordinatePlane::update().

#### 9.56.4.35 void TernaryCoordinatePlane::paint (QPainter \*) [virtual]

Implements [KDChart::AbstractLayoutItem](#).

Definition at line 162 of file KDChartTernaryCoordinatePlane.cpp.

References [KDChart::AbstractArea::areaGeometry\(\)](#), [d](#), and [KDChart::AbstractCoordinatePlane::diagrams\(\)](#).

```

163 {
164     PainterSaver s( painter );
165     // FIXME: this is not a good location for that:
166     painter->setRenderHint(QPainter::Antialiasing, true );
167
168     // painter->setPen( QColor( "gold" ) );
169     // painter->setBrush( QColor( "gold" ) );
170     // painter->drawRect( d->diagramRectContainer );
171
172     AbstractDiagramList diags = diagrams();
173     if ( !diags.isEmpty() )
174     {
175         PaintContext ctx;

```

```

176         ctx.setPainter ( painter );
177         ctx.setCoordinatePlane ( this );
178         const QRectF drawArea( areaGeometry() );
179         ctx.setRectangle ( drawArea );
180
181         // enabling clipping so that we're not drawing outside
182         //     QRect clipRect = drawArea.toRect().adjusted( -1, -1, 1, 1 );
183         //     QRegion clipRegion( clipRect );
184         //     painter->setClipRegion( clipRegion );
185
186         // paint the coordinate system rulers:
187         Q_ASSERT( d->grid != 0 );
188         d->grid->drawGrid( &ctx );
189
190         // paint the diagrams:
191         for ( int i = 0; i < diags.size(); i++ )
192         {
193             PainterSaver diagramPainterSaver( painter );
194             diags[i]->paint ( &ctx );
195         }
196     }
197 }

```

#### 9.56.4.36 void AbstractArea::paintAll (QPainter & *painter*) [virtual, inherited]

Call `paintAll`, if you want the background and the frame to be drawn before the normal `paint()` is invoked automatically.

Reimplemented from [KDChart::AbstractLayoutItem](#).

Reimplemented in [KDChart::TernaryAxis](#).

Definition at line 123 of file `KDChartAbstractArea.cpp`.

References `KDChart::AbstractArea::areaGeometry()`, `d`, `KDChart::AbstractAreaBase::innerRect()`, `KDChart::AbstractLayoutItem::paint()`, `KDChart::AbstractAreaBase::paintBackground()`, and `KDChart::AbstractAreaBase::paintFrame()`.

Referenced by `KDChart::AbstractArea::paintIntoRect()`.

```

124 {
125     // Paint the background and frame
126     const QRect overlappingArea( geometry().adjusted(
127         -d->amountOfLeftOverlap,
128         -d->amountOfTopOverlap,
129         d->amountOfRightOverlap,
130         d->amountOfBottomOverlap ) );
131     paintBackground( painter, overlappingArea );
132     paintFrame( painter, overlappingArea );
133
134     // temporarily adjust the widget size, to be sure all content gets calculated
135     // to fit into the inner rectangle
136     const QRect oldGeometry( areaGeometry() );
137     QRect inner( innerRect() );
138     inner.moveTo(
139         oldGeometry.left() + inner.left(),
140         oldGeometry.top() + inner.top() );
141     const bool needAdjustGeometry = oldGeometry != inner;
142     if( needAdjustGeometry )
143         setGeometry( inner );
144     paint( &painter );
145     if( needAdjustGeometry )
146         setGeometry( oldGeometry );
147     //qDebug() << "AbstractAreaWidget::paintAll() done.";
148 }

```

#### 9.56.4.37 void AbstractAreaBase::paintBackground (QPainter & *painter*, const QRect & *rectangle*) [virtual, inherited]

Definition at line 196 of file KDChartAbstractAreaBase.cpp.

References `d`, and `KDChart::AbstractAreaBase::paintBackgroundAttributes()`.

Referenced by `KDChart::TextArea::paintAll()`, `KDChart::AbstractAreaWidget::paintAll()`, and `KDChart::AbstractArea::paintAll()`.

```

197 {
198     Q_ASSERT_X ( d != 0, "AbstractAreaBase::paintBackground()",
199                 "Private class was not initialized!" );
200     paintBackgroundAttributes( painter, rect, d->backgroundAttributes );
201 }
```

#### 9.56.4.38 void AbstractAreaBase::paintBackgroundAttributes (QPainter & *painter*, const QRect & *rectangle*, const KDChart::BackgroundAttributes & *attributes*) [static, inherited]

Definition at line 127 of file KDChartAbstractAreaBase.cpp.

References `KDChart::BackgroundAttributes::BackgroundPixmapModeCentered`, `KDChart::BackgroundAttributes::BackgroundPixmapModeNone`, `KDChart::BackgroundAttributes::BackgroundPixmapModeScaled`, `KDChart::BackgroundAttributes::BackgroundPixmapModeStretched`, `KDChart::BackgroundAttributes::brush()`, `KDChart::BackgroundAttributes::isVisible()`, `KDChart::BackgroundAttributes::pixmap()`, and `KDChart::BackgroundAttributes::pixmapMode()`.

Referenced by `KDChart::AbstractAreaBase::paintBackground()`.

```

129 {
130     if( !attributes.isVisible() ) return;
131
132     /* first draw the brush (may contain a pixmap)*/
133     if( Qt::NoBrush != attributes.brush().style() ) {
134         KDChart::PainterSaver painterSaver( &painter );
135         painter.setPen( Qt::NoPen );
136         const QPointF newTopLeft( painter.deviceMatrix().map( rect.topLeft() ) );
137         painter.setBrushOrigin( newTopLeft );
138         painter.setBrush( attributes.brush() );
139         painter.drawRect( rect.adjusted( 0, 0, -1, -1 ) );
140     }
141     /* next draw the backPixmap over the brush */
142     if( !attributes.pixmap().isNull() &&
143         attributes.pixmapMode() != BackgroundAttributes::BackgroundPixmapModeNone ) {
144         QPointF ol = rect.topLeft();
145         if( BackgroundAttributes::BackgroundPixmapModeCentered == attributes.pixmapMode() )
146         {
147             ol.setX( rect.center().x() - attributes.pixmap().width() / 2 );
148             ol.setY( rect.center().y() - attributes.pixmap().height() / 2 );
149             painter.drawPixmap( ol, attributes.pixmap() );
150         } else {
151             QMatrix m;
152             double zW = (double)rect.width() / (double)attributes.pixmap().width();
153             double zH = (double)rect.height() / (double)attributes.pixmap().height();
154             switch( attributes.pixmapMode() ) {
155             case BackgroundAttributes::BackgroundPixmapModeScaled:
156                 {
157                     double z;
158                     z = qMin( zW, zH );
159                     m.scale( z, z );
160                 }
```

```

161         break;
162         case BackgroundAttributes::BackgroundPixmapModeStretched:
163             m.scale( zW, zH );
164             break;
165         default:
166             ; // Cannot happen, previously checked
167     }
168     QPixmap pm = attributes.pixmap().transformed( m );
169     ol.setX( rect.center().x() - pm.width() / 2 );
170     ol.setY( rect.center().y() - pm.height() / 2 );
171     painter.drawPixmap( ol, pm );
172 }
173 }
174 }

```

#### 9.56.4.39 void KDChart::AbstractLayoutItem::paintCtx ([PaintContext](#) \* *context*) [virtual, inherited]

Default impl: Paint the complete item using its layouted position and size.

Reimplemented in [KDChart::CartesianAxis](#), and [KDChart::TernaryAxis](#).

Definition at line 77 of file KDChartLayoutItems.cpp.

References [KDChart::AbstractLayoutItem::paint\(\)](#), and [KDChart::PaintContext::painter\(\)](#).

```

78 {
79     if( context )
80         paint( context->painter() );
81 }

```

#### 9.56.4.40 void AbstractAreaBase::paintFrame (QPainter & *painter*, const QRect & *rectangle*) [virtual, inherited]

Definition at line 204 of file KDChartAbstractAreaBase.cpp.

References [d](#), and [KDChart::AbstractAreaBase::paintFrameAttributes\(\)](#).

Referenced by [KDChart::TextArea::paintAll\(\)](#), [KDChart::AbstractAreaWidget::paintAll\(\)](#), and [KDChart::AbstractArea::paintAll\(\)](#).

```

205 {
206     Q_ASSERT_X ( d != 0, "AbstractAreaBase::paintFrame()",
207                 "Private class was not initialized!" );
208     paintFrameAttributes( painter, rect, d->frameAttributes );
209 }

```

#### 9.56.4.41 void AbstractAreaBase::paintFrameAttributes (QPainter & *painter*, const QRect & *rectangle*, const [KDChart::FrameAttributes](#) & *attributes*) [static, inherited]

Definition at line 177 of file KDChartAbstractAreaBase.cpp.

References [KDChart::FrameAttributes::isVisible\(\)](#), and [KDChart::FrameAttributes::pen\(\)](#).

Referenced by [KDChart::AbstractAreaBase::paintFrame\(\)](#).

```

179 {
180
181     if( !attributes.isVisible() ) return;
182
183     // Note: We set the brush to NoBrush explicitly here.
184     //       Otherwise we might get a filled rectangle, so any
185     //       previously drawn background would be overwritten by that area.
186
187     const QPen    oldPen( painter.pen() );
188     const QBrush  oldBrush( painter.brush() );
189     painter.setPen( attributes.pen() );
190     painter.setBrush( Qt::NoBrush );
191     painter.drawRect( rect.adjusted( 0, 0, -1, -1 ) );
192     painter.setBrush( oldBrush );
193     painter.setPen( oldPen );
194 }

```

#### 9.56.4.42 void AbstractArea::paintIntoRect (QPainter & *painter*, const QRect & *rect*) [virtual, inherited]

Draws the background and frame, then calls [paint\(\)](#).

In most cases there is no need to overwrite this method in a derived class, but you would overwrite [AbstractLayoutItem::paint\(\)](#) instead.

Definition at line 111 of file KDChartAbstractArea.cpp.

References [KDChart::AbstractArea::paintAll\(\)](#).

```

112 {
113     const QRect oldGeometry( geometry() );
114     if( oldGeometry != rect )
115         setGeometry( rect );
116     painter.translate( rect.left(), rect.top() );
117     paintAll( painter );
118     painter.translate( -rect.left(), -rect.top() );
119     if( oldGeometry != rect )
120         setGeometry( oldGeometry );
121 }

```

#### 9.56.4.43 const [KDChart::Chart](#) \* KDChart::AbstractCoordinatePlane::parent () const [inherited]

Definition at line 194 of file KDChartAbstractCoordinatePlane.cpp.

References [d](#).

```

195 {
196     return d->parent;
197 }

```

#### 9.56.4.44 [KDChart::Chart](#) \* KDChart::AbstractCoordinatePlane::parent () [inherited]

Definition at line 199 of file KDChartAbstractCoordinatePlane.cpp.

References [d](#).

Referenced by `KDChart::AbstractCoordinatePlane::AbstractCoordinatePlane()`, `KDChart::CartesianAxis::maximumSize()`, `KDChart::AbstractCoordinatePlane::mousePressEvent()`, and `KDChart::AbstractCoordinatePlane::setParent()`.

```
200 {
201     return d->parent;
202 }
```

#### 9.56.4.45 `QLayout* KDChart::AbstractLayoutItem::parentLayout ()` [inherited]

Definition at line 76 of file `KDChartLayoutItems.h`.

```
77     {
78         return mParentLayout;
79     }
```

#### 9.56.4.46 `void KDChart::AbstractArea::positionChanged (AbstractArea *)` [signal, inherited]

Referenced by `KDChart::AbstractArea::positionHasChanged()`.

#### 9.56.4.47 `void AbstractArea::positionHasChanged ()` [protected, virtual, inherited]

Reimplemented from `KDChart::AbstractAreaBase`.

Definition at line 155 of file `KDChartAbstractArea.cpp`.

References `KDChart::AbstractArea::positionChanged()`.

```
156 {
157     emit positionChanged( this );
158 }
```

#### 9.56.4.48 `void KDChart::AbstractCoordinatePlane::propertiesChanged ()` [signal, inherited]

Emitted upon change of a property of the Coordinate Plane or any of its components.

Referenced by `KDChart::CartesianCoordinatePlane::addDiagram()`, `KDChart::CartesianCoordinatePlane::adjustHorizontalRangeToData()`, `KDChart::CartesianCoordinatePlane::adjustRangesToData()`, `KDChart::CartesianCoordinatePlane::adjustVerticalRangeToData()`, `KDChart::CartesianCoordinatePlane::setAutoAdjustGridToZoom()`, `KDChart::CartesianCoordinatePlane::setAutoAdjustHorizontalRangeToData()`, `KDChart::CartesianCoordinatePlane::setAutoAdjustVerticalRangeToData()`, `KDChart::CartesianCoordinatePlane::setAxesCalcModes()`, `KDChart::CartesianCoordinatePlane::setAxesCalcModeX()`, `KDChart::CartesianCoordinatePlane::setAxesCalcModeY()`, `KDChart::PolarCoordinatePlane::setGridAttributes()`, `KDChart::CartesianCoordinatePlane::setGridAttributes()`, `KDChart::CartesianCoordinatePlane::setHorizontalRange()`, `KDChart::CartesianCoordinatePlane::setHorizontalRangeReversed()`, `KDChart::CartesianCoordinatePlane::setIsometricScaling()`, `KDChart::CartesianCoordinatePlane::setVerticalRange()`, `KDChart::CartesianCoordinatePlane::setVerticalRangeReversed()`, `KDChart::CartesianCoordinatePlane::setZoomCenter()`, `KDChart::CartesianCoordinatePlane::setZoomFactorX()`, and `KDChart::CartesianCoordinatePlane::setZoomFactorY()`.



#### 9.56.4.49 [AbstractCoordinatePlane](#) \* KDChart::AbstractCoordinatePlane::referenceCoordinatePlane () const [inherited]

There are two ways, in which planes can be caused to interact, in where they are put layouting wise: The first is the reference plane.

If such a reference plane is set, on a plane, it will use the same cell in the layout as that one. In addition to this, planes can share an axis. In that case they will be layed out in relation to each other as suggested by the position of the axis. If, for example Plane1 and Plane2 share an axis at position Left, that will result in the layout: Axis Plane1 Plane 2, vertically. If Plane1 also happens to be Plane2's reference plane, both planes are drawn over each other. The reference plane concept allows two planes to share the same space even if neither has any axis, and in case there are shared axis, it is used to decided, whether the planes should be painted on top of each other or layed out vertically or horizontally next to each other.

#### Returns:

The reference coordinate plane associated with this one.

Definition at line 184 of file KDChartAbstractCoordinatePlane.cpp.

References d.

```
185 {
186     return d->referenceCoordinatePlane;
187 }
```

#### 9.56.4.50 void KDChart::AbstractCoordinatePlane::relayout () [slot, inherited]

Calling [relayout\(\)](#) on the plane triggers the global KDChart::Chart::slotRelayout().

Definition at line 257 of file KDChartAbstractCoordinatePlane.cpp.

References KDChart::AbstractCoordinatePlane::needRelayout().

```
258 {
259     //qDebug("KDChart::AbstractCoordinatePlane::relayout() called");
260     emit needRelayout();
261 }
```

#### 9.56.4.51 void KDChart::AbstractLayoutItem::removeFromParentLayout () [inherited]

Definition at line 80 of file KDChartLayoutItems.h.

Referenced by KDChart::Chart::takeCoordinatePlane().

```
81     {
82         if( mParentLayout ){
83             if( widget() )
84                 mParentLayout->removeWidget( widget() );
85             else
86                 mParentLayout->removeItem( this );
87         }
88     }
```

**9.56.4.52** `void AbstractCoordinatePlane::replaceDiagram (AbstractDiagram * diagram, AbstractDiagram * oldDiagram = 0)` [virtual, inherited]

Replaces the old diagram, or appends the diagram, if there is none yet.

**Parameters:**

*diagram* The diagram to be used instead of the old diagram. This parameter must not be zero, or the method will do nothing.

*oldDiagram* The diagram to be removed by the new diagram. This diagram will be deleted automatically. If the parameter is omitted, the very first diagram will be replaced. In case, there was no diagram yet, the new diagram will just be added.

**Note:**

If you want to re-use the old diagram, call `takeDiagram` and `addDiagram`, instead of using `replaceDiagram`.

**See also:**

[addDiagram](#), [takeDiagram](#)

Definition at line 86 of file `KDChartAbstractCoordinatePlane.cpp`.

References `KDChart::AbstractCoordinatePlane::addDiagram()`, `d`, `KDChart::AbstractCoordinatePlane::diagram()`, `KDChart::AbstractCoordinatePlane::layoutDiagrams()`, `KDChart::AbstractCoordinatePlane::layoutPlanes()`, `KDChart::AbstractCoordinatePlane::takeDiagram()`, and `KDChart::AbstractCoordinatePlane::update()`.

Referenced by `KDChart::Widget::setType()`.

```

87 {
88     if( diagram && oldDiagram_ != diagram ){
89         AbstractDiagram* oldDiagram = oldDiagram_;
90         if( d->diagrams.count() ){
91             if( ! oldDiagram )
92                 oldDiagram = d->diagrams.first();
93             takeDiagram( oldDiagram );
94         }
95         delete oldDiagram;
96         addDiagram( diagram );
97         layoutDiagrams();
98         layoutPlanes(); // there might be new axes, etc
99         update();
100     }
101 }
```

**9.56.4.53** `int AbstractArea::rightOverlap (bool doNotRecalculate = false) const` [virtual, inherited]

This is called at layout time by [KDChart::AutoSpacerLayoutItem::sizeHint\(\)](#).

The method triggers `AbstractArea::sizeHint()` to find out the amount of overlap at the right edge of the area.

**Note:**

The default implementation is not using any caching, it might make sense to implement a more sophisticated solution for derived classes that have complex work to do in `sizeHint()`. All we have here is a primitive flag to be set by the caller if it is sure that no `sizeHint()` needs to be called.

Definition at line 85 of file KDChartAbstractArea.cpp.

References [d](#).

Referenced by [KDChart::CartesianAxis::maximumSize\(\)](#).

```
86 {
87     // Re-calculate the sizes,
88     // so we also get the amountOf..Overlap members set newly:
89     if( ! doNotRecalculate )
90         sizeHint();
91     return d->amountOfRightOverlap;
92 }
```

#### 9.56.4.54 void AbstractAreaBase::setBackgroundAttributes (const [BackgroundAttributes](#) & *a*) [inherited]

Definition at line 111 of file KDChartAbstractAreaBase.cpp.

References [d](#), and [KDChart::AbstractAreaBase::positionHasChanged\(\)](#).

```
112 {
113     if( d->backgroundAttributes == a )
114         return;
115
116     d->backgroundAttributes = a;
117     positionHasChanged();
118 }
```

#### 9.56.4.55 void AbstractAreaBase::setFrameAttributes (const [FrameAttributes](#) & *a*) [inherited]

Definition at line 97 of file KDChartAbstractAreaBase.cpp.

References [d](#), and [KDChart::AbstractAreaBase::positionHasChanged\(\)](#).

Referenced by [KDChart::Legend::clone\(\)](#).

```
98 {
99     if( d->frameAttributes == a )
100         return;
101
102     d->frameAttributes = a;
103     positionHasChanged();
104 }
```

#### 9.56.4.56 void KDChart::AbstractCoordinatePlane::setGeometry (const [QRect](#) & *r*) [virtual, inherited]

pure virtual in [QLayoutItem](#)

#### Note:

Do not call this function directly, unless you know exactly what you are doing. Geometry management is done by KD Chart's internal layouting measures.

Reimplemented in [KDChart::CartesianCoordinatePlane](#).

Definition at line 236 of file KDChartAbstractCoordinatePlane.cpp.

References [d](#).

Referenced by [KDChart::CartesianCoordinatePlane::setGeometry\(\)](#).

```

237 {
238 //      qDebug() << "KDChart::AbstractCoordinatePlane::setGeometry(" << r << ") called";
239     if( d->geometry != r ){
240         d->geometry = r;
241         // Note: We do *not* call update() here
242         //      because it would invoke KDChart::update() recursively.
243     }
244 }
```

#### 9.56.4.57 void KDChart::AbstractCoordinatePlane::setGlobalGridAttributes (const [GridAttributes](#) &) [inherited]

Set the grid attributes to be used by this coordinate plane.

To disable grid painting, for example, your code should like this:

```

GridAttributes ga = plane->globalGridAttributes();
ga.setGlobalGridVisible( false );
plane->setGlobalGridAttributes( ga );
```

See also:

[globalGridAttributes](#)  
[CartesianCoordinatePlane::setGridAttributes](#)

Definition at line 155 of file KDChartAbstractCoordinatePlane.cpp.

References [d](#), and [KDChart::AbstractCoordinatePlane::update\(\)](#).

```

156 {
157     d->gridAttributes = a;
158     update();
159 }
```

#### 9.56.4.58 void KDChart::AbstractCoordinatePlane::setGridNeedsRecalculate () [slot, inherited]

Used by the chart to clear the cached grid data.

Definition at line 174 of file KDChartAbstractCoordinatePlane.cpp.

References [d](#).

Referenced by [KDChart::Chart::resizeEvent\(\)](#).

```

175 {
176     d->grid->setNeedRecalculate();
177 }
```

**9.56.4.59 void KDChart::AbstractCoordinatePlane::setParent ([Chart](#) \* *parent*)** [inherited]

Called internally by [KDChart::Chart](#).

Definition at line 189 of file KDChartAbstractCoordinatePlane.cpp.

References [d](#), and [KDChart::AbstractCoordinatePlane::parent\(\)](#).

Referenced by [KDChart::Chart::addCoordinatePlane\(\)](#), and [KDChart::Chart::takeCoordinatePlane\(\)](#).

```
190 {
191     d->parent = parent;
192 }
```

**9.56.4.60 void KDChart::AbstractLayoutItem::setParentLayout (QLayout \* *lay*)** [inherited]

Definition at line 72 of file KDChartLayoutItems.h.

```
73     {
74         mParentLayout = lay;
75     }
```

**9.56.4.61 void KDChart::AbstractLayoutItem::setParentWidget ([QWidget](#) \* *widget*)**  
[virtual, inherited]

Inform the item about its widget: This enables the item, to trigger that widget's update, whenever the size of the item's contents has changed.

Thus, you need to call setParentWidget on every item, that has a non-fixed size.

Definition at line 64 of file KDChartLayoutItems.cpp.

References [KDChart::AbstractLayoutItem::mParent](#).

Referenced by [KDChart::HeaderFooter::setParent\(\)](#), and [KDChart::AbstractCartesianDiagram::takeAxis\(\)](#).

```
65 {
66     mParent = widget;
67 }
```

**9.56.4.62 void KDChart::AbstractCoordinatePlane::setReferenceCoordinatePlane ([AbstractCoordinatePlane](#) \* *plane*)** [inherited]

Set another coordinate plane to be used as the reference plane for this one.

**Parameters:**

*plane* The coordinate plane to be used the reference plane for this one.

**See also:**

[referenceCoordinatePlane](#)

Definition at line 179 of file KDChartAbstractCoordinatePlane.cpp.

References d.

```
180 {
181     d->referenceCoordinatePlane = plane;
182 }
```

#### 9.56.4.63 void KDChart::AbstractCoordinatePlane::setRubberBandZoomingEnabled (bool *enable*) [inherited]

Enables or disables zooming with a rubber band using the mouse.

Definition at line 269 of file KDChartAbstractCoordinatePlane.cpp.

References d.

```
270 {
271     d->enableRubberBandZooming = enable;
272
273     if( !enable && d->rubberBand != 0 )
274     {
275         delete d->rubberBand;
276         d->rubberBand = 0;
277     }
278 }
```

#### 9.56.4.64 virtual void KDChart::AbstractCoordinatePlane::setZoomCenter (const QPointF & *center*) [virtual, inherited]

Set the point (in value coordinates) to be used as the center point in zoom operations.

##### Parameters:

*center* The point to use.

Reimplemented in [KDChart::CartesianCoordinatePlane](#), and [KDChart::PolarCoordinatePlane](#).

Definition at line 182 of file KDChartAbstractCoordinatePlane.h.

Referenced by [KDChart::AbstractCoordinatePlane::mousePressEvent\(\)](#), and [KDChart::AbstractCoordinatePlane::mouseReleaseEvent\(\)](#).

```
182 { Q_UNUSED( center ); }
```

#### 9.56.4.65 virtual void KDChart::AbstractCoordinatePlane::setZoomFactorX (double *factor*) [virtual, inherited]

Sets the zoom factor in horizontal direction, that is applied to all coordinate transformations.

##### Parameters:

*factor* The new zoom factor

Reimplemented in [KDChart::CartesianCoordinatePlane](#), and [KDChart::PolarCoordinatePlane](#).

Definition at line 162 of file KDChartAbstractCoordinatePlane.h.

Referenced by [KDChart::AbstractCoordinatePlane::mousePressEvent\(\)](#), and [KDChart::AbstractCoordinatePlane::mouseReleaseEvent\(\)](#).

```
162 { Q_UNUSED( factor ); }
```

#### 9.56.4.66 virtual void KDChart::AbstractCoordinatePlane::setZoomFactorY (double *factor*) [virtual, inherited]

Sets the zoom factor in vertical direction, that is applied to all coordinate transformations.

##### Parameters:

*factor* The new zoom factor

Reimplemented in [KDChart::CartesianCoordinatePlane](#), and [KDChart::PolarCoordinatePlane](#).

Definition at line 169 of file KDChartAbstractCoordinatePlane.h.

Referenced by [KDChart::AbstractCoordinatePlane::mousePressEvent\(\)](#), and [KDChart::AbstractCoordinatePlane::mouseReleaseEvent\(\)](#).

```
169 { Q_UNUSED( factor ); }
```

#### 9.56.4.67 [AbstractCoordinatePlane](#) \* KDChart::AbstractCoordinatePlane::sharedAxisMaster-Plane (QPainter \**p* = 0) [virtual, inherited]

Reimplemented in [KDChart::CartesianCoordinatePlane](#).

Definition at line 408 of file KDChartAbstractCoordinatePlane.cpp.

Referenced by [KDChart::Plotter::paint\(\)](#), [KDChart::LineDiagram::paint\(\)](#), and [KDChart::BarDiagram::paint\(\)](#).

```
409 {
410     Q_UNUSED( p );
411     return this;
412 }
```

#### 9.56.4.68 QSize KDChart::AbstractCoordinatePlane::sizeHint () const [virtual, inherited]

pure virtual in [QLayoutItem](#)

Definition at line 229 of file KDChartAbstractCoordinatePlane.cpp.

References [KDChart::AbstractCoordinatePlane::maximumSize\(\)](#).

```
230 {
231     // we return our maximum (which is the full size of the Chart)
232     // even if we know the plane will be smaller
233     return maximumSize();
234 }
```

**9.56.4.69** `void KDChart::AbstractLayoutItem::sizeHintChanged () const` [virtual, inherited]

Report changed size hint: ask the parent widget to recalculate the layout.

Definition at line 86 of file `KDChartLayoutItems.cpp`.

References `KDChart::AbstractLayoutItem::mParent`.

Referenced by `KDChart::TextLayoutItem::sizeHint()`.

```

87 {
88     // This is exactly like what QWidget::updateGeometry does.
89     // qDebug("KDChart::AbstractLayoutItem::sizeHintChanged() called");
90     if( mParent ) {
91         if ( mParent->layout() )
92             mParent->layout()->invalidate();
93         else
94             QApplication::postEvent( mParent, new QEvent( QEvent::LayoutRequest ) );
95     }
96 }
```

**9.56.4.70** `QSizePolicy TernaryCoordinatePlane::sizePolicy () const` [virtual]

[reimplemented]

Reimplemented from [KDChart::AbstractCoordinatePlane](#).

Definition at line 156 of file `KDChartTernaryCoordinatePlane.cpp`.

```

157 {
158     return QSizePolicy( QSizePolicy::MinimumExpanding,
159                        QSizePolicy::MinimumExpanding );
160 }
```

**9.56.4.71** `void AbstractCoordinatePlane::takeDiagram (AbstractDiagram * diagram)` [virtual, inherited]

Removes the diagram from the plane, without deleting it.

The plane no longer owns the diagram, so it is the caller's responsibility to delete the diagram.

**See also:**

[addDiagram](#), [replaceDiagram](#)

Definition at line 104 of file `KDChartAbstractCoordinatePlane.cpp`.

References `d`, `KDChart::AbstractCoordinatePlane::diagram()`, `KDChart::AbstractCoordinatePlane::layoutDiagrams()`, `KDChart::AbstractDiagram::setCoordinatePlane()`, and `KDChart::AbstractCoordinatePlane::update()`.

Referenced by `KDChart::AbstractCoordinatePlane::replaceDiagram()`.

```

105 {
106     const int idx = d->diagrams.indexOf( diagram );
107     if( idx != -1 ){
108         d->diagrams.removeAt( idx );
109     }
```



```

109         diagram->setParent( 0 );
110         diagram->setCoordinatePlane( 0 );
111         layoutDiagrams();
112         update();
113     }
114 }
```

#### 9.56.4.72 int AbstractArea::topOverlap (bool *doNotRecalculate* = false) const [virtual, inherited]

This is called at layout time by [KDChart::AutoSpacerLayoutItem::sizeHint\(\)](#).

The method triggers AbstractArea::sizeHint() to find out the amount of overlap at the top edge of the area.

##### Note:

The default implementation is not using any caching, it might make sense to implement a more sophisticated solution for derived classes that have complex work to do in sizeHint(). All we have here is a primitive flag to be set by the caller if it is sure that no sizeHint() needs to be called.

Definition at line 93 of file KDChartAbstractArea.cpp.

References d.

Referenced by KDChart::CartesianAxis::maximumSize().

```

94 {
95     // Re-calculate the sizes,
96     // so we also get the amountOf..Overlap members set newly:
97     if( ! doNotRecalculate )
98         sizeHint();
99     return d->amountOfTopOverlap;
100 }
```

#### 9.56.4.73 const QPointF TernaryCoordinatePlane::translate (const QPointF & *diagramPoint*) const [virtual]

Translate the given point in value space coordinates to a position in pixel space.

##### Parameters:

*diagramPoint* The point in value coordinates.

##### Returns:

The translated point.

Implements [KDChart::AbstractCoordinatePlane](#).

Definition at line 144 of file KDChartTernaryCoordinatePlane.cpp.

References d.

Referenced by KDChart::TernaryPointDiagram::paint(), KDChart::TernaryLineDiagram::paint(), and KDChart::TernaryAxis::paintCtx().

```

145 {
146     return QPointF( d->diagramRect.bottomLeft().x() + point.x() * d->xUnit,
147                   d->diagramRect.bottomLeft().y() + point.y() * d->yUnit );
148 }
```

**9.56.4.74 void KDChart::AbstractCoordinatePlane::update ()** [slot, inherited]

Calling [update\(\)](#) on the plane triggers the global `KDChart::Chart::update()`.

Definition at line 251 of file `KDChartAbstractCoordinatePlane.cpp`.

References `KDChart::AbstractCoordinatePlane::needUpdate()`.

Referenced by `KDChart::AbstractCoordinatePlane::addDiagram()`, `KDChart::CartesianCoordinatePlane::layoutDiagrams()`, `KDChart::AbstractCoordinatePlane::replaceDiagram()`, `KDChart::PolarCoordinatePlane::resetGridAttributes()`, `KDChart::CartesianCoordinatePlane::resetGridAttributes()`, `KDChart::AbstractCoordinatePlane::setGlobalGridAttributes()`, `KDChart::PolarCoordinatePlane::setGridAttributes()`, `KDChart::CartesianCoordinatePlane::setGridAttributes()`, and `KDChart::AbstractCoordinatePlane::takeDiagram()`.

```
252 {
253     //qDebug("KDChart::AbstractCoordinatePlane::update() called");
254     emit needUpdate();
255 }
```

**9.56.4.75 virtual QPointF KDChart::AbstractCoordinatePlane::zoomCenter () const**  
[virtual, inherited]**Returns:**

The center point (in value coordinates) of the coordinate plane, that is used for zoom operations.

Reimplemented in [KDChart::CartesianCoordinatePlane](#), and [KDChart::PolarCoordinatePlane](#).

Definition at line 175 of file `KDChartAbstractCoordinatePlane.h`.

Referenced by `KDChart::AbstractCoordinatePlane::mouseReleaseEvent()`.

```
175 { return QPointF(0.0, 0.0); }
```

**9.56.4.76 virtual double KDChart::AbstractCoordinatePlane::zoomFactorX () const**  
[virtual, inherited]**Returns:**

The zoom factor in horizontal direction, that is applied to all coordinate transformations.

Reimplemented in [KDChart::CartesianCoordinatePlane](#), and [KDChart::PolarCoordinatePlane](#).

Definition at line 149 of file `KDChartAbstractCoordinatePlane.h`.

Referenced by `KDChart::AbstractCoordinatePlane::mouseReleaseEvent()`.

```
149 { return 1.0; }
```

**9.56.4.77 virtual double KDChart::AbstractCoordinatePlane::zoomFactorY () const**  
[virtual, inherited]**Returns:**

The zoom factor in vertical direction, that is applied to all coordinate transformations.

Reimplemented in [KDChart::CartesianCoordinatePlane](#), and [KDChart::PolarCoordinatePlane](#).

Definition at line 155 of file KDChartAbstractCoordinatePlane.h.

Referenced by [KDChart::AbstractCoordinatePlane::mouseReleaseEvent\(\)](#).

```
155 { return 1.0; }
```

## 9.56.5 Member Data Documentation

### 9.56.5.1 QWidget\* [KDChart::AbstractLayoutItem::mParent](#) [protected, inherited]

Definition at line 90 of file KDChartLayoutItems.h.

Referenced by [KDChart::AbstractLayoutItem::setParentWidget\(\)](#), [KDChart::TextLayoutItem::setText\(\)](#), [KDChart::TextLayoutItem::setTextAttributes\(\)](#), and [KDChart::AbstractLayoutItem::sizeHintChanged\(\)](#).

### 9.56.5.2 QLayout\* [KDChart::AbstractLayoutItem::mParentLayout](#) [protected, inherited]

Definition at line 91 of file KDChartLayoutItems.h.

Referenced by [KDChart::AutoSpacerLayoutItem::paint\(\)](#).

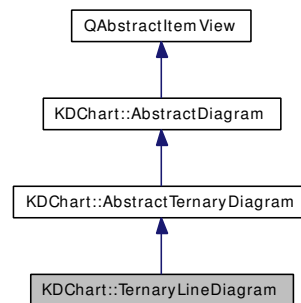
The documentation for this class was generated from the following files:

- [KDChartTernaryCoordinatePlane.h](#)
- [KDChartTernaryCoordinatePlane.cpp](#)

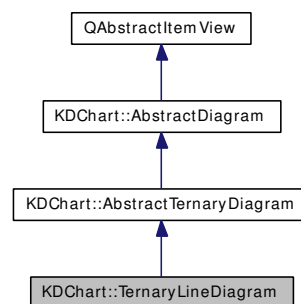
## 9.57 KDChart::TernaryLineDiagram Class Reference

```
#include <KDChartTernaryLineDiagram.h>
```

Inheritance diagram for KDChart::TernaryLineDiagram:



Collaboration diagram for KDChart::TernaryLineDiagram:



### 9.57.1 Detailed Description

A [TernaryLineDiagram](#) is a line diagram with a ternary coordinate plane.

Definition at line 42 of file KDChartTernaryLineDiagram.h.

### Signals

- void [dataHidden](#) ()  
*This signal is emitted, when the hidden status of at least one data cell was (un)set.*
- void [layoutChanged](#) ([AbstractDiagram](#) \*)  
*Diagrams are supposed to emit this signal, when the layout of one of their element changes.*
- void [modelsChanged](#) ()  
*This signal is emitted, when either the model or the [AttributesModel](#) is replaced.*
- void [propertiesChanged](#) ()  
*Emitted upon change of a property of the Diagram.*

## Public Member Functions

- bool [allowOverlappingDataValueTexts](#) () const  
*Returns:*  
*Whether data value labels are allowed to overlap.*
- bool [antiAliasing](#) () const  
*Returns:*  
*Whether anti-aliasing is to be used for rendering this diagram.*
- virtual [AttributesModel](#) \* [attributesModel](#) () const  
*Returns the [AttributesModel](#), that is used by this diagram.*
- QBrush [brush](#) (const QModelIndex &index) const  
*Retrieve the brush to be used, for painting the datapoint at the given index in the model.*
- QBrush [brush](#) (int dataset) const  
*Retrieve the brush to be used for the given dataset.*
- QBrush [brush](#) () const  
*Retrieve the brush to be used for painting datapoints globally.*
- bool [compare](#) (const [AbstractDiagram](#) \*other) const  
*Returns true if both diagrams have the same settings.*
- [AbstractCoordinatePlane](#) \* [coordinatePlane](#) () const  
*The coordinate plane associated with the diagram.*
- const QPair< QPointF, QPointF > [dataBoundaries](#) () const  
*Return the bottom left and top right data point, that the diagram will display (unless the grid adjusts these values).*
- virtual void [dataChanged](#) (const QModelIndex &topLeft, const QModelIndex &bottomRight)  
*[reimplemented]*
- QList< QBrush > [datasetBrushes](#) () const  
*The set of dataset brushes currently used, for use in legends, etc.*
- int [datasetDimension](#) () const  
*The dataset dimension of a diagram determines, how many value dimensions it expects each datapoint to have.*
- QStringList [datasetLabels](#) () const  
*The set of dataset labels currently displayed, for use in legends, etc.*
- QList< [MarkerAttributes](#) > [datasetMarkers](#) () const  
*The set of dataset markers currently used, for use in legends, etc.*
- QList< QPen > [datasetPens](#) () const  
*The set of dataset pens currently used, for use in legends, etc.*

- [DataValueAttributes dataValueAttributes](#) (const QModelIndex &index) const  
*Retrieve the [DataValueAttributes](#) for the given index.*
- [DataValueAttributes dataValueAttributes](#) (int column) const  
*Retrieve the [DataValueAttributes](#) for the given dataset.*
- [DataValueAttributes dataValueAttributes](#) () const  
*Retrieve the [DataValueAttributes](#) specified globally.*
- virtual void [doItemsLayout](#) ()  
*[reimplemented]*
- virtual int [horizontalOffset](#) () const  
*[reimplemented]*
- virtual QModelIndex [indexAt](#) (const QPoint &point) const  
*[reimplemented]*
- QModelIndexList [indexesAt](#) (const QPoint &point) const  
*This method is added alongside with [indexAt](#) from QAIM, since in kdchart multiple indexes can be displayed at the same spot.*
- bool [isHidden](#) (const QModelIndex &index) const  
*Retrieve the hidden status for the given index.*
- bool [isHidden](#) (int column) const  
*Retrieve the hidden status for the given dataset.*
- bool [isHidden](#) () const  
*Retrieve the hidden status specified globally.*
- virtual bool [isIndexHidden](#) (const QModelIndex &index) const  
*[reimplemented]*
- QStringList [itemRowLabels](#) () const  
*The set of item row labels currently displayed, for use in Abscissa axes, etc.*
- virtual QModelIndex [moveCursor](#) (CursorAction cursorAction, Qt::KeyboardModifiers modifiers)  
*[reimplemented]*
- void [paint](#) (PaintContext \*paintContext)  
*Draw the diagram contents to the rectangle and painter, that are passed in as part of the paint context.*
- void [paintDataValueText](#) (QPainter \*painter, const QModelIndex &index, const QPointF &pos, double value)
- void [paintMarker](#) (QPainter \*painter, const QModelIndex &index, const QPointF &pos)
- virtual void [paintMarker](#) (QPainter \*painter, const [MarkerAttributes](#) &markerAttributes, const QBrush &brush, const QPen &, const QPointF &point, const QSizeF &size)
- QPen [pen](#) (const QModelIndex &index) const

*Retrieve the pen to be used, for painting the datapoint at the given index in the model.*

- QPen [pen](#) (int dataset) const

*Retrieve the pen to be used for the given dataset.*

- QPen [pen](#) () const

*Retrieve the pen to be used for painting datapoints globally.*

- bool [percentMode](#) () const

- void [resize](#) (const QSizeF &area)

*Called by the widget's sizeEvent.*

- virtual void [scrollTo](#) (const QModelIndex &index, ScrollHint hint=EnsureVisible)  
*[reimplemented]*

- void [setAllowOverlappingDataValueTexts](#) (bool allow)

*Set whether data value labels are allowed to overlap.*

- void [setAntiAliasing](#) (bool enabled)

*Set whether anti-aliasing is to be used while rendering this diagram.*

- virtual void [setAttributesModel](#) ([AttributesModel](#) \*model)

*Associate an [AttributesModel](#) with this diagram.*

- void [setBrush](#) (const QBrush &brush)

*Set the brush to be used, for painting all datasets in the model.*

- void [setBrush](#) (int dataset, const QBrush &brush)

*Set the brush to be used, for painting the given dataset.*

- void [setBrush](#) (const QModelIndex &index, const QBrush &brush)

*Set the brush to be used, for painting the datapoint at the given index.*

- virtual void [setCoordinatePlane](#) ([AbstractCoordinatePlane](#) \*plane)

*Set the coordinate plane associated with the diagram.*

- void [setDatasetDimension](#) (int dimension)

*Sets the dataset dimension of the diagram.*

- void [setDataValueAttributes](#) (const [DataValueAttributes](#) &a)

*Set the [DataValueAttributes](#) for all datapoints in the model.*

- void [setDataValueAttributes](#) (int dataset, const [DataValueAttributes](#) &a)

*Set the [DataValueAttributes](#) for the given dataset.*

- void [setDataValueAttributes](#) (const QModelIndex &index, const [DataValueAttributes](#) &a)

*Set the [DataValueAttributes](#) for the given index.*

- void [setHidden](#) (bool hidden)

*Hide (or unhide, resp.)*

- void [setHidden](#) (int column, bool hidden)  
*Hide (or unhide, resp).*
- void [setHidden](#) (const QModelIndex &index, bool hidden)  
*Hide (or unhide, resp).*
- virtual void [setModel](#) (QAbstractItemModel \*model)  
*Associate a model with the diagram.*
- void [setPen](#) (const QPen &pen)  
*Set the pen to be used, for painting all datasets in the model.*
- void [setPen](#) (int dataset, const QPen &pen)  
*Set the pen to be used, for painting the given dataset.*
- void [setPen](#) (const QModelIndex &index, const QPen &pen)  
*Set the pen to be used, for painting the datapoint at the given index.*
- void [setPercentMode](#) (bool percent)
- virtual void [setRootIndex](#) (const QModelIndex &idx)  
*Set the root index in the model, where the diagram starts referencing data for display.*
- virtual void [setSelection](#) (const QRect &rect, QItemSelectionModel::SelectionFlags command)  
*[reimplemented]*
- void [setUnitPrefix](#) (const QString &prefix, Qt::Orientation orientation)  
*Sets the unit prefix for all values.*
- void [setUnitPrefix](#) (const QString &prefix, int column, Qt::Orientation orientation)  
*Sets the unit prefix for one value.*
- void [setUnitSuffix](#) (const QString &suffix, Qt::Orientation orientation)  
*Sets the unit suffix for all values.*
- void [setUnitSuffix](#) (const QString &suffix, int column, Qt::Orientation orientation)  
*Sets the unit suffix for one value.*
- [TernaryLineDiagram](#) (QWidget \*parent=0, [TernaryCoordinatePlane](#) \*plane=0)
- QString [unitPrefix](#) (Qt::Orientation orientation) const  
*Returns the global unit prefix.*
- QString [unitPrefix](#) (int column, Qt::Orientation orientation, bool fallback=false) const  
*Returns the unit prefix for a special value.*
- QString [unitSuffix](#) (Qt::Orientation orientation) const  
*Returns the global unit suffix.*
- QString [unitSuffix](#) (int column, Qt::Orientation orientation, bool fallback=false) const  
*Returns the unit suffix for a special value.*



- void [update](#) () const
- void [useDefaultColors](#) ()  
*Set the palette to be used, for painting datasets to the default palette.*
- void [useRainbowColors](#) ()  
*Set the palette to be used, for painting datasets to the rainbow palette.*
- virtual bool [usesExternalAttributesModel](#) () const  
*Returns whether the diagram is using its own built-in attributes model or an attributes model that was set via [setAttributesModel](#).*
- void [useSubduedColors](#) ()  
*Set the palette to be used, for painting datasets to the subdued palette.*
- virtual int [verticalOffset](#) () const  
*[reimplemented]*
- virtual QRect [visualRect](#) (const QModelIndex &index) const  
*[reimplemented]*
- virtual QRegion [visualRegionForSelection](#) (const QItemSelection &selection) const  
*[reimplemented]*
- virtual ~[TernaryLineDiagram](#) ()

## Protected Member Functions

- QModelIndex [attributesModelRootIndex](#) () const
- const QPair< QPointF, QPointF > [calculateDataBoundaries](#) () const
- virtual bool [checkInvariants](#) (bool justReturnTheStatus=false) const
- QModelIndex [columnToIndex](#) (int column) const
- virtual void [paintDataValueTexts](#) (QPainter \*painter)
- virtual void [paintMarkers](#) (QPainter \*painter)
- void [setAttributesModelRootIndex](#) (const QModelIndex &)
- void [setDataBoundariesDirty](#) () const
- double [valueForCell](#) (int row, int column) const  
*Helper method, retrieving the data value (DisplayRole) for a given row and column.*

## 9.57.2 Constructor & Destructor Documentation

### 9.57.2.1 TernaryLineDiagram::TernaryLineDiagram (QWidget \*parent = 0, TernaryCoordinatePlane \*plane = 0) [explicit]

Definition at line 55 of file KDChartTernaryLineDiagram.cpp.

References [KDChart::AbstractDiagram::attributesModel\(\)](#), [KDChart::AbstractDiagram::dataValueAttributes\(\)](#), [KDChart::DataValueLabelAttributesRole](#), [KDChart::MarkerAttributes::MarkerCircle](#), [KDChart::AbstractDiagram::setDatasetDimension\(\)](#), [KDChart::AttributesModel::setDefaultForRole\(\)](#), [KDChart::DataValueAttributes::setMarkerAttributes\(\)](#), [KDChart::MarkerAttributes::setMarkerStyle\(\)](#), [KDChart::MarkerAttributes::setVisible\(\)](#), and [KDChart::DataValueAttributes::setVisible\(\)](#).

```

57 : AbstractTernaryDiagram( new Private(), parent, plane )
58 {
59     init();
60     setDatasetDimension( 3 ); // the third column is implicit
61
62     DataValueAttributes dataValueAttributes;
63     dataValueAttributes.setVisible( true );
64     MarkerAttributes markerAttributes;
65     markerAttributes.setMarkerStyle( MarkerAttributes::MarkerCircle );
66     markerAttributes.setVisible( true );
67     dataValueAttributes.setMarkerAttributes( markerAttributes );
68     attributesModel()->setDefaultForRole(
69         KDChart::DataValueLabelAttributesRole,
70         qVariantFromValue( dataValueAttributes ) );
71 }

```

### 9.57.2.2 TernaryLineDiagram::~TernaryLineDiagram () [virtual]

Definition at line 73 of file KDChartTernaryLineDiagram.cpp.

```

74 {
75 }

```

## 9.57.3 Member Function Documentation

### 9.57.3.1 bool AbstractDiagram::allowOverlappingDataValueTexts () const [inherited]

#### Returns:

Whether data value labels are allowed to overlap.

Definition at line 440 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::compare().

```

441 {
442     return d->allowOverlappingDataValueTexts;
443 }

```

### 9.57.3.2 bool AbstractDiagram::antiAliasing () const [inherited]

#### Returns:

Whether anti-aliasing is to be used for rendering this diagram.

Definition at line 451 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::compare().

```

452 {
453     return d->antiAliasing;
454 }

```

### 9.57.3.3 [AttributesModel](#) \* AbstractDiagram::attributesModel () const [virtual, inherited]

Returns the [AttributesModel](#), that is used by this diagram.

By default each diagram owns its own [AttributesModel](#), which should never be deleted. Only if a user-supplied [AttributesModel](#) has been set does the pointer returned here not belong to the diagram.

#### Returns:

The [AttributesModel](#) associated with the diagram.

#### See also:

[setAttributesModel](#)

Definition at line 280 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::brush(), KDChart::AbstractDiagram::compare(), KDChart::AbstractDiagram::datasetBrushes(), KDChart::AbstractDiagram::datasetLabels(), KDChart::AbstractDiagram::datasetMarkers(), KDChart::AbstractDiagram::datasetPens(), KDChart::AbstractDiagram::dataValueAttributes(), KDChart::AbstractDiagram::isHidden(), KDChart::AbstractDiagram::itemRowLabels(), KDChart::RingDiagram::paint(), KDChart::PolarDiagram::paint(), KDChart::AbstractDiagram::pen(), KDChart::AbstractCartesianDiagram::setAttributesModel(), KDChart::BarDiagram::setBarAttributes(), KDChart::AbstractDiagram::setBrush(), KDChart::AbstractCartesianDiagram::setModel(), KDChart::AbstractDiagram::setPen(), and TernaryLineDiagram().

```
281 {
282     return d->attributesModel;
283 }
```

### 9.57.3.4 [QModelIndex](#) AbstractDiagram::attributesModelRootIndex () const [protected, inherited]

returns a QModelIndex pointing into the [AttributesModel](#) that corresponds to the root index of the diagram.

Definition at line 302 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::datasetBrushes(), KDChart::AbstractDiagram::datasetLabels(), KDChart::AbstractDiagram::datasetMarkers(), KDChart::AbstractDiagram::datasetPens(), KDChart::AbstractDiagram::itemRowLabels(), KDChart::Plotter::numberOfAbscissaSegments(), KDChart::LineDiagram::numberOfAbscissaSegments(), KDChart::BarDiagram::numberOfAbscissaSegments(), KDChart::Plotter::numberOfOrdinateSegments(), KDChart::LineDiagram::numberOfOrdinateSegments(), KDChart::BarDiagram::numberOfOrdinateSegments(), KDChart::AbstractDiagram::valueForCell(), and KDChart::LineDiagram::valueForCellTesting().

```
303 {
304     if ( !d->attributesModelRootIndex.isValid() )
305         d->attributesModelRootIndex = d->attributesModel->mapFromSource( rootIndex() );
306     return d->attributesModelRootIndex;
307 }
```

**9.57.3.5 QBrush AbstractDiagram::brush (const QModelIndex & *index*) const** [inherited]

Retrieve the brush to be used, for painting the datapoint at the given index in the model.

**Parameters:**

*index* The index of the datapoint in the model.

**Returns:**

The brush to use for painting.

Definition at line 838 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::brush(), KDChart::AttributesModel::data(), KDChart::DatasetBrushRole, and KDChart::AbstractDiagram::datasetDimension().

```

839 {
840     if( datasetDimension() > 1 )
841         return brush( index.column() );
842     return qVariantValue<QBrush>(
843         attributesModel()->data(
844             attributesModel()->mapFromSource( index ),
845             DatasetBrushRole ) );
846 }
```

**9.57.3.6 QBrush AbstractDiagram::brush (int *dataset*) const** [inherited]

Retrieve the brush to be used for the given dataset.

This will fall back automatically to what was set at model level, if there are no dataset specific settings.

**Parameters:**

*dataset* The dataset to retrieve the brush for.

**Returns:**

The brush to use for painting.

Definition at line 828 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::columnToIndex(), KDChart::AttributesModel::data(), KDChart::DatasetBrushRole, and KDChart::AbstractDiagram::datasetDimension().

```

829 {
830     if( datasetDimension() > 1 )
831         dataset /= datasetDimension();
832     return qVariantValue<QBrush>(
833         attributesModel()->data(
834             attributesModel()->mapFromSource( columnToIndex( dataset ) ),
835             DatasetBrushRole ) );
836 }
```

**9.57.3.7 QBrush AbstractDiagram::brush () const** [inherited]

Retrieve the brush to be used for painting datapoints globally.

This will fall back automatically to the default settings, if there are no specific settings.

**Returns:**

The brush to use for painting.

Definition at line 822 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AttributesModel::data(), and KDChart::DatasetBrushRole.

Referenced by KDChart::AbstractDiagram::brush(), KDChart::TernaryPointDiagram::paint(), paint(), KDChart::RingDiagram::paint(), KDChart::PolarDiagram::paint(), and KDChart::AbstractDiagram::paintMarker().

```
823 {
824     return qVariantValue<QBrush>(
825         attributesModel()->data( DatasetBrushRole ) );
826 }
```

**9.57.3.8 const QPair< QPointF, QPointF > TernaryLineDiagram::calculateDataBoundaries () const** [protected, virtual]

Implements [KDChart::AbstractTernaryDiagram](#).

Definition at line 153 of file KDChartTernaryLineDiagram.cpp.

References TriangleBottomLeft, TriangleBottomRight, and TriangleHeight.

```
154 {
155     // this is a constant, because we defined it to be one:
156     static QPair<QPointF, QPointF> Boundaries(
157         TriangleBottomLeft,
158         QPointF( TriangleBottomRight.x(), TriangleHeight ) );
159     return Boundaries;
160 }
```

**9.57.3.9 bool AbstractDiagram::checkInvariants (bool *justReturnTheStatus* = false) const** [protected, virtual, inherited]

Definition at line 1060 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::coordinatePlane().

Referenced by KDChart::RingDiagram::calculateDataBoundaries(), KDChart::PolarDiagram::calculateDataBoundaries(), KDChart::Plotter::calculateDataBoundaries(), KDChart::PieDiagram::calculateDataBoundaries(), KDChart::LineDiagram::calculateDataBoundaries(), KDChart::BarDiagram::calculateDataBoundaries(), KDChart::RingDiagram::paint(), KDChart::PolarDiagram::paint(), KDChart::Plotter::paint(), KDChart::PieDiagram::paint(), KDChart::LineDiagram::paint(), KDChart::BarDiagram::paint(), KDChart::AbstractDiagram::paintDataValueTexts(), KDChart::AbstractDiagram::paintMarker(), and KDChart::AbstractDiagram::paintMarkers().

```

1061 {
1062     if( ! justReturnTheStatus ){
1063         Q_ASSERT_X ( model(), "AbstractDiagram::checkInvariants()",
1064                     "There is no usable model set, for the diagram." );
1065
1066         Q_ASSERT_X ( coordinatePlane(), "AbstractDiagram::checkInvariants()",
1067                     "There is no usable coordinate plane set, for the diagram." );
1068     }
1069     return model() && coordinatePlane();
1070 }

```

### 9.57.3.10 QModelIndex AbstractDiagram::columnToIndex (int *column*) const [protected, inherited]

Definition at line 309 of file KDChartAbstractDiagram.cpp.

Referenced by KDChart::BarDiagram::barAttributes(), KDChart::AbstractDiagram::brush(), KDChart::AbstractDiagram::dataValueAttributes(), KDChart::AbstractDiagram::isHidden(), KDChart::Plotter::lineAttributes(), KDChart::LineDiagram::lineAttributes(), KDChart::AbstractDiagram::pen(), KDChart::AbstractPieDiagram::pieAttributes(), KDChart::BarDiagram::threeDBarAttributes(), KDChart::Plotter::threeDLineAttributes(), KDChart::LineDiagram::threeDLineAttributes(), and KDChart::AbstractPieDiagram::threeDPieAttributes().

```

310 { // FIXME (Mirko): shouldn't this be headerData? instead of the index for the first row?
311     if( model() )
312         return QModelIndex( model()->index( 0, column, rootIndex() ) );
313     return QModelIndex();
314 }

```

### 9.57.3.11 bool AbstractDiagram::compare (const [AbstractDiagram](#) \* *other*) const [inherited]

Returns true if both diagrams have the same settings.

Definition at line 129 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::allowOverlappingDataValueTexts(), KDChart::AbstractDiagram::antiAliasing(), KDChart::AbstractDiagram::attributesModel(), KDChart::AttributesModel::compare(), KDChart::AbstractDiagram::datasetDimension(), and KDChart::AbstractDiagram::percentMode().

```

130 {
131     if( other == this ) return true;
132     if( ! other ){
133         //qDebug() << "AbstractDiagram::compare() cannot compare to Null pointer";
134         return false;
135     }
136     /*
137     qDebug() << "\n                AbstractDiagram::compare() QAbstractScrollArea:";
138     // compare QAbstractScrollArea properties
139     qDebug() <<
140         ((horizontalScrollBarPolicy() == other->horizontalScrollBarPolicy()) &&
141         (verticalScrollBarPolicy() == other->verticalScrollBarPolicy()));
142     qDebug() << "AbstractDiagram::compare() QFrame:";
143     // compare QFrame properties
144     qDebug() <<
145         ((frameShadow() == other->frameShadow()) &&
146         (frameShape() == other->frameShape()) &&

```

```

147         (frameWidth() == other->frameWidth()) &&
148         (lineWidth() == other->lineWidth()) &&
149         (midLineWidth() == other->midLineWidth()));
150     qDebug() << "AbstractDiagram::compare() QAbstractItemView:";
151     // compare QAbstractItemView properties
152     qDebug() <<
153         ((alternatingRowColors() == other->alternatingRowColors()) &&
154         (hasAutoScroll() == other->hasAutoScroll()) &&
155 #if QT_VERSION > 0x040199
156         (dragDropMode() == other->dragDropMode()) &&
157         (dragDropOverwriteMode() == other->dragDropOverwriteMode()) &&
158         (horizontalScrollMode() == other->horizontalScrollMode()) &&
159         (verticalScrollMode() == other->verticalScrollMode()) &&
160 #endif
161         (dragEnabled() == other->dragEnabled()) &&
162         (editTriggers() == other->editTriggers()) &&
163         (iconSize() == other->iconSize()) &&
164         (selectionBehavior() == other->selectionBehavior()) &&
165         (selectionMode() == other->selectionMode()) &&
166         (showDropIndicator() == other->showDropIndicator()) &&
167         (tabKeyNavigation() == other->tabKeyNavigation()) &&
168         (textElideMode() == other->textElideMode()));
169     qDebug() << "AbstractDiagram::compare() AttributesModel: ";
170     // compare all of the properties stored in the attributes model
171     qDebug() << attributesModel()->compare( other->attributesModel() );
172     qDebug() << "AbstractDiagram::compare() own:";
173     // compare own properties
174     qDebug() <<
175         ((rootIndex().column() == other->rootIndex().column()) &&
176         (rootIndex().row() == other->rootIndex().row()) &&
177         (allowOverlappingDataValueTexts() == other->allowOverlappingDataValueTexts()) &&
178         (antiAliasing() == other->antiAliasing()) &&
179         (percentMode() == other->percentMode()) &&
180         (datasetDimension() == other->datasetDimension()));
181     */
182     return // compare QAbstractScrollArea properties
183         (horizontalScrollBarPolicy() == other->horizontalScrollBarPolicy()) &&
184         (verticalScrollBarPolicy() == other->verticalScrollBarPolicy()) &&
185         // compare QFrame properties
186         (frameShadow() == other->frameShadow()) &&
187         (frameShape() == other->frameShape()) &&
188     // frameWidth is a read-only property defined by the style, it should not be in here:
189     // (frameWidth() == other->frameWidth()) &&
190     (lineWidth() == other->lineWidth()) &&
191     (midLineWidth() == other->midLineWidth()) &&
192     // compare QAbstractItemView properties
193     (alternatingRowColors() == other->alternatingRowColors()) &&
194     (hasAutoScroll() == other->hasAutoScroll()) &&
195 #if QT_VERSION > 0x040199
196     (dragDropMode() == other->dragDropMode()) &&
197     (dragDropOverwriteMode() == other->dragDropOverwriteMode()) &&
198     (horizontalScrollMode() == other->horizontalScrollMode()) &&
199     (verticalScrollMode() == other->verticalScrollMode()) &&
200 #endif
201     (dragEnabled() == other->dragEnabled()) &&
202     (editTriggers() == other->editTriggers()) &&
203     (iconSize() == other->iconSize()) &&
204     (selectionBehavior() == other->selectionBehavior()) &&
205     (selectionMode() == other->selectionMode()) &&
206     (showDropIndicator() == other->showDropIndicator()) &&
207     (tabKeyNavigation() == other->tabKeyNavigation()) &&
208     (textElideMode() == other->textElideMode()) &&
209     // compare all of the properties stored in the attributes model
210     attributesModel()->compare( other->attributesModel() ) &&
211     // compare own properties
212     (rootIndex().column() == other->rootIndex().column()) &&
213     (rootIndex().row() == other->rootIndex().row()) &&

```

```

214         (allowOverlappingDataValueTexts() == other->allowOverlappingDataValueTexts()) &&
215         (antiAliasing() == other->antiAliasing()) &&
216         (percentMode() == other->percentMode()) &&
217         (datasetDimension() == other->datasetDimension());
218     }

```

### 9.57.3.12 **AbstractCoordinatePlane \* AbstractDiagram::coordinatePlane () const** [inherited]

The coordinate plane associated with the diagram.

This determines how coordinates in value space are mapped into pixel space. By default this is a [Cartesian-CoordinatePlane](#).

#### Returns:

The coordinate plane associated with the diagram.

Definition at line 220 of file `KDChartAbstractDiagram.cpp`.

References `d`.

Referenced by `KDChart::AbstractDiagram::checkInvariants()`, `KDChart::AbstractCartesianDiagram::layoutPlanes()`, `KDChart::PolarDiagram::paint()`, `KDChart::AbstractDiagram::paintDataValueTexts()`, `KDChart::AbstractDiagram::paintMarkers()`, `KDChart::AbstractPolarDiagram::polarCoordinatePlane()`, and `KDChart::AbstractCartesianDiagram::setCoordinatePlane()`.

```

221 {
222     return d->plane;
223 }

```

### 9.57.3.13 **const QPair< QPointF, QPointF > AbstractDiagram::dataBoundaries () const** [inherited]

Return the bottom left and top right data point, that the diagram will display (unless the grid adjusts these values).

This method returns a cached result of calculations done by `calculateDataBoundaries`. Classes derived from [AbstractDiagram](#) must implement the `calculateDataBoundaries` function, to specify their own way of calculating the data boundaries. If derived classes want to force recalculation of the data boundaries, they can call [setDataBoundariesDirty\(\)](#)

Returned value is in diagram coordinates.

Definition at line 225 of file `KDChartAbstractDiagram.cpp`.

References `KDChart::AbstractDiagram::calculateDataBoundaries()`, and `d`.

Referenced by `KDChart::CartesianCoordinatePlane::getRawDataBoundingRectFromDiagrams()`, `KDChart::PolarCoordinatePlane::layoutDiagrams()`, `KDChart::Plotter::paint()`, `KDChart::LineDiagram::paint()`, and `KDChart::BarDiagram::paint()`.

```

226 {
227     if ( d->databoundariesDirty ){
228         d->databoundaries = calculateDataBoundaries ();
229         d->databoundariesDirty = false;
230     }
231     return d->databoundaries;
232 }

```



### 9.57.3.14 void AbstractDiagram::dataChanged (const QModelIndex & *topLeft*, const QModelIndex & *bottomRight*) [virtual, inherited]

[reimplemented]

Definition at line 330 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::setDataBoundariesDirty().

```

332 {
333     Q_UNUSED( topLeft );
334     Q_UNUSED( bottomRight );
335     // We are still too dumb to do intelligent updates...
336     setDataBoundariesDirty();
337     scheduleDelayedItemsLayout();
338 }
```

### 9.57.3.15 void KDChart::AbstractDiagram::dataHidden () [signal, inherited]

This signal is emitted, when the hidden status of at least one data cell was (un)set.

Referenced by KDChart::AbstractDiagram::setHidden().

### 9.57.3.16 QList< QBrush > AbstractDiagram::datasetBrushes () const [inherited]

The set of dataset brushes currently used, for use in legends, etc.

#### Note:

Cell-level override brushes, if set, take precedence over the dataset values, so you might need to check these too, in order to find the brush, that is used for a single cell.

#### Returns:

The current set of dataset brushes.

Definition at line 1018 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::attributesModelRootIndex(), KDChart::AttributesModel::columnCount(), KDChart::DatasetBrushRole, and KDChart::AbstractDiagram::datasetDimension().

Referenced by KDChart::Legend::setBrushesFromDiagram().

```

1019 {
1020     QList<QBrush> ret;
1021     if( model() == 0 )
1022         return ret;
1023
1024     const int columnCount = attributesModel()->columnCount(attributesModelRootIndex());
1025     for( int i = 0; i < columnCount / datasetDimension(); ++i )
1026         ret << qVariantValue<QBrush>( attributesModel()->headerData( i, Qt::Vertical, DatasetBrushRole ));
1027
1028     return ret;
1029 }
```

**9.57.3.17 int AbstractDiagram::datasetDimension () const** [inherited]

The dataset dimension of a diagram determines, how many value dimensions it expects each datapoint to have.

For each dimension it will expect one column of values in the model. If the dimensionality is 1, automatic values will be used for the abscissa.

For example a diagram with the default dimension of 1, will have one column per datapoint (the y values) and will use automatic values for the x axis (1, 2, 3, ... n). If the dimension is 2, the diagram will use the first, (and the third, fifth, etc) columns as X values, and the second, (and the fourth, sixth, etc) column as Y values.

**Returns:**

The dataset dimension of the diagram.

Definition at line 1072 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::brush(), KDChart::AbstractDiagram::compare(), KDChart::AbstractDiagram::datasetBrushes(), KDChart::AbstractDiagram::datasetLabels(), KDChart::AbstractDiagram::datasetMarkers(), KDChart::AbstractDiagram::datasetPens(), KDChart::LineDiagram::getCellValues(), KDChart::CartesianCoordinatePlane::getDataDimensionsList(), KDChart::TernaryPointDiagram::paint(), paint(), KDChart::AbstractDiagram::paintDataValueTexts(), KDChart::AbstractDiagram::paintMarkers(), KDChart::AbstractDiagram::pen(), KDChart::AbstractDiagram::setPen(), KDChart::Plotter::setType(), and KDChart::LineDiagram::setType().

```
1073 {
1074     return d->datasetDimension;
1075 }
```

**9.57.3.18 QStringList AbstractDiagram::datasetLabels () const** [inherited]

The set of dataset labels currently displayed, for use in legends, etc.

**Returns:**

The set of dataset labels currently displayed.

Definition at line 1005 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::attributesModelRootIndex(), KDChart::AttributesModel::columnCount(), KDChart::AbstractDiagram::datasetDimension(), and KDChart::AttributesModel::headerData().

```
1006 {
1007     QStringList ret;
1008     if( model() == 0 )
1009         return ret;
1010
1011     const int columnCount = attributesModel()->columnCount(attributesModelRootIndex());
1012     for( int i = 0; i < columnCount / datasetDimension(); ++i )
1013         ret << attributesModel()->headerData( i, Qt::Horizontal, Qt::DisplayRole ).toString();
1014
1015     return ret;
1016 }
```

### 9.57.3.19 QList< MarkerAttributes > AbstractDiagram::datasetMarkers () const [inherited]

The set of dataset markers currently used, for use in legends, etc.

#### Note:

Cell-level override markers, if set, take precedence over the dataset values, so you might need to check these too, in order to find the marker, that is shown for a single cell.

#### Returns:

The current set of dataset brushes.

Definition at line 1044 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::attributesModelRootIndex(), KDChart::AttributesModel::columnCount(), KDChart::AbstractDiagram::datasetDimension(), and KDChart::DataValueLabelAttributesRole.

```

1045 {
1046     QList<MarkerAttributes> ret;
1047     if( model() == 0 )
1048         return ret;
1049
1050     const int columnCount = attributesModel()->columnCount(attributesModelRootIndex());
1051     for( int i = 0; i < columnCount / datasetDimension(); ++i )
1052     {
1053         const DataValueAttributes a =
1054             qVariantValue<DataValueAttributes>( attributesModel()->headerData( i, Qt::Vertical, DataValueLabelAttributesRole ));
1055         ret << a.markerAttributes();
1056     }
1057     return ret;
1058 }
```

### 9.57.3.20 QList< QPen > AbstractDiagram::datasetPens () const [inherited]

The set of dataset pens currently used, for use in legends, etc.

#### Note:

Cell-level override pens, if set, take precedence over the dataset values, so you might need to check these too, in order to find the pens, that is used for a single cell.

#### Returns:

The current set of dataset pens.

Definition at line 1031 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::attributesModelRootIndex(), KDChart::AttributesModel::columnCount(), KDChart::AbstractDiagram::datasetDimension(), and KDChart::DatasetPenRole.

```

1032 {
1033     QList<QPen> ret;
1034     if( model() == 0 )
1035         return ret;
```

```

1036
1037     const int columnCount = attributesModel()->columnCount(attributesModelRootIndex());
1038     for( int i = 0; i < columnCount / datasetDimension(); ++i )
1039         ret << qVariantValue<QPen>( attributesModel()->headerData( i, Qt::Vertical, DatasetPenRole )
1040
1041     return ret;
1042 }

```

### 9.57.3.21 [DataValueAttributes](#) AbstractDiagram::dataValueAttributes (const QModelIndex & index) const [inherited]

Retrieve the [DataValueAttributes](#) for the given index.

This will fall back automatically to what was set at dataset or model level, if there are no datapoint specific settings.

#### Parameters:

**index** The datapoint to retrieve the attributes for.

#### Returns:

The [DataValueAttributes](#) for the given index.

Definition at line 421 of file KDChartAbstractDiagram.cpp.

References [KDChart::AbstractDiagram::attributesModel\(\)](#), [KDChart::AttributesModel::data\(\)](#), and [KDChart::DataValueLabelAttributesRole](#).

```

422 {
423     return qVariantValue<DataValueAttributes>(
424         attributesModel()->data( attributesModel()->mapFromSource(index),
425                                 KDChart::DataValueLabelAttributesRole ) );
426 }

```

### 9.57.3.22 [DataValueAttributes](#) AbstractDiagram::dataValueAttributes (int column) const [inherited]

Retrieve the [DataValueAttributes](#) for the given dataset.

This will fall back automatically to what was set at model level, if there are no dataset specific settings.

#### Parameters:

**column** The dataset to retrieve the attributes for.

#### Returns:

The [DataValueAttributes](#) for the given dataset.

Definition at line 414 of file KDChartAbstractDiagram.cpp.

References [KDChart::AbstractDiagram::attributesModel\(\)](#), [KDChart::AbstractDiagram::columnToIndex\(\)](#), [KDChart::AttributesModel::data\(\)](#), and [KDChart::DataValueLabelAttributesRole](#).

```

415 {
416     return qVariantValue<DataValueAttributes>(
417         attributesModel()->data( attributesModel()->mapFromSource(columnToIndex( column ) ),
418                                 KDChart::DataValueLabelAttributesRole ) );
419 }

```

**9.57.3.23 DataValueAttributes** AbstractDiagram::dataValueAttributes () const [inherited]

Retrieve the [DataValueAttributes](#) specified globally.

This will fall back automatically to the default settings, if there are no specific settings.

**Returns:**

The global [DataValueAttributes](#).

Definition at line 408 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DataValueLabelAttributesRole, and KDChart::AttributesModel::modelData().

Referenced by KDChart::AbstractDiagram::paintDataValueText(), KDChart::AbstractDiagram::paintMarker(), and TernaryLineDiagram().

```
409 {
410     return qVariantValue<DataValueAttributes>(
411         attributesModel()->modelData( KDChart::DataValueLabelAttributesRole ) );
412 }
```

**9.57.3.24 void** AbstractDiagram::doItemsLayout () [virtual, inherited]

[reimplemented]

Definition at line 321 of file KDChartAbstractDiagram.cpp.

References d, and KDChart::AbstractDiagram::update().

```
322 {
323     if ( d->plane ) {
324         d->plane->layoutDiagrams();
325         update();
326     }
327     QAbstractItemView::doItemsLayout();
328 }
```

**9.57.3.25 int** AbstractDiagram::horizontalOffset () const [virtual, inherited]

[reimplemented]

Definition at line 950 of file KDChartAbstractDiagram.cpp.

```
951 { return 0; }
```

**9.57.3.26 QModelIndex** AbstractDiagram::indexAt (const QPoint & point) const [virtual, inherited]

[reimplemented]

Definition at line 1098 of file KDChartAbstractDiagram.cpp.

References d.

```
1099 {
1100     return d->indexAt( point );
1101 }
```

### 9.57.3.27 QModelIndexList AbstractDiagram::indexesAt (const QPoint & *point*) const [inherited]

This method is added alongside with `indexAt` from QAIM, since in `kdchart` multiple indexes can be displayed at the same spot.

Definition at line 1103 of file `KDChartAbstractDiagram.cpp`.

References `d`.

```
1104 {
1105     return d->indexesAt( point );
1106 }
```

### 9.57.3.28 bool AbstractDiagram::isHidden (const QModelIndex & *index*) const [inherited]

Retrieve the hidden status for the given index.

This will fall back automatically to what was set at dataset or diagram level, if there are no datapoint specific settings.

#### Parameters:

*index* The datapoint to retrieve the hidden status for.

#### Returns:

The hidden status for the given index.

Definition at line 380 of file `KDChartAbstractDiagram.cpp`.

References `KDChart::AbstractDiagram::attributesModel()`, `KDChart::AttributesModel::data()`, and `KDChart::DataHiddenRole`.

```
381 {
382     return qVariantValue<bool>(
383         attributesModel()->data(
384             attributesModel()->mapFromSource(index),
385             DataHiddenRole ) );
386 }
```

### 9.57.3.29 bool AbstractDiagram::isHidden (int *column*) const [inherited]

Retrieve the hidden status for the given dataset.

This will fall back automatically to what was set at diagram level, if there are no dataset specific settings.

#### Parameters:

*column* The dataset to retrieve the hidden status for.

#### Returns:

The hidden status for the given dataset.

Definition at line 373 of file `KDChartAbstractDiagram.cpp`.

References `KDChart::AbstractDiagram::attributesModel()`, `KDChart::AbstractDiagram::columnToIndex()`, `KDChart::AttributesModel::data()`, and `KDChart::DataHiddenRole`.

```

374 {
375     return qVariantValue<bool>(
376         attributesModel()->data(
377             attributesModel()->mapFromSource(columnToIndex( column )),
378             DataHiddenRole ) );
379 }

```

### 9.57.3.30 bool AbstractDiagram::isHidden () const [inherited]

Retrieve the hidden status specified globally.

This will fall back automatically to the default settings ( = not hidden), if there are no specific settings.

#### Returns:

The global hidden status.

Definition at line 367 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DataHiddenRole, and KDChart::AttributesModel::modelData().

Referenced by KDChart::LineDiagram::valueForCellTesting().

```

368 {
369     return qVariantValue<bool>(
370         attributesModel()->modelData( DataHiddenRole ) );
371 }

```

### 9.57.3.31 bool AbstractDiagram::isIndexHidden (const QModelIndex & index) const [virtual, inherited]

[reimplemented]

Definition at line 956 of file KDChartAbstractDiagram.cpp.

```

957 { return true; }

```

### 9.57.3.32 QStringList AbstractDiagram::itemRowLabels () const [inherited]

The set of item row labels currently displayed, for use in Abscissa axes, etc.

#### Returns:

The set of item row labels currently displayed.

Definition at line 989 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::attributesModel-RootIndex(), KDChart::AttributesModel::rowCount(), KDChart::AbstractDiagram::unitPrefix(), and KDChart::AbstractDiagram::unitSuffix().

```

990 {
991     QStringList ret;

```

```

992     if( model() ){
993         //qDebug() << "AbstractDiagram::itemRowLabels(): " << attributesModel()->rowCount(attributesModelRootIndex());
994         const int rowCount = attributesModel()->rowCount(attributesModelRootIndex());
995         for( int i = 0; i < rowCount; ++i ){
996             //qDebug() << "item row label: " << attributesModel()->headerData( i, Qt::Vertical, Qt::DisplayRole );
997             ret << unitPrefix( i, Qt::Horizontal, true ) +
998                 attributesModel()->headerData( i, Qt::Vertical, Qt::DisplayRole ).toString() +
999                 unitSuffix( i, Qt::Horizontal, true );
1000         }
1001     }
1002     return ret;
1003 }

```

### 9.57.3.33 void KDChart::AbstractDiagram::layoutChanged ([AbstractDiagram \\*](#)) [signal, inherited]

Diagrams are supposed to emit this signal, when the layout of one of their element changes.

Layouts can change, for example, when axes are added or removed, or when the configuration was changed in a way that the axes or the diagram itself are displayed in a different geometry. Changes in the diagrams coordinate system also result in the [layoutChanged\(\)](#) signal being emitted.

Referenced by [KDChart::AbstractDiagram::setDatasetDimension\(\)](#), [KDChart::AbstractPieDiagram::setPieAttributes\(\)](#), [KDChart::AbstractPieDiagram::setThreeDPieAttributes\(\)](#), [KDChart::Plotter::setType\(\)](#), [KDChart::LineDiagram::setType\(\)](#), and [KDChart::BarDiagram::setType\(\)](#).

### 9.57.3.34 void KDChart::AbstractDiagram::modelsChanged () [signal, inherited]

This signal is emitted, when either the model or the [AttributesModel](#) is replaced.

Referenced by [KDChart::AbstractDiagram::setAttributesModel\(\)](#), and [KDChart::AbstractDiagram::setModel\(\)](#).

### 9.57.3.35 QModelIndex AbstractDiagram::moveCursor ([CursorAction cursorAction](#), [Qt::KeyboardModifiers modifiers](#)) [virtual, inherited]

[reimplemented]

Definition at line 947 of file [KDChartAbstractDiagram.cpp](#).

```

948 { return QModelIndex(); }

```

### 9.57.3.36 void TernaryLineDiagram::paint ([PaintContext \\* paintContext](#)) [virtual]

Draw the diagram contents to the rectangle and painter, that are passed in as part of the paint context.

#### Parameters:

*paintContext* All information needed for painting.

Reimplemented from [KDChart::AbstractTernaryDiagram](#).

Definition at line 86 of file [KDChartTernaryLineDiagram.cpp](#).



References KDChart::AbstractDiagram::brush(), KDChart::PaintContext::coordinatePlane(), d, KDChart::AbstractDiagram::datasetDimension(), KDChart::PaintContext::painter(), KDChart::AbstractDiagram::paintMarker(), KDChart::AbstractDiagram::pen(), KDChart::TernaryCoordinatePlane::translate(), and translate().

```

87 {
88     d->reverseMapper.clear();
89
90     d->paint( paintContext );
91     // sanity checks:
92     if ( model() == 0 ) return;
93
94     QPainter* p = paintContext->painter();
95     PainterSaver s( p );
96
97     TernaryCoordinatePlane* plane =
98         (TernaryCoordinatePlane*) paintContext->coordinatePlane();
99     Q_ASSERT( plane );
100
101     double x, y, z;
102
103     int columnCount = model()->columnCount( rootIndex() );
104     QPointF start;
105     for(int column=0; column<columnCount; column+=datasetDimension() )
106     {
107         int numRows = model()->rowCount( rootIndex() );
108         for( int row = 0; row < numRows; row++ )
109         {
110             // see if there is data otherwise skip
111             QModelIndex base = model()->index( row, column );
112             if( ! model()->data( base ).isNull() )
113             {
114                 p->setPen( pen( base ) );
115                 p->setBrush( brush( base ) );
116
117                 // retrieve data
118                 x = qMax( model()->data( model()->index( row, column ) ).toDouble(),
119                         0.0 );
120                 y = qMax( model()->data( model()->index( row, column+1 ) ).toDouble(),
121                         0.0 );
122                 z = qMax( model()->data( model()->index( row, column+2 ) ).toDouble(),
123                         0.0 );
124
125                 double total = x + y + z;
126                 if ( fabs( total ) > 3 * std::numeric_limits<double>::epsilon() ) {
127                     TernaryPoint tPunkt( x / total, y / total );
128                     QPointF diagramLocation = translate( tPunkt );
129                     QPointF widgetLocation = plane->translate( diagramLocation );
130
131                     if ( row > 0 ) {
132                         p->drawLine( start, widgetLocation );
133                     }
134                     paintMarker( p, model()->index( row, column ), widgetLocation );
135                     start = widgetLocation;
136                     // retrieve text and data value attributes
137                     // FIXME use data model DisplayRole text
138                     QString text = tr( "%1, %2, %3" )
139                                     .arg( x * 100, 0, 'f', 0 )
140                                     .arg( y * 100, 0, 'f', 0 )
141                                     .arg( z * 100, 0, 'f', 0 );
142                     d->paintDataValueText( p, base, widgetLocation, text );
143                 } else {
144                     // ignore and do not paint this point, garbage data
145                     qDebug() << "TernaryPointDiagram::paint: data point x/y/z:"
146                             << x << "/" << y << "/" << z << "ignored, unusable.";
147                 }
148             }
149         }
150     }

```

```

149     }
150 }
151 }

```

### 9.57.3.37 void AbstractDiagram::paintDataValueText (QPainter \*painter, const QModelIndex &index, const QPointF &pos, double value) [inherited]

Definition at line 468 of file KDChartAbstractDiagram.cpp.

References `d`, `KDChart::DataValueAttributes::dataLabel()`, `KDChart::AbstractDiagram::dataValueAttributes()`, `KDChart::DataValueAttributes::decimalDigits()`, `KDChart::DataValueAttributes::isVisible()`, `KDChartEnums::MeasureOrientationMinimum`, `KDChart::DataValueAttributes::position()`, `KDChart::DataValueAttributes::prefix()`, `KDChart::DataValueAttributes::suffix()`, and `KDChart::DataValueAttributes::textAttributes()`.

Referenced by `KDChart::RingDiagram::paint()`, `KDChart::PolarDiagram::paint()`, and `KDChart::AbstractDiagram::paintDataValueTexts()`.

```

472 {
473     // paint one data series
474     const DataValueAttributes a( dataValueAttributes(index) );
475     if ( !a.isVisible() ) return;
476
477     // handle decimal digits
478     int decimalDigits = a.decimalDigits();
479     int decimalPos = QString::number( value ).indexOf( QLatin1Char( '.' ) );
480     QString roundedValue;
481     if ( a.dataLabel().isNull() ) {
482         if ( decimalPos > 0 && value != 0 )
483             roundedValue = roundValues( value, decimalPos, decimalDigits );
484         else
485             roundedValue = QString::number( value );
486     } else
487         roundedValue = a.dataLabel();
488     // handle prefix and suffix
489     if ( !a.prefix().isNull() )
490         roundedValue.prepend( a.prefix() );
491
492     if ( !a.suffix().isNull() )
493         roundedValue.append( a.suffix() );
494
495     const TextAttributes ta( a.textAttributes() );
496     // FIXME draw the non-text bits, background, etc
497     if ( ta.isVisible() ) {
498
499         QPointF pt( pos );
500         /* for debugging:
501         PainterSaver painterSaver( painter );
502         painter->setPen( Qt::black );
503         painter->drawLine( pos - QPointF( 1,1), pos + QPointF( 1,1) );
504         painter->drawLine( pos - QPointF(-1,1), pos + QPointF(-1,1) );
505         */
506
507         QTextDocument doc;
508         if( Qt::mightBeRichText( roundedValue ) )
509             doc.setHtml( roundedValue );
510         else
511             doc.setPlainText( roundedValue );
512
513         const RelativePosition relPos( a.position( value >= 0.0 ) );
514         const Qt::Alignment alignBottomLeft = Qt::AlignBottom | Qt::AlignLeft;
515         const QFont calculatedFont( ta.calculatedFont( d->plane, KDChartEnums::MeasureOrientationMinimum
516         const QRectF boundRect( d->cachedFontMetrics( calculatedFont, painter->device() )->boundingRect

```

```

517
518     // To place correctly
519     pt.ry() -= boundRect.height();
520
521     //qDebug() << "calculatedFont's point size:" << calculatedFont.pointSizeF();
522     // adjust the text start point position, if alignment is not Bottom/Left
523     if( (relPos.alignment() & alignBottomLeft) != alignBottomLeft ){
524         if( relPos.alignment() & Qt::AlignRight )
525             pt.rx() -= boundRect.width();
526         else if( relPos.alignment() & Qt::AlignHCenter )
527             pt.rx() -= 0.5 * boundRect.width();
528
529         if( relPos.alignment() & Qt::AlignTop )
530             pt.ry() += boundRect.height();
531         else if( relPos.alignment() & Qt::AlignVCenter )
532             pt.ry() += 0.5 * boundRect.height();
533     }
534
535     // FIXME draw the non-text bits, background, etc
536
537     if ( a.showRepetitiveDataLabels() ||
538         pos.x() <= d->lastX ||
539         d->lastRoundedValue != roundedValue ) {
540         d->lastRoundedValue = roundedValue;
541         d->lastX = pos.x();
542         PainterSaver painterSaver( painter );
543
544         doc.setDefaultFont( calculatedFont );
545         QAbstractTextDocumentLayout::PaintContext context;
546         context.palette = palette();
547         context.palette.setColor(QPalette::Text, ta.pen().color() );
548
549         painter->translate( pt );
550         painter->rotate( ta.rotation() );
551
552         QAbstractTextDocumentLayout* layout = doc.documentLayout();
553         layout->draw( painter, context );
554     }
555 }
556 }

```

### 9.57.3.38 void AbstractDiagram::paintDataValueTexts (QPainter \* *painter*) [protected, virtual, inherited]

Definition at line 584 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::checkInvariants(), KDChart::AbstractDiagram::coordinatePlane(), KDChart::AbstractDiagram::datasetDimension(), KDChart::AbstractDiagram::paintDataValueText(), and KDChart::AbstractCoordinatePlane::translate().

```

585 {
586     if ( !checkInvariants() ) return;
587     const int rowCount = model()->rowCount( rootIndex() );
588     const int columnCount = model()->columnCount( rootIndex() );
589     for ( int i=datasetDimension()-1; i<columnCount; i += datasetDimension() ) {
590         for ( int j=0; j< rowCount; ++j ) {
591             const QModelIndex index = model()->index( j, i, rootIndex() );
592             double value = model()->data( index ).toDouble();
593             const QPointF pos = coordinatePlane()->translate( QPointF( j, value ) );
594             paintDataValueText( painter, index, pos, value );
595         }
596     }
597 }

```

### 9.57.3.39 void AbstractDiagram::paintMarker (QPainter \* *painter*, const QModelIndex & *index*, const QPointF & *pos*) [inherited]

Definition at line 600 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::brush(), KDChart::AbstractDiagram::checkInvariants(), d, KDChart::AbstractDiagram::dataValueAttributes(), KDChart::MarkerAttributes::isVisible(), KDChart::DataValueAttributes::isVisible(), KDChart::DataValueAttributes::markerAttributes(), KDChart::MarkerAttributes::markerColor(), KDChart::MarkerAttributes::markerSize(), KDChart::AbstractDiagram::paintMarker(), and KDChart::MarkerAttributes::pen().

```

603 {
604     if ( !checkInvariants() ) return;
605     DataValueAttributes a = dataValueAttributes(index);
606     if ( !a.isVisible() ) return;
607     const MarkerAttributes &ma = a.markerAttributes();
608     if ( !ma.isVisible() ) return;
609
610     PainterSaver painterSaver( painter );
611     QSizeF maSize( ma.markerSize() );
612     QBrush indexBrush( brush( index ) );
613     QPen indexPen( ma.pen() );
614     if ( ma.markerColor().isValid() )
615         indexBrush.setColor( ma.markerColor() );
616
617     paintMarker( painter, ma, indexBrush, indexPen, pos, maSize );
618
619     // workaround: BC cannot be changed, otherwise we would pass the
620     // index down to next-lower paintMarker function. So far, we
621     // basically save a circle of radius maSize at pos in the
622     // reverseMapper. This means that ^^^ this version of paintMarker
623     // needs to be called to reverse-map the marker.
624     d->reverseMapper.addCircle( index.row(), index.column(), pos, 2 * maSize );
625 }
```

### 9.57.3.40 void AbstractDiagram::paintMarker (QPainter \* *painter*, const MarkerAttributes & *markerAttributes*, const QBrush & *brush*, const QPen & *pen*, const QPointF & *point*, const QSizeF & *size*) [virtual, inherited]

Definition at line 627 of file KDChartAbstractDiagram.cpp.

References KDChart::MarkerAttributes::Marker1Pixel, KDChart::MarkerAttributes::Marker4Pixels, KDChart::MarkerAttributes::MarkerCircle, KDChart::MarkerAttributes::MarkerCross, KDChart::MarkerAttributes::MarkerDiamond, KDChart::MarkerAttributes::MarkerFastCross, KDChart::MarkerAttributes::MarkerRing, KDChart::MarkerAttributes::MarkerSquare, and KDChart::MarkerAttributes::markerStyle().

Referenced by KDChart::TernaryPointDiagram::paint(), paint(), KDChart::MarkerLayoutItem::paintIntoRect(), KDChart::AbstractDiagram::paintMarker(), and KDChart::AbstractDiagram::paintMarkers().

```

633 {
634     const QPen oldPen( painter->pen() );
635     // Pen is used to paint 4Pixels - 1 Pixel - Ring and FastCross types.
636     // make sure to use the brush color - see above in those cases.
637     const bool isFourPixels = (markerAttributes.markerStyle() == MarkerAttributes::Marker4Pixels);
638     if( isFourPixels || (markerAttributes.markerStyle() == MarkerAttributes::Marker1Pixel) ){
639         // for high-performance point charts with tiny point markers:
640         painter->setPen( QPen( brush.color().light() ) );
641         if( isFourPixels ){
642             const qreal x = pos.x();
```

```

643         const qreal y = pos.y();
644         painter->drawLine( QPointF(x-1.0,y-1.0),
645                           QPointF(x+1.0,y-1.0) );
646         painter->drawLine( QPointF(x-1.0,y),
647                           QPointF(x+1.0,y) );
648         painter->drawLine( QPointF(x-1.0,y+1.0),
649                           QPointF(x+1.0,y+1.0) );
650     }
651     painter->drawPoint( pos );
652 }else{
653     PainterSaver painterSaver( painter );
654     // we only a solid line surrounding the markers
655     QPen painterPen( pen );
656     painterPen.setStyle( Qt::SolidLine );
657     painter->setPen( painterPen );
658     painter->setBrush( brush );
659     painter->setRenderHint ( QPainter::Antialiasing );
660     painter->translate( pos );
661     switch ( markerAttributes.markerStyle() ) {
662     case MarkerAttributes::MarkerCircle:
663         painter->drawEllipse( QRectF( 0 - maSize.height()/2, 0 - maSize.width()/2,
664                                     maSize.height(), maSize.width() ) );
665         break;
666     case MarkerAttributes::MarkerSquare:
667     {
668         QRectF rect( 0 - maSize.width()/2, 0 - maSize.height()/2,
669                     maSize.width(), maSize.height() );
670         painter->drawRect( rect );
671         break;
672     }
673     case MarkerAttributes::MarkerDiamond:
674     {
675         QVector <QPointF > diamondPoints;
676         QPointF top, left, bottom, right;
677         top = QPointF( 0, 0 - maSize.height()/2 );
678         left = QPointF( 0 - maSize.width()/2, 0 );
679         bottom = QPointF( 0, maSize.height()/2 );
680         right = QPointF( maSize.width()/2, 0 );
681         diamondPoints << top << left << bottom << right;
682         painter->drawPolygon( diamondPoints );
683         break;
684     }
685     // both handled on top of the method:
686     case MarkerAttributes::Marker1Pixel:
687     case MarkerAttributes::Marker4Pixels:
688         break;
689     case MarkerAttributes::MarkerRing:
690     {
691         painter->setPen( QPen( brush.color() ) );
692         painter->setBrush( Qt::NoBrush );
693         painter->drawEllipse( QRectF( 0 - maSize.height()/2, 0 - maSize.width()/2,
694                                     maSize.height(), maSize.width() ) );
695         break;
696     }
697     case MarkerAttributes::MarkerCross:
698     {
699         QRectF rect( maSize.width()*-0.5, maSize.height()*-0.2,
700                     maSize.width(), maSize.height()*0.4 );
701         painter->drawRect( rect );
702         rect.setTopLeft( QPointF( maSize.width()*-0.2, maSize.height()*-0.5 ) );
703         rect.setSize( QSizeF( maSize.width()*0.4, maSize.height() ) );
704         painter->drawRect( rect );
705         break;
706     }
707     case MarkerAttributes::MarkerFastCross:
708     {
709         QPointF left, right, top, bottom;

```

```

710         left = QPointF( -maSize.width()/2, 0 );
711         right = QPointF( maSize.width()/2, 0 );
712         top = QPointF( 0, -maSize.height()/2 );
713         bottom= QPointF( 0, maSize.height()/2 );
714         painter->setPen( QPen( brush.color() ) );
715         painter->drawLine( left, right );
716         painter->drawLine( top, bottom );
717         break;
718     }
719     default:
720         Q_ASSERT_X ( false, "paintMarkers()",
721                     "Type item does not match a defined Marker Type." );
722     }
723 }
724 painter->setPen( oldPen );
725 }

```

**9.57.3.41 void AbstractDiagram::paintMarkers (QPainter \* *painter*)** [protected, virtual, inherited]

Definition at line 727 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::checkInvariants(), KDChart::AbstractDiagram::coordinatePlane(), KDChart::AbstractDiagram::datasetDimension(), KDChart::AbstractDiagram::paintMarker(), and KDChart::AbstractCoordinatePlane::translate().

```

728 {
729     if ( !checkInvariants() ) return;
730     const int rowCount = model()->rowCount(rootIndex());
731     const int columnCount = model()->columnCount(rootIndex());
732     for ( int i=datasetDimension()-1; i<columnCount; i += datasetDimension() ) {
733         for ( int j=0; j<rowCount; ++j ) {
734             const QModelIndex index = model()->index( j, i, rootIndex() );
735             double value = model()->data( index ).toDouble();
736             const QPointF pos = coordinatePlane()->translate( QPointF( j, value ) );
737             paintMarker( painter, index, pos );
738         }
739     }
740 }

```

**9.57.3.42 QPen AbstractDiagram::pen (const QModelIndex & *index*) const** [inherited]

Retrieve the pen to be used, for painting the datapoint at the given index in the model.

#### Parameters:

*index* The index of the datapoint in the model.

#### Returns:

The pen to use for painting.

Definition at line 788 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AttributesModel::data(), KDChart::AbstractDiagram::datasetDimension(), KDChart::DatasetPenRole, and KDChart::AbstractDiagram::pen().

```

789 {
790     if( datasetDimension() > 1 )
791         return pen( index.column() );
792     return qVariantValue<QPen>(
793         attributesModel()->data(
794             attributesModel()->mapFromSource( index ),
795             DatasetPenRole ) );
796 }

```

### 9.57.3.43 QPen AbstractDiagram::pen (int *dataset*) const [inherited]

Retrieve the pen to be used for the given dataset.

This will fall back automatically to what was set at model level, if there are no dataset specific settings.

#### Parameters:

*dataset* The dataset to retrieve the pen for.

#### Returns:

The pen to use for painting.

Definition at line 777 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::columnToIndex(), KDChart::AttributesModel::data(), KDChart::AbstractDiagram::datasetDimension(), and KDChart::DatasetPenRole.

```

778 {
779     if( datasetDimension() > 1 )
780         dataset /= datasetDimension();
781
782     return qVariantValue<QPen>(
783         attributesModel()->data(
784             attributesModel()->mapFromSource( columnToIndex( dataset ) ),
785             DatasetPenRole ) );
786 }

```

### 9.57.3.44 QPen AbstractDiagram::pen () const [inherited]

Retrieve the pen to be used for painting datapoints globally.

This will fall back automatically to the default settings, if there are no specific settings.

#### Returns:

The pen to use for painting.

Definition at line 771 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AttributesModel::data(), and KDChart::DatasetPenRole.

Referenced by KDChart::TernaryPointDiagram::paint(), paint(), and KDChart::AbstractDiagram::pen().

```

772 {
773     return qVariantValue<QPen>(
774         attributesModel()->data( DatasetPenRole ) );
775 }

```

**9.57.3.45 bool AbstractDiagram::percentMode () const** [inherited]

Definition at line 462 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::compare(), and KDChart::CartesianCoordinatePlane::getDataDimensionsList().

```
463 {
464     return d->percent;
465 }
```

**9.57.3.46 void KDChart::AbstractDiagram::propertiesChanged ()** [signal, inherited]

Emitted upon change of a property of the Diagram.

Referenced by KDChart::Plotter::resetLineAttributes(), KDChart::LineDiagram::resetLineAttributes(), KDChart::AbstractDiagram::setAllowOverlappingDataValueTexts(), KDChart::AbstractDiagram::setAntiAliasing(), KDChart::BarDiagram::setBarAttributes(), KDChart::AbstractDiagram::setBrush(), KDChart::AbstractDiagram::setDataValueAttributes(), KDChart::Plotter::setLineAttributes(), KDChart::LineDiagram::setLineAttributes(), KDChart::AbstractDiagram::setPen(), KDChart::AbstractDiagram::setPercentMode(), KDChart::BarDiagram::setThreeDBarAttributes(), KDChart::Plotter::setThreeDLineAttributes(), KDChart::LineDiagram::setThreeDLineAttributes(), KDChart::Plotter::setType(), KDChart::LineDiagram::setType(), KDChart::BarDiagram::setType(), KDChart::Plotter::setValueTrackerAttributes(), and KDChart::LineDiagram::setValueTrackerAttributes().

**9.57.3.47 void TernaryLineDiagram::resize (const QSizeF & area)** [virtual]

Called by the widget's sizeEvent.

Adjust all internal structures, that are calculated, depending on the size of the widget.

**Parameters:**

*area*

Implements [KDChart::AbstractTernaryDiagram](#).

Definition at line 81 of file KDChartTernaryLineDiagram.cpp.

```
82 {
83     Q_UNUSED( area );
84 }
```

**9.57.3.48 void AbstractDiagram::scrollTo (const QModelIndex & index, ScrollHint hint = EnsureVisible)** [virtual, inherited]

[reimplemented]

Definition at line 942 of file KDChartAbstractDiagram.cpp.

```
943 {}
```



### 9.57.3.49 void AbstractDiagram::setAllowOverlappingDataValueTexts (bool *allow*) [inherited]

Set whether data value labels are allowed to overlap.

#### Parameters:

*allow* True means that overlapping labels are allowed.

Definition at line 434 of file KDChartAbstractDiagram.cpp.

References [d](#), and [KDChart::AbstractDiagram::propertiesChanged\(\)](#).

```
435 {  
436     d->allowOverlappingDataValueTexts = allow;  
437     emit propertiesChanged();  
438 }
```

### 9.57.3.50 void AbstractDiagram::setAntiAliasing (bool *enabled*) [inherited]

Set whether anti-aliasing is to be used while rendering this diagram.

#### Parameters:

*enabled* True means that AA is enabled.

Definition at line 445 of file KDChartAbstractDiagram.cpp.

References [d](#), and [KDChart::AbstractDiagram::propertiesChanged\(\)](#).

```
446 {  
447     d->antiAliasing = enabled;  
448     emit propertiesChanged();  
449 }
```

### 9.57.3.51 void AbstractDiagram::setAttributesModel ([AttributesModel](#) \* *model*) [virtual, inherited]

Associate an [AttributesModel](#) with this diagram.

Note that the diagram does `_not_` take ownership of the [AttributesModel](#). This should thus only be used with [AttributesModels](#) that have been explicitly created by the user, and are owned by her. Setting an [AttributesModel](#) that is internal to another diagram is an error.

Correct:

```
AttributesModel *am = new AttributesModel( model, 0 );  
diagram1->setAttributesModel( am );  
diagram2->setAttributesModel( am );
```

Wrong:

```
diagram1->setAttributesModel( diagram2->attributesModel() );
```

**Parameters:**

*model* The [AttributesModel](#) to use for this diagram.

**See also:**

[AttributesModel](#), [usesExternalAttributesModel](#)

Reimplemented in [KDChart::AbstractCartesianDiagram](#).

Definition at line 255 of file `KDChartAbstractDiagram.cpp`.

References `d`, `KDChart::AbstractDiagram::modelsChanged()`, and `KDChart::AbstractDiagram::setDataBoundariesDirty()`.

Referenced by `KDChart::AbstractCartesianDiagram::setAttributesModel()`.

```

256 {
257     if( amodel->sourceModel() != model() ) {
258         qWarning("KDChart::AbstractDiagram::setAttributesModel() failed: "
259                 "Trying to set an attributesmodel which works on a different "
260                 "model than the diagram.");
261         return;
262     }
263     if( qobject_cast<PrivateAttributesModel*>(amodel) ) {
264         qWarning("KDChart::AbstractDiagram::setAttributesModel() failed: "
265                 "Trying to set an attributesmodel that is private to another diagram.");
266         return;
267     }
268     d->setAttributesModel(amodel);
269     scheduleDelayedItemsLayout();
270     setDataBoundariesDirty();
271     emit modelsChanged();
272 }
```

### 9.57.3.52 void AbstractDiagram::setAttributesModelRootIndex (const QModelIndex &) [protected, inherited]

Definition at line 293 of file `KDChartAbstractDiagram.cpp`.

References `d`, and `KDChart::AbstractDiagram::setDataBoundariesDirty()`.

Referenced by `KDChart::AbstractDiagram::setRootIndex()`.

```

294 {
295     d->attributesModelRootIndex=idx;
296     setDataBoundariesDirty();
297     scheduleDelayedItemsLayout();
298 }
```

### 9.57.3.53 void AbstractDiagram::setBrush (const QBrush & brush) [inherited]

Set the brush to be used, for painting all datasets in the model.

**Parameters:**

*brush* The brush to use.

Definition at line 806 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DatasetBrushRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setModelData().

```
807 {
808     attributesModel()->setModelData(
809         qVariantFromValue( brush ), DatasetBrushRole );
810     emit propertiesChanged();
811 }
```

### 9.57.3.54 void AbstractDiagram::setBrush (int *dataset*, const QBrush & *brush*) [inherited]

Set the brush to be used, for painting the given dataset.

#### Parameters:

*dataset* The dataset's column in the model.

*brush* The brush to use.

Definition at line 813 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DatasetBrushRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setHeaderData().

```
814 {
815     attributesModel()->setHeaderData(
816         column, Qt::Vertical,
817         qVariantFromValue( brush ),
818         DatasetBrushRole );
819     emit propertiesChanged();
820 }
```

### 9.57.3.55 void AbstractDiagram::setBrush (const QModelIndex & *index*, const QBrush & *brush*) [inherited]

Set the brush to be used, for painting the datapoint at the given index.

#### Parameters:

*index* The datapoint's index in the model.

*brush* The brush to use.

Definition at line 798 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DatasetBrushRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setData().

```
799 {
800     attributesModel()->setData(
801         attributesModel()->mapFromSource( index ),
802         qVariantFromValue( brush ), DatasetBrushRole );
803     emit propertiesChanged();
804 }
```

**9.57.3.56** `void AbstractDiagram::setCoordinatePlane (AbstractCoordinatePlane * plane)`  
`[virtual, inherited]`

Set the coordinate plane associated with the diagram.

This determines how coordinates in value space are mapped into pixel space. The chart takes ownership.

**Returns:**

The coordinate plane associated with the diagram.

Reimplemented in [KDChart::AbstractCartesianDiagram](#).

Definition at line 316 of file `KDChartAbstractDiagram.cpp`.

References d.

Referenced by `KDChart::AbstractCoordinatePlane::addDiagram()`, `KDChart::AbstractCartesianDiagram::setCoordinatePlane()`, and `KDChart::AbstractCoordinatePlane::takeDiagram()`.

```
317 {
318     d->plane = parent;
319 }
```

**9.57.3.57** `void AbstractDiagram::setDataBoundariesDirty () const` `[protected, inherited]`

Definition at line 234 of file `KDChartAbstractDiagram.cpp`.

References d.

Referenced by `KDChart::AbstractDiagram::dataChanged()`, `KDChart::Plotter::resize()`, `KDChart::LineDiagram::resize()`, `KDChart::BarDiagram::resize()`, `KDChart::AbstractDiagram::setAttributesModel()`, `KDChart::AbstractDiagram::setAttributesModelRootIndex()`, `KDChart::AbstractDiagram::setDatasetDimension()`, `KDChart::AbstractDiagram::setModel()`, `KDChart::BarDiagram::setThreeDBarAttributes()`, `KDChart::Plotter::setThreeDLineAttributes()`, `KDChart::LineDiagram::setThreeDLineAttributes()`, `KDChart::Plotter::setType()`, `KDChart::LineDiagram::setType()`, and `KDChart::BarDiagram::setType()`.

```
235 {
236     d->databoundariesDirty = true;
237 }
```

**9.57.3.58** `void AbstractDiagram::setDatasetDimension (int dimension)` `[inherited]`

Sets the dataset dimension of the diagram.

**See also:**

[datasetDimension](#).

**Parameters:**

*dimension*

Definition at line 1077 of file `KDChartAbstractDiagram.cpp`.

References `d`, `KDChart::AbstractDiagram::layoutChanged()`, and `KDChart::AbstractDiagram::setDataBoundariesDirty()`.

Referenced by `KDChart::Widget::setType()`, `TernaryLineDiagram()`, and `KDChart::TernaryPointDiagram::TernaryPointDiagram()`.

```
1078 {
1079     if ( d->datasetDimension == dimension ) return;
1080     d->datasetDimension = dimension;
1081     setDataBoundariesDirty();
1082     emit layoutChanged( this );
1083 }
```

### 9.57.3.59 void AbstractDiagram::setDataValueAttributes (const [DataValueAttributes](#) & *a*) [inherited]

Set the [DataValueAttributes](#) for all datapoints in the model.

#### Parameters:

*a* The attributes to set.

Definition at line 428 of file `KDChartAbstractDiagram.cpp`.

References `d`, `KDChart::DataValueLabelAttributesRole`, and `KDChart::AbstractDiagram::propertiesChanged()`.

```
429 {
430     d->attributesModel->setModelData( qVariantFromValue( a ), DataValueLabelAttributesRole );
431     emit propertiesChanged();
432 }
```

### 9.57.3.60 void AbstractDiagram::setDataValueAttributes (int *dataset*, const [DataValueAttributes](#) & *a*) [inherited]

Set the [DataValueAttributes](#) for the given dataset.

#### Parameters:

*dataset* The dataset to set the attributes for.

*a* The attributes to set.

Definition at line 400 of file `KDChartAbstractDiagram.cpp`.

References `d`, `KDChart::DataValueLabelAttributesRole`, and `KDChart::AbstractDiagram::propertiesChanged()`.

```
401 {
402     d->attributesModel->setHeaderData(
403         column, Qt::Vertical,
404         qVariantFromValue( a ), DataValueLabelAttributesRole );
405     emit propertiesChanged();
406 }
```

### 9.57.3.61 void AbstractDiagram::setDataValueAttributes (const QModelIndex & *index*, const [DataValueAttributes](#) & *a*) [inherited]

Set the [DataValueAttributes](#) for the given index.

#### Parameters:

- index* The datapoint to set the attributes for.
- a* The attributes to set.

Definition at line 389 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::DataValueLabelAttributesRole](#), and [KDChart::AbstractDiagram::propertiesChanged\(\)](#).

```

391 {
392     d->attributesModel->setData(
393         d->attributesModel->mapFromSource( index ),
394         QVariantFromValue( a ),
395         DataValueLabelAttributesRole );
396     emit propertiesChanged();
397 }
```

### 9.57.3.62 void AbstractDiagram::setHidden (bool *hidden*) [inherited]

Hide (or unhide, resp.

) all datapoints in the model.

#### Note:

Hidden data are still taken into account by the coordinate plane, so neither the grid nor your axes' ranges will change, when you hide data. For totally removing data from KD Chart's view you can use another approach: e.g. you could define a proxy model on top of your data model, and register the proxy model calling [setModel\(\)](#) instead of registering your real data model.

#### Parameters:

- hidden* The hidden status to set.

Definition at line 359 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::AbstractDiagram::dataHidden\(\)](#), and [KDChart::DataHiddenRole](#).

```

360 {
361     d->attributesModel->setModelData(
362         QVariantFromValue( hidden ),
363         DataHiddenRole );
364     emit dataHidden();
365 }
```

### 9.57.3.63 void AbstractDiagram::setHidden (int *column*, bool *hidden*) [inherited]

Hide (or unhide, resp.

) a dataset.

**Note:**

Hidden data are still taken into account by the coordinate plane, so neither the grid nor your axes' ranges will change, when you hide data. For totally removing data from KD Chart's view you can use another approach: e.g. you could define a proxy model on top of your data model, and register the proxy model calling [setModel\(\)](#) instead of registering your real data model.

**Parameters:**

***column*** The dataset to set the hidden status for.

***hidden*** The hidden status to set.

Definition at line 350 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::AbstractDiagram::dataHidden\(\)](#), and [KDChart::DataHiddenRole](#).

```
351 {  
352     d->attributesModel->setHeaderData(  
353         column, Qt::Vertical,  
354         QVariantFromValue( hidden ),  
355         DataHiddenRole );  
356     emit dataHidden();  
357 }
```

### 9.57.3.64 void AbstractDiagram::setHidden (const QModelIndex & *index*, bool *hidden*) [inherited]

Hide (or unhide, resp.

) a data cell.

**Note:**

Hidden data are still taken into account by the coordinate plane, so neither the grid nor your axes' ranges will change, when you hide data. For totally removing data from KD Chart's view you can use another approach: e.g. you could define a proxy model on top of your data model, and register the proxy model calling [setModel\(\)](#) instead of registering your real data model.

**Parameters:**

***index*** The datapoint to set the hidden status for.

***hidden*** The hidden status to set.

Definition at line 341 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::AbstractDiagram::dataHidden\(\)](#), and [KDChart::DataHiddenRole](#).

```
342 {  
343     d->attributesModel->setData(  
344         d->attributesModel->mapFromSource( index ),  
345         QVariantFromValue( hidden ),  
346         DataHiddenRole );  
347     emit dataHidden();  
348 }
```

### 9.57.3.65 void AbstractDiagram::setModel (QAbstractItemModel \* *model*) [virtual, inherited]

Associate a model with the diagram.

Reimplemented in [KDChart::AbstractCartesianDiagram](#).

Definition at line 239 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::AttributesModel::initFrom\(\)](#), [KDChart::AbstractDiagram::modelsChanged\(\)](#), and [KDChart::AbstractDiagram::setDataBoundariesDirty\(\)](#).

Referenced by [KDChart::AbstractCartesianDiagram::setModel\(\)](#), and [KDChart::Widget::setType\(\)](#).

```

240 {
241     QAbstractItemView::setModel( newModel );
242     AttributesModel* amodel = new PrivateAttributesModel( newModel, this );
243     amodel->initFrom( d->attributesModel );
244     d->setAttributesModel(amodel);
245     scheduleDelayedItemsLayout();
246     setDataBoundariesDirty();
247     emit modelsChanged();
248 }
```

### 9.57.3.66 void AbstractDiagram::setPen (const QPen & *pen*) [inherited]

Set the pen to be used, for painting all datasets in the model.

#### Parameters:

*pen* The pen to use.

Definition at line 753 of file KDChartAbstractDiagram.cpp.

References [KDChart::AbstractDiagram::attributesModel\(\)](#), [KDChart::DatasetPenRole](#), [KDChart::AbstractDiagram::propertiesChanged\(\)](#), and [KDChart::AttributesModel::setModelData\(\)](#).

```

754 {
755     attributesModel()->setModelData(
756         qVariantFromValue( pen ), DatasetPenRole );
757     emit propertiesChanged();
758 }
```

### 9.57.3.67 void AbstractDiagram::setPen (int *dataset*, const QPen & *pen*) [inherited]

Set the pen to be used, for painting the given dataset.

#### Parameters:

*dataset* The dataset's row in the model.

*pen* The pen to use.

Definition at line 760 of file KDChartAbstractDiagram.cpp.

References [KDChart::AbstractDiagram::attributesModel\(\)](#), [KDChart::AbstractDiagram::datasetDimension\(\)](#), [KDChart::DatasetPenRole](#), [KDChart::AbstractDiagram::propertiesChanged\(\)](#), and [KDChart::AttributesModel::setHeaderData\(\)](#).



```

761 {
762     if( datasetDimension() > 1 )
763         column *= datasetDimension();
764     attributesModel()->setHeaderData(
765         column, Qt::Vertical,
766         qVariantFromValue( pen ),
767         DatasetPenRole );
768     emit propertiesChanged();
769 }

```

### 9.57.3.68 void AbstractDiagram::setPen (const QModelIndex & *index*, const QPen & *pen*) [inherited]

Set the pen to be used, for painting the datapoint at the given index.

#### Parameters:

***index*** The datapoint's index in the model.

***pen*** The pen to use.

Definition at line 743 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::datasetDimension(), KDChart::DatasetPenRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setData().

```

744 {
745     if( datasetDimension() > 1 )
746         return setPen( index.column(), pen );
747     attributesModel()->setData(
748         attributesModel()->mapFromSource( index ),
749         qVariantFromValue( pen ), DatasetPenRole );
750     emit propertiesChanged();
751 }

```

### 9.57.3.69 void AbstractDiagram::setPercentMode (bool *percent*) [inherited]

Definition at line 456 of file KDChartAbstractDiagram.cpp.

References d, and KDChart::AbstractDiagram::propertiesChanged().

Referenced by KDChart::LineDiagram::setType(), and KDChart::BarDiagram::setType().

```

457 {
458     d->percent = percent;
459     emit propertiesChanged();
460 }

```

### 9.57.3.70 void AbstractDiagram::setRootIndex (const QModelIndex & *idx*) [virtual, inherited]

Set the root index in the model, where the diagram starts referencing data for display.

[reimplemented]

Reimplemented in [KDChart::AbstractCartesianDiagram](#).

Definition at line 286 of file KDChartAbstractDiagram.cpp.

References `d`, and `KDChart::AbstractDiagram::setAttributesModelRootIndex()`.

Referenced by `KDChart::AbstractCartesianDiagram::setRootIndex()`.

```
287 {
288     QAbstractItemView::setRootIndex(idx);
289     setAttributesModelRootIndex( d->attributesModel->mapFromSource(idx) );
290 }
```

#### 9.57.3.71 void AbstractDiagram::setSelection (const QRect & *rect*, QItemSelectionModel::SelectionFlags *command*) [virtual, inherited]

[reimplemented]

Definition at line 959 of file KDChartAbstractDiagram.cpp.

References `d`.

```
960 {
961     const QModelIndexList indexes = d->indexesIn( rect );
962     QItemSelection selection;
963     KDAB_FOREACH( const QModelIndex& index, indexes )
964     {
965         selection.append( QItemSelectionRange( index ) );
966     }
967     selectionModel()->select( selection, command );
968 }
```

#### 9.57.3.72 void AbstractDiagram::setUnitPrefix (const QString & *prefix*, Qt::Orientation *orientation*) [inherited]

Sets the unit prefix for all values.

##### Parameters:

*prefix* the prefix to be set

*orientation* the orientation of the axis to set

Definition at line 864 of file KDChartAbstractDiagram.cpp.

References `d`.

```
865 {
866     d->unitPrefix[ orientation ] = prefix;
867 }
```

#### 9.57.3.73 void AbstractDiagram::setUnitPrefix (const QString & *prefix*, int *column*, Qt::Orientation *orientation*) [inherited]

Sets the unit prefix for one value.

##### Parameters:

*prefix* the prefix to be set

*column* the value using that prefix

*orientation* the orientation of the axis to set

Definition at line 854 of file KDChartAbstractDiagram.cpp.

References d.

```
855 {  
856     d->unitPrefixMap[ column ][ orientation ]= prefix;  
857 }
```

#### 9.57.3.74 void AbstractDiagram::setUnitSuffix (const QString & *suffix*, Qt::Orientation *orientation*) [inherited]

Sets the unit suffix for all values.

##### Parameters:

*suffix* the suffix to be set

*orientation* the orientation of the axis to set

Definition at line 885 of file KDChartAbstractDiagram.cpp.

References d.

```
886 {  
887     d->unitSuffix[ orientation ] = suffix;  
888 }
```

#### 9.57.3.75 void AbstractDiagram::setUnitSuffix (const QString & *suffix*, int *column*, Qt::Orientation *orientation*) [inherited]

Sets the unit suffix for one value.

##### Parameters:

*suffix* the suffix to be set

*column* the value using that suffix

*orientation* the orientation of the axis to set

Definition at line 875 of file KDChartAbstractDiagram.cpp.

References d.

```
876 {  
877     d->unitSuffixMap[ column ][ orientation ]= suffix;  
878 }
```

### 9.57.3.76 QString AbstractDiagram::unitPrefix (Qt::Orientation *orientation*) const [inherited]

Returns the global unit prefix.

#### Parameters:

*orientation* the orientation of the axis

#### Returns:

the unit prefix

Definition at line 908 of file KDChartAbstractDiagram.cpp.

References d.

```
909 {
910     return d->unitPrefix[ orientation ];
911 }
```

### 9.57.3.77 QString AbstractDiagram::unitPrefix (int *column*, Qt::Orientation *orientation*, bool *fallback* = false) const [inherited]

Returns the unit prefix for a special value.

#### Parameters:

*column* the value which's prefix is requested

*orientation* the orientation of the axis

*fallback* if true, the global prefix is return when no specific one is set for that value

#### Returns:

the unit prefix

Definition at line 897 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::itemRowLabels(), and KDChart::CartesianAxis::paintCtx().

```
898 {
899     if( !fallback || d->unitPrefixMap[ column ].contains( orientation ) )
900         return d->unitPrefixMap[ column ][ orientation ];
901     return d->unitPrefix[ orientation ];
902 }
```

### 9.57.3.78 QString AbstractDiagram::unitSuffix (Qt::Orientation *orientation*) const [inherited]

Returns the global unit suffix.

#### Parameters:

*orientation* the orientation of the axis

**Returns:**

the unit suffix

Definition at line 931 of file KDChartAbstractDiagram.cpp.

References d.

```
932 {
933     return d->unitSuffix[ orientation ];
934 }
```

### 9.57.3.79 QString AbstractDiagram::unitSuffix (int *column*, Qt::Orientation *orientation*, bool *fallback* = false) const [inherited]

Returns the unit suffix for a special value.

**Parameters:**

*column* the value which's suffix is requested

*orientation* the orientation of the axis

*fallback* if true, the global suffix is return when no specific one is set for that value

**Returns:**

the unit suffix

Definition at line 920 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::itemRowLabels(), and KDChart::CartesianAxis::paintCtx().

```
921 {
922     if( !fallback || d->unitSuffixMap[ column ].contains( orientation ) )
923         return d->unitSuffixMap[ column ][ orientation ];
924     return d->unitSuffix[ orientation ];
925 }
```

### 9.57.3.80 void AbstractDiagram::update () const [inherited]

Definition at line 1091 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::doItemsLayout().

```
1092 {
1093     //QDebug("KDChart::AbstractDiagram::update() called");
1094     if( d->plane )
1095         d->plane->update();
1096 }
```

**9.57.3.81 void KDChart::AbstractDiagram::useDefaultColors ()** [inherited]

Set the palette to be used, for painting datasets to the default palette.

**See also:**

[KDChart::Palette](#). FIXME: fold into one usePalette ([KDChart::Palette&](#)) method

Definition at line 974 of file KDChartAbstractDiagram.cpp.

References [d](#), and [KDChart::AttributesModel::PaletteTypeDefault](#).

```
975 {  
976     d->attributesModel->setPaletteType( AttributesModel::PaletteTypeDefault );  
977 }
```

**9.57.3.82 void KDChart::AbstractDiagram::useRainbowColors ()** [inherited]

Set the palette to be used, for painting datasets to the rainbow palette.

**See also:**

[KDChart::Palette](#).

Definition at line 984 of file KDChartAbstractDiagram.cpp.

References [d](#), and [KDChart::AttributesModel::PaletteTypeRainbow](#).

```
985 {  
986     d->attributesModel->setPaletteType( AttributesModel::PaletteTypeRainbow );  
987 }
```

**9.57.3.83 bool AbstractDiagram::usesExternalAttributesModel () const** [virtual, inherited]

Returns whether the diagram is using its own built-in attributes model or an attributes model that was set via [setAttributesModel](#).

**See also:**

[setAttributesModel](#)

Definition at line 274 of file KDChartAbstractDiagram.cpp.

References [d](#).

```
275 {  
276     return d->usesExternalAttributesModel();  
277 }
```

**9.57.3.84 void KDChart::AbstractDiagram::useSubduedColors ()** [inherited]

Set the palette to be used, for painting datasets to the subdued palette.

**See also:**

[KDChart::Palette](#).

Definition at line 979 of file KDChartAbstractDiagram.cpp.

References [d](#), and [KDChart::AttributesModel::PaletteTypeSubdued](#).

```
980 {  
981     d->attributesModel->setPaletteType( AttributesModel::PaletteTypeSubdued );  
982 }
```

**9.57.3.85 double AbstractDiagram::valueForCell (int row, int column) const** [protected, inherited]

Helper method, retrieving the data value (DisplayRole) for a given row and column.

**Parameters:**

*row* The row to query.

*column* The column to query.

**Returns:**

The value of the display role at the given row and column as a double.

**Deprecated**

Definition at line 1085 of file KDChartAbstractDiagram.cpp.

References [KDChart::AbstractDiagram::attributesModelRootIndex\(\)](#), and [d](#).

```
1086 {  
1087     return d->attributesModel->data(  
1088         d->attributesModel->index( row, column, attributesModelRootIndex() ) ).toDouble();  
1089 }
```

**9.57.3.86 int AbstractDiagram::verticalOffset () const** [virtual, inherited]

[reimplemented]

Definition at line 953 of file KDChartAbstractDiagram.cpp.

```
954 { return 0; }
```

**9.57.3.87** **QRect AbstractDiagram::visualRect (const QModelIndex & *index*) const** [virtual, inherited]

[reimplemented]

Definition at line 937 of file KDChartAbstractDiagram.cpp.

```
938 {  
939     return QRect();  
940 }
```

**9.57.3.88** **QRegion AbstractDiagram::visualRegionForSelection (const QItemSelection & *selection*) const** [virtual, inherited]

[reimplemented]

Definition at line 970 of file KDChartAbstractDiagram.cpp.

```
971 { return QRegion(); }
```

The documentation for this class was generated from the following files:

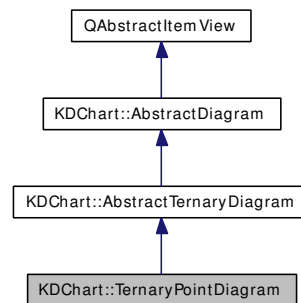
- [KDChartTernaryLineDiagram.h](#)
- [KDChartTernaryLineDiagram.cpp](#)



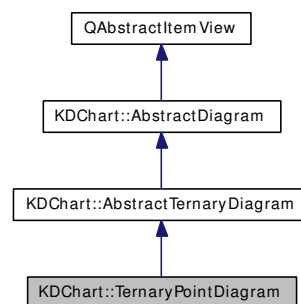
## 9.58 KDChart::TernaryPointDiagram Class Reference

```
#include <KDChartTernaryPointDiagram.h>
```

Inheritance diagram for KDChart::TernaryPointDiagram:



Collaboration diagram for KDChart::TernaryPointDiagram:



### 9.58.1 Detailed Description

A [TernaryPointDiagram](#) is a point diagram within a ternary coordinate plane.

Definition at line 41 of file KDChartTernaryPointDiagram.h.

### Signals

- void [dataHidden](#) ()  
*This signal is emitted, when the hidden status of at least one data cell was (un)set.*
- void [layoutChanged](#) ([AbstractDiagram](#) \*)  
*Diagrams are supposed to emit this signal, when the layout of one of their element changes.*
- void [modelsChanged](#) ()  
*This signal is emitted, when either the model or the [AttributesModel](#) is replaced.*
- void [propertiesChanged](#) ()  
*Emitted upon change of a property of the Diagram.*

## Public Member Functions

- bool [allowOverlappingDataValueTexts](#) () const  
*Returns:*  
*Whether data value labels are allowed to overlap.*
- bool [antiAliasing](#) () const  
*Returns:*  
*Whether anti-aliasing is to be used for rendering this diagram.*
- virtual [AttributesModel](#) \* [attributesModel](#) () const  
*Returns the [AttributesModel](#), that is used by this diagram.*
- [QBrush](#) [brush](#) (const [QModelIndex](#) &index) const  
*Retrieve the brush to be used, for painting the datapoint at the given index in the model.*
- [QBrush](#) [brush](#) (int dataset) const  
*Retrieve the brush to be used for the given dataset.*
- [QBrush](#) [brush](#) () const  
*Retrieve the brush to be used for painting datapoints globally.*
- bool [compare](#) (const [AbstractDiagram](#) \*other) const  
*Returns true if both diagrams have the same settings.*
- [AbstractCoordinatePlane](#) \* [coordinatePlane](#) () const  
*The coordinate plane associated with the diagram.*
- const [QPair](#)< [QPointF](#), [QPointF](#) > [dataBoundaries](#) () const  
*Return the bottom left and top right data point, that the diagram will display (unless the grid adjusts these values).*
- virtual void [dataChanged](#) (const [QModelIndex](#) &topLeft, const [QModelIndex](#) &bottomRight)  
*[reimplemented]*
- [QList](#)< [QBrush](#) > [datasetBrushes](#) () const  
*The set of dataset brushes currently used, for use in legends, etc.*
- int [datasetDimension](#) () const  
*The dataset dimension of a diagram determines, how many value dimensions it expects each datapoint to have.*
- [QStringList](#) [datasetLabels](#) () const  
*The set of dataset labels currently displayed, for use in legends, etc.*
- [QList](#)< [MarkerAttributes](#) > [datasetMarkers](#) () const  
*The set of dataset markers currently used, for use in legends, etc.*
- [QList](#)< [QPen](#) > [datasetPens](#) () const  
*The set of dataset pens currently used, for use in legends, etc.*

- [DataValueAttributes dataValueAttributes](#) (const QModelIndex &index) const  
*Retrieve the [DataValueAttributes](#) for the given index.*
- [DataValueAttributes dataValueAttributes](#) (int column) const  
*Retrieve the [DataValueAttributes](#) for the given dataset.*
- [DataValueAttributes dataValueAttributes](#) () const  
*Retrieve the [DataValueAttributes](#) specified globally.*
- virtual void [doItemsLayout](#) ()  
*[reimplemented]*
- virtual int [horizontalOffset](#) () const  
*[reimplemented]*
- virtual QModelIndex [indexAt](#) (const QPoint &point) const  
*[reimplemented]*
- QModelIndexList [indexesAt](#) (const QPoint &point) const  
*This method is added alongside with [indexAt](#) from QAIM, since in kdchart multiple indexes can be displayed at the same spot.*
- bool [isHidden](#) (const QModelIndex &index) const  
*Retrieve the hidden status for the given index.*
- bool [isHidden](#) (int column) const  
*Retrieve the hidden status for the given dataset.*
- bool [isHidden](#) () const  
*Retrieve the hidden status specified globally.*
- virtual bool [isIndexHidden](#) (const QModelIndex &index) const  
*[reimplemented]*
- QStringList [itemRowLabels](#) () const  
*The set of item row labels currently displayed, for use in Abscissa axes, etc.*
- virtual QModelIndex [moveCursor](#) (CursorAction cursorAction, Qt::KeyboardModifiers modifiers)  
*[reimplemented]*
- virtual void [paint](#) (PaintContext \*paintContext)  
*Draw the diagram contents to the rectangle and painter, that are passed in as part of the paint context.*
- void [paintDataValueText](#) (QPainter \*painter, const QModelIndex &index, const QPointF &pos, double value)
- void [paintMarker](#) (QPainter \*painter, const QModelIndex &index, const QPointF &pos)
- virtual void [paintMarker](#) (QPainter \*painter, const [MarkerAttributes](#) &markerAttributes, const QBrush &brush, const QPen &, const QPointF &point, const QSizeF &size)
- QPen [pen](#) (const QModelIndex &index) const

*Retrieve the pen to be used, for painting the datapoint at the given index in the model.*

- QPen [pen](#) (int dataset) const  
*Retrieve the pen to be used for the given dataset.*
- QPen [pen](#) () const  
*Retrieve the pen to be used for painting datapoints globally.*
- bool [percentMode](#) () const
- virtual void [resize](#) (const QSizeF &area)  
*Called by the widget's sizeEvent.*
- virtual void [scrollTo](#) (const QModelIndex &index, ScrollHint hint=EnsureVisible)  
*[reimplemented]*
- void [setAllowOverlappingDataValueTexts](#) (bool allow)  
*Set whether data value labels are allowed to overlap.*
- void [setAntiAliasing](#) (bool enabled)  
*Set whether anti-aliasing is to be used while rendering this diagram.*
- virtual void [setAttributesModel](#) ([AttributesModel](#) \*model)  
*Associate an [AttributesModel](#) with this diagram.*
- void [setBrush](#) (const QBrush &brush)  
*Set the brush to be used, for painting all datasets in the model.*
- void [setBrush](#) (int dataset, const QBrush &brush)  
*Set the brush to be used, for painting the given dataset.*
- void [setBrush](#) (const QModelIndex &index, const QBrush &brush)  
*Set the brush to be used, for painting the datapoint at the given index.*
- virtual void [setCoordinatePlane](#) ([AbstractCoordinatePlane](#) \*plane)  
*Set the coordinate plane associated with the diagram.*
- void [setDatasetDimension](#) (int dimension)  
*Sets the dataset dimension of the diagram.*
- void [setDataValueAttributes](#) (const [DataValueAttributes](#) &a)  
*Set the [DataValueAttributes](#) for all datapoints in the model.*
- void [setDataValueAttributes](#) (int dataset, const [DataValueAttributes](#) &a)  
*Set the [DataValueAttributes](#) for the given dataset.*
- void [setDataValueAttributes](#) (const QModelIndex &index, const [DataValueAttributes](#) &a)  
*Set the [DataValueAttributes](#) for the given index.*
- void [setHidden](#) (bool hidden)  
*Hide (or unhide, resp.*

- void [setHidden](#) (int column, bool hidden)  
*Hide (or unhide, resp).*
- void [setHidden](#) (const QModelIndex &index, bool hidden)  
*Hide (or unhide, resp).*
- virtual void [setModel](#) (QAbstractItemModel \*model)  
*Associate a model with the diagram.*
- void [setPen](#) (const QPen &pen)  
*Set the pen to be used, for painting all datasets in the model.*
- void [setPen](#) (int dataset, const QPen &pen)  
*Set the pen to be used, for painting the given dataset.*
- void [setPen](#) (const QModelIndex &index, const QPen &pen)  
*Set the pen to be used, for painting the datapoint at the given index.*
- void [setPercentMode](#) (bool percent)
- virtual void [setRootIndex](#) (const QModelIndex &idx)  
*Set the root index in the model, where the diagram starts referencing data for display.*
- virtual void [setSelection](#) (const QRect &rect, QItemSelectionModel::SelectionFlags command)  
*[reimplemented]*
- void [setUnitPrefix](#) (const QString &prefix, Qt::Orientation orientation)  
*Sets the unit prefix for all values.*
- void [setUnitPrefix](#) (const QString &prefix, int column, Qt::Orientation orientation)  
*Sets the unit prefix for one value.*
- void [setUnitSuffix](#) (const QString &suffix, Qt::Orientation orientation)  
*Sets the unit suffix for all values.*
- void [setUnitSuffix](#) (const QString &suffix, int column, Qt::Orientation orientation)  
*Sets the unit suffix for one value.*
- [TernaryPointDiagram](#) (QWidget \*parent=0, [TernaryCoordinatePlane](#) \*plane=0)
- QString [unitPrefix](#) (Qt::Orientation orientation) const  
*Returns the global unit prefix.*
- QString [unitPrefix](#) (int column, Qt::Orientation orientation, bool fallback=false) const  
*Returns the unit prefix for a special value.*
- QString [unitSuffix](#) (Qt::Orientation orientation) const  
*Returns the global unit suffix.*
- QString [unitSuffix](#) (int column, Qt::Orientation orientation, bool fallback=false) const  
*Returns the unit suffix for a special value.*

- void [update](#) () const
- void [useDefaultColors](#) ()  
*Set the palette to be used, for painting datasets to the default palette.*
- void [useRainbowColors](#) ()  
*Set the palette to be used, for painting datasets to the rainbow palette.*
- virtual bool [usesExternalAttributesModel](#) () const  
*Returns whether the diagram is using its own built-in attributes model or an attributes model that was set via [setAttributesModel](#).*
- void [useSubduedColors](#) ()  
*Set the palette to be used, for painting datasets to the subdued palette.*
- virtual int [verticalOffset](#) () const  
*[reimplemented]*
- virtual QRect [visualRect](#) (const QModelIndex &index) const  
*[reimplemented]*
- virtual QRegion [visualRegionForSelection](#) (const QItemSelection &selection) const  
*[reimplemented]*
- virtual [~TernaryPointDiagram](#) ()

## Protected Member Functions

- QModelIndex [attributesModelRootIndex](#) () const
- virtual const QPair< QPointF, QPointF > [calculateDataBoundaries](#) () const
- virtual bool [checkInvariants](#) (bool justReturnTheStatus=false) const
- QModelIndex [columnToIndex](#) (int column) const
- virtual void [paintDataValueTexts](#) (QPainter \*painter)
- virtual void [paintMarkers](#) (QPainter \*painter)
- void [setAttributesModelRootIndex](#) (const QModelIndex &)
- void [setDataBoundariesDirty](#) () const
- double [valueForCell](#) (int row, int column) const  
*Helper method, retrieving the data value (DisplayRole) for a given row and column.*

## 9.58.2 Constructor & Destructor Documentation

### 9.58.2.1 TernaryPointDiagram::TernaryPointDiagram (QWidget \*parent = 0, TernaryCoordinatePlane \*plane = 0) [explicit]

Definition at line 50 of file `KDChartTernaryPointDiagram.cpp`.

References `KDChart::AbstractDiagram::setDatasetDimension()`.

```
52 : AbstractTernaryDiagram( new Private(), parent, plane )
53 {
54     init();
55     setDatasetDimension( 3 ); // the third column is implicit
56 }
```

#### 9.58.2.2 TernaryPointDiagram::~TernaryPointDiagram () [virtual]

Definition at line 58 of file KDChartTernaryPointDiagram.cpp.

```
59 {
60 }
```

### 9.58.3 Member Function Documentation

#### 9.58.3.1 bool AbstractDiagram::allowOverlappingDataValueTexts () const [inherited]

##### Returns:

Whether data value labels are allowed to overlap.

Definition at line 440 of file KDChartAbstractDiagram.cpp.

References [d](#).

Referenced by [KDChart::AbstractDiagram::compare\(\)](#).

```
441 {
442     return d->allowOverlappingDataValueTexts;
443 }
```

#### 9.58.3.2 bool AbstractDiagram::antiAliasing () const [inherited]

##### Returns:

Whether anti-aliasing is to be used for rendering this diagram.

Definition at line 451 of file KDChartAbstractDiagram.cpp.

References [d](#).

Referenced by [KDChart::AbstractDiagram::compare\(\)](#).

```
452 {
453     return d->antiAliasing;
454 }
```

#### 9.58.3.3 [AttributesModel](#) \* AbstractDiagram::attributesModel () const [virtual, inherited]

Returns the [AttributesModel](#), that is used by this diagram.

By default each diagram owns its own [AttributesModel](#), which should never be deleted. Only if a user-supplied [AttributesModel](#) has been set does the pointer returned here not belong to the diagram.

**Returns:**

The [AttributesModel](#) associated with the diagram.

**See also:**

[setAttributesModel](#)

Definition at line 280 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::brush(), KDChart::AbstractDiagram::compare(), KDChart::AbstractDiagram::datasetBrushes(), KDChart::AbstractDiagram::datasetLabels(), KDChart::AbstractDiagram::datasetMarkers(), KDChart::AbstractDiagram::datasetPens(), KDChart::AbstractDiagram::dataValueAttributes(), KDChart::AbstractDiagram::isHidden(), KDChart::AbstractDiagram::itemRowLabels(), KDChart::RingDiagram::paint(), KDChart::PolarDiagram::paint(), KDChart::AbstractDiagram::pen(), KDChart::AbstractCartesianDiagram::setAttributesModel(), KDChart::BarDiagram::setBarAttributes(), KDChart::AbstractDiagram::setBrush(), KDChart::AbstractCartesianDiagram::setModel(), KDChart::AbstractDiagram::setPen(), and KDChart::TernaryLineDiagram::TernaryLineDiagram().

```
281 {
282     return d->attributesModel;
283 }
```

#### 9.58.3.4 QModelIndex AbstractDiagram::attributesModelRootIndex () const [protected, inherited]

returns a QModelIndex pointing into the [AttributesModel](#) that corresponds to the root index of the diagram.

Definition at line 302 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::datasetBrushes(), KDChart::AbstractDiagram::datasetLabels(), KDChart::AbstractDiagram::datasetMarkers(), KDChart::AbstractDiagram::datasetPens(), KDChart::AbstractDiagram::itemRowLabels(), KDChart::Plotter::numberOfAbscissaSegments(), KDChart::LineDiagram::numberOfAbscissaSegments(), KDChart::BarDiagram::numberOfAbscissaSegments(), KDChart::Plotter::numberOfOrdinateSegments(), KDChart::LineDiagram::numberOfOrdinateSegments(), KDChart::BarDiagram::numberOfOrdinateSegments(), KDChart::AbstractDiagram::valueForCell(), and KDChart::LineDiagram::valueForCellTesting().

```
303 {
304     if ( !d->attributesModelRootIndex.isValid() )
305         d->attributesModelRootIndex = d->attributesModel->mapFromSource( rootIndex() );
306     return d->attributesModelRootIndex;
307 }
```

#### 9.58.3.5 QBrush AbstractDiagram::brush (const QModelIndex & index) const [inherited]

Retrieve the brush to be used, for painting the datapoint at the given index in the model.

**Parameters:**

*index* The index of the datapoint in the model.



**Returns:**

The brush to use for painting.

Definition at line 838 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::brush(), KDChart::AttributesModel::data(), KDChart::DatasetBrushRole, and KDChart::AbstractDiagram::datasetDimension().

```

839 {
840     if( datasetDimension() > 1 )
841         return brush( index.column() );
842     return qVariantValue<QBrush>(
843         attributesModel()->data(
844             attributesModel()->mapFromSource( index ),
845             DatasetBrushRole ) );
846 }
```

**9.58.3.6 QBrush AbstractDiagram::brush (int *dataset*) const** [inherited]

Retrieve the brush to be used for the given dataset.

This will fall back automatically to what was set at model level, if there are no dataset specific settings.

**Parameters:**

*dataset* The dataset to retrieve the brush for.

**Returns:**

The brush to use for painting.

Definition at line 828 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::columnToIndex(), KDChart::AttributesModel::data(), KDChart::DatasetBrushRole, and KDChart::AbstractDiagram::datasetDimension().

```

829 {
830     if( datasetDimension() > 1 )
831         dataset /= datasetDimension();
832     return qVariantValue<QBrush>(
833         attributesModel()->data(
834             attributesModel()->mapFromSource( columnToIndex( dataset ) ),
835             DatasetBrushRole ) );
836 }
```

**9.58.3.7 QBrush AbstractDiagram::brush () const** [inherited]

Retrieve the brush to be used for painting datapoints globally.

This will fall back automatically to the default settings, if there are no specific settings.

**Returns:**

The brush to use for painting.

Definition at line 822 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AttributesModel::data(), and KDChart::DatasetBrushRole.

Referenced by KDChart::AbstractDiagram::brush(), paint(), KDChart::TernaryLineDiagram::paint(), KDChart::RingDiagram::paint(), KDChart::PolarDiagram::paint(), and KDChart::AbstractDiagram::paintMarker().

```
823 {
824     return qVariantValue<QBrush>(
825         attributesModel()->data( DatasetBrushRole ) );
826 }
```

### 9.58.3.8 const QPair< QPointF, QPointF > TernaryPointDiagram::calculateDataBoundaries () const [protected, virtual]

Implements [KDChart::AbstractTernaryDiagram](#).

Definition at line 134 of file KDChartTernaryPointDiagram.cpp.

References TriangleBottomLeft, TriangleBottomRight, and TriangleHeight.

```
135 {
136     // this is a constant, because we defined it to be one:
137     static QPair<QPointF, QPointF> Boundaries(
138         TriangleBottomLeft,
139         QPointF( TriangleBottomRight.x(), TriangleHeight ) );
140     return Boundaries;
141 }
```

### 9.58.3.9 bool AbstractDiagram::checkInvariants (bool *justReturnTheStatus* = false) const [protected, virtual, inherited]

Definition at line 1060 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::coordinatePlane().

Referenced by KDChart::RingDiagram::calculateDataBoundaries(), KDChart::PolarDiagram::calculateDataBoundaries(), KDChart::Plotter::calculateDataBoundaries(), KDChart::PieDiagram::calculateDataBoundaries(), KDChart::LineDiagram::calculateDataBoundaries(), KDChart::BarDiagram::calculateDataBoundaries(), KDChart::RingDiagram::paint(), KDChart::PolarDiagram::paint(), KDChart::Plotter::paint(), KDChart::PieDiagram::paint(), KDChart::LineDiagram::paint(), KDChart::BarDiagram::paint(), KDChart::AbstractDiagram::paintDataValueTexts(), KDChart::AbstractDiagram::paintMarker(), and KDChart::AbstractDiagram::paintMarkers().

```
1061 {
1062     if( ! justReturnTheStatus ){
1063         Q_ASSERT_X ( model(), "AbstractDiagram::checkInvariants()",
1064                     "There is no usable model set, for the diagram." );
1065
1066         Q_ASSERT_X ( coordinatePlane(), "AbstractDiagram::checkInvariants()",
1067                     "There is no usable coordinate plane set, for the diagram." );
1068     }
1069     return model() && coordinatePlane();
1070 }
```

### 9.58.3.10 QModelIndex AbstractDiagram::columnToIndex (int *column*) const [protected, inherited]

Definition at line 309 of file KDChartAbstractDiagram.cpp.

Referenced by KDChart::BarDiagram::barAttributes(), KDChart::AbstractDiagram::brush(), KDChart::AbstractDiagram::dataValueAttributes(), KDChart::AbstractDiagram::isHidden(), KDChart::Plotter::lineAttributes(), KDChart::LineDiagram::lineAttributes(), KDChart::AbstractDiagram::pen(), KDChart::AbstractPieDiagram::pieAttributes(), KDChart::BarDiagram::threeDBarAttributes(), KDChart::Plotter::threeDLineAttributes(), KDChart::LineDiagram::threeDLineAttributes(), and KDChart::AbstractPieDiagram::threeDPieAttributes().

```

310 {    // FIXME (Mirko): shouldn't this be headerData? instead of the index for the first row?
311     if( model() )
312         return QModelIndex( model()->index( 0, column, rootIndex() ) );
313     return QModelIndex();
314 }
```

### 9.58.3.11 bool AbstractDiagram::compare (const AbstractDiagram \* *other*) const [inherited]

Returns true if both diagrams have the same settings.

Definition at line 129 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::allowOverlappingDataValueTexts(), KDChart::AbstractDiagram::antiAliasing(), KDChart::AbstractDiagram::attributesModel(), KDChart::AttributesModel::compare(), KDChart::AbstractDiagram::datasetDimension(), and KDChart::AbstractDiagram::percentMode().

```

130 {
131     if( other == this ) return true;
132     if( ! other ){
133         //qDebug() << "AbstractDiagram::compare() cannot compare to Null pointer";
134         return false;
135     }
136     /*
137     qDebug() << "\n                AbstractDiagram::compare() QAbstractScrollArea:";
138     // compare QAbstractScrollArea properties
139     qDebug() <<
140     ((horizontalScrollBarPolicy() == other->horizontalScrollBarPolicy()) &&
141     (verticalScrollBarPolicy() == other->verticalScrollBarPolicy()));
142     qDebug() << "AbstractDiagram::compare() QFrame:";
143     // compare QFrame properties
144     qDebug() <<
145     ((frameShadow() == other->frameShadow()) &&
146     (frameShape() == other->frameShape()) &&
147     (frameWidth() == other->frameWidth()) &&
148     (lineWidth() == other->lineWidth()) &&
149     (midLineWidth() == other->midLineWidth()));
150     qDebug() << "AbstractDiagram::compare() QAbstractItemView:";
151     // compare QAbstractItemView properties
152     qDebug() <<
153     ((alternatingRowColors() == other->alternatingRowColors()) &&
154     (hasAutoScroll() == other->hasAutoScroll()) &&
155     #if QT_VERSION > 0x040199
156     (dragDropMode() == other->dragDropMode()) &&
157     (dragDropOverwriteMode() == other->dragDropOverwriteMode()) &&
158     (horizontalScrollMode() == other->horizontalScrollMode()) &&
159     (verticalScrollMode() == other->verticalScrollMode()) &&
160     #endif
```

```

161         (dragEnabled() == other->dragEnabled()) &&
162         (editTriggers() == other->editTriggers()) &&
163         (iconSize() == other->iconSize()) &&
164         (selectionBehavior() == other->selectionBehavior()) &&
165         (selectionMode() == other->selectionMode()) &&
166         (showDropIndicator() == other->showDropIndicator()) &&
167         (tabKeyNavigation() == other->tabKeyNavigation()) &&
168         (textElideMode() == other->textElideMode()));
169     qDebug() << "AbstractDiagram::compare() AttributesModel: ";
170     // compare all of the properties stored in the attributes model
171     qDebug() << attributesModel()->compare( other->attributesModel() );
172     qDebug() << "AbstractDiagram::compare() own:";
173     // compare own properties
174     qDebug() <<
175         ((rootIndex().column() == other->rootIndex().column()) &&
176         (rootIndex().row() == other->rootIndex().row()) &&
177         (allowOverlappingDataValueTexts() == other->allowOverlappingDataValueTexts()) &&
178         (antiAliasing() == other->antiAliasing()) &&
179         (percentMode() == other->percentMode()) &&
180         (datasetDimension() == other->datasetDimension()));
181     */
182     return // compare QAbstractScrollArea properties
183         (horizontalScrollBarPolicy() == other->horizontalScrollBarPolicy()) &&
184         (verticalScrollBarPolicy() == other->verticalScrollBarPolicy()) &&
185         // compare QFrame properties
186         (frameShadow() == other->frameShadow()) &&
187         (frameShape() == other->frameShape()) &&
188     // frameWidth is a read-only property defined by the style, it should not be in here:
189     // (frameWidth() == other->frameWidth()) &&
190     (lineWidth() == other->lineWidth()) &&
191     (midLineWidth() == other->midLineWidth()) &&
192     // compare QAbstractItemView properties
193     (alternatingRowColors() == other->alternatingRowColors()) &&
194     (hasAutoScroll() == other->hasAutoScroll()) &&
195 #if QT_VERSION > 0x040199
196     (dragDropMode() == other->dragDropMode()) &&
197     (dragDropOverwriteMode() == other->dragDropOverwriteMode()) &&
198     (horizontalScrollMode() == other->horizontalScrollMode()) &&
199     (verticalScrollMode() == other->verticalScrollMode()) &&
200 #endif
201     (dragEnabled() == other->dragEnabled()) &&
202     (editTriggers() == other->editTriggers()) &&
203     (iconSize() == other->iconSize()) &&
204     (selectionBehavior() == other->selectionBehavior()) &&
205     (selectionMode() == other->selectionMode()) &&
206     (showDropIndicator() == other->showDropIndicator()) &&
207     (tabKeyNavigation() == other->tabKeyNavigation()) &&
208     (textElideMode() == other->textElideMode()) &&
209     // compare all of the properties stored in the attributes model
210     attributesModel()->compare( other->attributesModel() ) &&
211     // compare own properties
212     ((rootIndex().column() == other->rootIndex().column()) &&
213     (rootIndex().row() == other->rootIndex().row()) &&
214     (allowOverlappingDataValueTexts() == other->allowOverlappingDataValueTexts()) &&
215     (antiAliasing() == other->antiAliasing()) &&
216     (percentMode() == other->percentMode()) &&
217     (datasetDimension() == other->datasetDimension()));
218 }

```

### 9.58.3.12 [AbstractCoordinatePlane](#) \* [AbstractDiagram::coordinatePlane](#) () const [inherited]

The coordinate plane associated with the diagram.

This determines how coordinates in value space are mapped into pixel space. By default this is a [Cartesian-](#)

CoordinatePlane.

#### Returns:

The coordinate plane associated with the diagram.

Definition at line 220 of file KDChartAbstractDiagram.cpp.

References `d`.

Referenced by `KDChart::AbstractDiagram::checkInvariants()`, `KDChart::AbstractCartesianDiagram::layoutPlanes()`, `KDChart::PolarDiagram::paint()`, `KDChart::AbstractDiagram::paintDataValueTexts()`, `KDChart::AbstractDiagram::paintMarkers()`, `KDChart::AbstractPolarDiagram::polarCoordinatePlane()`, and `KDChart::AbstractCartesianDiagram::setCoordinatePlane()`.

```
221 {
222     return d->plane;
223 }
```

#### 9.58.3.13 `const QPair< QPointF, QPointF > AbstractDiagram::dataBoundaries () const` [inherited]

Return the bottom left and top right data point, that the diagram will display (unless the grid adjusts these values).

This method returns a cached result of calculations done by `calculateDataBoundaries`. Classes derived from [AbstractDiagram](#) must implement the `calculateDataBoundaries` function, to specify their own way of calculating the data boundaries. If derived classes want to force recalculation of the data boundaries, they can call [setDataBoundariesDirty\(\)](#)

Returned value is in diagram coordinates.

Definition at line 225 of file KDChartAbstractDiagram.cpp.

References `KDChart::AbstractDiagram::calculateDataBoundaries()`, and `d`.

Referenced by `KDChart::CartesianCoordinatePlane::getRawDataBoundingRectFromDiagrams()`, `KDChart::PolarCoordinatePlane::layoutDiagrams()`, `KDChart::Plotter::paint()`, `KDChart::LineDiagram::paint()`, and `KDChart::BarDiagram::paint()`.

```
226 {
227     if( d->databoundariesDirty ){
228         d->databoundaries = calculateDataBoundaries ();
229         d->databoundariesDirty = false;
230     }
231     return d->databoundaries;
232 }
```

#### 9.58.3.14 `void AbstractDiagram::dataChanged (const QModelIndex & topLeft, const QModelIndex & bottomRight)` [virtual, inherited]

[reimplemented]

Definition at line 330 of file KDChartAbstractDiagram.cpp.

References `KDChart::AbstractDiagram::setDataBoundariesDirty()`.

```

332 {
333     Q_UNUSED( topLeft );
334     Q_UNUSED( bottomRight );
335     // We are still too dumb to do intelligent updates...
336     setDataBoundariesDirty();
337     scheduleDelayedItemsLayout();
338 }

```

### 9.58.3.15 void KDChart::AbstractDiagram::dataHidden () [signal, inherited]

This signal is emitted, when the hidden status of at least one data cell was (un)set.

Referenced by KDChart::AbstractDiagram::setHidden().

### 9.58.3.16 QList< QBrush > AbstractDiagram::datasetBrushes () const [inherited]

The set of dataset brushes currently used, for use in legends, etc.

#### Note:

Cell-level override brushes, if set, take precedence over the dataset values, so you might need to check these too, in order to find the brush, that is used for a single cell.

#### Returns:

The current set of dataset brushes.

Definition at line 1018 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::attributesModelRootIndex(), KDChart::AttributesModel::columnCount(), KDChart::DatasetBrushRole, and KDChart::AbstractDiagram::datasetDimension().

Referenced by KDChart::Legend::setBrushesFromDiagram().

```

1019 {
1020     QList<QBrush> ret;
1021     if( model() == 0 )
1022         return ret;
1023
1024     const int columnCount = attributesModel()->columnCount(attributesModelRootIndex());
1025     for( int i = 0; i < columnCount / datasetDimension(); ++i )
1026         ret << QVariantValue<QBrush>( attributesModel()->headerData( i, Qt::Vertical, DatasetBrushRole ));
1027
1028     return ret;
1029 }

```

### 9.58.3.17 int AbstractDiagram::datasetDimension () const [inherited]

The dataset dimension of a diagram determines, how many value dimensions it expects each datapoint to have.

For each dimension it will expect one column of values in the model. If the dimensionality is 1, automatic values will be used for the abscissa.

For example a diagram with the default dimension of 1, will have one column per datapoint (the y values) and will use automatic values for the x axis (1, 2, 3, ... n). If the dimension is 2, the diagram will use the

first, (and the third, fifth, etc) columns as X values, and the second, (and the fourth, sixth, etc) column as Y values.

#### Returns:

The dataset dimension of the diagram.

Definition at line 1072 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::brush(), KDChart::AbstractDiagram::compare(), KDChart::AbstractDiagram::datasetBrushes(), KDChart::AbstractDiagram::datasetLabels(), KDChart::AbstractDiagram::datasetMarkers(), KDChart::AbstractDiagram::datasetPens(), KDChart::LineDiagram::getCellValues(), KDChart::CartesianCoordinatePlane::getDataDimensionsList(), paint(), KDChart::TernaryLineDiagram::paint(), KDChart::AbstractDiagram::paintDataValueTexts(), KDChart::AbstractDiagram::paintMarkers(), KDChart::AbstractDiagram::pen(), KDChart::AbstractDiagram::setPen(), KDChart::Plotter::setType(), and KDChart::LineDiagram::setType().

```
1073 {
1074     return d->datasetDimension;
1075 }
```

#### 9.58.3.18 QStringList AbstractDiagram::datasetLabels () const [inherited]

The set of dataset labels currently displayed, for use in legends, etc.

#### Returns:

The set of dataset labels currently displayed.

Definition at line 1005 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::attributesModelRootIndex(), KDChart::AttributesModel::columnCount(), KDChart::AbstractDiagram::datasetDimension(), and KDChart::AttributesModel::headerData().

```
1006 {
1007     QStringList ret;
1008     if( model() == 0 )
1009         return ret;
1010
1011     const int columnCount = attributesModel()->columnCount(attributesModelRootIndex());
1012     for( int i = 0; i < columnCount / datasetDimension(); ++i )
1013         ret << attributesModel()->headerData( i, Qt::Horizontal, Qt::DisplayRole ).toString();
1014
1015     return ret;
1016 }
```

#### 9.58.3.19 QList< [MarkerAttributes](#) > AbstractDiagram::datasetMarkers () const [inherited]

The set of dataset markers currently used, for use in legends, etc.

#### Note:

Cell-level override markers, if set, take precedence over the dataset values, so you might need to check these too, in order to find the marker, that is shown for a single cell.

**Returns:**

The current set of dataset brushes.

Definition at line 1044 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::attributesModelRootIndex(), KDChart::AttributesModel::columnCount(), KDChart::AbstractDiagram::datasetDimension(), and KDChart::DataValueLabelAttributesRole.

```

1045 {
1046     QList<MarkerAttributes> ret;
1047     if( model() == 0 )
1048         return ret;
1049
1050     const int columnCount = attributesModel()->columnCount(attributesModelRootIndex());
1051     for( int i = 0; i < columnCount / datasetDimension(); ++i )
1052     {
1053         const DataValueAttributes a =
1054             qVariantValue<DataValueAttributes>( attributesModel()->headerData( i, Qt::Vertical, DataValueLabelAttributesRole ) );
1055         ret << a.markerAttributes();
1056     }
1057     return ret;
1058 }
```

**9.58.3.20 QList< QPen > AbstractDiagram::datasetPens () const** [inherited]

The set of dataset pens currently used, for use in legends, etc.

**Note:**

Cell-level override pens, if set, take precedence over the dataset values, so you might need to check these too, in order to find the pens, that is used for a single cell.

**Returns:**

The current set of dataset pens.

Definition at line 1031 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::attributesModelRootIndex(), KDChart::AttributesModel::columnCount(), KDChart::AbstractDiagram::datasetDimension(), and KDChart::DatasetPenRole.

```

1032 {
1033     QList<QPen> ret;
1034     if( model() == 0 )
1035         return ret;
1036
1037     const int columnCount = attributesModel()->columnCount(attributesModelRootIndex());
1038     for( int i = 0; i < columnCount / datasetDimension(); ++i )
1039         ret << qVariantValue<QPen>( attributesModel()->headerData( i, Qt::Vertical, DatasetPenRole ) );
1040
1041     return ret;
1042 }
```



### 9.58.3.21 [DataValueAttributes](#) AbstractDiagram::dataValueAttributes (const QModelIndex & *index*) const [inherited]

Retrieve the [DataValueAttributes](#) for the given index.

This will fall back automatically to what was set at dataset or model level, if there are no datapoint specific settings.

#### Parameters:

*index* The datapoint to retrieve the attributes for.

#### Returns:

The [DataValueAttributes](#) for the given index.

Definition at line 421 of file KDChartAbstractDiagram.cpp.

References [KDChart::AbstractDiagram::attributesModel\(\)](#), [KDChart::AttributesModel::data\(\)](#), and [KDChart::DataValueLabelAttributesRole](#).

```
422 {
423     return qVariantValue<DataValueAttributes>(
424         attributesModel()->data( attributesModel()->mapFromSource(index),
425                                 KDChart::DataValueLabelAttributesRole ));
426 }
```

### 9.58.3.22 [DataValueAttributes](#) AbstractDiagram::dataValueAttributes (int *column*) const [inherited]

Retrieve the [DataValueAttributes](#) for the given dataset.

This will fall back automatically to what was set at model level, if there are no dataset specific settings.

#### Parameters:

*column* The dataset to retrieve the attributes for.

#### Returns:

The [DataValueAttributes](#) for the given dataset.

Definition at line 414 of file KDChartAbstractDiagram.cpp.

References [KDChart::AbstractDiagram::attributesModel\(\)](#), [KDChart::AbstractDiagram::columnToIndex\(\)](#), [KDChart::AttributesModel::data\(\)](#), and [KDChart::DataValueLabelAttributesRole](#).

```
415 {
416     return qVariantValue<DataValueAttributes>(
417         attributesModel()->data( attributesModel()->mapFromSource(columnToIndex( column )),
418                                 KDChart::DataValueLabelAttributesRole ));
419 }
```

### 9.58.3.23 [DataValueAttributes](#) AbstractDiagram::dataValueAttributes () const [inherited]

Retrieve the [DataValueAttributes](#) specified globally.

This will fall back automatically to the default settings, if there are no specific settings.

**Returns:**

The global [DataValueAttributes](#).

Definition at line 408 of file `KDChartAbstractDiagram.cpp`.

References `KDChart::AbstractDiagram::attributesModel()`, `KDChart::DataValueLabelAttributesRole`, and `KDChart::AttributesModel::modelData()`.

Referenced by `KDChart::AbstractDiagram::paintDataValueText()`, `KDChart::AbstractDiagram::paintMarker()`, and `KDChart::TernaryLineDiagram::TernaryLineDiagram()`.

```
409 {
410     return qVariantValue<DataValueAttributes>(
411         attributesModel()->modelData( KDChart::DataValueLabelAttributesRole ) );
412 }
```

**9.58.3.24 void AbstractDiagram::doItemsLayout () [virtual, inherited]**

[reimplemented]

Definition at line 321 of file `KDChartAbstractDiagram.cpp`.

References `d`, and `KDChart::AbstractDiagram::update()`.

```
322 {
323     if ( d->plane ) {
324         d->plane->layoutDiagrams();
325         update();
326     }
327     QAbstractItemView::doItemsLayout();
328 }
```

**9.58.3.25 int AbstractDiagram::horizontalOffset () const [virtual, inherited]**

[reimplemented]

Definition at line 950 of file `KDChartAbstractDiagram.cpp`.

```
951 { return 0; }
```

**9.58.3.26 QModelIndex AbstractDiagram::indexAt (const QPoint & point) const [virtual, inherited]**

[reimplemented]

Definition at line 1098 of file `KDChartAbstractDiagram.cpp`.

References `d`.

```
1099 {
1100     return d->indexAt( point );
1101 }
```

### 9.58.3.27 QModelIndexList AbstractDiagram::indexesAt (const QPoint & *point*) const [inherited]

This method is added alongside with `indexAt` from QAIM, since in `kdchart` multiple indexes can be displayed at the same spot.

Definition at line 1103 of file `KDChartAbstractDiagram.cpp`.

References `d`.

```
1104 {
1105     return d->indexesAt( point );
1106 }
```

### 9.58.3.28 bool AbstractDiagram::isHidden (const QModelIndex & *index*) const [inherited]

Retrieve the hidden status for the given index.

This will fall back automatically to what was set at dataset or diagram level, if there are no datapoint specific settings.

#### Parameters:

*index* The datapoint to retrieve the hidden status for.

#### Returns:

The hidden status for the given index.

Definition at line 380 of file `KDChartAbstractDiagram.cpp`.

References `KDChart::AbstractDiagram::attributesModel()`, `KDChart::AttributesModel::data()`, and `KDChart::DataHiddenRole`.

```
381 {
382     return qVariantValue<bool>(
383         attributesModel()->data(
384             attributesModel()->mapFromSource(index),
385             DataHiddenRole ) );
386 }
```

### 9.58.3.29 bool AbstractDiagram::isHidden (int *column*) const [inherited]

Retrieve the hidden status for the given dataset.

This will fall back automatically to what was set at diagram level, if there are no dataset specific settings.

#### Parameters:

*column* The dataset to retrieve the hidden status for.

#### Returns:

The hidden status for the given dataset.

Definition at line 373 of file `KDChartAbstractDiagram.cpp`.

References `KDChart::AbstractDiagram::attributesModel()`, `KDChart::AbstractDiagram::columnToIndex()`, `KDChart::AttributesModel::data()`, and `KDChart::DataHiddenRole`.

```

374 {
375     return qVariantValue<bool>(
376         attributesModel()->data(
377             attributesModel()->mapFromSource(columnToIndex( column )),
378             DataHiddenRole ) );
379 }

```

### 9.58.3.30 bool AbstractDiagram::isHidden () const [inherited]

Retrieve the hidden status specified globally.

This will fall back automatically to the default settings (= not hidden), if there are no specific settings.

#### Returns:

The global hidden status.

Definition at line 367 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DataHiddenRole, and KDChart::AttributesModel::modelData().

Referenced by KDChart::LineDiagram::valueForCellTesting().

```

368 {
369     return qVariantValue<bool>(
370         attributesModel()->modelData( DataHiddenRole ) );
371 }

```

### 9.58.3.31 bool AbstractDiagram::isIndexHidden (const QModelIndex & index) const [virtual, inherited]

[reimplemented]

Definition at line 956 of file KDChartAbstractDiagram.cpp.

```

957 { return true; }

```

### 9.58.3.32 QStringList AbstractDiagram::itemRowLabels () const [inherited]

The set of item row labels currently displayed, for use in Abscissa axes, etc.

#### Returns:

The set of item row labels currently displayed.

Definition at line 989 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::attributesModel-RootIndex(), KDChart::AttributesModel::rowCount(), KDChart::AbstractDiagram::unitPrefix(), and KDChart::AbstractDiagram::unitSuffix().

```

990 {
991     QStringList ret;

```

```

992     if( model() ){
993         //QDebug() << "AbstractDiagram::itemRowLabels(): " << attributesModel()->rowCount(attributesModelRootIndex());
994         const int rowCount = attributesModel()->rowCount(attributesModelRootIndex());
995         for( int i = 0; i < rowCount; ++i ){
996             //QDebug() << "item row label: " << attributesModel()->headerData( i, Qt::Vertical, Qt::DisplayRole );
997             ret << unitPrefix( i, Qt::Horizontal, true ) +
998                 attributesModel()->headerData( i, Qt::Vertical, Qt::DisplayRole ).toString() +
999                 unitSuffix( i, Qt::Horizontal, true );
1000         }
1001     }
1002     return ret;
1003 }

```

### 9.58.3.33 void KDChart::AbstractDiagram::layoutChanged ([AbstractDiagram \\*](#)) [signal, inherited]

Diagrams are supposed to emit this signal, when the layout of one of their element changes.

Layouts can change, for example, when axes are added or removed, or when the configuration was changed in a way that the axes or the diagram itself are displayed in a different geometry. Changes in the diagrams coordinate system also result in the [layoutChanged\(\)](#) signal being emitted.

Referenced by [KDChart::AbstractDiagram::setDatasetDimension\(\)](#), [KDChart::AbstractPieDiagram::setPieAttributes\(\)](#), [KDChart::AbstractPieDiagram::setThreeDPieAttributes\(\)](#), [KDChart::Plotter::setType\(\)](#), [KDChart::LineDiagram::setType\(\)](#), and [KDChart::BarDiagram::setType\(\)](#).

### 9.58.3.34 void KDChart::AbstractDiagram::modelsChanged () [signal, inherited]

This signal is emitted, when either the model or the [AttributesModel](#) is replaced.

Referenced by [KDChart::AbstractDiagram::setAttributesModel\(\)](#), and [KDChart::AbstractDiagram::setModel\(\)](#).

### 9.58.3.35 QModelIndex AbstractDiagram::moveCursor ([CursorAction cursorAction](#), [Qt::KeyboardModifiers modifiers](#)) [virtual, inherited]

[reimplemented]

Definition at line 947 of file [KDChartAbstractDiagram.cpp](#).

```

948 { return QModelIndex(); }

```

### 9.58.3.36 void TernaryPointDiagram::paint ([PaintContext \\*paintContext](#)) [virtual]

Draw the diagram contents to the rectangle and painter, that are passed in as part of the paint context.

#### Parameters:

***paintContext*** All information needed for painting.

Reimplemented from [KDChart::AbstractTernaryDiagram](#).

Definition at line 72 of file [KDChartTernaryPointDiagram.cpp](#).

References `KDChart::AbstractDiagram::brush()`, `KDChart::PaintContext::coordinatePlane()`, `d`, `KDChart::AbstractDiagram::datasetDimension()`, `KDChart::PaintContext::painter()`, `KDChart::AbstractDiagram::paintMarker()`, `KDChart::AbstractDiagram::pen()`, `KDChart::TernaryCoordinatePlane::translate()`, and `translate()`.

```

73 {
74     d->reverseMapper.clear();
75
76     d->paint( paintContext );
77
78     // sanity checks:
79     if ( model() == 0 ) return;
80
81     QPainter* p = paintContext->painter();
82     PainterSaver s( p );
83
84     TernaryCoordinatePlane* plane =
85         (TernaryCoordinatePlane*) paintContext->coordinatePlane();
86     Q_ASSERT( plane );
87
88     double x, y, z;
89
90     int columnCount = model()->columnCount( rootIndex() );
91     for(int column=0; column<columnCount; column+=datasetDimension() )
92     {
93         int numRows = model()->rowCount( rootIndex() );
94         for( int row = 0; row < numRows; row++ )
95         {
96             QModelIndex base = model()->index( row, column );
97             // see if there is data otherwise skip
98             if( ! model()->data( model()->index( row, column+0 ) ).isNull() )
99             {
100                 p->setPen( pen( base ) );
101                 p->setBrush( brush( base ) );
102
103                 // retrieve data
104                 x = qMax( model()->data( model()->index( row, column+0 ) ).toDouble(),
105                     0.0 );
106                 y = qMax( model()->data( model()->index( row, column+1 ) ).toDouble(),
107                     0.0 );
108                 z = qMax( model()->data( model()->index( row, column+2 ) ).toDouble(),
109                     0.0 );
110
111                 // fix messed up data values (paint as much as possible)
112                 double total = x + y + z;
113                 if ( fabs( total ) > 3 * std::numeric_limits<double>::epsilon() ) {
114                     TernaryPoint tPunkt( x / total, y / total );
115                     QPointF diagramLocation = translate( tPunkt );
116                     QPointF widgetLocation = plane->translate( diagramLocation );
117
118                     paintMarker( p, model()->index( row, column ), widgetLocation );
119                     QString text = tr( "(%1, %2, %3)" )
120                         .arg( x * 100, 0, 'f', 0 )
121                         .arg( y * 100, 0, 'f', 0 )
122                         .arg( z * 100, 0, 'f', 0 );
123                     d->paintDataValueText( p, base, widgetLocation, text );
124                 } else {
125                     // ignore and do not paint this point, garbage data
126                     qDebug() << "TernaryPointDiagram::paint: data point x/y/z:"
127                         << x << "/" << y << "/" << z << "ignored, unusable.";
128                 }
129             }
130         }
131     }
132 }

```

### 9.58.3.37 void AbstractDiagram::paintDataValueText (QPainter \* *painter*, const QModelIndex & *index*, const QPointF & *pos*, double *value*) [inherited]

Definition at line 468 of file KDChartAbstractDiagram.cpp.

References `d`, `KDChart::DataValueAttributes::dataLabel()`, `KDChart::AbstractDiagram::dataValueAttributes()`, `KDChart::DataValueAttributes::decimalDigits()`, `KDChart::DataValueAttributes::isVisible()`, `KDChartEnums::MeasureOrientationMinimum`, `KDChart::DataValueAttributes::position()`, `KDChart::DataValueAttributes::prefix()`, `KDChart::DataValueAttributes::suffix()`, and `KDChart::DataValueAttributes::textAttributes()`.

Referenced by `KDChart::RingDiagram::paint()`, `KDChart::PolarDiagram::paint()`, and `KDChart::AbstractDiagram::paintDataValueTexts()`.

```

472 {
473     // paint one data series
474     const DataValueAttributes a( dataValueAttributes(index) );
475     if ( !a.isVisible() ) return;
476
477     // handle decimal digits
478     int decimalDigits = a.decimalDigits();
479     int decimalPos = QString::number( value ).indexOf( QLatin1Char( '.' ) );
480     QString roundedValue;
481     if ( a.dataLabel().isNull() ) {
482         if ( decimalPos > 0 && value != 0 )
483             roundedValue = roundValues( value, decimalPos, decimalDigits );
484         else
485             roundedValue = QString::number( value );
486     } else
487         roundedValue = a.dataLabel();
488     // handle prefix and suffix
489     if ( !a.prefix().isNull() )
490         roundedValue.prepend( a.prefix() );
491
492     if ( !a.suffix().isNull() )
493         roundedValue.append( a.suffix() );
494
495     const TextAttributes ta( a.textAttributes() );
496     // FIXME draw the non-text bits, background, etc
497     if ( ta.isVisible() ) {
498
499         QPointF pt( pos );
500         /* for debugging:
501         PainterSaver painterSaver( painter );
502         painter->setPen( Qt::black );
503         painter->drawLine( pos - QPointF( 1,1), pos + QPointF( 1,1) );
504         painter->drawLine( pos - QPointF(-1,1), pos + QPointF(-1,1) );
505         */
506
507         QTextDocument doc;
508         if( Qt::mightBeRichText( roundedValue ) )
509             doc.setHtml( roundedValue );
510         else
511             doc.setPlainText( roundedValue );
512
513         const RelativePosition relPos( a.position( value >= 0.0 ) );
514         const Qt::Alignment alignBottomLeft = Qt::AlignBottom | Qt::AlignLeft;
515         const QFont calculatedFont( ta.calculatedFont( d->plane, KDChartEnums::MeasureOrientationMinimum ) );
516         const QRectF boundRect( d->cachedFontMetrics( calculatedFont, painter->device() )->boundingRect( roundedValue, calculatedFont, alignBottomLeft ) );
517
518         // To place correctly
519         pt.ry() -= boundRect.height();
520
521         //qDebug() << "calculatedFont's point size:" << calculatedFont.pointSizeF();
522         // adjust the text start point position, if alignment is not Bottom/Left

```

```

523         if( relPos.alignment() & alignBottomLeft) != alignBottomLeft ){
524             if( relPos.alignment() & Qt::AlignRight )
525                 pt.rx() -= boundRect.width();
526             else if( relPos.alignment() & Qt::AlignHCenter )
527                 pt.rx() -= 0.5 * boundRect.width();
528
529             if( relPos.alignment() & Qt::AlignTop )
530                 pt.ry() += boundRect.height();
531             else if( relPos.alignment() & Qt::AlignVCenter )
532                 pt.ry() += 0.5 * boundRect.height();
533         }
534
535         // FIXME draw the non-text bits, background, etc
536
537         if ( a.showRepetitiveDataLabels() ||
538             pos.x() <= d->lastX ||
539             d->lastRoundedValue != roundedValue ) {
540             d->lastRoundedValue = roundedValue;
541             d->lastX = pos.x();
542             PainterSaver painterSaver( painter );
543
544             doc.setDefaultFont( calculatedFont );
545             QAbstractTextDocumentLayout::PaintContext context;
546             context.palette = palette();
547             context.palette.setColor( QPalette::Text, ta.pen().color() );
548
549             painter->translate( pt );
550             painter->rotate( ta.rotation() );
551
552             QAbstractTextDocumentLayout* layout = doc.documentLayout();
553             layout->draw( painter, context );
554         }
555     }
556 }

```

**9.58.3.38** `void AbstractDiagram::paintDataValueTexts (QPainter * painter)` [protected, virtual, inherited]

Definition at line 584 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::checkInvariants(), KDChart::AbstractDiagram::coordinatePlane(), KDChart::AbstractDiagram::datasetDimension(), KDChart::AbstractDiagram::paintDataValueText(), and KDChart::AbstractCoordinatePlane::translate().

```

585 {
586     if ( !checkInvariants() ) return;
587     const int rowCount = model()->rowCount( rootIndex() );
588     const int columnCount = model()->columnCount( rootIndex() );
589     for ( int i=datasetDimension()-1; i<columnCount; i += datasetDimension() ) {
590         for ( int j=0; j<rowCount; ++j ) {
591             const QModelIndex index = model()->index( j, i, rootIndex() );
592             double value = model()->data( index ).toDouble();
593             const QPointF pos = coordinatePlane()->translate( QPointF( j, value ) );
594             paintDataValueText( painter, index, pos, value );
595         }
596     }
597 }

```

**9.58.3.39** `void AbstractDiagram::paintMarker (QPainter * painter, const QModelIndex & index, const QPointF & pos)` [inherited]

Definition at line 600 of file KDChartAbstractDiagram.cpp.



References KDChart::AbstractDiagram::brush(), KDChart::AbstractDiagram::checkInvariants(),  
 d, KDChart::AbstractDiagram::dataValueAttributes(), KDChart::MarkerAttributes::isVisible(),  
 KDChart::DataValueAttributes::isVisible(), KDChart::DataValueAttributes::markerAttributes(),  
 KDChart::MarkerAttributes::markerColor(), KDChart::MarkerAttributes::markerSize(),  
 KDChart::AbstractDiagram::paintMarker(), and KDChart::MarkerAttributes::pen().

```

603 {
604     if ( !checkInvariants() ) return;
605     DataValueAttributes a = dataValueAttributes(index);
606     if ( !a.isVisible() ) return;
607     const MarkerAttributes &ma = a.markerAttributes();
608     if ( !ma.isVisible() ) return;
609
610     PainterSaver painterSaver( painter );
611     QSizeF maSize( ma.markerSize() );
612     QBrush indexBrush( brush( index ) );
613     QPen indexPen( ma.pen() );
614     if ( ma.markerColor().isValid() )
615         indexBrush.setColor( ma.markerColor() );
616
617     paintMarker( painter, ma, indexBrush, indexPen, pos, maSize );
618
619     // workaround: BC cannot be changed, otherwise we would pass the
620     // index down to next-lower paintMarker function. So far, we
621     // basically save a circle of radius maSize at pos in the
622     // reverseMapper. This means that ^^ this version of paintMarker
623     // needs to be called to reverse-map the marker.
624     d->reverseMapper.addCircle( index.row(), index.column(), pos, 2 * maSize );
625 }
```

### 9.58.3.40 void AbstractDiagram::paintMarker (QPainter \*painter, const MarkerAttributes &markerAttributes, const QBrush &brush, const QPen &, const QPointF &point, const QSizeF &size) [virtual, inherited]

Definition at line 627 of file KDChartAbstractDiagram.cpp.

References KDChart::MarkerAttributes::Marker1Pixel, KDChart::MarkerAttributes::Marker4Pixels, KDChart::MarkerAttributes::MarkerCircle, KDChart::MarkerAttributes::MarkerCross, KDChart::MarkerAttributes::MarkerDiamond, KDChart::MarkerAttributes::MarkerFastCross, KDChart::MarkerAttributes::MarkerRing, KDChart::MarkerAttributes::MarkerSquare, and KDChart::MarkerAttributes::markerStyle().

Referenced by paint(), KDChart::TernaryLineDiagram::paint(), KDChart::MarkerLayoutItem::paintIntoRect(), KDChart::AbstractDiagram::paintMarker(), and KDChart::AbstractDiagram::paintMarkers().

```

633 {
634     const QPen oldPen( painter->pen() );
635     // Pen is used to paint 4Pixels - 1 Pixel - Ring and FastCross types.
636     // make sure to use the brush color - see above in those cases.
637     const bool isFourPixels = (markerAttributes.markerStyle() == MarkerAttributes::Marker4Pixels);
638     if( isFourPixels || (markerAttributes.markerStyle() == MarkerAttributes::Marker1Pixel) ){
639         // for high-performance point charts with tiny point markers:
640         painter->setPen( QPen( brush.color().light() ) );
641         if( isFourPixels ){
642             const qreal x = pos.x();
643             const qreal y = pos.y();
644             painter->drawLine( QPointF(x-1.0,y-1.0),
645                             QPointF(x+1.0,y-1.0) );
646             painter->drawLine( QPointF(x-1.0,y),
647                             QPointF(x+1.0,y) );
648             painter->drawLine( QPointF(x-1.0,y+1.0),
```

```

649             QPointF(x+1.0,y+1.0) );
650         }
651         painter->drawPoint( pos );
652     }else{
653         PainterSaver painterSaver( painter );
654         // we only a solid line surrounding the markers
655         QPen painterPen( pen );
656         painterPen.setStyle( Qt::SolidLine );
657         painter->setPen( painterPen );
658         painter->setBrush( brush );
659         painter->setRenderHint ( QPainter::Antialiasing );
660         painter->translate( pos );
661         switch ( markerAttributes.markerStyle() ) {
662             case MarkerAttributes::MarkerCircle:
663                 painter->drawEllipse( QRectF( 0 - maSize.height()/2, 0 - maSize.width()/2,
664                     maSize.height(), maSize.width() ) );
665                 break;
666             case MarkerAttributes::MarkerSquare:
667                 {
668                     QRectF rect( 0 - maSize.width()/2, 0 - maSize.height()/2,
669                         maSize.width(), maSize.height() );
670                     painter->drawRect( rect );
671                     break;
672                 }
673             case MarkerAttributes::MarkerDiamond:
674                 {
675                     QVector <QPointF > diamondPoints;
676                     QPointF top, left, bottom, right;
677                     top = QPointF( 0, 0 - maSize.height()/2 );
678                     left = QPointF( 0 - maSize.width()/2, 0 );
679                     bottom = QPointF( 0, maSize.height()/2 );
680                     right = QPointF( maSize.width()/2, 0 );
681                     diamondPoints << top << left << bottom << right;
682                     painter->drawPolygon( diamondPoints );
683                     break;
684                 }
685             // both handled on top of the method:
686             case MarkerAttributes::Marker1Pixel:
687             case MarkerAttributes::Marker4Pixels:
688                 break;
689             case MarkerAttributes::MarkerRing:
690                 {
691                     painter->setPen( QPen( brush.color() ) );
692                     painter->setBrush( Qt::NoBrush );
693                     painter->drawEllipse( QRectF( 0 - maSize.height()/2, 0 - maSize.width()/2,
694                         maSize.height(), maSize.width() ) );
695                     break;
696                 }
697             case MarkerAttributes::MarkerCross:
698                 {
699                     QRectF rect( maSize.width()*-0.5, maSize.height()*-0.2,
700                         maSize.width(), maSize.height()*0.4 );
701                     painter->drawRect( rect );
702                     rect.setTopLeft( QPointF( maSize.width()*-0.2, maSize.height()*-0.5 ) );
703                     rect.setSize( QSizeF( maSize.width()*0.4, maSize.height() ) );
704                     painter->drawRect( rect );
705                     break;
706                 }
707             case MarkerAttributes::MarkerFastCross:
708                 {
709                     QPointF left, right, top, bottom;
710                     left = QPointF( -maSize.width()/2, 0 );
711                     right = QPointF( maSize.width()/2, 0 );
712                     top = QPointF( 0, -maSize.height()/2 );
713                     bottom= QPointF( 0, maSize.height()/2 );
714                     painter->setPen( QPen( brush.color() ) );
715                     painter->drawLine( left, right );

```

```

716             painter->drawLine( top, bottom );
717             break;
718         }
719         default:
720             Q_ASSERT_X ( false, "paintMarkers()",
721                 "Type item does not match a defined Marker Type." );
722     }
723 }
724 painter->setPen( oldPen );
725 }

```

### 9.58.3.41 void AbstractDiagram::paintMarkers (QPainter \* *painter*) [protected, virtual, inherited]

Definition at line 727 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::checkInvariants(), KDChart::AbstractDiagram::coordinatePlane(), KDChart::AbstractDiagram::datasetDimension(), KDChart::AbstractDiagram::paintMarker(), and KDChart::AbstractCoordinatePlane::translate().

```

728 {
729     if ( !checkInvariants() ) return;
730     const int rowCount = model()->rowCount( rootIndex() );
731     const int columnCount = model()->columnCount( rootIndex() );
732     for ( int i=datasetDimension()-1; i<columnCount; i += datasetDimension() ) {
733         for ( int j=0; j< rowCount; ++j ) {
734             const QModelIndex index = model()->index( j, i, rootIndex() );
735             double value = model()->data( index ).toDouble();
736             const QPointF pos = coordinatePlane()->translate( QPointF( j, value ) );
737             paintMarker( painter, index, pos );
738         }
739     }
740 }

```

### 9.58.3.42 QPen AbstractDiagram::pen (const QModelIndex & *index*) const [inherited]

Retrieve the pen to be used, for painting the datapoint at the given index in the model.

#### Parameters:

*index* The index of the datapoint in the model.

#### Returns:

The pen to use for painting.

Definition at line 788 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AttributesModel::data(), KDChart::AbstractDiagram::datasetDimension(), KDChart::DatasetPenRole, and KDChart::AbstractDiagram::pen().

```

789 {
790     if( datasetDimension() > 1 )
791         return pen( index.column() );
792     return QVariantValue<QPen>(
793         attributesModel()->data(
794             attributesModel()->mapFromSource( index ),
795             DatasetPenRole ) );
796 }

```

**9.58.3.43 QPen AbstractDiagram::pen (int *dataset*) const** [inherited]

Retrieve the pen to be used for the given dataset.

This will fall back automatically to what was set at model level, if there are no dataset specific settings.

**Parameters:**

*dataset* The dataset to retrieve the pen for.

**Returns:**

The pen to use for painting.

Definition at line 777 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::columnToIndex(), KDChart::AttributesModel::data(), KDChart::AbstractDiagram::datasetDimension(), and KDChart::DatasetPenRole.

```

778 {
779     if( datasetDimension() > 1 )
780         dataset /= datasetDimension();
781
782     return qVariantValue<QPen>(
783         attributesModel()->data(
784             attributesModel()->mapFromSource( columnToIndex( dataset ) ),
785             DatasetPenRole ) );
786 }
```

**9.58.3.44 QPen AbstractDiagram::pen () const** [inherited]

Retrieve the pen to be used for painting datapoints globally.

This will fall back automatically to the default settings, if there are no specific settings.

**Returns:**

The pen to use for painting.

Definition at line 771 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AttributesModel::data(), and KDChart::DatasetPenRole.

Referenced by paint(), KDChart::TernaryLineDiagram::paint(), and KDChart::AbstractDiagram::pen().

```

772 {
773     return qVariantValue<QPen>(
774         attributesModel()->data( DatasetPenRole ) );
775 }
```

**9.58.3.45 bool AbstractDiagram::percentMode () const** [inherited]

Definition at line 462 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::compare(), and KDChart::CartesianCoordinatePlane::getDataDimensionsList().

```

463 {
464     return d->percent;
465 }

```

### 9.58.3.46 void KDChart::AbstractDiagram::propertiesChanged () [signal, inherited]

Emitted upon change of a property of the Diagram.

Referenced by KDChart::Plotter::resetLineAttributes(), KDChart::LineDiagram::resetLineAttributes(), KDChart::AbstractDiagram::setAllowOverlappingDataValueTexts(), KDChart::AbstractDiagram::setAntiAliasing(), KDChart::BarDiagram::setBarAttributes(), KDChart::AbstractDiagram::setBrush(), KDChart::AbstractDiagram::setDataValueAttributes(), KDChart::Plotter::setLineAttributes(), KDChart::LineDiagram::setLineAttributes(), KDChart::AbstractDiagram::setPen(), KDChart::AbstractDiagram::setPercentMode(), KDChart::BarDiagram::setThreeDBarAttributes(), KDChart::Plotter::setThreeDLineAttributes(), KDChart::LineDiagram::setThreeDLineAttributes(), KDChart::Plotter::setType(), KDChart::LineDiagram::setType(), KDChart::BarDiagram::setType(), KDChart::Plotter::setValueTrackerAttributes(), and KDChart::LineDiagram::setValueTrackerAttributes().

### 9.58.3.47 void TernaryPointDiagram::resize (const QSizeF & area) [virtual]

Called by the widget's sizeEvent.

Adjust all internal structures, that are calculated, depending on the size of the widget.

#### Parameters:

*area*

Implements [KDChart::AbstractTernaryDiagram](#).

Definition at line 67 of file KDChartTernaryPointDiagram.cpp.

```

68 {
69     Q_UNUSED( area );
70 }

```

### 9.58.3.48 void AbstractDiagram::scrollTo (const QModelIndex & index, ScrollHint hint = EnsureVisible) [virtual, inherited]

[reimplemented]

Definition at line 942 of file KDChartAbstractDiagram.cpp.

```

943 {}

```

### 9.58.3.49 void AbstractDiagram::setAllowOverlappingDataValueTexts (bool allow) [inherited]

Set whether data value labels are allowed to overlap.

#### Parameters:

*allow* True means that overlapping labels are allowed.

Definition at line 434 of file KDChartAbstractDiagram.cpp.

References `d`, and `KDChart::AbstractDiagram::propertiesChanged()`.

```
435 {  
436     d->allowOverlappingDataValueTexts = allow;  
437     emit propertiesChanged();  
438 }
```

### 9.58.3.50 void AbstractDiagram::setAntiAliasing (bool *enabled*) [inherited]

Set whether anti-aliasing is to be used while rendering this diagram.

#### Parameters:

*enabled* True means that AA is enabled.

Definition at line 445 of file KDChartAbstractDiagram.cpp.

References `d`, and `KDChart::AbstractDiagram::propertiesChanged()`.

```
446 {  
447     d->antiAliasing = enabled;  
448     emit propertiesChanged();  
449 }
```

### 9.58.3.51 void AbstractDiagram::setAttributesModel ([AttributesModel](#) \* *model*) [virtual, inherited]

Associate an [AttributesModel](#) with this diagram.

Note that the diagram does *\_not\_* take ownership of the [AttributesModel](#). This should thus only be used with [AttributesModels](#) that have been explicitly created by the user, and are owned by her. Setting an [AttributesModel](#) that is internal to another diagram is an error.

Correct:

```
AttributesModel *am = new AttributesModel( model, 0 );  
diagram1->setAttributesModel( am );  
diagram2->setAttributesModel( am );
```

Wrong:

```
diagram1->setAttributesModel( diagram2->attributesModel() );
```

#### Parameters:

*model* The [AttributesModel](#) to use for this diagram.

See also:

[AttributesModel](#), [usesExternalAttributesModel](#)

Reimplemented in [KDChart::AbstractCartesianDiagram](#).

Definition at line 255 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::AbstractDiagram::modelsChanged\(\)](#), and [KDChart::AbstractDiagram::setDataBoundariesDirty\(\)](#).

Referenced by [KDChart::AbstractCartesianDiagram::setAttributesModel\(\)](#).

```

256 {
257     if( amodel->sourceModel() != model() ) {
258         qWarning("KDChart::AbstractDiagram::setAttributesModel() failed: "
259                "Trying to set an attributesmodel which works on a different "
260                "model than the diagram.");
261         return;
262     }
263     if( qobject_cast<PrivateAttributesModel*>(amodel) ) {
264         qWarning("KDChart::AbstractDiagram::setAttributesModel() failed: "
265                "Trying to set an attributesmodel that is private to another diagram.");
266         return;
267     }
268     d->setAttributesModel(amodel);
269     scheduleDelayedItemsLayout();
270     setDataBoundariesDirty();
271     emit modelsChanged();
272 }
```

### 9.58.3.52 void AbstractDiagram::setAttributesModelRootIndex (const QModelIndex &) [protected, inherited]

Definition at line 293 of file KDChartAbstractDiagram.cpp.

References [d](#), and [KDChart::AbstractDiagram::setDataBoundariesDirty\(\)](#).

Referenced by [KDChart::AbstractDiagram::setRootIndex\(\)](#).

```

294 {
295     d->attributesModelRootIndex=idx;
296     setDataBoundariesDirty();
297     scheduleDelayedItemsLayout();
298 }
```

### 9.58.3.53 void AbstractDiagram::setBrush (const QBrush & brush) [inherited]

Set the brush to be used, for painting all datasets in the model.

#### Parameters:

**brush** The brush to use.

Definition at line 806 of file KDChartAbstractDiagram.cpp.

References [KDChart::AbstractDiagram::attributesModel\(\)](#), [KDChart::DatasetBrushRole](#), [KDChart::AbstractDiagram::propertiesChanged\(\)](#), and [KDChart::AttributesModel::setModelData\(\)](#).

```

807 {
808     attributesModel()->setModelData(
809         qVariantFromValue( brush ), DatasetBrushRole );
810     emit propertiesChanged();
811 }
```

**9.58.3.54 void AbstractDiagram::setBrush (int *dataset*, const QBrush & *brush*)** [inherited]

Set the brush to be used, for painting the given dataset.

**Parameters:**

*dataset* The dataset's column in the model.

*brush* The brush to use.

Definition at line 813 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DatasetBrushRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setHeaderData().

```

814 {
815     attributesModel()->setHeaderData(
816         column, Qt::Vertical,
817         qVariantFromValue( brush ),
818         DatasetBrushRole );
819     emit propertiesChanged();
820 }
```

**9.58.3.55 void AbstractDiagram::setBrush (const QModelIndex & *index*, const QBrush & *brush*)** [inherited]

Set the brush to be used, for painting the datapoint at the given index.

**Parameters:**

*index* The datapoint's index in the model.

*brush* The brush to use.

Definition at line 798 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DatasetBrushRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setData().

```

799 {
800     attributesModel()->setData(
801         attributesModel()->mapFromSource( index ),
802         qVariantFromValue( brush ), DatasetBrushRole );
803     emit propertiesChanged();
804 }
```

**9.58.3.56 void AbstractDiagram::setCoordinatePlane ([AbstractCoordinatePlane](#) \* *plane*)** [virtual, inherited]

Set the coordinate plane associated with the diagram.

This determines how coordinates in value space are mapped into pixel space. The chart takes ownership.

**Returns:**

The coordinate plane associated with the diagram.



Reimplemented in [KDChart::AbstractCartesianDiagram](#).

Definition at line 316 of file KDChartAbstractDiagram.cpp.

References [d](#).

Referenced by [KDChart::AbstractCoordinatePlane::addDiagram\(\)](#), [KDChart::AbstractCartesianDiagram::setCoordinatePlane\(\)](#), and [KDChart::AbstractCoordinatePlane::takeDiagram\(\)](#).

```
317 {
318     d->plane = parent;
319 }
```

### 9.58.3.57 void AbstractDiagram::setDataBoundariesDirty () const [protected, inherited]

Definition at line 234 of file KDChartAbstractDiagram.cpp.

References [d](#).

Referenced by [KDChart::AbstractDiagram::dataChanged\(\)](#), [KDChart::Plotter::resize\(\)](#), [KDChart::LineDiagram::resize\(\)](#), [KDChart::BarDiagram::resize\(\)](#), [KDChart::AbstractDiagram::setAttributesModel\(\)](#), [KDChart::AbstractDiagram::setAttributesModelRootIndex\(\)](#), [KDChart::AbstractDiagram::setDatasetDimension\(\)](#), [KDChart::AbstractDiagram::setModel\(\)](#), [KDChart::BarDiagram::setThreeDBarAttributes\(\)](#), [KDChart::Plotter::setThreeDLineAttributes\(\)](#), [KDChart::LineDiagram::setThreeDLineAttributes\(\)](#), [KDChart::Plotter::setType\(\)](#), [KDChart::LineDiagram::setType\(\)](#), and [KDChart::BarDiagram::setType\(\)](#).

```
235 {
236     d->databoundariesDirty = true;
237 }
```

### 9.58.3.58 void AbstractDiagram::setDatasetDimension (int *dimension*) [inherited]

Sets the dataset dimension of the diagram.

See also:

[datasetDimension](#).

Parameters:

*dimension*

Definition at line 1077 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::AbstractDiagram::layoutChanged\(\)](#), and [KDChart::AbstractDiagram::setDataBoundariesDirty\(\)](#).

Referenced by [KDChart::Widget::setType\(\)](#), [KDChart::TernaryLineDiagram::TernaryLineDiagram\(\)](#), and [TernaryPointDiagram\(\)](#).

```
1078 {
1079     if ( d->datasetDimension == dimension ) return;
1080     d->datasetDimension = dimension;
1081     setDataBoundariesDirty();
1082     emit layoutChanged( this );
1083 }
```

### 9.58.3.59 void AbstractDiagram::setDataValueAttributes (const [DataValueAttributes](#) & *a*) [inherited]

Set the [DataValueAttributes](#) for all datapoints in the model.

#### Parameters:

*a* The attributes to set.

Definition at line 428 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::DataValueLabelAttributesRole](#), and [KDChart::AbstractDiagram::propertiesChanged\(\)](#).

```
429 {  
430     d->attributesModel->setModelData( qVariantFromValue( a ), DataValueLabelAttributesRole );  
431     emit propertiesChanged();  
432 }
```

### 9.58.3.60 void AbstractDiagram::setDataValueAttributes (int *dataset*, const [DataValueAttributes](#) & *a*) [inherited]

Set the [DataValueAttributes](#) for the given dataset.

#### Parameters:

*dataset* The dataset to set the attributes for.

*a* The attributes to set.

Definition at line 400 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::DataValueLabelAttributesRole](#), and [KDChart::AbstractDiagram::propertiesChanged\(\)](#).

```
401 {  
402     d->attributesModel->setHeaderData(  
403         column, Qt::Vertical,  
404         qVariantFromValue( a ), DataValueLabelAttributesRole );  
405     emit propertiesChanged();  
406 }
```

### 9.58.3.61 void AbstractDiagram::setDataValueAttributes (const [QModelIndex](#) & *index*, const [DataValueAttributes](#) & *a*) [inherited]

Set the [DataValueAttributes](#) for the given index.

#### Parameters:

*index* The datapoint to set the attributes for.

*a* The attributes to set.

Definition at line 389 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::DataValueLabelAttributesRole](#), and [KDChart::AbstractDiagram::propertiesChanged\(\)](#).

```

391 {
392     d->attributesModel->setData(
393         d->attributesModel->mapFromSource( index ),
394         qVariantFromValue( a ),
395         DataValueLabelAttributesRole );
396     emit propertiesChanged();
397 }

```

### 9.58.3.62 void AbstractDiagram::setHidden (bool *hidden*) [inherited]

Hide (or unhide, resp.

) all datapoints in the model.

#### Note:

Hidden data are still taken into account by the coordinate plane, so neither the grid nor your axes' ranges will change, when you hide data. For totally removing data from KD Chart's view you can use another approach: e.g. you could define a proxy model on top of your data model, and register the proxy model calling [setModel\(\)](#) instead of registering your real data model.

#### Parameters:

***hidden*** The hidden status to set.

Definition at line 359 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::AbstractDiagram::dataHidden\(\)](#), and [KDChart::DataHiddenRole](#).

```

360 {
361     d->attributesModel->setModelData(
362         qVariantFromValue( hidden ),
363         DataHiddenRole );
364     emit dataHidden();
365 }

```

### 9.58.3.63 void AbstractDiagram::setHidden (int *column*, bool *hidden*) [inherited]

Hide (or unhide, resp.

) a dataset.

#### Note:

Hidden data are still taken into account by the coordinate plane, so neither the grid nor your axes' ranges will change, when you hide data. For totally removing data from KD Chart's view you can use another approach: e.g. you could define a proxy model on top of your data model, and register the proxy model calling [setModel\(\)](#) instead of registering your real data model.

#### Parameters:

***column*** The dataset to set the hidden status for.

***hidden*** The hidden status to set.

Definition at line 350 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::AbstractDiagram::dataHidden\(\)](#), and [KDChart::DataHiddenRole](#).

```

351 {
352     d->attributesModel->setHeaderData(
353         column, Qt::Vertical,
354         qVariantFromValue( hidden ),
355         DataHiddenRole );
356     emit dataHidden();
357 }

```

### 9.58.3.64 void AbstractDiagram::setHidden (const QModelIndex & *index*, bool *hidden*) [inherited]

Hide (or unhide, resp.

) a data cell.

#### Note:

Hidden data are still taken into account by the coordinate plane, so neither the grid nor your axes' ranges will change, when you hide data. For totally removing data from KD Chart's view you can use another approach: e.g. you could define a proxy model on top of your data model, and register the proxy model calling [setModel\(\)](#) instead of registering your real data model.

#### Parameters:

***index*** The datapoint to set the hidden status for.

***hidden*** The hidden status to set.

Definition at line 341 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::AbstractDiagram::dataHidden\(\)](#), and [KDChart::DataHiddenRole](#).

```

342 {
343     d->attributesModel->setData(
344         d->attributesModel->mapFromSource( index ),
345         qVariantFromValue( hidden ),
346         DataHiddenRole );
347     emit dataHidden();
348 }

```

### 9.58.3.65 void AbstractDiagram::setModel (QAbstractItemModel \* *model*) [virtual, inherited]

Associate a model with the diagram.

Reimplemented in [KDChart::AbstractCartesianDiagram](#).

Definition at line 239 of file KDChartAbstractDiagram.cpp.

References [d](#), [KDChart::AttributesModel::initFrom\(\)](#), [KDChart::AbstractDiagram::modelsChanged\(\)](#), and [KDChart::AbstractDiagram::setDataBoundariesDirty\(\)](#).

Referenced by [KDChart::AbstractCartesianDiagram::setModel\(\)](#), and [KDChart::Widget::setType\(\)](#).

```

240 {
241     QAbstractItemView::setModel( newModel );
242     AttributesModel* amodel = new PrivateAttributesModel( newModel, this );
243     amodel->initFrom( d->attributesModel );

```

```

244     d->setAttributesModel(amodel);
245     scheduleDelayedItemsLayout();
246     setDataBoundariesDirty();
247     emit modelsChanged();
248 }

```

### 9.58.3.66 void AbstractDiagram::setPen (const QPen & *pen*) [inherited]

Set the pen to be used, for painting all datasets in the model.

#### Parameters:

*pen* The pen to use.

Definition at line 753 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::DatasetPenRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setModelData().

```

754 {
755     attributesModel()->setModelData(
756         qVariantFromValue( pen ), DatasetPenRole );
757     emit propertiesChanged();
758 }

```

### 9.58.3.67 void AbstractDiagram::setPen (int *dataset*, const QPen & *pen*) [inherited]

Set the pen to be used, for painting the given dataset.

#### Parameters:

*dataset* The dataset's row in the model.

*pen* The pen to use.

Definition at line 760 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::datasetDimension(), KDChart::DatasetPenRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setHeaderData().

```

761 {
762     if( datasetDimension() > 1 )
763         column *= datasetDimension();
764     attributesModel()->setHeaderData(
765         column, Qt::Vertical,
766         qVariantFromValue( pen ),
767         DatasetPenRole );
768     emit propertiesChanged();
769 }

```

### 9.58.3.68 void AbstractDiagram::setPen (const QModelIndex & *index*, const QPen & *pen*) [inherited]

Set the pen to be used, for painting the datapoint at the given index.

**Parameters:**

*index* The datapoint's index in the model.

*pen* The pen to use.

Definition at line 743 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::datasetDimension(), KDChart::DatasetPenRole, KDChart::AbstractDiagram::propertiesChanged(), and KDChart::AttributesModel::setData().

```

744 {
745     if( datasetDimension() > 1 )
746         return setPen( index.column(), pen );
747     attributesModel()->setData(
748         attributesModel()->mapFromSource( index ),
749         QVariantFromValue( pen ), DatasetPenRole );
750     emit propertiesChanged();
751 }
```

### 9.58.3.69 void AbstractDiagram::setPercentMode (bool *percent*) [inherited]

Definition at line 456 of file KDChartAbstractDiagram.cpp.

References d, and KDChart::AbstractDiagram::propertiesChanged().

Referenced by KDChart::LineDiagram::setType(), and KDChart::BarDiagram::setType().

```

457 {
458     d->percent = percent;
459     emit propertiesChanged();
460 }
```

### 9.58.3.70 void AbstractDiagram::setRootIndex (const QModelIndex & *idx*) [virtual, inherited]

Set the root index in the model, where the diagram starts referencing data for display.

[reimplemented]

Reimplemented in [KDChart::AbstractCartesianDiagram](#).

Definition at line 286 of file KDChartAbstractDiagram.cpp.

References d, and KDChart::AbstractDiagram::setAttributesModelRootIndex().

Referenced by KDChart::AbstractCartesianDiagram::setRootIndex().

```

287 {
288     QAbstractItemView::setRootIndex(idx);
289     setAttributesModelRootIndex( d->attributesModel->mapFromSource(idx) );
290 }
```

### 9.58.3.71 void AbstractDiagram::setSelection (const QRect & *rect*, QItemSelectionModel::SelectionFlags *command*) [virtual, inherited]

[reimplemented]

Definition at line 959 of file KDChartAbstractDiagram.cpp.

References d.

```

960 {
961     const QModelIndexList indexes = d->indexesIn( rect );
962     QItemSelection selection;
963     KDAB_FOREACH( const QModelIndex& index, indexes )
964     {
965         selection.append( QItemSelectionRange( index ) );
966     }
967     selectionModel()->select( selection, command );
968 }
```

### 9.58.3.72 void AbstractDiagram::setUnitPrefix (const QString & *prefix*, Qt::Orientation *orientation*) [inherited]

Sets the unit prefix for all values.

#### Parameters:

*prefix* the prefix to be set

*orientation* the orientation of the axis to set

Definition at line 864 of file KDChartAbstractDiagram.cpp.

References d.

```

865 {
866     d->unitPrefix[ orientation ] = prefix;
867 }
```

### 9.58.3.73 void AbstractDiagram::setUnitPrefix (const QString & *prefix*, int *column*, Qt::Orientation *orientation*) [inherited]

Sets the unit prefix for one value.

#### Parameters:

*prefix* the prefix to be set

*column* the value using that prefix

*orientation* the orientation of the axis to set

Definition at line 854 of file KDChartAbstractDiagram.cpp.

References d.

```

855 {
856     d->unitPrefixMap[ column ][ orientation ]= prefix;
857 }
```

**9.58.3.74 void AbstractDiagram::setUnitSuffix (const QString & *suffix*, Qt::Orientation *orientation*)** [inherited]

Sets the unit suffix for all values.

**Parameters:**

*suffix* the suffix to be set

*orientation* the orientation of the axis to set

Definition at line 885 of file KDChartAbstractDiagram.cpp.

References d.

```
886 {  
887     d->unitSuffix[ orientation ] = suffix;  
888 }
```

**9.58.3.75 void AbstractDiagram::setUnitSuffix (const QString & *suffix*, int *column*, Qt::Orientation *orientation*)** [inherited]

Sets the unit suffix for one value.

**Parameters:**

*suffix* the suffix to be set

*column* the value using that suffix

*orientation* the orientation of the axis to set

Definition at line 875 of file KDChartAbstractDiagram.cpp.

References d.

```
876 {  
877     d->unitSuffixMap[ column ][ orientation ]= suffix;  
878 }
```

**9.58.3.76 QString AbstractDiagram::unitPrefix (Qt::Orientation *orientation*)** const [inherited]

Returns the global unit prefix.

**Parameters:**

*orientation* the orientation of the axis

**Returns:**

the unit prefix

Definition at line 908 of file KDChartAbstractDiagram.cpp.

References d.

```
909 {  
910     return d->unitPrefix[ orientation ];  
911 }
```



**9.58.3.77 QString AbstractDiagram::unitPrefix (int *column*, Qt::Orientation *orientation*, bool *fallback* = false) const** [inherited]

Returns the unit prefix for a special value.

**Parameters:**

*column* the value which's prefix is requested

*orientation* the orientation of the axis

*fallback* if true, the global prefix is return when no specific one is set for that value

**Returns:**

the unit prefix

Definition at line 897 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::itemRowLabels(), and KDChart::CartesianAxis::paintCtx().

```
898 {  
899     if( !fallback || d->unitPrefixMap[ column ].contains( orientation ) )  
900         return d->unitPrefixMap[ column ][ orientation ];  
901     return d->unitPrefix[ orientation ];  
902 }
```

**9.58.3.78 QString AbstractDiagram::unitSuffix (Qt::Orientation *orientation*) const** [inherited]

Returns the global unit suffix.

**Parameters:**

*orientation* the orientation of the axis

**Returns:**

the unit siffix

Definition at line 931 of file KDChartAbstractDiagram.cpp.

References d.

```
932 {  
933     return d->unitSuffix[ orientation ];  
934 }
```

**9.58.3.79 QString AbstractDiagram::unitSuffix (int *column*, Qt::Orientation *orientation*, bool *fallback* = false) const** [inherited]

Returns the unit suffix for a special value.

**Parameters:**

*column* the value which's suffix is requested

*orientation* the orientation of the axis

*fallback* if true, the global suffix is return when no specific one is set for that value

#### Returns:

the unit suffix

Definition at line 920 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::itemRowLabels(), and KDChart::CartesianAxis::paintCtx().

```

921 {
922     if( !fallback || d->unitSuffixMap[ column ].contains( orientation ) )
923         return d->unitSuffixMap[ column ][ orientation ];
924     return d->unitSuffix[ orientation ];
925 }
```

#### 9.58.3.80 void AbstractDiagram::update () const [inherited]

Definition at line 1091 of file KDChartAbstractDiagram.cpp.

References d.

Referenced by KDChart::AbstractDiagram::doItemsLayout().

```

1092 {
1093     //QDebug("KDChart::AbstractDiagram::update() called");
1094     if( d->plane )
1095         d->plane->update();
1096 }
```

#### 9.58.3.81 void KDChart::AbstractDiagram::useDefaultColors () [inherited]

Set the palette to be used, for painting datasets to the default palette.

#### See also:

[KDChart::Palette](#). FIXME: fold into one usePalette ([KDChart::Palette&](#)) method

Definition at line 974 of file KDChartAbstractDiagram.cpp.

References d, and KDChart::AttributesModel::PaletteTypeDefault.

```

975 {
976     d->attributesModel->setPaletteType( AttributesModel::PaletteTypeDefault );
977 }
```

#### 9.58.3.82 void KDChart::AbstractDiagram::useRainbowColors () [inherited]

Set the palette to be used, for painting datasets to the rainbow palette.

See also:

[KDChart::Palette](#).

Definition at line 984 of file KDChartAbstractDiagram.cpp.

References [d](#), and [KDChart::AttributesModel::PaletteTypeRainbow](#).

```
985 {  
986     d->attributesModel->setPaletteType( AttributesModel::PaletteTypeRainbow );  
987 }
```

#### 9.58.3.83 bool AbstractDiagram::usesExternalAttributesModel () const [virtual, inherited]

Returns whether the diagram is using its own built-in attributes model or an attributes model that was set via [setAttributesModel](#).

See also:

[setAttributesModel](#)

Definition at line 274 of file KDChartAbstractDiagram.cpp.

References [d](#).

```
275 {  
276     return d->usesExternalAttributesModel();  
277 }
```

#### 9.58.3.84 void KDChart::AbstractDiagram::useSubduedColors () [inherited]

Set the palette to be used, for painting datasets to the subdued palette.

See also:

[KDChart::Palette](#).

Definition at line 979 of file KDChartAbstractDiagram.cpp.

References [d](#), and [KDChart::AttributesModel::PaletteTypeSubdued](#).

```
980 {  
981     d->attributesModel->setPaletteType( AttributesModel::PaletteTypeSubdued );  
982 }
```

#### 9.58.3.85 double AbstractDiagram::valueForCell (int *row*, int *column*) const [protected, inherited]

Helper method, retrieving the data value ([DisplayRole](#)) for a given row and column.

Parameters:

*row* The row to query.

*column* The column to query.

**Returns:**

The value of the display role at the given row and column as a double.

**Deprecated**

Definition at line 1085 of file KDChartAbstractDiagram.cpp.

References KDChart::AbstractDiagram::attributesModelRootIndex(), and d.

```
1086 {  
1087     return d->attributesModel->data(  
1088         d->attributesModel->index( row, column, attributesModelRootIndex() ) ).toDouble();  
1089 }
```

**9.58.3.86 int AbstractDiagram::verticalOffset () const** [virtual, inherited]

[reimplemented]

Definition at line 953 of file KDChartAbstractDiagram.cpp.

```
954 { return 0; }
```

**9.58.3.87 QRect AbstractDiagram::visualRect (const QModelIndex & *index*) const** [virtual, inherited]

[reimplemented]

Definition at line 937 of file KDChartAbstractDiagram.cpp.

```
938 {  
939     return QRect();  
940 }
```

**9.58.3.88 QRegion AbstractDiagram::visualRegionForSelection (const QItemSelection & *selection*) const** [virtual, inherited]

[reimplemented]

Definition at line 970 of file KDChartAbstractDiagram.cpp.

```
971 { return QRegion(); }
```

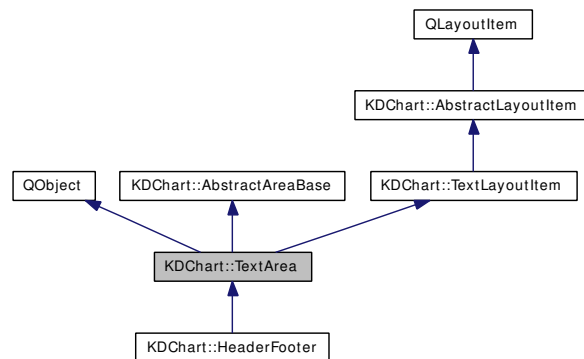
The documentation for this class was generated from the following files:

- [KDChartTernaryPointDiagram.h](#)
- [KDChartTernaryPointDiagram.cpp](#)

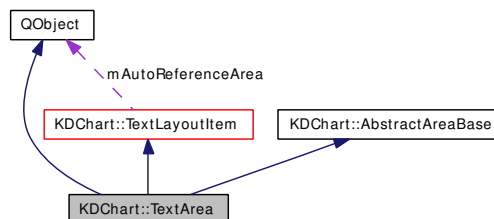
## 9.59 KDChart::TextArea Class Reference

```
#include <KDChartTextArea.h>
```

Inheritance diagram for KDChart::TextArea:



Collaboration diagram for KDChart::TextArea:



### 9.59.1 Detailed Description

A text area in the chart with a background, a frame, etc.

**TextArea** is the base class for all text containing non-widget chart elements that have a set of background attributes and frame attributes, such as headers or footers.

#### Note:

This class inherits from **AbstractAreaBase**, **TextLayoutItem**, **QObject**. The reason for this tripple inheritance is that neither **AbstractAreaBase** nor **TextLayoutItem** are **QObject**.

Definition at line 54 of file `KDChartTextArea.h`.

### Signals

- void **positionChanged** (**TextArea** \*)

### Public Member Functions

- void **alignToReferencePoint** (const **RelativePosition** &position)

- const [QObject](#) \* [autoReferenceArea](#) () const
- [BackgroundAttributes](#) [backgroundAttributes](#) () const
- bool [compare](#) (const [AbstractAreaBase](#) \*other) const  
*Returns true if both areas have the same settings.*
- virtual Qt::Orientations [expandingDirections](#) () const  
*pure virtual in [QLayoutItem](#)*
- [FrameAttributes](#) [frameAttributes](#) () const
- virtual QRect [geometry](#) () const  
*pure virtual in [QLayoutItem](#)*
- void [getFrameLeadings](#) (int &left, int &top, int &right, int &bottom) const
- virtual bool [intersects](#) (const [TextLayoutItem](#) &other, const QPoint &myPos, const QPoint &otherPos) const
- virtual bool [intersects](#) (const [TextLayoutItem](#) &other, const QPointF &myPos, const QPointF &otherPos) const
- virtual bool [isEmpty](#) () const  
*pure virtual in [QLayoutItem](#)*
- virtual QSize [maximumSize](#) () const  
*pure virtual in [QLayoutItem](#)*
- virtual QSize [minimumSize](#) () const  
*pure virtual in [QLayoutItem](#)*
- virtual void [paint](#) (QPainter \*)
- void [paintAll](#) (QPainter &painter)  
*Call paintAll, if you want the background and the frame to be drawn before the normal [paint\(\)](#) is invoked automatically.*
- virtual void [paintBackground](#) (QPainter &painter, const QRect &rectangle)
- virtual void [paintCtx](#) ([PaintContext](#) \*context)  
*Default impl: Paint the complete item using its layouted position and size.*
- virtual void [paintFrame](#) (QPainter &painter, const QRect &rectangle)
- virtual void [paintIntoRect](#) (QPainter &painter, const QRect &rect)  
*Draws the background and frame, then calls [paint\(\)](#).*
- QLayout \* [parentLayout](#) ()
- virtual QFont [realFont](#) () const
- virtual qreal [realFontSize](#) () const
- void [removeFromParentLayout](#) ()
- void [setAutoReferenceArea](#) (const [QObject](#) \*area)
- void [setBackgroundAttributes](#) (const [BackgroundAttributes](#) &a)
- void [setFrameAttributes](#) (const [FrameAttributes](#) &a)
- virtual void [setGeometry](#) (const QRect &r)  
*pure virtual in [QLayoutItem](#)*
- void [setParentLayout](#) (QLayout \*lay)

- virtual void [setParentWidget](#) ([QWidget](#) \*widget)  
*Inform the item about its widget: This enables the item, to trigger that widget's update, whenever the size of the item's contents has changed.*
- void [setText](#) (const [QString](#) &text)
- void [setTextAttributes](#) (const [TextAttributes](#) &a)  
*Use this to specify the text attributes to be used for this item.*
- virtual [QSize](#) [sizeHint](#) () const  
*pure virtual in [QLayoutItem](#)*
- virtual void [sizeHintChanged](#) () const  
*Report changed size hint: ask the parent widget to recalculate the layout.*
- [QString](#) [text](#) () const
- [TextAttributes](#) [textAttributes](#) () const  
*Returns the text attributes to be used for this item.*
- virtual [~TextArea](#) ()

## Static Public Member Functions

- static void [paintBackgroundAttributes](#) ([QPainter](#) &painter, const [QRect](#) &rectangle, const [KDChart::BackgroundAttributes](#) &attributes)
- static void [paintFrameAttributes](#) ([QPainter](#) &painter, const [QRect](#) &rectangle, const [KDChart::FrameAttributes](#) &attributes)

## Protected Member Functions

- virtual [QRect](#) [areaGeometry](#) () const
- [QRect](#) [innerRect](#) () const
- virtual void [positionHasChanged](#) ()
- [TextArea](#) ()

## Protected Attributes

- [QWidget](#) \* [mParent](#)
- [QLayout](#) \* [mParentLayout](#)

## 9.59.2 Constructor & Destructor Documentation

### 9.59.2.1 [TextArea::~TextArea](#) () [virtual]

Definition at line 60 of file [KDChartTextArea.cpp](#).

```
61 {  
62     // this bloc left empty intentionally  
63 }
```

**9.59.2.2   TextArea::TextArea ()   [protected]**

Definition at line 52 of file KDChartTextArea.cpp.

```

53      : QObject()
54      , KDChart::AbstractAreaBase()
55      , KDChart::TextLayoutItem()
56 {
57     // this bloc left empty intentionally
58 }
```

**9.59.3   Member Function Documentation****9.59.3.1   void AbstractAreaBase::alignToReferencePoint (const [RelativePosition](#) & position)**  
[inherited]

Definition at line 90 of file KDChartAbstractAreaBase.cpp.

```

91 {
92     Q_UNUSED( position );
93     // PENDING(kalle) FIXME
94     qWarning( "Sorry, not implemented: void AbstractAreaBase::alignToReferencePoint( const RelativePosi
95 }
```

**9.59.3.2   QRect TextArea::areaGeometry () const   [protected, virtual]**

Implements [KDChart::AbstractAreaBase](#).

Definition at line 105 of file KDChartTextArea.cpp.

References [KDChart::TextLayoutItem::geometry\(\)](#).

Referenced by [paintAll\(\)](#).

```

106 {
107     return geometry();
108 }
```

**9.59.3.3   const [QObject](#) \* KDChart::TextLayoutItem::autoReferenceArea () const**  
[inherited]

Definition at line 135 of file KDChartLayoutItems.cpp.

Referenced by [KDChart::HeaderFooter::compare\(\)](#), and [KDChart::HeaderFooter::setParent\(\)](#).

```

136 {
137     return mAutoReferenceArea;
138 }
```

**9.59.3.4   [BackgroundAttributes](#) AbstractAreaBase::backgroundAttributes () const**  
[inherited]

Definition at line 120 of file KDChartAbstractAreaBase.cpp.



References d.

Referenced by KDChart::AbstractAreaBase::compare(), and updateCommonBrush().

```
121 {
122     return d->backgroundAttributes;
123 }
```

### 9.59.3.5 bool AbstractAreaBase::compare (const [AbstractAreaBase](#) \* other) const [inherited]

Returns true if both areas have the same settings.

Definition at line 75 of file KDChartAbstractAreaBase.cpp.

References KDChart::AbstractAreaBase::backgroundAttributes(), and KDChart::AbstractAreaBase::frameAttributes().

```
76 {
77     if( other == this ) return true;
78     if( ! other ){
79         //qDebug() << "CartesianAxis::compare() cannot compare to Null pointer";
80         return false;
81     }
82     /*
83     qDebug() << "AbstractAreaBase:" << (frameAttributes() == other->frameAttributes())
84     << (backgroundAttributes() == other->backgroundAttributes()) << "\n";
85     */
86     return (frameAttributes() == other->frameAttributes()) &&
87            (backgroundAttributes() == other->backgroundAttributes());
88 }
```

### 9.59.3.6 Qt::Orientations KDChart::TextLayoutItem::expandingDirections () const [virtual, inherited]

pure virtual in [QLayoutItem](#)

Definition at line 179 of file KDChartLayoutItems.cpp.

```
180 {
181     return 0; // Grow neither vertically nor horizontally
182 }
```

### 9.59.3.7 [FrameAttributes](#) AbstractAreaBase::frameAttributes () const [inherited]

Definition at line 106 of file KDChartAbstractAreaBase.cpp.

References d.

Referenced by KDChart::Legend::clone(), KDChart::AbstractAreaBase::compare(), and updateCommonBrush().

```
107 {
108     return d->frameAttributes;
109 }
```

**9.59.3.8 QRect KDChart::TextLayoutItem::geometry () const** [virtual, inherited]

pure virtual in [QLayoutItem](#)

Definition at line 184 of file KDChartLayoutItems.cpp.

Referenced by [areaGeometry\(\)](#), [KDChart::TextLayoutItem::paint\(\)](#), [paintAll\(\)](#), [KDChart::Cartesian-Axis::paintCtx\(\)](#), and [paintIntoRect\(\)](#).

```
185 {
186     return mRect;
187 }
```

**9.59.3.9 void AbstractAreaBase::getFrameLeadings (int & left, int & top, int & right, int & bottom) const** [inherited]

Definition at line 212 of file KDChartAbstractAreaBase.cpp.

References [d](#).

Referenced by [KDChart::AbstractAreaBase::innerRect\(\)](#), and [KDChart::AbstractAreaWidget::paintAll\(\)](#).

```
213 {
214     if( d && d->frameAttributes.isVisible() ){
215         const int padding = qMax( d->frameAttributes.padding(), 0 );
216         left    = padding;
217         top     = padding;
218         right   = padding;
219         bottom  = padding;
220     }else{
221         left    = 0;
222         top     = 0;
223         right   = 0;
224         bottom  = 0;
225     }
226 }
```

**9.59.3.10 QRect AbstractAreaBase::innerRect () const** [protected, inherited]

Definition at line 228 of file KDChartAbstractAreaBase.cpp.

References [KDChart::AbstractAreaBase::areaGeometry\(\)](#), and [KDChart::AbstractAreaBase::getFrameLeadings\(\)](#).

Referenced by [paintAll\(\)](#), and [KDChart::AbstractArea::paintAll\(\)](#).

```
229 {
230     int left;
231     int top;
232     int right;
233     int bottom;
234     getFrameLeadings( left, top, right, bottom );
235     return
236         QRect( QPoint(0,0), areaGeometry().size() )
237         .adjusted( left, top, -right, -bottom );
238 }
```

### 9.59.3.11 bool KDChart::TextLayoutItem::intersects (const [TextLayoutItem](#) & *other*, const QPoint & *myPos*, const QPoint & *otherPos*) const [virtual, inherited]

Definition at line 258 of file KDChartLayoutItems.cpp.

References KDChart::TextLayoutItem::mAttributes, PI, KDChart::TextLayoutItem::rotatedCorners(), KDChart::TextAttributes::rotation(), and KDChart::TextLayoutItem::unrotatedSizeHint().

```

259 {
260     if ( mAttributes.rotation() != other.mAttributes.rotation() )
261     {
262         // that's the code for the common case: the rotation angles don't need to match here
263         QPolygon myPolygon( rotatedCorners() );
264         QPolygon otherPolygon( other.rotatedCorners() );
265
266         // move the polygons to their positions
267         myPolygon.translate( myPos );
268         otherPolygon.translate( otherPos );
269
270         // create regions out of it
271         QRegion myRegion( myPolygon );
272         QRegion otherRegion( otherPolygon );
273
274         // now the question - do they intersect or not?
275         return ! myRegion.intersect( otherRegion ).isEmpty();
276     }
277     else {
278         // and that's the code for the special case: the rotation angles match, which is less time consuming
279         const qreal angle = mAttributes.rotation() * PI / 180.0;
280         // both sizes
281         const QSizeF mySize( unrotatedSizeHint() );
282         const QSizeF otherSize( other.unrotatedSizeHint() );
283
284         // that's myP1 relative to myPos
285         QPointF myP1( mySize.height() * sin( angle ), 0.0 );
286         // that's otherP1 to myPos
287         QPointF otherP1 = QPointF( otherSize.height() * sin( angle ), 0.0 ) + otherPos - myPos;
288
289         // now rotate both points the negative angle around myPos
290         myP1 = QPointF( myP1.x() * cos( -angle ), myP1.x() * sin( -angle ) );
291         qreal r = sqrt( otherP1.x() * otherP1.x() + otherP1.y() * otherP1.y() );
292         otherP1 = QPointF( r * cos( -angle ), r * sin( -angle ) );
293
294         // finally we look, whether both rectangles intersect or even not
295         return QRectF( myP1, mySize ).intersects( QRectF( otherP1, otherSize ) );
296     }
297 }

```

### 9.59.3.12 bool KDChart::TextLayoutItem::intersects (const [TextLayoutItem](#) & *other*, const QPointF & *myPos*, const QPointF & *otherPos*) const [virtual, inherited]

Definition at line 253 of file KDChartLayoutItems.cpp.

Referenced by KDChart::CartesianAxis::paintCtx().

```

254 {
255     return intersects( other, myPos.toPoint(), otherPos.toPoint() );
256 }

```

**9.59.3.13 bool KDChart::TextLayoutItem::isEmpty () const** [virtual, inherited]

pure virtual in [QLayoutItem](#)

Definition at line 189 of file KDChartLayoutItems.cpp.

```
190 {
191     return false; // never empty, otherwise the layout item would not exist
192 }
```

**9.59.3.14 QSize KDChart::TextLayoutItem::maximumSize () const** [virtual, inherited]

pure virtual in [QLayoutItem](#)

Definition at line 194 of file KDChartLayoutItems.cpp.

References KDChart::TextLayoutItem::sizeHint().

```
195 {
196     return sizeHint(); // PENDING(kalle) Review, quite inflexible
197 }
```

**9.59.3.15 QSize KDChart::TextLayoutItem::minimumSize () const** [virtual, inherited]

pure virtual in [QLayoutItem](#)

Definition at line 199 of file KDChartLayoutItems.cpp.

References KDChart::TextLayoutItem::sizeHint().

```
200 {
201     return sizeHint(); // PENDING(kalle) Review, quite inflexible
202 }
```

**9.59.3.16 void KDChart::TextLayoutItem::paint (QPainter \*)** [virtual, inherited]

Implements [KDChart::AbstractLayoutItem](#).

Definition at line 386 of file KDChartLayoutItems.cpp.

References KDChart::TextLayoutItem::geometry(), KDChart::TextAttributes::pen(), rotatedRect(), and KDChart::TextAttributes::rotation().

Referenced by paintAll(), and KDChart::CartesianAxis::paintCtx().

```
387 {
388     // make sure, cached font is updated, if needed:
389     // sizeHint();
390
391     if( !mRect.isValid() )
392         return;
393
394     PainterSaver painterSaver( painter );
395     painter->setFont( cachedFont );
396     QRectF rect( geometry() );
397
398     // #ifdef DEBUG_ITEMS_PAINT
```

```

399 //      painter->setPen( Qt::black );
400 //      painter->drawRect( rect );
401 // #endif
402      painter->translate( rect.center() );
403      rect.moveTopLeft( QPointF( - rect.width() / 2, - rect.height() / 2 ) );
404 #ifdef DEBUG_ITEMS_PAINT
405      painter->setPen( Qt::blue );
406      painter->drawRect( rect );
407 #endif
408      painter->rotate( mAttributes.rotation() );
409      rect = rotatedRect( rect, mAttributes.rotation() );
410 #ifdef DEBUG_ITEMS_PAINT
411      painter->setPen( Qt::red );
412      painter->drawRect( rect );
413 #endif
414      painter->setPen( mAttributes.pen() );
415      painter->drawText( rect, Qt::AlignHCenter | Qt::AlignVCenter, mText );
416 //      if ( calcSizeHint( cachedFont ).width() > rect.width() )
417 //          qDebug() << "rect.width()" << rect.width() << "text.width()" << calcSizeHint( cachedFont ).width();
418 //
419 //      //painter->drawText( rect, Qt::AlignHCenter | Qt::AlignVCenter, mText );
420 }

```

### 9.59.3.17 void TextArea::paintAll (QPainter &painter) [virtual]

Call `paintAll`, if you want the background and the frame to be drawn before the normal `paint()` is invoked automatically.

Reimplemented from [KDChart::AbstractLayoutItem](#).

Definition at line 83 of file `KDChartTextArea.cpp`.

References `areaGeometry()`, `KDChart::TextLayoutItem::geometry()`, `KDChart::AbstractAreaBase::innerRect()`, `KDChart::TextLayoutItem::paint()`, `KDChart::AbstractAreaBase::paintBackground()`, `KDChart::AbstractAreaBase::paintFrame()`, and `KDChart::TextLayoutItem::setGeometry()`.

Referenced by `paintIntoRect()`.

```

84 {
85     // Paint the background and frame
86     paintBackground( painter, geometry() );
87     paintFrame(      painter, geometry() );
88
89     // temporarily adjust the widget size, to be sure all content gets calculated
90     // to fit into the inner rectangle
91     const QRect oldGeometry( areaGeometry() );
92     QRect inner( innerRect() );
93     inner.moveTo(
94         oldGeometry.left() + inner.left(),
95         oldGeometry.top()  + inner.top() );
96     const bool needAdjustGeometry = oldGeometry != inner;
97     if( needAdjustGeometry )
98         setGeometry( inner );
99     paint( &painter );
100     if( needAdjustGeometry )
101         setGeometry( oldGeometry );
102     //qDebug() << "TextAreaWidget::paintAll() done.";
103 }

```

### 9.59.3.18 void AbstractAreaBase::paintBackground (QPainter & *painter*, const QRect & *rectangle*) [virtual, inherited]

Definition at line 196 of file KDChartAbstractAreaBase.cpp.

References `d`, and `KDChart::AbstractAreaBase::paintBackgroundAttributes()`.

Referenced by `paintAll()`, `KDChart::AbstractAreaWidget::paintAll()`, and `KDChart::AbstractArea::paintAll()`.

```

197 {
198     Q_ASSERT_X ( d != 0, "AbstractAreaBase::paintBackground()",
199                 "Private class was not initialized!" );
200     paintBackgroundAttributes( painter, rect, d->backgroundAttributes );
201 }
```

### 9.59.3.19 void AbstractAreaBase::paintBackgroundAttributes (QPainter & *painter*, const QRect & *rectangle*, const [KDChart::BackgroundAttributes](#) & *attributes*) [static, inherited]

Definition at line 127 of file KDChartAbstractAreaBase.cpp.

References `KDChart::BackgroundAttributes::BackgroundPixmapModeCentered`, `KDChart::BackgroundAttributes::BackgroundPixmapModeNone`, `KDChart::BackgroundAttributes::BackgroundPixmapModeScaled`, `KDChart::BackgroundAttributes::BackgroundPixmapModeStretched`, `KDChart::BackgroundAttributes::brush()`, `KDChart::BackgroundAttributes::isVisible()`, `KDChart::BackgroundAttributes::pixmap()`, and `KDChart::BackgroundAttributes::pixmapMode()`.

Referenced by `KDChart::AbstractAreaBase::paintBackground()`.

```

129 {
130     if( !attributes.isVisible() ) return;
131
132     /* first draw the brush (may contain a pixmap)*/
133     if( Qt::NoBrush != attributes.brush().style() ) {
134         KDChart::PainterSaver painterSaver( &painter );
135         painter.setPen( Qt::NoPen );
136         const QPointF newTopLeft( painter.deviceMatrix().map( rect.topLeft() ) );
137         painter.setBrushOrigin( newTopLeft );
138         painter.setBrush( attributes.brush() );
139         painter.drawRect( rect.adjusted( 0, 0, -1, -1 ) );
140     }
141     /* next draw the backPixmap over the brush */
142     if( !attributes.pixmap().isNull() &&
143         attributes.pixmapMode() != BackgroundAttributes::BackgroundPixmapModeNone ) {
144         QPointF ol = rect.topLeft();
145         if( BackgroundAttributes::BackgroundPixmapModeCentered == attributes.pixmapMode() )
146         {
147             ol.setX( rect.center().x() - attributes.pixmap().width() / 2 );
148             ol.setY( rect.center().y() - attributes.pixmap().height() / 2 );
149             painter.drawPixmap( ol, attributes.pixmap() );
150         } else {
151             QMatrix m;
152             double zW = (double)rect.width() / (double)attributes.pixmap().width();
153             double zH = (double)rect.height() / (double)attributes.pixmap().height();
154             switch( attributes.pixmapMode() ) {
155             case BackgroundAttributes::BackgroundPixmapModeScaled:
156                 {
157                     double z;
158                     z = qMin( zW, zH );
159                     m.scale( z, z );
160                 }
```

```

161         break;
162         case BackgroundAttributes::BackgroundPixmapModeStretched:
163             m.scale( zW, zH );
164             break;
165         default:
166             ; // Cannot happen, previously checked
167     }
168     QPixmap pm = attributes.pixmap().transformed( m );
169     ol.setX( rect.center().x() - pm.width() / 2 );
170     ol.setY( rect.center().y() - pm.height() / 2 );
171     painter.drawPixmap( ol, pm );
172 }
173 }
174 }

```

### 9.59.3.20 void KDChart::AbstractLayoutItem::paintCtx ([PaintContext](#) \* *context*) [virtual, inherited]

Default impl: Paint the complete item using its layouted position and size.

Reimplemented in [KDChart::CartesianAxis](#), and [KDChart::TernaryAxis](#).

Definition at line 77 of file KDChartLayoutItems.cpp.

References [KDChart::AbstractLayoutItem::paint\(\)](#), and [KDChart::PaintContext::painter\(\)](#).

```

78 {
79     if( context )
80         paint( context->painter() );
81 }

```

### 9.59.3.21 void AbstractAreaBase::paintFrame (QPainter & *painter*, const QRect & *rectangle*) [virtual, inherited]

Definition at line 204 of file KDChartAbstractAreaBase.cpp.

References [d](#), and [KDChart::AbstractAreaBase::paintFrameAttributes\(\)](#).

Referenced by [paintAll\(\)](#), [KDChart::AbstractAreaWidget::paintAll\(\)](#), and [KDChart::AbstractArea::paintAll\(\)](#).

```

205 {
206     Q_ASSERT_X ( d != 0, "AbstractAreaBase::paintFrame()",
207                 "Private class was not initialized!" );
208     paintFrameAttributes( painter, rect, d->frameAttributes );
209 }

```

### 9.59.3.22 void AbstractAreaBase::paintFrameAttributes (QPainter & *painter*, const QRect & *rectangle*, const [KDChart::FrameAttributes](#) & *attributes*) [static, inherited]

Definition at line 177 of file KDChartAbstractAreaBase.cpp.

References [KDChart::FrameAttributes::isVisible\(\)](#), and [KDChart::FrameAttributes::pen\(\)](#).

Referenced by [KDChart::AbstractAreaBase::paintFrame\(\)](#).

```

179 {
180
181     if( !attributes.isVisible() ) return;
182
183     // Note: We set the brush to NoBrush explicitly here.
184     //         Otherwise we might get a filled rectangle, so any
185     //         previously drawn background would be overwritten by that area.
186
187     const QPen    oldPen( painter.pen() );
188     const QBrush  oldBrush( painter.brush() );
189     painter.setPen( attributes.pen() );
190     painter.setBrush( Qt::NoBrush );
191     painter.drawRect( rect.adjusted( 0, 0, -1, -1 ) );
192     painter.setBrush( oldBrush );
193     painter.setPen( oldPen );
194 }

```

### 9.59.3.23 void TextArea::paintIntoRect (QPainter &painter, const QRect &rect) [virtual]

Draws the background and frame, then calls [paint\(\)](#).

In most cases there is no need to overwrite this method in a derived class, but you would overwrite [TextLayoutItem::paint\(\)](#) instead.

Definition at line 71 of file KDChartTextArea.cpp.

References [KDChart::TextLayoutItem::geometry\(\)](#), [paintAll\(\)](#), and [KDChart::TextLayoutItem::setGeometry\(\)](#).

```

72 {
73     const QRect oldGeometry( geometry() );
74     if( oldGeometry != rect )
75         setGeometry( rect );
76     painter.translate( rect.left(), rect.top() );
77     paintAll( painter );
78     painter.translate( -rect.left(), -rect.top() );
79     if( oldGeometry != rect )
80         setGeometry( oldGeometry );
81 }

```

### 9.59.3.24 QLayout\* KDChart::AbstractLayoutItem::parentLayout () [inherited]

Definition at line 76 of file KDChartLayoutItems.h.

```

77     {
78         return mParentLayout;
79     }

```

### 9.59.3.25 void KDChart::TextArea::positionChanged (TextArea \*) [signal]

Referenced by [positionHasChanged\(\)](#).

### 9.59.3.26 void TextArea::positionHasChanged () [protected, virtual]

Reimplemented from [KDChart::AbstractAreaBase](#).



Definition at line 110 of file KDChartTextArea.cpp.

References `positionChanged()`.

```
111 {  
112     emit positionChanged( this );  
113 }
```

### 9.59.3.27 QFont KDChart::TextLayoutItem::realFont () const [virtual, inherited]

Definition at line 230 of file KDChartLayoutItems.cpp.

Referenced by `KDChart::CartesianAxis::maximumSize()`, and `KDChart::CartesianAxis::paintCtx()`.

```
231 {  
232     realFontWasRecalculated(); // we can safely ignore the boolean return value  
233     return cachedFont;  
234 }
```

### 9.59.3.28 qreal KDChart::TextLayoutItem::realFontSize () const [virtual, inherited]

Definition at line 210 of file KDChartLayoutItems.cpp.

References `KDChart::TextAttributes::calculatedFontSize()`.

```
211 {  
212     return mAttributes.calculatedFontSize( mAutoReferenceArea, mAutoReferenceOrientation );  
213 }
```

### 9.59.3.29 void KDChart::AbstractLayoutItem::removeFromParentLayout () [inherited]

Definition at line 80 of file KDChartLayoutItems.h.

Referenced by `KDChart::Chart::takeCoordinatePlane()`.

```
81     {  
82         if( mParentLayout ){  
83             if( widget() )  
84                 mParentLayout->removeWidget( widget() );  
85             else  
86                 mParentLayout->removeItem( this );  
87         }  
88     }
```

### 9.59.3.30 void KDChart::TextLayoutItem::setAutoReferenceArea (const [QObject](#) \* area) [inherited]

Definition at line 128 of file KDChartLayoutItems.cpp.

References `KDChart::TextLayoutItem::sizeHint()`.

Referenced by `KDChart::HeaderFooter::setParent()`.

```

129 {
130     mAutoReferenceArea = area;
131     cachedSizeHint = QSize();
132     sizeHint();
133 }

```

### 9.59.3.31 void AbstractAreaBase::setBackgroundAttributes (const [BackgroundAttributes](#) & a) [inherited]

Definition at line 111 of file KDChartAbstractAreaBase.cpp.

References [d](#), and [KDChart::AbstractAreaBase::positionHasChanged\(\)](#).

```

112 {
113     if( d->backgroundAttributes == a )
114         return;
115
116     d->backgroundAttributes = a;
117     positionHasChanged();
118 }

```

### 9.59.3.32 void AbstractAreaBase::setFrameAttributes (const [FrameAttributes](#) & a) [inherited]

Definition at line 97 of file KDChartAbstractAreaBase.cpp.

References [d](#), and [KDChart::AbstractAreaBase::positionHasChanged\(\)](#).

Referenced by [KDChart::Legend::clone\(\)](#).

```

98 {
99     if( d->frameAttributes == a )
100         return;
101
102     d->frameAttributes = a;
103     positionHasChanged();
104 }

```

### 9.59.3.33 void KDChart::TextLayoutItem::setGeometry (const QRect & r) [virtual, inherited]

pure virtual in [QLayoutItem](#)

Definition at line 204 of file KDChartLayoutItems.cpp.

Referenced by [paintAll\(\)](#), [KDChart::CartesianAxis::paintCtx\(\)](#), and [paintIntoRect\(\)](#).

```

205 {
206     mRect = r;
207 }

```

### 9.59.3.34 void KDChart::AbstractLayoutItem::setParentLayout (QLayout \* lay) [inherited]

Definition at line 72 of file KDChartLayoutItems.h.

```
73     {
74         mParentLayout = lay;
75     }
```

**9.59.3.35 void KDChart::AbstractLayoutItem::setParentWidget (QWidget \* widget)**  
[virtual, inherited]

Inform the item about its widget: This enables the item, to trigger that widget's update, whenever the size of the item's contents has changed.

Thus, you need to call setParentWidget on every item, that has a non-fixed size.

Definition at line 64 of file KDChartLayoutItems.cpp.

References KDChart::AbstractLayoutItem::mParent.

Referenced by KDChart::HeaderFooter::setParent(), and KDChart::AbstractCartesianDiagram::takeAxis().

```
65 {
66     mParent = widget;
67 }
```

**9.59.3.36 void KDChart::TextLayoutItem::setText (const QString & text)** [inherited]

Definition at line 140 of file KDChartLayoutItems.cpp.

References KDChart::AbstractLayoutItem::mParent, and KDChart::TextLayoutItem::sizeHint().

Referenced by KDChart::Widget::addHeaderFooter(), KDChart::CartesianAxis::maximumSize(), and KDChart::CartesianAxis::paintCtx().

```
141 {
142     mText = text;
143     cachedSizeHint = QSize();
144     sizeHint();
145     if ( mParent )
146         mParent->update();
147 }
```

**9.59.3.37 void KDChart::TextLayoutItem::setTextAttributes (const TextAttributes & a)**  
[inherited]

Use this to specify the text attributes to be used for this item.

See also:

[textAttributes](#)

Definition at line 159 of file KDChartLayoutItems.cpp.

References KDChart::AbstractLayoutItem::mParent, and KDChart::TextLayoutItem::sizeHint().

Referenced by KDChart::HeaderFooter::clone().

```

160 {
161     mAttributes = a;
162     cachedSizeHint = QSize(); // invalidate size hint
163     sizeHint();
164     if( mParent )
165         mParent->update();
166 }

```

### 9.59.3.38 QSize KDChart::TextLayoutItem::sizeHint () const [virtual, inherited]

pure virtual in [QLayoutItem](#)

Definition at line 299 of file KDChartLayoutItems.cpp.

References KDChart::AbstractLayoutItem::sizeHintChanged().

Referenced by KDChart::TextLayoutItem::maximumSize(), KDChart::CartesianAxis::maximumSize(), KDChart::TextLayoutItem::minimumSize(), KDChart::CartesianAxis::paintCtx(), KDChart::TextLayoutItem::setAutoReferenceArea(), KDChart::TextLayoutItem::setText(), and KDChart::TextLayoutItem::setTextAttributes().

```

300 {
301     if( realFontWasRecalculated() )
302     {
303         const QSize newSizeHint( calcSizeHint( cachedFont ) );
304         if( newSizeHint != cachedSizeHint ){
305             cachedSizeHint = newSizeHint;
306             sizeHintChanged();
307         }
308     }
309     //qDebug() << "----- KDChart::TextLayoutItem::sizeHint() returns:"<<cachedSizeHint<<" -----
310     return cachedSizeHint;
311 }

```

### 9.59.3.39 void KDChart::AbstractLayoutItem::sizeHintChanged () const [virtual, inherited]

Report changed size hint: ask the parent widget to recalculate the layout.

Definition at line 86 of file KDChartLayoutItems.cpp.

References KDChart::AbstractLayoutItem::mParent.

Referenced by KDChart::TextLayoutItem::sizeHint().

```

87 {
88     // This is exactly like what QWidget::updateGeometry does.
89     // qDebug("KDChart::AbstractLayoutItem::sizeHintChanged() called");
90     if( mParent ) {
91         if ( mParent->layout() )
92             mParent->layout()->invalidate();
93         else
94             QApplication::postEvent( mParent, new QEvent( QEvent::LayoutRequest ) );
95     }
96 }

```

### 9.59.3.40 QString KDChart::TextLayoutItem::text () const [inherited]

Definition at line 149 of file KDChartLayoutItems.cpp.

Referenced by KDChart::HeaderFooter::compare(), and KDChart::CartesianAxis::paintCtx().

```
150 {  
151     return mText;  
152 }
```

#### 9.59.3.41 [KDChart::TextAttributes](#) KDChart::TextLayoutItem::textAttributes () const [inherited]

Returns the text attributes to be used for this item.

See also:

[setTextAttributes](#)

Definition at line 173 of file KDChartLayoutItems.cpp.

Referenced by KDChart::HeaderFooter::clone(), and KDChart::HeaderFooter::compare().

```
174 {  
175     return mAttributes;  
176 }
```

### 9.59.4 Member Data Documentation

#### 9.59.4.1 [QWidget\\*](#) [KDChart::AbstractLayoutItem::mParent](#) [protected, inherited]

Definition at line 90 of file KDChartLayoutItems.h.

Referenced by KDChart::AbstractLayoutItem::setParentWidget(), KDChart::TextLayoutItem::setText(), KDChart::TextLayoutItem::setTextAttributes(), and KDChart::AbstractLayoutItem::sizeHintChanged().

#### 9.59.4.2 [QLayout\\*](#) [KDChart::AbstractLayoutItem::mParentLayout](#) [protected, inherited]

Definition at line 91 of file KDChartLayoutItems.h.

Referenced by KDChart::AutoSpacerLayoutItem::paint().

The documentation for this class was generated from the following files:

- [KDChartTextArea.h](#)
- [KDChartTextArea.cpp](#)

## 9.60 KDChart::TextAttributes Class Reference

```
#include <KDChartTextAttributes.h>
```

### 9.60.1 Detailed Description

A set of text attributes.

[TextAttributes](#) encapsulates settings that have to do with text. This includes font, fontsize, color, whether the text is rotated, etc

Definition at line 50 of file `KDChartTextAttributes.h`.

### Public Member Functions

- bool [autoRotate](#) () const  
**Returns:**  
*Whether text is automatically rotated when space is constrained.*
- bool [autoShrink](#) () const  
**Returns:**  
*Whether text is automatically shrunk if space is constraint.*
- const QFont [calculatedFont](#) (const [QObject](#) \*autoReferenceArea, [KDChartEnums::MeasureOrientation](#) autoReferenceOrientation) const  
*Returns the font in the size that is used at drawing time.*
- const qreal [calculatedFontSize](#) (const [QObject](#) \*autoReferenceArea, [KDChartEnums::MeasureOrientation](#) autoReferenceOrientation) const  
*Returns the font size that is used at drawing time.*
- QFont [font](#) () const  
**Returns:**  
*The font that is used for rendering text.*
- [Measure](#) [fontSize](#) () const  
**Returns:**  
*The measure used for the font size.*
- bool [hasAbsoluteFontSize](#) () const  
**Returns:**  
*Whether the text has an absolute font size set.*
- bool [isVisible](#) () const  
**Returns:**  
*Whether the text is visible.*
- [Measure](#) [minimalFontSize](#) () const  
**Returns:**  
*The measure used for the minimal font size.*

- bool `operator!=` (const [TextAttributes](#) &other) const
- [TextAttributes](#) & `operator=` (const [TextAttributes](#) &)
- bool `operator==` (const [TextAttributes](#) &) const
- QPen `pen` () const

**Returns:**

*The pen used for rendering the text.*

- int `rotation` () const

**Returns:**

*The rotation angle used for rendering the text.*

- void `setAutoRotate` (bool autoRotate)

*Set whether the text should be automatically rotated as needed when space is constraint.*

- void `setAutoShrink` (bool autoShrink)

*Set whether the text should automatically be shrunk, if space is constraint.*

- void `setFont` (const QFont &font)

*Set the font to be used for rendering the text.*

- void `setFontSize` (const [Measure](#) &measure)

*Set the size of the font used for rendering text.*

- void `setMinimalFontSize` (const [Measure](#) &measure)

*Set the minimal size of the font used for rendering text.*

- void `setPen` (const QPen &pen)

*Set the pen to use for rendering the text.*

- void `setRotation` (int rotation)

*Set the rotation angle to use for the text.*

- void `setVisible` (bool visible)

*Set whether the text is to be rendered at all.*

- [TextAttributes](#) (const [TextAttributes](#) &)
- [TextAttributes](#) ()
- [~TextAttributes](#) ()

## 9.60.2 Constructor & Destructor Documentation

### 9.60.2.1 TextAttributes::TextAttributes ()

Definition at line 62 of file KDChartTextAttributes.cpp.

References `setAutoRotate()`, `setAutoShrink()`, `setFont()`, `setPen()`, `setRotation()`, and `setVisible()`.

```
63      : _d( new Private() )
64  {
65      setVisible( true );
66      setFont( QApplication::font() );
67      setAutoRotate( false );
68      setAutoShrink( false );
69      setRotation( 0 );
70      setPen( QPen( Qt::black ) );
71  }
```

#### 9.60.2.2 TextAttributes::TextAttributes (const [TextAttributes](#) &)

Definition at line 73 of file KDChartTextAttributes.cpp.

```
74      : _d( new Private( *r.d ) )
75  {
76
77  }
```

#### 9.60.2.3 TextAttributes::~TextAttributes ()

Definition at line 89 of file KDChartTextAttributes.cpp.

```
90  {
91      delete _d; _d = 0;
92  }
```

### 9.60.3 Member Function Documentation

#### 9.60.3.1 bool TextAttributes::autoRotate () const

##### Returns:

Whether text is automatically rotated when space is constrained.

Definition at line 197 of file KDChartTextAttributes.cpp.

References [d](#).

Referenced by [operator<<\(\)](#), and [operator==\(\)](#).

```
198  {
199      return d->autoRotate;
200  }
```

#### 9.60.3.2 bool TextAttributes::autoShrink () const

##### Returns:

Whether text is automatically shrunk if space is constraint.

Definition at line 207 of file KDChartTextAttributes.cpp.

References [d](#).

Referenced by [operator<<\(\)](#), and [operator==\(\)](#).



```

208 {
209     return d->autoShrink;
210 }

```

### 9.60.3.3 const QFont TextAttributes::calculatedFont (const QObject \* *autoReferenceArea*, KDChartEnums::MeasureOrientation *autoReferenceOrientation*) const

Returns the font in the size that is used at drawing time.

This method is called at drawing time. It returns the font as it is used for rendering text, taking into account any measures that were set via setFontSize and/or setMinimalFontSize.

Definition at line 178 of file KDChartTextAttributes.cpp.

References calculatedFontSize(), and d.

```

181 {
182     const qreal size = calculatedFontSize( autoReferenceArea, autoReferenceOrientation );
183     //qDebug() << "TextAttributes::calculatedFont() has d->cachedFontSize" << d->cachedFontSize << "
184     if( size > 0.0 && d->cachedFontSize != size ){
185         d->cachedFontSize = size;
186         d->cachedFont.setPointSizeF( d->cachedFontSize );
187     }
188     return d->cachedFont;
189 }

```

### 9.60.3.4 const qreal TextAttributes::calculatedFontSize (const QObject \* *autoReferenceArea*, KDChartEnums::MeasureOrientation *autoReferenceOrientation*) const

Returns the font size that is used at drawing time.

This method is called at drawing time. It returns the font size as it is used for rendering text, taking into account any measures that were set via setFontSize and/or setMinimalFontSize.

Definition at line 167 of file KDChartTextAttributes.cpp.

References KDChart::Measure::calculatedValue(), fontSize(), and minimalFontSize().

Referenced by calculatedFont(), and KDChart::TextLayoutItem::realFontSize().

```

170 {
171     const qreal normalSize = fontSize().calculatedValue( autoReferenceArea, autoReferenceOrientation );
172     const qreal minimalSize = minimalFontSize().calculatedValue( autoReferenceArea, autoReferenceOrientation );
173     //qDebug() << "TextAttributes::calculatedFontSize() finds" << normalSize << "and" << minimalSize;
174     return qMax( normalSize, minimalSize );
175 }

```

### 9.60.3.5 QFont TextAttributes::font () const

#### Returns:

The font that is used for rendering text.

Definition at line 135 of file KDChartTextAttributes.cpp.

References d.

Referenced by operator<<(), and operator==().

```
136 {  
137     return d->font;  
138 }
```

#### 9.60.3.6 [Measure](#) TextAttributes::fontSize () const

##### Returns:

The measure used for the font size.

Definition at line 145 of file KDChartTextAttributes.cpp.

References [d](#).

Referenced by [calculatedFontSize\(\)](#), [operator<<\(\)](#), [operator==\(\)](#), and [KDChart::CartesianAxis::titleTextAttributes\(\)](#).

```
146 {  
147     return d->fontSize;  
148 }
```

#### 9.60.3.7 [bool](#) TextAttributes::hasAbsoluteFontSize () const

##### Returns:

Whether the text has an absolute font size set.

Definition at line 160 of file KDChartTextAttributes.cpp.

References [d](#), and [KDChartEnums::MeasureCalculationModeAbsolute](#).

```
161 {  
162     return d->fontSize.calculationMode() == KDChartEnums::MeasureCalculationModeAbsolute  
163         && d->minimalFontSize.calculationMode() == KDChartEnums::MeasureCalculationModeAbsolute;  
164 }
```

#### 9.60.3.8 [bool](#) TextAttributes::isVisible () const

##### Returns:

Whether the text is visible.

Definition at line 123 of file KDChartTextAttributes.cpp.

References [d](#).

Referenced by [KDChart::CartesianAxis::maximumSize\(\)](#), [operator<<\(\)](#), [operator==\(\)](#), and [KDChart::CartesianAxis::paintCtx\(\)](#).

```
124 {  
125     return d->visible;  
126 }
```

**9.60.3.9 Measure** TextAttributes::minimalFontSize () const**Returns:**

The measure used for the minimal font size.

Definition at line 155 of file KDChartTextAttributes.cpp.

References d.

Referenced by calculatedFontSize(), operator<<(), and operator==().

```
156 {
157     return d->minimalFontSize;
158 }
```

**9.60.3.10 bool** KDChart::TextAttributes::operator!= (const TextAttributes & other) const

Definition at line 57 of file KDChartTextAttributes.h.

```
58 { return !operator==(other); }
```

**9.60.3.11 TextAttributes & TextAttributes::operator=** (const TextAttributes &)

Definition at line 79 of file KDChartTextAttributes.cpp.

References d.

```
80 {
81     if( this == &r )
82         return *this;
83
84     *d = *r.d;
85
86     return *this;
87 }
```

**9.60.3.12 bool** TextAttributes::operator== (const TextAttributes &) const

Definition at line 95 of file KDChartTextAttributes.cpp.

References autoRotate(), autoShrink(), font(), fontSize(), isVisible(), minimalFontSize(), pen(), and rotation().

```
96 {
97     /*
98     qDebug() << "\n" << "TextAttributes::operator== :" << ( isVisible() == r.isVisible())
99         << (font() == r.font())
100         << (fontSize() == r.fontSize())
101         << (minimalFontSize() == r.minimalFontSize())
102         << (autoRotate() == r.autoRotate())
103         << (autoShrink() == r.autoShrink())
104         << (rotation() == rotation())
105         << (pen() == r.pen());
106     */
107     return ( isVisible() == r.isVisible() &&
```

```
108         font() == r.font() &&
109         fontSize() == r.fontSize() &&
110         minimalFontSize() == r.minimalFontSize() &&
111         autoRotate() == r.autoRotate() &&
112         autoShrink() == r.autoShrink() &&
113         rotation() == r.rotation() &&
114         pen() == r.pen() );
115 }
```

### 9.60.3.13 QPen TextAttributes::pen () const

#### Returns:

The pen used for rendering the text.

Definition at line 227 of file KDChartTextAttributes.cpp.

References d.

Referenced by operator<<(), operator==(), KDChart::TextLayoutItem::paint(), and KDChart::Cartesian-Axis::paintCtx().

```
228 {
229     return d->pen;
230 }
```

### 9.60.3.14 int TextAttributes::rotation () const

#### Returns:

The rotation angle used for rendering the text.

Definition at line 217 of file KDChartTextAttributes.cpp.

References d.

Referenced by KDChart::TextLayoutItem::intersects(), operator<<(), operator==(), and KDChart::TextLayoutItem::paint().

```
218 {
219     return d->rotation;
220 }
```

### 9.60.3.15 void TextAttributes::setAutoRotate (bool *autoRotate*)

Set whether the text should be automatically rotated as needed when space is constraint.

#### Parameters:

*autoRotate* Whether text should be automatically rotated.

Definition at line 192 of file KDChartTextAttributes.cpp.

References d.

Referenced by TextAttributes().

```
193 {  
194     d->autoRotate = autoRotate;  
195 }
```

#### 9.60.3.16 void TextAttributes::setAutoShrink (bool *autoShrink*)

Set whether the text should automatically be shrunk, if space is constraint.

##### Parameters:

*autoShrink* Whether text should be auto-shrunk.

Definition at line 202 of file KDChartTextAttributes.cpp.

References [d](#).

Referenced by [TextAttributes\(\)](#).

```
203 {  
204     d->autoShrink = autoShrink;  
205 }
```

#### 9.60.3.17 void TextAttributes::setFont (const QFont & *font*)

Set the font to be used for rendering the text.

##### Parameters:

*font* The font to use.

Definition at line 128 of file KDChartTextAttributes.cpp.

References [d](#).

Referenced by [TextAttributes\(\)](#).

```
129 {  
130     d->font = font;  
131     d->cachedFont = font; // note: we do not set the font's size here, but in calculatedFont()  
132     d->cachedFontSize = -1.0;  
133 }
```

#### 9.60.3.18 void TextAttributes::setFontSize (const [Measure](#) & *measure*)

Set the size of the font used for rendering text.

##### Parameters:

*measure* The measure to use.

##### See also:

[Measure](#)

Definition at line 140 of file KDChartTextAttributes.cpp.

References d.

Referenced by KDChart::Chart::addLegend(), and KDChart::CartesianAxis::titleTextAttributes().

```
141 {  
142     d->fontSize = measure;  
143 }
```

#### 9.60.3.19 void TextAttributes::setMinimalFontSize (const [Measure](#) & *measure*)

Set the minimal size of the font used for rendering text.

##### Parameters:

*measure* The measure to use.

##### See also:

[Measure](#)

Definition at line 150 of file KDChartTextAttributes.cpp.

References d.

```
151 {  
152     d->minimalFontSize = measure;  
153 }
```

#### 9.60.3.20 void TextAttributes::setPen (const QPen & *pen*)

Set the pen to use for rendering the text.

##### Parameters:

*pen* The pen to use.

Definition at line 222 of file KDChartTextAttributes.cpp.

References d.

Referenced by TextAttributes().

```
223 {  
224     d->pen = pen;  
225 }
```

#### 9.60.3.21 void TextAttributes::setRotation (int *rotation*)

Set the rotation angle to use for the text.

##### Note:

For axis titles the rotation angle can be set to one of the following angles: 0, 90, 180, 270 Any other values specified will be replaced by the next smaller one of the allowed values, so no matter what you set the rotation will always be one of these four values.

**Parameters:**

*rotation* The rotation angle.

Definition at line 212 of file KDChartTextAttributes.cpp.

References d.

Referenced by TextAttributes().

```
213 {  
214     d->rotation = rotation;  
215 }
```

**9.60.3.22 void TextAttributes::setVisible (bool *visible*)**

Set whether the text is to be rendered at all.

**Parameters:**

*visible* Whether the text is visible.

Definition at line 118 of file KDChartTextAttributes.cpp.

References d.

Referenced by TextAttributes().

```
119 {  
120     d->visible = visible;  
121 }
```

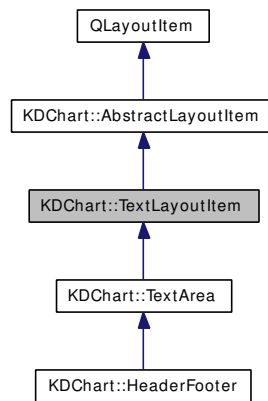
The documentation for this class was generated from the following files:

- [KDChartTextAttributes.h](#)
- [KDChartTextAttributes.cpp](#)

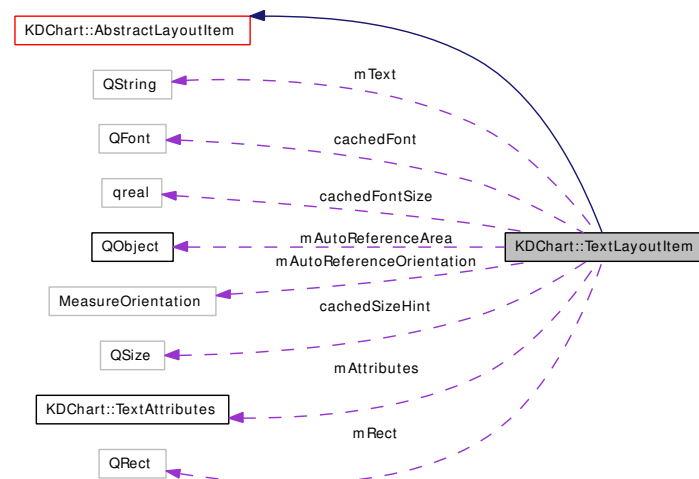
## 9.61 KDChart::TextLayoutItem Class Reference

```
#include <KDChartLayoutItems.h>
```

Inheritance diagram for KDChart::TextLayoutItem:



Collaboration diagram for KDChart::TextLayoutItem:



### 9.61.1 Detailed Description

Layout item showing a text.

Definition at line 98 of file `KDChartLayoutItems.h`.

### Public Member Functions

- const `QObject` \* `autoReferenceArea` () const
- virtual Qt::Orientations `expandingDirections` () const  
*pure virtual in `QLayoutItem`*



- virtual QRect [geometry](#) () const  
*pure virtual in [QLayoutItem](#)*
- virtual bool [intersects](#) (const [TextLayoutItem](#) &other, const QPoint &myPos, const QPoint &otherPos) const
- virtual bool [intersects](#) (const [TextLayoutItem](#) &other, const QPointF &myPos, const QPointF &otherPos) const
- virtual bool [isEmpty](#) () const  
*pure virtual in [QLayoutItem](#)*
- virtual QSize [maximumSize](#) () const  
*pure virtual in [QLayoutItem](#)*
- virtual QSize [minimumSize](#) () const  
*pure virtual in [QLayoutItem](#)*
- virtual void [paint](#) (QPainter \*)
- virtual void [paintAll](#) (QPainter &painter)  
*Default impl: just call paint.*
- virtual void [paintCtx](#) ([PaintContext](#) \*context)  
*Default impl: Paint the complete item using its layouted position and size.*
- QLayout \* [parentLayout](#) ()
- virtual QFont [realFont](#) () const
- virtual qreal [realFontSize](#) () const
- void [removeFromParentLayout](#) ()
- void [setAutoReferenceArea](#) (const [QObject](#) \*area)
- virtual void [setGeometry](#) (const QRect &r)  
*pure virtual in [QLayoutItem](#)*
- void [setParentLayout](#) (QLayout \*lay)
- virtual void [setParentWidget](#) ([QWidget](#) \*widget)  
*Inform the item about its widget: This enables the item, to trigger that widget's update, whenever the size of the item's contents has changed.*
- void [setText](#) (const QString &text)
- void [setTextAttributes](#) (const [TextAttributes](#) &a)  
*Use this to specify the text attributes to be used for this item.*
- virtual QSize [sizeHint](#) () const  
*pure virtual in [QLayoutItem](#)*
- virtual void [sizeHintChanged](#) () const  
*Report changed size hint: ask the parent widget to recalculate the layout.*
- QString [text](#) () const
- [TextAttributes](#) [textAttributes](#) () const  
*Returns the text attributes to be used for this item.*

- [TextLayoutItem](#) (const QString &text, const [TextAttributes](#) &attributes, const [QObject](#) \*autoReferenceArea, [KDChartEnums::MeasureOrientation](#) autoReferenceOrientation, Qt::Alignment alignment=0)
- [TextLayoutItem](#) ()

## Protected Attributes

- [QWidget](#) \* mParent
- [QLayout](#) \* mParentLayout

## 9.61.2 Constructor & Destructor Documentation

### 9.61.2.1 KDChart::TextLayoutItem::TextLayoutItem ()

Definition at line 115 of file KDChartLayoutItems.cpp.

```

116      : AbstractLayoutItem( Qt::AlignLeft )
117      , mText ()
118      , mAttributes ()
119      , mAutoReferenceArea ( 0 )
120      , mAutoReferenceOrientation ( KDChartEnums::MeasureOrientationHorizontal )
121      , cachedSizeHint () // default this to invalid to force just-in-time calculation before first use c
122      , cachedFontSize ( 0.0 )
123      , cachedFont ( mAttributes.font () )
124  {
125
126  }
```

### 9.61.2.2 KDChart::TextLayoutItem::TextLayoutItem (const QString & text, const [TextAttributes](#) & attributes, const [QObject](#) \* autoReferenceArea, [KDChartEnums::MeasureOrientation](#) autoReferenceOrientation, Qt::Alignment alignment = 0)

Definition at line 99 of file KDChartLayoutItems.cpp.

```

104      : AbstractLayoutItem( alignment )
105      , mText ( text )
106      , mAttributes ( attributes )
107      , mAutoReferenceArea ( area )
108      , mAutoReferenceOrientation ( orientation )
109      , cachedSizeHint () // default this to invalid to force just-in-time calculation before first use c
110      , cachedFontSize ( 0.0 )
111      , cachedFont ( mAttributes.font () )
112  {
113  }
```

## 9.61.3 Member Function Documentation

### 9.61.3.1 const [QObject](#) \* KDChart::TextLayoutItem::autoReferenceArea () const

Definition at line 135 of file KDChartLayoutItems.cpp.

Referenced by KDChart::HeaderFooter::compare(), and KDChart::HeaderFooter::setParent().

```

136 {
137     return mAutoReferenceArea;
138 }

```

### 9.61.3.2 Qt::Orientations KDChart::TextLayoutItem::expandingDirections () const [virtual]

pure virtual in [QLayoutItem](#)

Definition at line 179 of file KDChartLayoutItems.cpp.

```

180 {
181     return 0; // Grow neither vertically nor horizontally
182 }

```

### 9.61.3.3 QRect KDChart::TextLayoutItem::geometry () const [virtual]

pure virtual in [QLayoutItem](#)

Definition at line 184 of file KDChartLayoutItems.cpp.

Referenced by KDChart::TextArea::areaGeometry(), paint(), KDChart::TextArea::paintAll(), KDChart::CartesianAxis::paintCtx(), and KDChart::TextArea::paintIntoRect().

```

185 {
186     return mRect;
187 }

```

### 9.61.3.4 bool KDChart::TextLayoutItem::intersects (const [TextLayoutItem](#) & other, const QPoint & myPos, const QPoint & otherPos) const [virtual]

Definition at line 258 of file KDChartLayoutItems.cpp.

References mAttributes, PI, rotatedCorners(), KDChart::TextAttributes::rotation(), and unrotatedSizeHint().

```

259 {
260     if ( mAttributes.rotation() != other.mAttributes.rotation() )
261     {
262         // that's the code for the common case: the rotation angles don't need to match here
263         QPolygon myPolygon( rotatedCorners() );
264         QPolygon otherPolygon( other.rotatedCorners() );
265
266         // move the polygons to their positions
267         myPolygon.translate( myPos );
268         otherPolygon.translate( otherPos );
269
270         // create regions out of it
271         QRegion myRegion( myPolygon );
272         QRegion otherRegion( otherPolygon );
273
274         // now the question - do they intersect or not?
275         return ! myRegion.intersect( otherRegion ).isEmpty();
276
277     } else {
278         // and that's the code for the special case: the rotation angles match, which is less time con
279         const qreal angle = mAttributes.rotation() * PI / 180.0;
280         // both sizes

```

```

281         const QSizeF mySize(          unrotatedSizeHint() );
282         const QSizeF otherSize( other.unrotatedSizeHint() );
283
284         // that's myPl relative to myPos
285         QPointF myPl( mySize.height() * sin( angle ), 0.0 );
286         // that's otherPl to myPos
287         QPointF otherPl = QPointF( otherSize.height() * sin( angle ), 0.0 ) + otherPos - myPos;
288
289         // now rotate both points the negative angle around myPos
290         myPl = QPointF( myPl.x() * cos( -angle ), myPl.x() * sin( -angle ) );
291         qreal r = sqrt( otherPl.x() * otherPl.x() + otherPl.y() * otherPl.y() );
292         otherPl = QPointF( r * cos( -angle ), r * sin( -angle ) );
293
294         // finally we look, whether both rectangles intersect or even not
295         return QRectF( myPl, mySize ).intersects( QRectF( otherPl, otherSize ) );
296     }
297 }

```

#### 9.61.3.5 bool KDChart::TextLayoutItem::intersects (const [TextLayoutItem](#) & *other*, const QPointF & *myPos*, const QPointF & *otherPos*) const [virtual]

Definition at line 253 of file KDChartLayoutItems.cpp.

Referenced by KDChart::CartesianAxis::paintCtx().

```

254 {
255     return intersects( other, myPos.toPoint(), otherPos.toPoint() );
256 }

```

#### 9.61.3.6 bool KDChart::TextLayoutItem::isEmpty () const [virtual]

pure virtual in [QLayoutItem](#)

Definition at line 189 of file KDChartLayoutItems.cpp.

```

190 {
191     return false; // never empty, otherwise the layout item would not exist
192 }

```

#### 9.61.3.7 QSize KDChart::TextLayoutItem::maximumSize () const [virtual]

pure virtual in [QLayoutItem](#)

Definition at line 194 of file KDChartLayoutItems.cpp.

References [sizeHint\(\)](#).

```

195 {
196     return sizeHint(); // PENDING(kalle) Review, quite inflexible
197 }

```

#### 9.61.3.8 QSize KDChart::TextLayoutItem::minimumSize () const [virtual]

pure virtual in [QLayoutItem](#)

Definition at line 199 of file KDChartLayoutItems.cpp.

References [sizeHint\(\)](#).

```
200 {
201     return sizeHint(); // PENDING(kalle) Review, quite inflexible
202 }
```

### 9.61.3.9 void KDChart::TextLayoutItem::paint (QPainter \*) [virtual]

Implements [KDChart::AbstractLayoutItem](#).

Definition at line 386 of file KDChartLayoutItems.cpp.

References [geometry\(\)](#), [KDChart::TextAttributes::pen\(\)](#), [rotatedRect\(\)](#), and [KDChart::TextAttributes::rotation\(\)](#).

Referenced by [KDChart::TextArea::paintAll\(\)](#), and [KDChart::CartesianAxis::paintCtx\(\)](#).

```
387 {
388     // make sure, cached font is updated, if needed:
389     // sizeHint();
390
391     if( !mRect.isValid() )
392         return;
393
394     PainterSaver painterSaver( painter );
395     painter->setFont( cachedFont );
396     QRectF rect( geometry() );
397
398     // #ifdef DEBUG_ITEMS_PAINT
399     //     painter->setPen( Qt::black );
400     //     painter->drawRect( rect );
401     // #endif
402     painter->translate( rect.center() );
403     rect.moveTopLeft( QPointF( - rect.width() / 2, - rect.height() / 2 ) );
404     #ifdef DEBUG_ITEMS_PAINT
405     painter->setPen( Qt::blue );
406     painter->drawRect( rect );
407     #endif
408     painter->rotate( mAttributes.rotation() );
409     rect = rotatedRect( rect, mAttributes.rotation() );
410     #ifdef DEBUG_ITEMS_PAINT
411     painter->setPen( Qt::red );
412     painter->drawRect( rect );
413     #endif
414     painter->setPen( mAttributes.pen() );
415     painter->drawText( rect, Qt::AlignHCenter | Qt::AlignVCenter, mText );
416     // if ( calcSizeHint( cachedFont ).width() > rect.width() )
417     //     qDebug() << "rect.width()" << rect.width() << "text.width()" << calcSizeHint( cachedFont ).width();
418     //
419     // //painter->drawText( rect, Qt::AlignHCenter | Qt::AlignVCenter, mText );
420 }
```

### 9.61.3.10 void KDChart::AbstractLayoutItem::paintAll (QPainter & painter) [virtual, inherited]

Default impl: just call paint.

Derived classes like [KDChart::AbstractArea](#) are providing additional action here.

Reimplemented in [KDChart::AbstractArea](#), [KDChart::TextArea](#), and [KDChart::TernaryAxis](#).

Definition at line 69 of file [KDChartLayoutItems.cpp](#).

References [KDChart::AbstractLayoutItem::paint\(\)](#).

```
70 {
71     paint( &painter );
72 }
```

#### 9.61.3.11 void KDChart::AbstractLayoutItem::paintCtx ([PaintContext](#) \* context) [virtual, inherited]

Default impl: Paint the complete item using its layouted position and size.

Reimplemented in [KDChart::CartesianAxis](#), and [KDChart::TernaryAxis](#).

Definition at line 77 of file [KDChartLayoutItems.cpp](#).

References [KDChart::AbstractLayoutItem::paint\(\)](#), and [KDChart::PaintContext::painter\(\)](#).

```
78 {
79     if( context )
80         paint( context->painter() );
81 }
```

#### 9.61.3.12 QLayout\* KDChart::AbstractLayoutItem::parentLayout () [inherited]

Definition at line 76 of file [KDChartLayoutItems.h](#).

```
77     {
78         return mParentLayout;
79     }
```

#### 9.61.3.13 QFont KDChart::TextLayoutItem::realFont () const [virtual]

Definition at line 230 of file [KDChartLayoutItems.cpp](#).

Referenced by [KDChart::CartesianAxis::maximumSize\(\)](#), and [KDChart::CartesianAxis::paintCtx\(\)](#).

```
231 {
232     realFontWasRecalculated(); // we can safely ignore the boolean return value
233     return cachedFont;
234 }
```

#### 9.61.3.14 qreal KDChart::TextLayoutItem::realFontSize () const [virtual]

Definition at line 210 of file [KDChartLayoutItems.cpp](#).

References [KDChart::TextAttributes::calculatedFontSize\(\)](#).

```
211 {
212     return mAttributes.calculatedFontSize( mAutoReferenceArea, mAutoReferenceOrientation );
213 }
```

**9.61.3.15 void KDChart::AbstractLayoutItem::removeFromParentLayout ()** [inherited]

Definition at line 80 of file KDChartLayoutItems.h.

Referenced by KDChart::Chart::takeCoordinatePlane().

```
81     {
82         if( mParentLayout ){
83             if( widget() )
84                 mParentLayout->removeWidget( widget() );
85             else
86                 mParentLayout->removeItem( this );
87         }
88     }
```

**9.61.3.16 void KDChart::TextLayoutItem::setAutoReferenceArea (const [QObject](#) \* area)**

Definition at line 128 of file KDChartLayoutItems.cpp.

References [sizeHint\(\)](#).

Referenced by KDChart::HeaderFooter::setParent().

```
129 {
130     mAutoReferenceArea = area;
131     cachedSizeHint = QSize();
132     sizeHint();
133 }
```

**9.61.3.17 void KDChart::TextLayoutItem::setGeometry (const [QRect](#) & r)** [virtual]

pure virtual in [QLayoutItem](#)

Definition at line 204 of file KDChartLayoutItems.cpp.

Referenced by KDChart::TextArea::paintAll(), KDChart::CartesianAxis::paintCtx(), and KDChart::TextArea::paintIntoRect().

```
205 {
206     mRect = r;
207 }
```

**9.61.3.18 void KDChart::AbstractLayoutItem::setParentLayout ([QLayout](#) \* lay)** [inherited]

Definition at line 72 of file KDChartLayoutItems.h.

```
73     {
74         mParentLayout = lay;
75     }
```

### 9.61.3.19 void KDChart::AbstractLayoutItem::setParentWidget (QWidget \* widget) [virtual, inherited]

Inform the item about its widget: This enables the item, to trigger that widget's update, whenever the size of the item's contents has changed.

Thus, you need to call setParentWidget on every item, that has a non-fixed size.

Definition at line 64 of file KDChartLayoutItems.cpp.

References KDChart::AbstractLayoutItem::mParent.

Referenced by KDChart::HeaderFooter::setParent(), and KDChart::AbstractCartesianDiagram::takeAxis().

```
65 {  
66     mParent = widget;  
67 }
```

### 9.61.3.20 void KDChart::TextLayoutItem::setText (const QString & text)

Definition at line 140 of file KDChartLayoutItems.cpp.

References KDChart::AbstractLayoutItem::mParent, and sizeHint().

Referenced by KDChart::Widget::addHeaderFooter(), KDChart::CartesianAxis::maximumSize(), and KDChart::CartesianAxis::paintCtx().

```
141 {  
142     mText = text;  
143     cachedSizeHint = QSize();  
144     sizeHint();  
145     if( mParent )  
146         mParent->update();  
147 }
```

### 9.61.3.21 void KDChart::TextLayoutItem::setTextAttributes (const TextAttributes & a)

Use this to specify the text attributes to be used for this item.

See also:

[textAttributes](#)

Definition at line 159 of file KDChartLayoutItems.cpp.

References KDChart::AbstractLayoutItem::mParent, and sizeHint().

Referenced by KDChart::HeaderFooter::clone().

```
160 {  
161     mAttributes = a;  
162     cachedSizeHint = QSize(); // invalidate size hint  
163     sizeHint();  
164     if( mParent )  
165         mParent->update();  
166 }
```



**9.61.3.22 QSize KDChart::TextLayoutItem::sizeHint () const** [virtual]

pure virtual in [QLayoutItem](#)

Definition at line 299 of file KDChartLayoutItems.cpp.

References KDChart::AbstractLayoutItem::sizeHintChanged().

Referenced by [maximumSize\(\)](#), [KDChart::CartesianAxis::maximumSize\(\)](#), [minimumSize\(\)](#), [KDChart::CartesianAxis::paintCtx\(\)](#), [setAutoReferenceArea\(\)](#), [setText\(\)](#), and [setTextAttributes\(\)](#).

```

300 {
301     if( realFontWasRecalculated() )
302     {
303         const QSize newSizeHint( calcSizeHint( cachedFont ) );
304         if( newSizeHint != cachedSizeHint ){
305             cachedSizeHint = newSizeHint;
306             sizeHintChanged();
307         }
308     }
309     //qDebug() << "----- KDChart::TextLayoutItem::sizeHint() returns:<<cachedSizeHint<<" -----
310     return cachedSizeHint;
311 }
```

**9.61.3.23 void KDChart::AbstractLayoutItem::sizeHintChanged () const** [virtual, inherited]

Report changed size hint: ask the parent widget to recalculate the layout.

Definition at line 86 of file KDChartLayoutItems.cpp.

References KDChart::AbstractLayoutItem::mParent.

Referenced by [sizeHint\(\)](#).

```

87 {
88     // This is exactly like what QWidget::updateGeometry does.
89     // qDebug("KDChart::AbstractLayoutItem::sizeHintChanged() called");
90     if( mParent ) {
91         if ( mParent->layout() )
92             mParent->layout()->invalidate();
93         else
94             QApplication::postEvent( mParent, new QEvent( QEvent::LayoutRequest ) );
95     }
96 }
```

**9.61.3.24 QString KDChart::TextLayoutItem::text () const**

Definition at line 149 of file KDChartLayoutItems.cpp.

Referenced by [KDChart::HeaderFooter::compare\(\)](#), and [KDChart::CartesianAxis::paintCtx\(\)](#).

```

150 {
151     return mText;
152 }
```

### 9.61.3.25 [KDChart::TextAttributes](#) KDChart::TextLayoutItem::textAttributes () const

Returns the text attributes to be used for this item.

See also:

[setTextAttributes](#)

Definition at line 173 of file KDChartLayoutItems.cpp.

Referenced by KDChart::HeaderFooter::clone(), and KDChart::HeaderFooter::compare().

```
174 {  
175     return mAttributes;  
176 }
```

## 9.61.4 Member Data Documentation

### 9.61.4.1 [QWidget\\*](#) [KDChart::AbstractLayoutItem::mParent](#) [protected, inherited]

Definition at line 90 of file KDChartLayoutItems.h.

Referenced by KDChart::AbstractLayoutItem::setParentWidget(), setText(), setTextAttributes(), and KDChart::AbstractLayoutItem::sizeHintChanged().

### 9.61.4.2 [QLayout\\*](#) [KDChart::AbstractLayoutItem::mParentLayout](#) [protected, inherited]

Definition at line 91 of file KDChartLayoutItems.h.

Referenced by KDChart::AutoSpacerLayoutItem::paint().

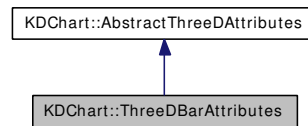
The documentation for this class was generated from the following files:

- [KDChartLayoutItems.h](#)
- [KDChartLayoutItems.cpp](#)

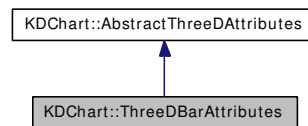
## 9.62 KDChart::ThreeDBarAttributes Class Reference

```
#include <KDChartThreeDBarAttributes.h>
```

Inheritance diagram for KDChart::ThreeDBarAttributes:



Collaboration diagram for KDChart::ThreeDBarAttributes:



### 9.62.1 Detailed Description

A set of 3D bar attributes.

Definition at line 38 of file KDChartThreeDBarAttributes.h.

### Public Member Functions

- uint [angle](#) () const
- double [depth](#) () const
- bool [isEnabled](#) () const
- bool [operator!=](#) (const [AbstractThreeDAttributes](#) &other) const
- bool [operator!=](#) (const [ThreeDBarAttributes](#) &other) const
- [ThreeDBarAttributes](#) & [operator=](#) (const [ThreeDBarAttributes](#) &)
- bool [operator==](#) (const [AbstractThreeDAttributes](#) &) const
- bool [operator==](#) (const [ThreeDBarAttributes](#) &) const
- void [setAngle](#) (uint threeDAngle)
- void [setDepth](#) (double depth)
- void [setEnabled](#) (bool enabled)
- void [setUseShadowColors](#) (bool useShadowColors)
- [ThreeDBarAttributes](#) (const [ThreeDBarAttributes](#) &)
- [ThreeDBarAttributes](#) ()
- bool [useShadowColors](#) () const
- double [validDepth](#) () const
- [~ThreeDBarAttributes](#) ()

## 9.62.2 Constructor & Destructor Documentation

### 9.62.2.1 ThreeDBarAttributes::ThreeDBarAttributes ()

Definition at line 44 of file KDChartThreeDBarAttributes.cpp.

```
45      : AbstractThreeDAttributes( new Private() )
46 {
47
48 }
```

### 9.62.2.2 ThreeDBarAttributes::ThreeDBarAttributes (const [ThreeDBarAttributes](#) &)

Definition at line 50 of file KDChartThreeDBarAttributes.cpp.

```
51      : AbstractThreeDAttributes( new Private( *r.d) )
52 {
53 }
```

### 9.62.2.3 ThreeDBarAttributes::~~ThreeDBarAttributes ()

Definition at line 65 of file KDChartThreeDBarAttributes.cpp.

```
66 {
67 }
```

## 9.62.3 Member Function Documentation

### 9.62.3.1 uint ThreeDBarAttributes::angle () const

Definition at line 98 of file KDChartThreeDBarAttributes.cpp.

References [d](#).

Referenced by [operator<<\(\)](#), and [operator==\(\)](#).

```
99 {
100     return d->angle;
101 }
```

### 9.62.3.2 double AbstractThreeDAttributes::depth () const [inherited]

Definition at line 103 of file KDChartAbstractThreeDAttributes.cpp.

References [d](#).

Referenced by [operator<<\(\)](#), [KDChart::AbstractThreeDAttributes::operator==\(\)](#), and [KDChart::Pie-Diagram::paint\(\)](#).

```
104 {
105     return d->depth;
106 }
```

**9.62.3.3 bool AbstractThreeDAttributes::isEnabled () const** [inherited]

Definition at line 92 of file KDChartAbstractThreeDAttributes.cpp.

References [d](#).

Referenced by [operator<<\(\)](#), [KDChart::AbstractThreeDAttributes::operator==\(\(\)\)](#), [KDChart::PieDiagram::paint\(\)](#), and [KDChart::AbstractThreeDAttributes::validDepth\(\)](#).

```
93 {
94     return d->enabled;
95 }
```

**9.62.3.4 bool KDChart::AbstractThreeDAttributes::operator!= (const AbstractThreeDAttributes & other) const** [inherited]

Definition at line 60 of file KDChartAbstractThreeDAttributes.h.

```
60 { return !operator==(other); }
```

**9.62.3.5 bool KDChart::ThreeDBarAttributes::operator!= (const ThreeDBarAttributes & other) const**

Definition at line 56 of file KDChartThreeDBarAttributes.h.

```
56 { return !operator==(other); }
```

**9.62.3.6 ThreeDBarAttributes & ThreeDBarAttributes::operator= (const ThreeDBarAttributes &)**

Definition at line 55 of file KDChartThreeDBarAttributes.cpp.

References [d](#).

```
56 {
57     if( this == &r )
58         return *this;
59
60     *d = *r.d;
61
62     return *this;
63 }
```

**9.62.3.7 bool AbstractThreeDAttributes::operator== (const AbstractThreeDAttributes &) const** [inherited]

Definition at line 72 of file KDChartAbstractThreeDAttributes.cpp.

References [KDChart::AbstractThreeDAttributes::depth\(\)](#), and [KDChart::AbstractThreeDAttributes::isEnabled\(\)](#).

Referenced by [KDChart::ThreeDPieAttributes::operator==\(\(\)\)](#), [KDChart::ThreeDLineAttributes::operator==\(\(\)\)](#), and [operator==\(\(\)\)](#).

```
73 {
74     if( isEnabled() == r.isEnabled() &&
75         depth() == r.depth() )
76         return true;
77     else
78         return false;
79 }
```

#### 9.62.3.8 bool ThreeDBarAttributes::operator==(const ThreeDBarAttributes &) const

Definition at line 74 of file KDChartThreeDBarAttributes.cpp.

References [angle\(\)](#), [KDChart::AbstractThreeDAttributes::operator==\(\(\)\)](#), and [useShadowColors\(\)](#).

```
75 {
76     return ( useShadowColors() == r.useShadowColors() &&
77             angle() == r.angle() &&
78             AbstractThreeDAttributes::operator==(r) );
79 }
```

#### 9.62.3.9 void ThreeDBarAttributes::setAngle (uint *threeDAngle*)

Definition at line 93 of file KDChartThreeDBarAttributes.cpp.

References [d](#).

```
94 {
95     d->angle = threeDAngle;
96 }
```

#### 9.62.3.10 void AbstractThreeDAttributes::setDepth (double *depth*) [inherited]

Definition at line 97 of file KDChartAbstractThreeDAttributes.cpp.

References [d](#).

```
98 {
99     d->depth = depth;
100 }
```

#### 9.62.3.11 void AbstractThreeDAttributes::setEnabled (bool *enabled*) [inherited]

Definition at line 87 of file KDChartAbstractThreeDAttributes.cpp.

References [d](#).

```
88 {
89     d->enabled = enabled;
90 }
```

**9.62.3.12 void ThreeDBarAttributes::setUseShadowColors (bool *useShadowColors*)**

Definition at line 83 of file KDChartThreeDBarAttributes.cpp.

References [d](#).

```
84 {  
85     d->useShadowColors = shadowColors;  
86 }
```

**9.62.3.13 bool ThreeDBarAttributes::useShadowColors () const**

Definition at line 88 of file KDChartThreeDBarAttributes.cpp.

References [d](#).

Referenced by [operator<<\(\)](#), and [operator==\(\)](#).

```
89 {  
90     return d->useShadowColors;  
91 }
```

**9.62.3.14 double AbstractThreeDAttributes::validDepth () const** [inherited]

Definition at line 109 of file KDChartAbstractThreeDAttributes.cpp.

References [d](#), and [KDChart::AbstractThreeDAttributes::isEnabled\(\)](#).

Referenced by [KDChart::Plotter::threeDItemDepth\(\)](#), [KDChart::LineDiagram::threeDItemDepth\(\)](#), and [KDChart::BarDiagram::threeDItemDepth\(\)](#).

```
110 {  
111     return isEnabled() ? d->depth : 0.0;  
112 }
```

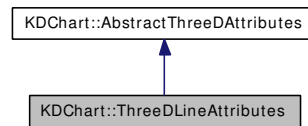
The documentation for this class was generated from the following files:

- [KDChartThreeDBarAttributes.h](#)
- [KDChartThreeDBarAttributes.cpp](#)

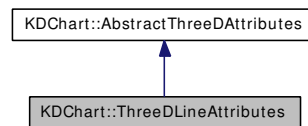
## 9.63 KDChart::ThreeDLineAttributes Class Reference

```
#include <KDChartThreeDLineAttributes.h>
```

Inheritance diagram for KDChart::ThreeDLineAttributes:



Collaboration diagram for KDChart::ThreeDLineAttributes:



### 9.63.1 Detailed Description

A set of 3D line attributes.

Definition at line 38 of file KDChartThreeDLineAttributes.h.

### Public Member Functions

- double [depth](#) () const
- bool [isEnabled](#) () const
- uint [lineXRotation](#) () const
- uint [lineYRotation](#) () const
- bool [operator!=](#) (const [AbstractThreeDAttributes](#) &other) const
- bool [operator!=](#) (const [ThreeDLineAttributes](#) &other) const
- [ThreeDLineAttributes](#) & [operator=](#) (const [ThreeDLineAttributes](#) &)
- bool [operator==](#) (const [AbstractThreeDAttributes](#) &) const
- bool [operator==](#) (const [ThreeDLineAttributes](#) &) const
- void [setDepth](#) (double depth)
- void [setEnabled](#) (bool enabled)
- void [setLineXRotation](#) (const uint degrees)
- void [setLineYRotation](#) (const uint degrees)
- [ThreeDLineAttributes](#) (const [ThreeDLineAttributes](#) &)
- [ThreeDLineAttributes](#) ()
- double [validDepth](#) () const
- [~ThreeDLineAttributes](#) ()



## 9.63.2 Constructor & Destructor Documentation

### 9.63.2.1 ThreeDLineAttributes::ThreeDLineAttributes ()

Definition at line 44 of file KDChartThreeDLineAttributes.cpp.

```
45      : AbstractThreeDAttributes( new Private() )
46  {
47
48  }
```

### 9.63.2.2 ThreeDLineAttributes::ThreeDLineAttributes (const [ThreeDLineAttributes](#) &)

Definition at line 50 of file KDChartThreeDLineAttributes.cpp.

```
51      : AbstractThreeDAttributes( new Private( *r.d) )
52  {
53  }
```

### 9.63.2.3 ThreeDLineAttributes::~~ThreeDLineAttributes ()

Definition at line 65 of file KDChartThreeDLineAttributes.cpp.

```
66  {
67  }
```

## 9.63.3 Member Function Documentation

### 9.63.3.1 double AbstractThreeDAttributes::depth () const [inherited]

Definition at line 103 of file KDChartAbstractThreeDAttributes.cpp.

References [d](#).

Referenced by [operator<<\(\)](#), [KDChart::AbstractThreeDAttributes::operator==\(\(\)\)](#), and [KDChart::Pie-Diagram::paint\(\)](#).

```
104 {
105     return d->depth;
106 }
```

### 9.63.3.2 bool AbstractThreeDAttributes::isEnabled () const [inherited]

Definition at line 92 of file KDChartAbstractThreeDAttributes.cpp.

References [d](#).

Referenced by [operator<<\(\)](#), [KDChart::AbstractThreeDAttributes::operator==\(\(\)\)](#), [KDChart::Pie-Diagram::paint\(\)](#), and [KDChart::AbstractThreeDAttributes::validDepth\(\)](#).

```
93 {
94     return d->enabled;
95 }
```

**9.63.3.3 uint ThreeDLineAttributes::lineXRotation () const**

Definition at line 88 of file KDChartThreeDLineAttributes.cpp.

References [d](#).

Referenced by [operator<<\(\)](#), and [operator==\(\)](#).

```
89 {
90     return d->lineXRotation;
91 }
```

**9.63.3.4 uint ThreeDLineAttributes::lineYRotation () const**

Definition at line 98 of file KDChartThreeDLineAttributes.cpp.

References [d](#).

Referenced by [operator<<\(\)](#), and [operator==\(\)](#).

```
99 {
100     return d->lineYRotation;
101 }
```

**9.63.3.5 bool KDChart::AbstractThreeDAttributes::operator!= (const [AbstractThreeDAttributes](#) & *other*) const** [inherited]

Definition at line 60 of file KDChartAbstractThreeDAttributes.h.

```
60 { return !operator==(other); }
```

**9.63.3.6 bool KDChart::ThreeDLineAttributes::operator!= (const [ThreeDLineAttributes](#) & *other*) const**

Definition at line 54 of file KDChartThreeDLineAttributes.h.

```
54 { return !operator==(other); }
```

**9.63.3.7 [ThreeDLineAttributes](#) & ThreeDLineAttributes::operator= (const [ThreeDLineAttributes](#) &)**

Definition at line 55 of file KDChartThreeDLineAttributes.cpp.

References [d](#).

```
56 {
57     if( this == &r )
58         return *this;
59
60     *d = *r.d;
61
62     return *this;
63 }
```

### 9.63.3.8 bool AbstractThreeDAttributes::operator==(const AbstractThreeDAttributes &) const [inherited]

Definition at line 72 of file KDChartAbstractThreeDAttributes.cpp.

References KDChart::AbstractThreeDAttributes::depth(), and KDChart::AbstractThreeDAttributes::isEnabled().

Referenced by KDChart::ThreeDPieAttributes::operator==(), operator==(), and KDChart::ThreeDBarAttributes::operator==().

```

73 {
74     if( isEnabled() == r.isEnabled() &&
75         depth() == r.depth() )
76         return true;
77     else
78         return false;
79 }
```

### 9.63.3.9 bool ThreeDLineAttributes::operator==(const ThreeDLineAttributes &) const

Definition at line 74 of file KDChartThreeDLineAttributes.cpp.

References lineXRotation(), lineYRotation(), and KDChart::AbstractThreeDAttributes::operator==().

```

75 {
76     return ( lineXRotation() == r.lineXRotation() &&
77             lineYRotation() == r.lineYRotation() &&
78             AbstractThreeDAttributes::operator==(r) );
79 }
```

### 9.63.3.10 void AbstractThreeDAttributes::setDepth (double *depth*) [inherited]

Definition at line 97 of file KDChartAbstractThreeDAttributes.cpp.

References d.

```

98 {
99     d->depth = depth;
100 }
```

### 9.63.3.11 void AbstractThreeDAttributes::setEnabled (bool *enabled*) [inherited]

Definition at line 87 of file KDChartAbstractThreeDAttributes.cpp.

References d.

```

88 {
89     d->enabled = enabled;
90 }
```

**9.63.3.12 void ThreeDLineAttributes::setLineXRotation (const uint *degrees*)**

Definition at line 83 of file KDChartThreeDLineAttributes.cpp.

References [d](#).

```
84 {  
85     d->lineXRotation = degrees;  
86 }
```

**9.63.3.13 void ThreeDLineAttributes::setLineYRotation (const uint *degrees*)**

Definition at line 93 of file KDChartThreeDLineAttributes.cpp.

References [d](#).

```
94 {  
95     d->lineYRotation = degrees;  
96 }
```

**9.63.3.14 double AbstractThreeDAttributes::validDepth () const** [inherited]

Definition at line 109 of file KDChartAbstractThreeDAttributes.cpp.

References [d](#), and [KDChart::AbstractThreeDAttributes::isEnabled\(\)](#).

Referenced by [KDChart::Plotter::threeDItemDepth\(\)](#), [KDChart::LineDiagram::threeDItemDepth\(\)](#), and [KDChart::BarDiagram::threeDItemDepth\(\)](#).

```
110 {  
111     return isEnabled() ? d->depth : 0.0;  
112 }
```

The documentation for this class was generated from the following files:

- [KDChartThreeDLineAttributes.h](#)
- [KDChartThreeDLineAttributes.cpp](#)

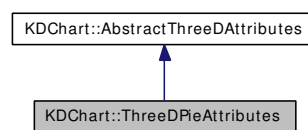
## 9.64 KDChart::ThreeDPieAttributes Class Reference

```
#include <KDChartThreeDPieAttributes.h>
```

Inheritance diagram for KDChart::ThreeDPieAttributes:



Collaboration diagram for KDChart::ThreeDPieAttributes:



### 9.64.1 Detailed Description

A set of 3D pie attributes.

Definition at line 38 of file KDChartThreeDPieAttributes.h.

### Public Member Functions

- double [depth](#) () const
- bool [isEnabled](#) () const
- bool [operator!=](#) (const [AbstractThreeDAttributes](#) &other) const
- bool [operator!=](#) (const [ThreeDPieAttributes](#) &other) const
- [ThreeDPieAttributes](#) & [operator=](#) (const [ThreeDPieAttributes](#) &)
- bool [operator==](#) (const [AbstractThreeDAttributes](#) &) const
- bool [operator==](#) (const [ThreeDPieAttributes](#) &) const
- void [setDepth](#) (double depth)
- void [setEnabled](#) (bool enabled)
- void [setUseShadowColors](#) (bool useShadowColors)
- [ThreeDPieAttributes](#) (const [ThreeDPieAttributes](#) &)
- [ThreeDPieAttributes](#) ()
- bool [useShadowColors](#) () const
- double [validDepth](#) () const
- ~[ThreeDPieAttributes](#) ()

### 9.64.2 Constructor & Destructor Documentation

#### 9.64.2.1 ThreeDPieAttributes::ThreeDPieAttributes ()

Definition at line 43 of file KDChartThreeDPieAttributes.cpp.

```

44      : AbstractThreeDAttributes( new Private() )
45  {
46
47  }

```

#### 9.64.2.2 ThreeDPieAttributes::ThreeDPieAttributes (const [ThreeDPieAttributes](#) &)

Definition at line 49 of file KDChartThreeDPieAttributes.cpp.

```

50      : AbstractThreeDAttributes( new Private( *r.d ) )
51  {
52  }

```

#### 9.64.2.3 ThreeDPieAttributes::~~ThreeDPieAttributes ()

Definition at line 64 of file KDChartThreeDPieAttributes.cpp.

```

65 {
66 }

```

### 9.64.3 Member Function Documentation

#### 9.64.3.1 double AbstractThreeDAttributes::depth () const [inherited]

Definition at line 103 of file KDChartAbstractThreeDAttributes.cpp.

References d.

Referenced by operator<<(), KDChart::AbstractThreeDAttributes::operator==(), and KDChart::PieDiagram::paint().

```

104 {
105     return d->depth;
106 }

```

#### 9.64.3.2 bool AbstractThreeDAttributes::isEnabled () const [inherited]

Definition at line 92 of file KDChartAbstractThreeDAttributes.cpp.

References d.

Referenced by operator<<(), KDChart::AbstractThreeDAttributes::operator==(), KDChart::PieDiagram::paint(), and KDChart::AbstractThreeDAttributes::validDepth().

```

93 {
94     return d->enabled;
95 }

```

#### 9.64.3.3 bool KDChart::AbstractThreeDAttributes::operator!= (const [AbstractThreeDAttributes](#) & *other*) const [inherited]

Definition at line 60 of file KDChartAbstractThreeDAttributes.h.

```

60 { return !operator==(other); }

```

#### 9.64.3.4 **bool KDChart::ThreeDPieAttributes::operator!= (const [ThreeDPieAttributes](#) & *other*) const**

Definition at line 52 of file KDChartThreeDPieAttributes.h.

```
52 { return !operator==(other); }
```

#### 9.64.3.5 **[ThreeDPieAttributes](#) & ThreeDPieAttributes::operator= (const [ThreeDPieAttributes](#) &)**

Definition at line 54 of file KDChartThreeDPieAttributes.cpp.

References [d](#).

```
55 {
56     if( this == &r )
57         return *this;
58
59     *d = *r.d;
60
61     return *this;
62 }
```

#### 9.64.3.6 **bool AbstractThreeDAttributes::operator== (const [AbstractThreeDAttributes](#) &) const** [inherited]

Definition at line 72 of file KDChartAbstractThreeDAttributes.cpp.

References [KDChart::AbstractThreeDAttributes::depth\(\)](#), and [KDChart::AbstractThreeDAttributes::isEnabled\(\)](#).

Referenced by [operator==\(\)](#), [KDChart::ThreeDLineAttributes::operator==\(\)](#), and [KDChart::ThreeDBarAttributes::operator==\(\)](#).

```
73 {
74     if( isEnabled() == r.isEnabled() &&
75         depth() == r.depth() )
76         return true;
77     else
78         return false;
79 }
```

#### 9.64.3.7 **bool ThreeDPieAttributes::operator== (const [ThreeDPieAttributes](#) &) const**

Definition at line 73 of file KDChartThreeDPieAttributes.cpp.

References [KDChart::AbstractThreeDAttributes::operator==\(\)](#), and [useShadowColors\(\)](#).

```
74 {
75     return ( useShadowColors() == r.useShadowColors() &&
76             AbstractThreeDAttributes::operator==(r) );
77 }
```

**9.64.3.8 void AbstractThreeDAttributes::setDepth (double *depth*)** [inherited]

Definition at line 97 of file KDChartAbstractThreeDAttributes.cpp.

References [d](#).

```
98 {  
99     d->depth = depth;  
100 }
```

**9.64.3.9 void AbstractThreeDAttributes::setEnabled (bool *enabled*)** [inherited]

Definition at line 87 of file KDChartAbstractThreeDAttributes.cpp.

References [d](#).

```
88 {  
89     d->enabled = enabled;  
90 }
```

**9.64.3.10 void ThreeDPieAttributes::setUseShadowColors (bool *useShadowColors*)**

Definition at line 81 of file KDChartThreeDPieAttributes.cpp.

References [d](#).

```
82 {  
83     d->useShadowColors = shadowColors;  
84 }
```

**9.64.3.11 bool ThreeDPieAttributes::useShadowColors () const**

Definition at line 86 of file KDChartThreeDPieAttributes.cpp.

References [d](#).

Referenced by [operator<<\(\)](#), and [operator==\(\)](#).

```
87 {  
88     return d->useShadowColors;  
89 }
```

**9.64.3.12 double AbstractThreeDAttributes::validDepth () const** [inherited]

Definition at line 109 of file KDChartAbstractThreeDAttributes.cpp.

References [d](#), and [KDChart::AbstractThreeDAttributes::isEnabled\(\)](#).

Referenced by [KDChart::Plotter::threeDItemDepth\(\)](#), [KDChart::LineDiagram::threeDItemDepth\(\)](#), and [KDChart::BarDiagram::threeDItemDepth\(\)](#).

```
110 {  
111     return isEnabled() ? d->depth : 0.0;  
112 }
```



The documentation for this class was generated from the following files:

- [KDChartThreeDPieAttributes.h](#)
- [KDChartThreeDPieAttributes.cpp](#)

## 9.65 KDChart::ValueTrackerAttributes Class Reference

```
#include <KDChartValueTrackerAttributes>
```

### 9.65.1 Detailed Description

Cell-specific attributes regarding value tracking.

[ValueTrackerAttributes](#) groups the properties regarding value tracking, and how it is displayed. Value tracking can be used to emphasize on one or several specific points in a line diagram.

Definition at line 49 of file `KDChartValueTrackerAttributes.h`.

### Public Member Functions

- `QBrush areaBrush () const`

**Returns:**

*The brush the area below the value tracking lines is filled with*

- `bool isEnabled () const`

**Returns:**

*Whether value tracking is enabled or not*

- `QSizeF markerSize () const`

**Returns:**

*The size of the markers*

- `bool operator!= (const ValueTrackerAttributes &other) const`
- `ValueTrackerAttributes & operator= (const ValueTrackerAttributes &)`
- `bool operator== (const ValueTrackerAttributes &) const`
- `QPen pen () const`

**Returns:**

*The pen the lines and markers are drawn with*

- `void setAreaBrush (const QBrush &brush)`

*Set the brush the area below the value tracking lines should be filled with.*

- `void setEnabled (bool enabled)`

*Set whether value tracking should be enabled for a specific index or not.*

- `void setMarkerSize (const QSizeF &size)`

*Set the size of the markers.*

- `void setPen (const QPen &pen)`

*Set the pen the value tracking lines and markers will be drawn with.*

- `ValueTrackerAttributes (const ValueTrackerAttributes &)`
- `ValueTrackerAttributes ()`
- `~ValueTrackerAttributes ()`

## 9.65.2 Constructor & Destructor Documentation

### 9.65.2.1 ValueTrackerAttributes::ValueTrackerAttributes ()

Definition at line 58 of file KDChartValueTrackerAttributes.cpp.

```
59      : _d( new Private() )
60 {
61 }
```

### 9.65.2.2 ValueTrackerAttributes::ValueTrackerAttributes (const ValueTrackerAttributes &)

Definition at line 63 of file KDChartValueTrackerAttributes.cpp.

```
64      : _d( new Private( *r.d ) )
65 {
66 }
```

### 9.65.2.3 ValueTrackerAttributes::~~ValueTrackerAttributes ()

Definition at line 78 of file KDChartValueTrackerAttributes.cpp.

```
79 {
80     delete _d; _d = 0;
81 }
```

## 9.65.3 Member Function Documentation

### 9.65.3.1 QBrush ValueTrackerAttributes::areaBrush () const

#### Returns:

The brush the area below the value tracking lines is filled with

Definition at line 107 of file KDChartValueTrackerAttributes.cpp.

References d.

Referenced by operator==( ).

```
108 {
109     return d->areaBrush;
110 }
```

### 9.65.3.2 bool ValueTrackerAttributes::isEnabled () const

#### Returns:

Whether value tracking is enabled or not

Definition at line 127 of file KDChartValueTrackerAttributes.cpp.

References d.

Referenced by operator<<( ), and operator==( ).

```

128 {
129     return d->enabled;
130 }

```

### 9.65.3.3 QSizeF ValueTrackerAttributes::markerSize () const

#### Returns:

The size of the markers

Definition at line 117 of file KDChartValueTrackerAttributes.cpp.

References d.

Referenced by operator<<(), and operator==( ).

```

118 {
119     return d->markerSize;
120 }

```

### 9.65.3.4 bool KDChart::ValueTrackerAttributes::operator!= (const ValueTrackerAttributes & other) const

Definition at line 103 of file KDChartValueTrackerAttributes.h.

```

103 { return !operator==(other); }

```

### 9.65.3.5 ValueTrackerAttributes & ValueTrackerAttributes::operator= (const ValueTrackerAttributes &)

Definition at line 68 of file KDChartValueTrackerAttributes.cpp.

References d.

```

69 {
70     if( this == &r )
71         return *this;
72
73     *d = *r.d;
74
75     return *this;
76 }

```

### 9.65.3.6 bool ValueTrackerAttributes::operator==(const ValueTrackerAttributes &) const

Definition at line 84 of file KDChartValueTrackerAttributes.cpp.

References areaBrush(), isEnabled(), markerSize(), and pen().

```

85 {
86     return ( pen() == r.pen() &&
87             areaBrush() == r.areaBrush() &&
88             markerSize() == r.markerSize() &&
89             isEnabled() == r.isEnabled() );
90 }

```

### 9.65.3.7 QPen ValueTrackerAttributes::pen () const

**Returns:**

The pen the lines and markers are drawn with

Definition at line 97 of file KDChartValueTrackerAttributes.cpp.

References d.

Referenced by operator<<(), and operator==().

```
98 {  
99     return d->pen;  
100 }
```

### 9.65.3.8 void ValueTrackerAttributes::setAreaBrush (const QBrush & *brush*)

Set the brush the area below the value tracking lines should be filled with.

Default is a black brush with the style Qt::NoBrush.

**Parameters:**

*brush* The brush the area should be filled with

Definition at line 102 of file KDChartValueTrackerAttributes.cpp.

References d.

```
103 {  
104     d->areaBrush = brush;  
105 }
```

### 9.65.3.9 void ValueTrackerAttributes::setEnabled (bool *enabled*)

Set whether value tracking should be enabled for a specific index or not.

**Parameters:**

*enabled* Whether value tracking should be enabled or not

Definition at line 122 of file KDChartValueTrackerAttributes.cpp.

References d.

```
123 {  
124     d->enabled = enabled;  
125 }
```

### 9.65.3.10 void ValueTrackerAttributes::setMarkerSize (const QSizeF & *size*)

Set the size of the markers.

This includes both the arrows at the axes and the circle at the data point.

**Parameters:**

*size* The size of the markers

Definition at line 112 of file KDChartValueTrackerAttributes.cpp.

References d.

```
113 {  
114     d->markerSize = size;  
115 }
```

**9.65.3.11 void ValueTrackerAttributes::setPen (const QPen & *pen*)**

Set the pen the value tracking lines and markers will be drawn with.

**Parameters:**

*pen* The pen the lines and markers will be drawn with

Definition at line 92 of file KDChartValueTrackerAttributes.cpp.

References d.

```
93 {  
94     d->pen = pen;  
95 }
```

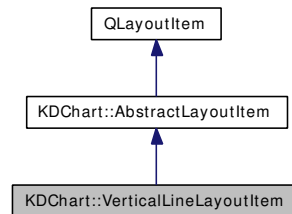
The documentation for this class was generated from the following files:

- [KDChartValueTrackerAttributes.h](#)
- [KDChartValueTrackerAttributes.cpp](#)

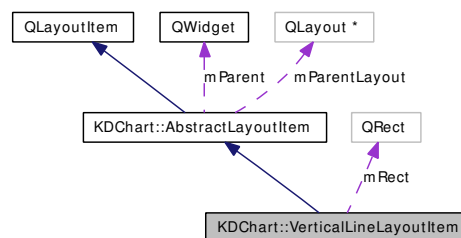
## 9.66 KDChart::VerticalLineLayoutItem Class Reference

```
#include <KDChartLayoutItems.h>
```

Inheritance diagram for KDChart::VerticalLineLayoutItem:



Collaboration diagram for KDChart::VerticalLineLayoutItem:



### 9.66.1 Detailed Description

Layout item showing a vertical line.

Definition at line 294 of file `KDChartLayoutItems.h`.

### Public Member Functions

- virtual Qt::Orientations `expandingDirections` () const
- virtual QRect `geometry` () const
- virtual bool `isEmpty` () const
- virtual QSize `maximumSize` () const
- virtual QSize `minimumSize` () const
- virtual void `paint` (QPainter \*)
- virtual void `paintAll` (QPainter &painter)

*Default impl: just call paint.*

- virtual void `paintCtx` (PaintContext \*context)

*Default impl: Paint the complete item using its layouted position and size.*

- QLayout \* `parentLayout` ()
- void `removeFromParentLayout` ()
- virtual void `setGeometry` (const QRect &r)
- void `setParentLayout` (QLayout \*lay)

- virtual void [setParentWidget](#) ([QWidget](#) \*widget)  
*Inform the item about its widget: This enables the item, to trigger that widget's update, whenever the size of the item's contents has changed.*
- virtual QSize [sizeHint](#) () const
- virtual void [sizeHintChanged](#) () const  
*Report changed size hint: ask the parent widget to recalculate the layout.*
- [VerticalLayoutItem](#) ()

## Protected Attributes

- [QWidget](#) \* [mParent](#)
- [QLayout](#) \* [mParentLayout](#)

## 9.66.2 Constructor & Destructor Documentation

### 9.66.2.1 [KDChart::VerticalLayoutItem::VerticalLayoutItem](#) ()

Definition at line 473 of file [KDChartLayoutItems.cpp](#).

```
474      : AbstractLayoutItem( Qt::AlignCenter )
475  {
476  }
```

## 9.66.3 Member Function Documentation

### 9.66.3.1 [Qt::Orientations KDChart::VerticalLayoutItem::expandingDirections](#) () const [virtual]

Definition at line 478 of file [KDChartLayoutItems.cpp](#).

```
479  {
480      return Qt::Vertical|Qt::Vertical; // Grow both vertically, and horizontally
481  }
```

### 9.66.3.2 [QRect KDChart::VerticalLayoutItem::geometry](#) () const [virtual]

Definition at line 483 of file [KDChartLayoutItems.cpp](#).

```
484  {
485      return mRect;
486  }
```

### 9.66.3.3 [bool KDChart::VerticalLayoutItem::isEmpty](#) () const [virtual]

Definition at line 488 of file [KDChartLayoutItems.cpp](#).

```
489  {
490      return false; // never empty, otherwise the layout item would not exist
491  }
```



#### 9.66.3.4 QSize KDChart::VerticalLineLayoutItem::maximumSize () const [virtual]

Definition at line 493 of file KDChartLayoutItems.cpp.

```
494 {  
495     return QSize( QWIDGETSIZE_MAX, QWIDGETSIZE_MAX );  
496 }
```

#### 9.66.3.5 QSize KDChart::VerticalLineLayoutItem::minimumSize () const [virtual]

Definition at line 498 of file KDChartLayoutItems.cpp.

```
499 {  
500     return QSize( 0, 0 );  
501 }
```

#### 9.66.3.6 void KDChart::VerticalLineLayoutItem::paint (QPainter \*) [virtual]

Implements [KDChart::AbstractLayoutItem](#).

Definition at line 514 of file KDChartLayoutItems.cpp.

```
515 {  
516     if( !mRect.isValid() )  
517         return;  
518  
519     painter->drawLine( QPointF( mRect.center().x(), mRect.top() ),  
520                      QPointF( mRect.center().x(), mRect.bottom() ) );  
521 }
```

#### 9.66.3.7 void KDChart::AbstractLayoutItem::paintAll (QPainter & *painter*) [virtual, inherited]

Default impl: just call paint.

Derived classes like [KDChart::AbstractArea](#) are providing additional action here.

Reimplemented in [KDChart::AbstractArea](#), [KDChart::TextArea](#), and [KDChart::TernaryAxis](#).

Definition at line 69 of file KDChartLayoutItems.cpp.

References [KDChart::AbstractLayoutItem::paint\(\)](#).

```
70 {  
71     paint( &painter );  
72 }
```

#### 9.66.3.8 void KDChart::AbstractLayoutItem::paintCtx ([PaintContext](#) \* *context*) [virtual, inherited]

Default impl: Paint the complete item using its layouted position and size.

Reimplemented in [KDChart::CartesianAxis](#), and [KDChart::TernaryAxis](#).

Definition at line 77 of file KDChartLayoutItems.cpp.

References KDChart::AbstractLayoutItem::paint(), and KDChart::PaintContext::painter().

```

78 {
79     if( context )
80         paint( context->painter() );
81 }
```

### 9.66.3.9 QLayout\* KDChart::AbstractLayoutItem::parentLayout () [inherited]

Definition at line 76 of file KDChartLayoutItems.h.

```

77     {
78         return mParentLayout;
79     }
```

### 9.66.3.10 void KDChart::AbstractLayoutItem::removeFromParentLayout () [inherited]

Definition at line 80 of file KDChartLayoutItems.h.

Referenced by KDChart::Chart::takeCoordinatePlane().

```

81     {
82         if( mParentLayout ){
83             if( widget() )
84                 mParentLayout->removeWidget( widget() );
85             else
86                 mParentLayout->removeItem( this );
87         }
88     }
```

### 9.66.3.11 void KDChart::VerticalLineLayoutItem::setGeometry (const QRect & r) [virtual]

Definition at line 503 of file KDChartLayoutItems.cpp.

```

504 {
505     mRect = r;
506 }
```

### 9.66.3.12 void KDChart::AbstractLayoutItem::setParentLayout (QLayout \* lay) [inherited]

Definition at line 72 of file KDChartLayoutItems.h.

```

73     {
74         mParentLayout = lay;
75     }
```

### 9.66.3.13 void KDChart::AbstractLayoutItem::setParentWidget (QWidget \* widget) [virtual, inherited]

Inform the item about its widget: This enables the item, to trigger that widget's update, whenever the size of the item's contents has changed.

Thus, you need to call setParentWidget on every item, that has a non-fixed size.

Definition at line 64 of file KDChartLayoutItems.cpp.

References KDChart::AbstractLayoutItem::mParent.

Referenced by KDChart::HeaderFooter::setParent(), and KDChart::AbstractCartesianDiagram::takeAxis().

```
65 {
66     mParent = widget;
67 }
```

### 9.66.3.14 QSize KDChart::VerticalLayoutItem::sizeHint () const [virtual]

Definition at line 508 of file KDChartLayoutItems.cpp.

```
509 {
510     return QSize( 3, -1 ); // see qframe.cpp
511 }
```

### 9.66.3.15 void KDChart::AbstractLayoutItem::sizeHintChanged () const [virtual, inherited]

Report changed size hint: ask the parent widget to recalculate the layout.

Definition at line 86 of file KDChartLayoutItems.cpp.

References KDChart::AbstractLayoutItem::mParent.

Referenced by KDChart::TextLayoutItem::sizeHint().

```
87 {
88     // This is exactly like what QWidget::updateGeometry does.
89     qDebug("KDChart::AbstractLayoutItem::sizeHintChanged() called");
90     if( mParent ) {
91         if ( mParent->layout() )
92             mParent->layout()->invalidate();
93         else
94             QApplication::postEvent( mParent, new QEvent( QEvent::LayoutRequest ) );
95     }
96 }
```

## 9.66.4 Member Data Documentation

### 9.66.4.1 QWidget\* KDChart::AbstractLayoutItem::mParent [protected, inherited]

Definition at line 90 of file KDChartLayoutItems.h.

Referenced by KDChart::AbstractLayoutItem::setParentWidget(), KDChart::TextLayoutItem::setText(), KDChart::TextLayoutItem::setTextAttributes(), and KDChart::AbstractLayoutItem::sizeHintChanged().

**9.66.4.2** `QLayout*` [KDChart::AbstractLayoutItem::mParentLayout](#) [protected,  
inherited]

Definition at line 91 of file [KDChartLayoutItems.h](#).

Referenced by [KDChart::AutoSpacerLayoutItem::paint\(\)](#).

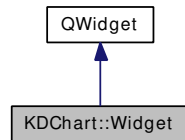
The documentation for this class was generated from the following files:

- [KDChartLayoutItems.h](#)
- [KDChartLayoutItems.cpp](#)

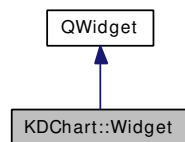
## 9.67 KDChart::Widget Class Reference

```
#include <KDChartWidget.h>
```

Inheritance diagram for KDChart::Widget:



Collaboration diagram for KDChart::Widget:



### 9.67.1 Detailed Description

The KD [Chart](#) widget for usage without Model/View.

If you want to use KD [Chart](#) with Model/View, use [KDChart::Chart](#) instead.

Definition at line 63 of file KDChartWidget.h.

### Public Types

- enum [ChartType](#) {  
    [NoType](#),  
    [Bar](#),  
    [Line](#),  
    [Pie](#),  
    [Ring](#),  
    [Polar](#) }  
• enum [SubType](#) {  
    [Normal](#),  
    [Stacked](#),  
    [Percent](#),  
    [Rows](#) }

*Sub type values, matching the values defines for the respective Diagram classes.*

## Public Slots

- void [setGlobalLeading](#) (int left, int top, int right, int bottom)  
*Sets all global leadings (borders).*
- void [setGlobalLeadingBottom](#) (int leading)  
*Sets the bottom leading (border).*
- void [setGlobalLeadingLeft](#) (int leading)  
*Sets the left leading (border).*
- void [setGlobalLeadingRight](#) (int leading)  
*Sets the right leading (border).*
- void [setGlobalLeadingTop](#) (int leading)  
*Sets the top leading (border).*
- void [setSubType](#) ([SubType](#) subType)  
*Sets the type of the chart without changing the main type.*
- void [setType](#) ([ChartType](#) chartType, [SubType](#) subType=Normal)  
*Sets the type of the chart.*

## Public Member Functions

- void [addHeaderFooter](#) ([HeaderFooter](#) \*header)  
*Adds the existing header / footer object header.*
- void [addHeaderFooter](#) (const QString &text, [HeaderFooter::HeaderFooterType](#) type, [Position](#) position)  
*Adds a new header/footer with the given text to the position.*
- void [addLegend](#) ([Legend](#) \*legend)  
*Adds a new, already existing, legend.*
- void [addLegend](#) ([Position](#) position)  
*Adds an empty legend on the given position.*
- [QList](#)< [HeaderFooter](#) \* > [allHeadersFooters](#) ()  
*Returns a list with all headers.*
- [QList](#)< [Legend](#) \* > [allLegends](#) ()  
*Returns a list with all legends.*
- [BarDiagram](#) \* [barDiagram](#) ()  
*If the current diagram is a [BarDiagram](#), it is returned; otherwise 0 is returned.*
- [AbstractCoordinatePlane](#) \* [coordinatePlane](#) ()  
*Returns a pointer to the current coordinate plane.*

- [AbstractDiagram \\* diagram \(\)](#)  
*Returns a pointer to the current diagram.*
- [HeaderFooter \\* firstHeaderFooter \(\)](#)  
*Returns the first of all headers.*
- [int globalLeadingBottom \(\) const](#)  
*Returns the bottom leading (border).*
- [int globalLeadingLeft \(\) const](#)  
*Returns the left leading (border).*
- [int globalLeadingRight \(\) const](#)  
*Returns the right leading (border).*
- [int globalLeadingTop \(\) const](#)  
*Returns the top leading (border).*
- [Legend \\* legend \(\)](#)  
*Returns the first of all legends.*
- [LineDiagram \\* lineDiagram \(\)](#)  
*If the current diagram is a [LineDiagram](#), it is returned; otherwise 0 is returned.*
- [PieDiagram \\* pieDiagram \(\)](#)  
*If the current diagram is a [PieDiagram](#), it is returned; otherwise 0 is returned.*
- [PolarDiagram \\* polarDiagram \(\)](#)  
*If the current diagram is a [PolarDiagram](#), it is returned; otherwise 0 is returned.*
- [void replaceHeaderFooter \(HeaderFooter \\*header, HeaderFooter \\*oldHeader=0\)](#)  
*Replaces the old header (or footer, resp).*
- [void replaceLegend \(Legend \\*legend, Legend \\*oldLegend=0\)](#)
- [void resetData \(\)](#)  
*Resets all data.*
- [RingDiagram \\* ringDiagram \(\)](#)  
*If the current diagram is a [RingDiagram](#), it is returned; otherwise 0 is returned.*
- [void setDataCell \(int row, int column, QPair< double, double > data\)](#)  
*Sets the data for a given column using an (X, Y) QPair of doubles.*
- [void setDataCell \(int row, int column, double data\)](#)  
*Sets the Y value data for a given cell.*
- [void setDataset \(int column, const QVector< QPair< double, double > > &data, const QString &title=QString\(\)\)](#)  
*Sets the data in the given column using a QVector of QPairs of double for the (X, Y) values.*

- void [setDataset](#) (int column, const QVector< double > &data, const QString &title=QString())  
*Sets the data in the given column using a QVector of double for the Y values.*
- [SubType subType](#) () const  
*Returns the sub-type of the chart.*
- void [takeHeaderFooter](#) (HeaderFooter \*header)  
*Remove the header (or footer, resp.*
- void [takeLegend](#) (Legend \*legend)
- [ChartType type](#) () const  
*Returns the type of the chart.*
- [Widget](#) (QWidget \*parent=0)  
*Standard Qt-style Constructor.*
- [~Widget](#) ()  
*Destructor.*

## 9.67.2 Member Enumeration Documentation

### 9.67.2.1 enum [KDChart::Widget::ChartType](#)

Enumerator:

*NoType*  
*Bar*  
*Line*  
*Pie*  
*Ring*  
*Polar*

Definition at line 203 of file KDChartWidget.h.

```
203 { NoType, Bar, Line, Pie, Ring, Polar };
```

### 9.67.2.2 enum [KDChart::Widget::SubType](#)

Sub type values, matching the values defines for the respective Diagram classes.

Enumerator:

*Normal*  
*Stacked*  
*Percent*  
*Rows*

Definition at line 209 of file KDChartWidget.h.

```
209 { Normal, Stacked, Percent, Rows };
```



### 9.67.3 Constructor & Destructor Documentation

#### 9.67.3.1 Widget::Widget ([QWidget](#) \* *parent* = 0) [explicit]

Standard Qt-style Constructor.

Creates a new widget with all data initialized empty.

**Parameters:**

*parent* the widget parent; passed on to [QWidget](#)

Definition at line 82 of file KDChartWidget.cpp.

References [Line](#), and [setType\(\)](#).

```
82         :  
83     QWidget(parent), _d( new Private( this ) )  
84 {  
85     // as default we have a cartesian coordinate plane ...  
86     // ... and a line diagram  
87     setType( Line );  
88 }
```

#### 9.67.3.2 Widget::~~Widget ()

Destructor.

Definition at line 93 of file KDChartWidget.cpp.

```
94 {  
95     delete _d; _d = 0;  
96 }
```

### 9.67.4 Member Function Documentation

#### 9.67.4.1 void Widget::addHeaderFooter ([HeaderFooter](#) \* *header*)

Adds the existing header / footer object *header*.

**See also:**

[replaceHeaderFooter](#), [takeHeaderFooter](#)

Definition at line 286 of file KDChartWidget.cpp.

References [d](#), and [KDChart::HeaderFooter::setParent\(\)](#).

```
287 {  
288     header->setParent( &d->m_chart );  
289     d->m_chart.addHeaderFooter( header ); // we need this explicit call !  
290 }
```

#### 9.67.4.2 void Widget::addHeaderFooter (const QString & *text*, [HeaderFooter::HeaderFooterType](#) *type*, [Position](#) *position*)

Adds a new header/footer with the given text to the position.

Definition at line 272 of file KDChartWidget.cpp.

References `d`, `KDChart::HeaderFooter::setPosition()`, `KDChart::TextLayoutItem::setText()`, and `KDChart::HeaderFooter::setType()`.

```
275 {
276     HeaderFooter* newHeader = new HeaderFooter( &d->m_chart );
277     newHeader->setType( type );
278     newHeader->setPosition( position );
279     newHeader->setText( text );
280     d->m_chart.addHeaderFooter( newHeader ); // we need this explicit call !
281 }
```

#### 9.67.4.3 void Widget::addLegend ([Legend](#) \* *legend*)

Adds a new, already existing, legend.

Definition at line 332 of file KDChartWidget.cpp.

References `d`, `diagram()`, `legend()`, and `KDChart::Legend::setDiagram()`.

```
333 {
334     legend->setDiagram( diagram() );
335     legend->setParent( &d->m_chart );
336     d->m_chart.addLegend( legend );
337 }
```

#### 9.67.4.4 void Widget::addLegend ([Position](#) *position*)

Adds an empty legend on the given position.

Definition at line 322 of file KDChartWidget.cpp.

References `d`, `diagram()`, `legend()`, and `KDChart::Legend::setPosition()`.

```
323 {
324     Legend* legend = new Legend( diagram(), &d->m_chart );
325     legend->setPosition( position );
326     d->m_chart.addLegend( legend );
327 }
```

#### 9.67.4.5 QList< [KDChart::HeaderFooter](#) \* > Widget::allHeadersFooters ()

Returns a list with all headers.

Definition at line 264 of file KDChartWidget.cpp.

References `d`.

```
265 {
266     return d->m_chart.headerFooters();
267 }
```

#### 9.67.4.6 QList< KDChart::Legend \* > Widget::allLegends ()

Returns a list with all legends.

Definition at line 314 of file KDChartWidget.cpp.

References d.

```
315 {  
316     return d->m_chart.legends();  
317 }
```

#### 9.67.4.7 BarDiagram \* Widget::barDiagram ()

If the current diagram is a [BarDiagram](#), it is returned; otherwise 0 is returned.

This function provides type-safe casting.

Definition at line 359 of file KDChartWidget.cpp.

References diagram().

```
360 {  
361     return dynamic_cast<BarDiagram*>( diagram() );  
362 }
```

#### 9.67.4.8 AbstractCoordinatePlane \* Widget::coordinatePlane ()

Returns a pointer to the current coordinate plane.

Definition at line 380 of file KDChartWidget.cpp.

References d.

Referenced by diagram(), and setType().

```
381 {  
382     return d->m_chart.coordinatePlane();  
383 }
```

#### 9.67.4.9 AbstractDiagram \* Widget::diagram ()

Returns a pointer to the current diagram.

Definition at line 351 of file KDChartWidget.cpp.

References coordinatePlane(), and KDChart::AbstractCoordinatePlane::diagram().

Referenced by addLegend(), barDiagram(), lineDiagram(), pieDiagram(), polarDiagram(), replaceLegend(), ringDiagram(), setSubType(), subType(), and type().

```
352 {  
353     if ( coordinatePlane() == 0 )  
354         qDebug() << "diagram(): coordinatePlane() was NULL";  
355     return coordinatePlane()->diagram();  
357 }
```

**9.67.4.10   `KDChart::HeaderFooter * Widget::firstHeaderFooter ()`**

Returns the first of all headers.

Definition at line 256 of file `KDChartWidget.cpp`.

References `d`.

```
257 {  
258     return d->m_chart.headerFooter();  
259 }
```

**9.67.4.11   `int Widget::globalLeadingBottom () const`**

Returns the bottom leading (border).

Definition at line 248 of file `KDChartWidget.cpp`.

References `d`.

```
249 {  
250     return d->m_chart.globalLeadingBottom();  
251 }
```

**9.67.4.12   `int Widget::globalLeadingLeft () const`**

Returns the left leading (border).

Definition at line 200 of file `KDChartWidget.cpp`.

References `d`.

```
201 {  
202     return d->m_chart.globalLeadingLeft();  
203 }
```

**9.67.4.13   `int Widget::globalLeadingRight () const`**

Returns the right leading (border).

Definition at line 232 of file `KDChartWidget.cpp`.

References `d`.

```
233 {  
234     return d->m_chart.globalLeadingRight();  
235 }
```

**9.67.4.14   `int Widget::globalLeadingTop () const`**

Returns the top leading (border).

Definition at line 216 of file `KDChartWidget.cpp`.

References `d`.

```
217 {  
218     return d->m_chart.globalLeadingTop();  
219 }
```

#### 9.67.4.15 [KDChart::Legend](#) \* [Widget::legend](#) ()

Returns the first of all legends.

Definition at line 306 of file KDChartWidget.cpp.

References [d](#).

Referenced by [addLegend\(\)](#), [replaceLegend\(\)](#), and [takeLegend\(\)](#).

```
307 {  
308     return d->m_chart.legend();  
309 }
```

#### 9.67.4.16 [LineDiagram](#) \* [Widget::lineDiagram](#) ()

If the current diagram is a [LineDiagram](#), it is returned; otherwise 0 is returned.

This function provides type-safe casting.

Definition at line 363 of file KDChartWidget.cpp.

References [diagram\(\)](#).

```
364 {  
365     return dynamic_cast<LineDiagram*>( diagram() );  
366 }
```

#### 9.67.4.17 [PieDiagram](#) \* [Widget::pieDiagram](#) ()

If the current diagram is a [PieDiagram](#), it is returned; otherwise 0 is returned.

This function provides type-safe casting.

Definition at line 367 of file KDChartWidget.cpp.

References [diagram\(\)](#).

```
368 {  
369     return dynamic_cast<PieDiagram*>( diagram() );  
370 }
```

#### 9.67.4.18 [PolarDiagram](#) \* [Widget::polarDiagram](#) ()

If the current diagram is a [PolarDiagram](#), it is returned; otherwise 0 is returned.

This function provides type-safe casting.

Definition at line 375 of file KDChartWidget.cpp.

References [diagram\(\)](#).

```

376 {
377     return dynamic_cast<PolarDiagram*>( diagram() );
378 }

```

#### 9.67.4.19 void Widget::replaceHeaderFooter ([HeaderFooter](#) \* header, [HeaderFooter](#) \* oldHeader = 0)

Replaces the old header (or footer, resp.

), or appends the new header or footer, if there is none yet.

##### Parameters:

**header** The header or footer to be used instead of the old one. This parameter must not be zero, or the method will do nothing.

**oldHeader** The header or footer to be removed by the new one. This header or footer will be deleted automatically. If the parameter is omitted, the very first header or footer will be replaced. In case, there was no header and no footer yet, the new header or footer will just be added.

##### Note:

If you want to re-use the old header or footer, call takeHeaderFooter and addHeaderFooter, instead of using replaceHeaderFooter.

##### See also:

[addHeaderFooter](#), [takeHeaderFooter](#)

Definition at line 292 of file KDChartWidget.cpp.

References [d](#), and [KDChart::HeaderFooter::setParent\(\)](#).

```

293 {
294     header->setParent( &d->m_chart );
295     d->m_chart.replaceHeaderFooter( header, oldHeader );
296 }

```

#### 9.67.4.20 void Widget::replaceLegend ([Legend](#) \* legend, [Legend](#) \* oldLegend = 0)

Definition at line 339 of file KDChartWidget.cpp.

References [d](#), [diagram\(\)](#), [legend\(\)](#), and [KDChart::Legend::setDiagram\(\)](#).

```

340 {
341     legend->setDiagram( diagram() );
342     legend->setParent( &d->m_chart );
343     d->m_chart.replaceLegend( legend, oldLegend );
344 }

```

#### 9.67.4.21 void Widget::resetData ()

Resets all data.

Definition at line 175 of file KDChartWidget.cpp.

References [d](#).

```
176 {  
177     d->m_model.clear();  
178     d->usedDatasetWidth = 0;  
179 }
```

#### 9.67.4.22 [RingDiagram](#) \* Widget::ringDiagram ()

If the current diagram is a [RingDiagram](#), it is returned; otherwise 0 is returned.

This function provides type-safe casting.

Definition at line 371 of file KDChartWidget.cpp.

References [diagram\(\)](#).

```
372 {  
373     return dynamic_cast<RingDiagram*>( diagram() );  
374 }
```

#### 9.67.4.23 void Widget::setDataCell (int row, int column, QPair< double, double > data)

Sets the data for a given column using an (X, Y) QPair of doubles.

Definition at line 156 of file KDChartWidget.cpp.

References [d](#).

```
157 {  
158     if ( ! checkDatasetWidth( 2 ) )  
159         return;  
160  
161     QStandardItemModel & model = d->m_model;  
162  
163     justifyModelSize( row + 1, (column + 1) * 2 );  
164  
165     QModelIndex index = model.index( row, column * 2 );  
166     model.setData( index, QVariant( data.first ), Qt::DisplayRole );  
167  
168     index = model.index( row, column * 2 + 1 );  
169     model.setData( index, QVariant( data.second ), Qt::DisplayRole );  
170 }
```

#### 9.67.4.24 void Widget::setDataCell (int row, int column, double data)

Sets the Y value data for a given cell.

Definition at line 143 of file KDChartWidget.cpp.

References [d](#).

```
144 {  
145     if ( ! checkDatasetWidth( 1 ) )  
146         return;  
147  
148     QStandardItemModel & model = d->m_model;  
149  
150     justifyModelSize( row + 1, column + 1 );  
151 }
```

```

152     const QModelIndex index = model.index( row, column );
153     model.setData( index, QVariant( data ), Qt::DisplayRole );
154 }

```

#### 9.67.4.25 void Widget::setDataset( int *column*, const QVector< QPair< double, double > > & *data*, const QString & *title* = QString())

Sets the data in the given column using a QVector of QPairs of double for the (X, Y) values.

Definition at line 120 of file KDChartWidget.cpp.

References [d](#).

```

121 {
122     if ( ! checkDatasetWidth( 2 ) )
123         return;
124
125     QStandardItemModel & model = d->m_model;
126
127     justifyModelSize( data.size(), (column + 1) * 2 );
128
129     for( int i = 0; i < data.size(); ++i )
130     {
131         QModelIndex index = model.index( i, column * 2 );
132         model.setData( index, QVariant( data[i].first ), Qt::DisplayRole );
133
134         index = model.index( i, column * 2 + 1 );
135         model.setData( index, QVariant( data[i].second ), Qt::DisplayRole );
136     }
137     if ( ! title.isEmpty() ){
138         model.setHeaderData( column * 2, Qt::Horizontal, QVariant( title ) );
139         model.setHeaderData( column * 2+1, Qt::Horizontal, QVariant( title ) );
140     }
141 }

```

#### 9.67.4.26 void Widget::setDataset( int *column*, const QVector< double > & *data*, const QString & *title* = QString())

Sets the data in the given column using a QVector of double for the Y values.

Definition at line 102 of file KDChartWidget.cpp.

References [d](#).

```

103 {
104     if ( ! checkDatasetWidth( 1 ) )
105         return;
106
107     QStandardItemModel & model = d->m_model;
108
109     justifyModelSize( data.size(), column + 1 );
110
111     for( int i = 0; i < data.size(); ++i )
112     {
113         const QModelIndex index = model.index( i, column );
114         model.setData( index, QVariant( data[i] ), Qt::DisplayRole );
115     }
116     if ( ! title.isEmpty() )
117         model.setHeaderData( column, Qt::Horizontal, QVariant( title ) );
118 }

```



**9.67.4.27 void Widget::setGlobalLeading (int *left*, int *top*, int *right*, int *bottom*)** [slot]

Sets all global leadings (borders).

Definition at line 184 of file KDChartWidget.cpp.

References d.

```
185 {  
186     d->m_chart.setGlobalLeading( left, top, right, bottom );  
187 }
```

**9.67.4.28 void Widget::setGlobalLeadingBottom (int *leading*)** [slot]

Sets the bottom leading (border).

Definition at line 240 of file KDChartWidget.cpp.

References d.

```
241 {  
242     d->m_chart.setGlobalLeadingBottom( leading );  
243 }
```

**9.67.4.29 void Widget::setGlobalLeadingLeft (int *leading*)** [slot]

Sets the left leading (border).

Definition at line 192 of file KDChartWidget.cpp.

References d.

```
193 {  
194     d->m_chart.setGlobalLeadingLeft( leading );  
195 }
```

**9.67.4.30 void Widget::setGlobalLeadingRight (int *leading*)** [slot]

Sets the right leading (border).

Definition at line 224 of file KDChartWidget.cpp.

References d.

```
225 {  
226     d->m_chart.setGlobalLeadingRight( leading );  
227 }
```

**9.67.4.31 void Widget::setGlobalLeadingTop (int *leading*)** [slot]

Sets the top leading (border).

Definition at line 208 of file KDChartWidget.cpp.

References d.

```

209 {
210     d->m_chart.setGlobalLeadingTop( leading );
211 }

```

#### 9.67.4.32 void Widget::setSubType (SubType subType) [slot]

Sets the type of the chart without changing the main type.

Make sure to use a sub-type that matches the main type, so e.g. setting sub-type Rows makes sense for Bar charts only, and it will be ignored for all other chart types.

**See also:**

KDChartBarDiagram::BarType, KDChartLineDiagram::LineType  
 KDChartPieDiagram::PieType, KDChartRingDiagram::RingType  
 KDChartPolarDiagram::PolarType

Definition at line 462 of file KDChartWidget.cpp.

References [diagram\(\)](#), [KDChart::LineDiagram::Normal](#), [KDChart::BarDiagram::Normal](#), [Normal](#), [KDChart::LineDiagram::Percent](#), [KDChart::BarDiagram::Percent](#), [Percent](#), [KDChart::BarDiagram::Rows](#), [Rows](#), [SET\\_SUB\\_TYPE](#), [KDChart::LineDiagram::Stacked](#), [KDChart::BarDiagram::Stacked](#), and [Stacked](#).

Referenced by [setType\(\)](#).

```

463 {
464     BarDiagram* barDia = qobject_cast< BarDiagram* >( diagram() );
465     LineDiagram* lineDia = qobject_cast< LineDiagram* >( diagram() );
466
467 //FIXME(khz): Add the impl for these chart types - or remove them from here:
468 //     PieDiagram* pieDia = qobject_cast< PieDiagram* >( diagram() );
469 //     PolarDiagram* polarDia = qobject_cast< PolarDiagram* >( diagram() );
470 //     RingDiagram* ringDia = qobject_cast< RingDiagram* >( diagram() );
471
472 #define SET_SUB_TYPE(DIAGRAM, SUBTYPE) \
473 { \
474     if( DIAGRAM ) \
475         DIAGRAM->setType( SUBTYPE ); \
476 }
477     switch ( subType )
478     {
479     case Normal:
480         SET_SUB_TYPE( barDia, BarDiagram::Normal );
481         SET_SUB_TYPE( lineDia, LineDiagram::Normal );
482         break;
483     case Stacked:
484         SET_SUB_TYPE( barDia, BarDiagram::Stacked );
485         SET_SUB_TYPE( lineDia, LineDiagram::Stacked );
486         break;
487     case Percent:
488         SET_SUB_TYPE( barDia, BarDiagram::Percent );
489         SET_SUB_TYPE( lineDia, LineDiagram::Percent );
490         break;
491     case Rows:
492         SET_SUB_TYPE( barDia, BarDiagram::Rows );
493         break;
494     default:
495         Q_ASSERT_X ( false,
496                     "Widget::setSubType", "Sub-type not supported!" );
497         break;
498     }
499 //     coordinatePlane()->show();
500 }

```

**9.67.4.33 void Widget::setType (ChartType chartType, SubType subType = Normal) [slot]**

Sets the type of the chart.

Definition at line 397 of file KDCartWidget.cpp.

References KDCart::AbstractCartesianDiagram::axes(), Bar, coordinatePlane(), d, isCartesian(), isPolar(), Line, NoType, Pie, Polar, KDCart::AbstractCoordinatePlane::replaceDiagram(), Ring, KDCart::AbstractDiagram::setDatasetDimension(), KDCart::Legend::setDiagram(), KDCart::AbstractDiagram::setModel(), setSubType(), subType(), KDCart::AbstractCartesianDiagram::takeAxis(), and type().

Referenced by Widget().

```

398 {
399     AbstractDiagram* diag = 0;
400     CartesianCoordinatePlane* cartPlane = 0;
401     PolarCoordinatePlane* polPlane = 0;
402
403
404     if ( chartType != type() ){
405         switch ( chartType )
406         {
407             case Bar:
408                 diag = new BarDiagram( &d->m_chart, cartPlane );
409                 break;
410             case Line:
411                 diag = new LineDiagram( &d->m_chart, cartPlane );
412                 break;
413             case Pie:
414                 diag = new PieDiagram( &d->m_chart, polPlane );
415                 break;
416             case Polar:
417                 diag = new PolarDiagram( &d->m_chart, polPlane );
418                 break;
419             case Ring:
420                 diag = new RingDiagram( &d->m_chart, polPlane );
421                 break;
422             case NoType:
423                 break;
424         }
425         if ( diag != NULL )
426         {
427             if ( isPolar( type() ) && isCartesian( chartType ) )
428             {
429                 cartPlane = new CartesianCoordinatePlane( &d->m_chart );
430                 d->m_chart.replaceCoordinatePlane( cartPlane );
431             }
432             else if ( isCartesian( type() ) && isPolar( chartType ) )
433             {
434                 polPlane = new PolarCoordinatePlane( &d->m_chart );
435                 d->m_chart.replaceCoordinatePlane( polPlane );
436             }
437             else if ( isCartesian( type() ) && isCartesian( chartType ) )
438             {
439                 AbstractCartesianDiagram *old =
440                     qobject_cast<AbstractCartesianDiagram*>( d->m_chart.coordinatePlane()->diagram
441                     Q_FOREACH( CartesianAxis* axis, old->axes() ) {
442                         old->takeAxis( axis );
443                         qobject_cast<AbstractCartesianDiagram*>(diag)->addAxis( axis );
444                     }
445             }
446             diag->setModel( &d->m_model );
447             coordinatePlane()->replaceDiagram( diag );
448
449             LegendList legends = d->m_chart.legends();

```

```

450         Q_FOREACH( Legend* l, legends)
451             l->setDiagram( diag );
452         if( d->usedDatasetWidth )
453             diag->setDatasetDimension( d->usedDatasetWidth );
454     }
455 }
456
457 if ( chartSubType != subType() )
458     setSubType( chartSubType );
459 //     coordinatePlane()->show();
460 }

```

#### 9.67.4.34 [Widget::SubType](#) [Widget::subType\(\)](#) const

Returns the sub-type of the chart.

Definition at line 523 of file `KDChartWidget.cpp`.

References `Bar`, `diagram()`, `Line`, `KDChart::LineDiagram::Normal`, `KDChart::BarDiagram::Normal`, `Normal`, `KDChart::LineDiagram::Percent`, `Percent`, `KDChart::BarDiagram::Percent`, `Pie`, `Polar`, `Ring`, `Rows`, `KDChart::BarDiagram::Rows`, `KDChart::LineDiagram::Stacked`, `Stacked`, `KDChart::BarDiagram::Stacked`, `TEST_SUB_TYPE`, and `type()`.

Referenced by `setType()`.

```

524 {
525     // PENDING(christoph) save the type out-of-band:
526     Widget::SubType retVal = Normal;
527
528     AbstractDiagram * const dia = const_cast<Widget*>( this )->diagram();
529     BarDiagram* barDia = qobject_cast< BarDiagram* >( dia );
530     LineDiagram* lineDia = qobject_cast< LineDiagram* >( dia );
531
532     //FIXME(khz): Add the impl for these chart types - or remove them from here:
533     //     PieDiagram* pieDia = qobject_cast< PieDiagram* >( diagram() );
534     //     PolarDiagram* polarDia = qobject_cast< PolarDiagram* >( diagram() );
535     //     RingDiagram* ringDia = qobject_cast< RingDiagram* >( diagram() );
536
537     #define TEST_SUB_TYPE(DIAGRAM, INTERNALSUBTYPE, SUBTYPE) \
538     { \
539         if( DIAGRAM && DIAGRAM->type() == INTERNALSUBTYPE ) \
540             retVal = SUBTYPE; \
541     }
542     const Widget::ChartType mainType = type();
543     switch ( mainType )
544     {
545         case Bar:
546             TEST_SUB_TYPE( barDia, BarDiagram::Normal, Normal );
547             TEST_SUB_TYPE( barDia, BarDiagram::Stacked, Stacked );
548             TEST_SUB_TYPE( barDia, BarDiagram::Percent, Percent );
549             TEST_SUB_TYPE( barDia, BarDiagram::Rows, Rows );
550             break;
551         case Line:
552             TEST_SUB_TYPE( lineDia, LineDiagram::Normal, Normal );
553             TEST_SUB_TYPE( lineDia, LineDiagram::Stacked, Stacked );
554             TEST_SUB_TYPE( lineDia, LineDiagram::Percent, Percent );
555             break;
556         case Pie:
557             // no impl. yet
558             break;
559         case Polar:
560             // no impl. yet
561             break;
562         case Ring:

```

```

563         // no impl. yet
564         break;
565     default:
566         Q_ASSERT_X ( false,
567                     "Widget::subType", "Chart type not supported!" );
568         break;
569     }
570     return retVal;
571 }

```

#### 9.67.4.35 void Widget::takeHeaderFooter (**HeaderFooter** \* *header*)

Remove the header (or footer, resp.

) from the widget, without deleting it. The chart no longer owns the header or footer, so it is the caller's responsibility to delete the header or footer.

**See also:**

[addHeaderFooter](#), [replaceHeaderFooter](#)

Definition at line 298 of file KDChartWidget.cpp.

References d.

```

299 {
300     d->m_chart.takeHeaderFooter( header );
301 }

```

#### 9.67.4.36 void Widget::takeLegend (**Legend** \* *legend*)

Definition at line 346 of file KDChartWidget.cpp.

References d, and legend().

```

347 {
348     d->m_chart.takeLegend( legend );
349 }

```

#### 9.67.4.37 **Widget::ChartType** Widget::type () const

Returns the type of the chart.

Definition at line 505 of file KDChartWidget.cpp.

References Bar, diagram(), Line, NoType, Pie, Polar, and Ring.

Referenced by setType(), and subType().

```

506 {
507     // PENDING(christoph) save the type out-of-band:
508     AbstractDiagram * const dia = const_cast<Widget*>( this )->diagram();
509     if ( qobject_cast< BarDiagram* >( dia ) )
510         return Bar;
511     else if ( qobject_cast< LineDiagram* >( dia ) )
512         return Line;

```

```
513     else if( qobject_cast< PieDiagram* >( dia ) )
514         return Pie;
515     else if( qobject_cast< PolarDiagram* >( dia ) )
516         return Polar;
517     else if( qobject_cast< RingDiagram* >( dia ) )
518         return Ring;
519     else
520         return NoType;
521 }
```

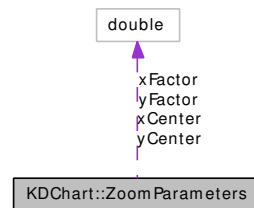
The documentation for this class was generated from the following files:

- [KDChartWidget.h](#)
- [KDChartWidget.cpp](#)

## 9.68 KDChart::ZoomParameters Class Reference

```
#include <KDChartZoomParameters.h>
```

Collaboration diagram for KDChart::ZoomParameters:



### 9.68.1 Detailed Description

[ZoomParameters](#) stores the center and the factor of zooming internally.

Definition at line 49 of file KDChartZoomParameters.h.

#### Public Member Functions

- const QPointF [center](#) () const
- void [setCenter](#) (const QPointF &center)
- [ZoomParameters](#) (double [xFactor](#), double [yFactor](#), const QPointF &center)
- [ZoomParameters](#) ()

#### Public Attributes

- double [xCenter](#)
- double [xFactor](#)
- double [yCenter](#)
- double [yFactor](#)

### 9.68.2 Constructor & Destructor Documentation

#### 9.68.2.1 KDChart::ZoomParameters::ZoomParameters ()

Definition at line 51 of file KDChartZoomParameters.h.

```

52         : xFactor( 1.0 ),
53           yFactor( 1.0 ),
54           xCenter( 0.5 ),
55           yCenter( 0.5 )
56     {
57     }
```

### 9.68.2.2 **KDChart::ZoomParameters::ZoomParameters** (double *xFactor*, double *yFactor*, const **QPointF** & *center*)

Definition at line 59 of file KDChartZoomParameters.h.

```
60         : xFactor( xFactor ),
61           yFactor( yFactor ),
62           xCenter( center.x() ),
63           yCenter( center.y() )
64     {
65     }
```

## 9.68.3 Member Function Documentation

### 9.68.3.1 **const QPointF KDChart::ZoomParameters::center** () const

Definition at line 72 of file KDChartZoomParameters.h.

References `xCenter`, and `yCenter`.

```
73     {
74         return QPointF( xCenter, yCenter );
75     }
```

### 9.68.3.2 **void KDChart::ZoomParameters::setCenter** (const **QPointF** & *center*)

Definition at line 67 of file KDChartZoomParameters.h.

References `xCenter`, and `yCenter`.

```
68     {
69         xCenter = center.x();
70         yCenter = center.y();
71     }
```

## 9.68.4 Member Data Documentation

### 9.68.4.1 **double KDChart::ZoomParameters::xCenter**

Definition at line 80 of file KDChartZoomParameters.h.

Referenced by `center()`, and `setCenter()`.

### 9.68.4.2 **double KDChart::ZoomParameters::xFactor**

Definition at line 77 of file KDChartZoomParameters.h.

### 9.68.4.3 **double KDChart::ZoomParameters::yCenter**

Definition at line 81 of file KDChartZoomParameters.h.

Referenced by `center()`, and `setCenter()`.



#### 9.68.4.4 double [KDChart::ZoomParameters::yFactor](#)

Definition at line 78 of file [KDChartZoomParameters.h](#).

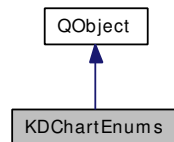
The documentation for this class was generated from the following file:

- [KDChartZoomParameters.h](#)

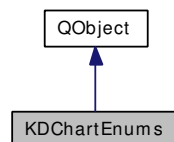
## 9.69 KDChartEnums Class Reference

```
#include <KDChartEnums.h>
```

Inheritance diagram for KDChartEnums:



Collaboration diagram for KDChartEnums:



### 9.69.1 Detailed Description

Project global class providing some enums needed both by KDChartParams and by KDChartCustomBox.

Definition at line 46 of file KDChartEnums.h.

#### Public Types

- enum [GranularitySequence](#) {  
[GranularitySequence\\_10\\_20](#),  
[GranularitySequence\\_10\\_50](#),  
[GranularitySequence\\_25\\_50](#),  
[GranularitySequence\\_125\\_25](#),  
[GranularitySequenceIrregular](#) }

*GranularitySequence specifies the values, that may be applied, to determine a step width within a given data range.*

- enum [MeasureCalculationMode](#) {  
[MeasureCalculationModeAbsolute](#),  
[MeasureCalculationModeRelative](#),  
[MeasureCalculationModeAuto](#),  
[MeasureCalculationModeAutoArea](#),  
[MeasureCalculationModeAutoOrientation](#) }

*Measure calculation mode: the way how the absolute value of a [KDChart::Measure](#) is determined during KD Chart's internal geometry calculation time.*

- enum [MeasureOrientation](#) {  
[MeasureOrientationAuto](#),  
[MeasureOrientationHorizontal](#),  
[MeasureOrientationVertical](#),  
[MeasureOrientationMinimum](#),  
[MeasureOrientationMaximum](#) }

*Measure orientation mode: the way how the absolute value of a [KDChart::Measure](#) is determined during KD Chart's internal geometry calculation time.*

- enum [PositionValue](#) {  
[PositionUnknown](#) = 0,  
[PositionCenter](#) = 1,  
[PositionNorthWest](#) = 2,  
[PositionNorth](#) = 3,  
[PositionNorthEast](#) = 4,  
[PositionEast](#) = 5,  
[PositionSouthEast](#) = 6,  
[PositionSouth](#) = 7,  
[PositionSouthWest](#) = 8,  
[PositionWest](#) = 9,  
[PositionFloating](#) = 10 }

*Numerical values of the static [KDChart::Position](#) instances, for using a [Position::value\(\)](#) with a [switch\(\)](#) statement.*

- enum [TextLayoutPolicy](#) {  
[LayoutJustOverwrite](#),  
[LayoutPolicyRotate](#),  
[LayoutPolicyShiftVertically](#),  
[LayoutPolicyShiftHorizontally](#),  
[LayoutPolicyShrinkFontSize](#) }

*Text layout policy: what to do if text that is to be drawn would cover neighboring text or neighboring areas.*

## Static Public Member Functions

- static QString [granularitySequenceToString](#) ([GranularitySequence](#) sequence)  
*Converts the specified granularity sequence enum to a string representation.*
- static QString [layoutPolicyToString](#) ([TextLayoutPolicy](#) type)  
*Converts the specified text layout policy enum to a string representation.*
- static QString [measureCalculationModeToString](#) ([MeasureCalculationMode](#) mode)  
*Converts the specified measure calculation mode enum to a string representation.*
- static QString [measureOrientationToString](#) ([MeasureOrientation](#) mode)

*Converts the specified measure orientation enum to a string representation.*

- static [GranularitySequence](#) [stringToGranularitySequence](#) (const QString &string)  
*Converts the specified string to a granularity sequence enum value.*
- static [TextLayoutPolicy](#) [stringToLayoutPolicy](#) (const QString &string)  
*Converts the specified string to a text layout policy enum value.*
- static [MeasureCalculationMode](#) [stringToMeasureCalculationMode](#) (const QString &string)  
*Converts the specified string to a measure calculation mode enum value.*
- static [MeasureOrientation](#) [stringToMeasureOrientation](#) (const QString &string)  
*Converts the specified string to a measure orientation enum value.*

## 9.69.2 Member Enumeration Documentation

### 9.69.2.1 enum [KDChartEnums::GranularitySequence](#)

GranularitySequence specifies the values, that may be applied, to determine a step width within a given data range.

#### Note:

Granularity with can be set for Linear axis calculation mode only, there is no way to specify a step width for Logarithmic axes.

Value occurring in the GranularitySequence names only are showing their respective relation ship. For real data they will most times not be used directly, but be multiplied by positive (or negative, resp.) powers of ten.

A granularity sequence is a sequence of values from the following set: 1, 1.25, 2, 2.5, 5.

The reason for using one of the following three pre-defined granularity sequences (instead of just using the best matching step width) is to follow a simple rule: If scaling becomes finer (== smaller step width) no value, that has been on a grid line before, shall loose its line and be NOT on a grid line anymore!

This means: Smaller step width may not remove any grid lines, but it may add additional lines in between.

- [GranularitySequence\\_10\\_20](#) Step widths can be 1, or 2, but they never can be 2.5 nor 5, nor 1.25.
- [GranularitySequence\\_10\\_50](#) Step widths can be 1, or 5, but they never can be 2, nor 2.5, nor 1.25.
- [GranularitySequence\\_25\\_50](#) Step widths can be 2.5, or 5, but they never can be 1, nor 2, nor 1.25.
- [GranularitySequence\\_125\\_25](#) Step widths can be 1.25 or 2.5 but they never can be 1, nor 2, nor 5.
- [GranularitySequenceIrregular](#) Step widths can be all of these values: 1, or 1.25, or 2, or 2.5, or 5.

**Note:**

When ever possible, try to avoid using GranularitySequenceIrregular! Allowing all possible step values, using this granularity sequence involves a serious risk: Your users might be irritated due to 'jumping' grid lines, when step size is changed from 2.5 to 2 (or vice versa, resp.). In case you still want to use GranularitySequenceIrregular just make sure to NOT draw any sub-grid lines, because in most cases you will get not-matching step widths for the sub-grid. In short: GranularitySequenceIrregular can safely be used if your data range is not changing at all AND (b) you will not allow the coordinate plane to be zoomed AND (c) you are not displaying any sub-grid lines.

Since you probably like having the value 1 as an allowed step width, the granularity sequence decision boils down to a boolean question:

- To get ten divided by five you use GranularitySequence\_10\_20, while
- for having it divided by two GranularitySequence\_10\_50 is your choice.

**Enumerator:**

*GranularitySequence\_10\_20*

*GranularitySequence\_10\_50*

*GranularitySequence\_25\_50*

*GranularitySequence\_125\_25*

*GranularitySequenceIrregular*

Definition at line 101 of file KDChartEnums.h.

```

101                                     {
102     GranularitySequence_10_20,
103     GranularitySequence_10_50,
104     GranularitySequence_25_50,
105     GranularitySequence_125_25,
106     GranularitySequenceIrregular };

```

**9.69.2.2 enum [KDChartEnums::MeasureCalculationMode](#)**

Measure calculation mode: the way how the absolute value of a [KDChart::Measure](#) is determined during KD Chart's internal geometry calculation time.

[KDChart::Measure](#) values either are relative (calculated in relation to a given AbstractArea), or they are absolute (used as fixed values).

Values stored in relative measure always are interpreted as per-mille of a reference area's height (or width, resp.) depending on the orientation set for the [KDChart::Measure](#).

- `MeasureCalculationModeAbsolute` Value set by `setValue()` is absolute, to be used unchanged.
- `MeasureCalculationModeRelative` Value is relative, the reference area is specified by `setReferenceArea()`, and orientation specified by `setOrientation()`.
- `MeasureCalculationModeAuto` Value is relative, KD Chart will automatically determine which reference area to use, and it will determine the orientation too.

- `MeasureCalculationModeAutoArea` Value is relative, Orientation is specified by `setOrientation()`, and KD Chart will automatically determine which reference area to use.
- `MeasureCalculationModeAutoOrientation` Value is relative, Area is specified by `setReferenceArea()`, and KD Chart will automatically determine which orientation to use.

See also:

[KDChart::Measure::setCalculationMode](#)

Enumerator:

*MeasureCalculationModeAbsolute*

*MeasureCalculationModeRelative*

*MeasureCalculationModeAuto*

*MeasureCalculationModeAutoArea*

*MeasureCalculationModeAutoOrientation*

Definition at line 229 of file `KDChartEnums.h`.

```

229                                     { MeasureCalculationModeAbsolute,
230     MeasureCalculationModeRelative,
231     MeasureCalculationModeAuto,
232     MeasureCalculationModeAutoArea,
233     MeasureCalculationModeAutoOrientation };

```

### 9.69.2.3 enum [KDChartEnums::MeasureOrientation](#)

Measure orientation mode: the way how the absolute value of a [KDChart::Measure](#) is determined during KD Chart's internal geometry calculation time.

[KDChart::Measure](#) values either are relative (calculated in relation to a given `AbstractArea`), or they are absolute (used as fixed values).

Values stored in relative measure take into account the width (and/or the height, resp.) of a so-called reference area, that is either specified by [KDChart::Measure::setReferenceArea](#), or determined by KD Chart automatically, respectively.

- `MeasureOrientationAuto` Value is calculated, based upon the width (or on the height, resp.) of the reference area: KD Chart will automatically determine an appropriate way.
- `MeasureOrientationHorizontal` Value is calculated, based upon the width of the reference area.
- `MeasureOrientationVertical` Value is calculated, based upon the height of the reference area.
- `MeasureOrientationMinimum` Value is calculated, based upon the width (or on the height, resp.) of the reference area - whichever is smaller.
- `MeasureOrientationMaximum` Value is calculated, based upon the width (or on the height, resp.) of the reference area - whichever is smaller.

See also:

[KDChart::Measure::setOrientationMode](#)

**Enumerator:**

*MeasureOrientationAuto*  
*MeasureOrientationHorizontal*  
*MeasureOrientationVertical*  
*MeasureOrientationMinimum*  
*MeasureOrientationMaximum*

Definition at line 298 of file KDChartEnums.h.

```

298                                     { MeasureOrientationAuto,
299     MeasureOrientationHorizontal,
300     MeasureOrientationVertical,
301     MeasureOrientationMinimum,
302     MeasureOrientationMaximum };

```

#### 9.69.2.4 enum [KDChartEnums::PositionValue](#)

Numerical values of the static [KDChart::Position](#) instances, for using a [Position::value\(\)](#) with a [switch\(\)](#) statement.

**See also:**

[Position](#)

**Enumerator:**

*PositionUnknown*  
*PositionCenter*  
*PositionNorthWest*  
*PositionNorth*  
*PositionNorthEast*  
*PositionEast*  
*PositionSouthEast*  
*PositionSouth*  
*PositionSouthWest*  
*PositionWest*  
*PositionFloating*

Definition at line 199 of file KDChartEnums.h.

```

199                                     {
200     PositionUnknown    = 0,
201     PositionCenter     = 1,
202     PositionNorthWest  = 2,
203     PositionNorth     = 3,
204     PositionNorthEast  = 4,
205     PositionEast       = 5,
206     PositionSouthEast  = 6,
207     PositionSouth     = 7,
208     PositionSouthWest  = 8,
209     PositionWest       = 9,
210     PositionFloating   =10
211 };

```

### 9.69.2.5 enum [KDChartEnums::TextLayoutPolicy](#)

Text layout policy: what to do if text that is to be drawn would cover neighboring text or neighboring areas.

- `LayoutJustOverwrite` Just ignore the layout collision and write the text nevertheless.
- `LayoutPolicyRotate` Try counter-clockwise rotation to make the text fit into the space.
- `LayoutPolicyShiftVertically` Shift the text baseline upwards (or downwards, resp.) and draw a connector line between the text and its anchor.
- `LayoutPolicyShiftHorizontally` Shift the text baseline to the left (or to the right, resp.) and draw a connector line between the text and its anchor.
- `LayoutPolicyShrinkFontSize` Reduce the text font size.

See also:

`KDChartParams::setPrintDataValues`

Enumerator:

*`LayoutJustOverwrite`*

*`LayoutPolicyRotate`*

*`LayoutPolicyShiftVertically`*

*`LayoutPolicyShiftHorizontally`*

*`LayoutPolicyShrinkFontSize`*

Definition at line 168 of file `KDChartEnums.h`.

```
168                                     { LayoutJustOverwrite,
169     LayoutPolicyRotate,
170     LayoutPolicyShiftVertically,
171     LayoutPolicyShiftHorizontally,
172     LayoutPolicyShrinkFontSize };
```

## 9.69.3 Member Function Documentation

### 9.69.3.1 static QString [KDChartEnums::granularitySequenceToString \(GranularitySequence sequence\)](#) [static]

Converts the specified granularity sequence enum to a string representation.

Parameters:

*`sequence`* the granularity sequence enum to convert

Returns:

the string representation of the granularity sequence

Definition at line 115 of file `KDChartEnums.h`.

Referenced by `KDChart::operator<<()`.



```

115                                     {
116         switch( sequence ) {
117             case GranularitySequence_10_20:
118                 return QString::fromLatin1("GranularitySequence_10_20");
119             case GranularitySequence_10_50:
120                 return QString::fromLatin1("GranularitySequence_10_50");
121             case GranularitySequence_25_50:
122                 return QString::fromLatin1("GranularitySequence_25_50");
123             case GranularitySequence_125_25:
124                 return QString::fromLatin1("GranularitySequence_125_25");
125             case GranularitySequenceIrregular:
126                 return QString::fromLatin1("GranularitySequenceIrregular");
127             default: // should not happen
128                 qDebug( "Unknown granularity sequence" );
129                 return QString::fromLatin1("GranularitySequence_10_20");
130             }
131     }

```

### 9.69.3.2 static QString KDChartEnums::layoutPolicyToString ([TextLayoutPolicy](#) type) [static]

Converts the specified text layout policy enum to a string representation.

#### Parameters:

*type* the text layout policy to convert

#### Returns:

the string representation of the text layout policy enum

### 9.69.3.3 static QString KDChartEnums::measureCalculationModeToString ([MeasureCalculationMode](#) mode) [static]

Converts the specified measure calculation mode enum to a string representation.

#### Parameters:

*mode* the measure calculation mode to convert

#### Returns:

the string representation of the Measure calculation mode enum

Definition at line 242 of file KDChartEnums.h.

```

242                                     {
243         switch( mode ) {
244             case MeasureCalculationModeAbsolute:
245                 return QString::fromLatin1("MeasureCalculationModeAbsolute");
246             case MeasureCalculationModeAuto:
247                 return QString::fromLatin1("MeasureCalculationModeAuto");
248             case MeasureCalculationModeAutoArea:
249                 return QString::fromLatin1("MeasureCalculationModeAutoArea");
250             case MeasureCalculationModeAutoOrientation:
251                 return QString::fromLatin1("MeasureCalculationModeAutoOrientation");
252             case MeasureCalculationModeRelative:
253                 return QString::fromLatin1("MeasureCalculationModeRelative");

```

```

254         default: // should not happen
255         qDebug( "Unknown measure calculation mode" );
256         return QString::fromLatin1("MeasureCalculationModeAuto");
257     }
258 }

```

#### 9.69.3.4 static QString KDChartEnums::measureOrientationToString ([MeasureOrientation mode](#)) [static]

Converts the specified measure orientation enum to a string representation.

##### Parameters:

*mode* the measure orientation to convert

##### Returns:

the string representation of the measure orientation enum

Definition at line 311 of file KDChartEnums.h.

```

311                                                                 {
312     switch( mode ) {
313         case MeasureOrientationAuto:
314             return QString::fromLatin1("MeasureOrientationAuto");
315         case MeasureOrientationHorizontal:
316             return QString::fromLatin1("MeasureOrientationHorizontal");
317         case MeasureOrientationVertical:
318             return QString::fromLatin1("MeasureOrientationVertical");
319         case MeasureOrientationMinimum:
320             return QString::fromLatin1("MeasureOrientationMinimum");
321         case MeasureOrientationMaximum:
322             return QString::fromLatin1("MeasureOrientationMaximum");
323         default: // should not happen
324             qDebug( "Unknown measure orientation mode" );
325             return QString::fromLatin1("MeasureOrientationAuto");
326     }
327 }

```

#### 9.69.3.5 static [GranularitySequence](#) KDChartEnums::stringToGranularitySequence (const QString & *string*) [static]

Converts the specified string to a granularity sequence enum value.

##### Parameters:

*string* the string to convert

##### Returns:

the granularity sequence enum value

Definition at line 140 of file KDChartEnums.h.

```

140                                                                 {
141     if( string == QString::fromLatin1("GranularitySequence_10_20") )

```

```

142         return GranularitySequence_10_20;
143     if( string == QString::fromLatin1("GranularitySequence_10_50") )
144         return GranularitySequence_10_50;
145     if( string == QString::fromLatin1("GranularitySequence_25_50") )
146         return GranularitySequence_25_50;
147     if( string == QString::fromLatin1("GranularitySequence_125") )
148         return GranularitySequence_125_25;
149     if( string == QString::fromLatin1("GranularitySequenceIrregular") )
150         return GranularitySequenceIrregular;
151     // default, should not happen
152     return GranularitySequence_10_20;
153 }

```

#### 9.69.3.6 static [TextLayoutPolicy](#) KDChartEnums::stringToLayoutPolicy (const QString & string) [static]

Converts the specified string to a text layout policy enum value.

##### Parameters:

*string* the string to convert

##### Returns:

the text layout policy enum value

#### 9.69.3.7 static [MeasureCalculationMode](#) KDChartEnums::stringToMeasureCalculationMode (const QString & string) [static]

Converts the specified string to a measure calculation mode enum value.

##### Parameters:

*string* the string to convert

##### Returns:

the measure calculation mode enum value

Definition at line 267 of file KDChartEnums.h.

```

267                                                                 {
268     if( string == QString::fromLatin1("MeasureCalculationModeAbsolute") )
269         return MeasureCalculationModeAbsolute;
270     if( string == QString::fromLatin1("MeasureCalculationModeAuto") )
271         return MeasureCalculationModeAuto;
272     if( string == QString::fromLatin1("MeasureCalculationModeAutoArea") )
273         return MeasureCalculationModeAutoArea;
274     if( string == QString::fromLatin1("MeasureCalculationModeAutoOrientation") )
275         return MeasureCalculationModeAutoOrientation;
276     if( string == QString::fromLatin1("MeasureCalculationModeRelative") )
277         return MeasureCalculationModeRelative;
278     // default, should not happen
279     return MeasureCalculationModeAuto;
280 }

```

### 9.69.3.8 static [MeasureOrientation](#) KDChartEnums::stringToMeasureOrientation (const QString & *string*) [static]

Converts the specified string to a measure orientation enum value.

#### Parameters:

*string* the string to convert

#### Returns:

the measure orientation enum value

Definition at line 336 of file KDChartEnums.h.

```
336                                                                                                     {
337     if( string == QString::fromLatin1("MeasureOrientationAuto") )
338         return MeasureOrientationAuto;
339     if( string == QString::fromLatin1("MeasureOrientationHorizontal") )
340         return MeasureOrientationHorizontal;
341     if( string == QString::fromLatin1("MeasureOrientationVertical") )
342         return MeasureOrientationVertical;
343     if( string == QString::fromLatin1("MeasureOrientationMinimum") )
344         return MeasureOrientationMinimum;
345     if( string == QString::fromLatin1("MeasureOrientationMaximum") )
346         return MeasureOrientationMaximum;
347     // default, should not happen
348     return MeasureOrientationAuto;
349 }
```

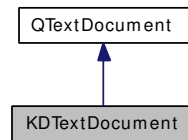
The documentation for this class was generated from the following file:

- [KDChartEnums.h](#)

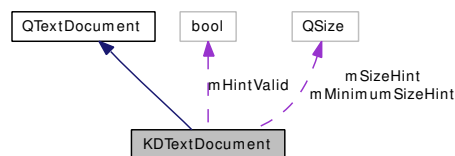
## 9.70 KDDocument Class Reference

```
#include <KDDocument.h>
```

Inheritance diagram for KDDocument:



Collaboration diagram for KDDocument:



### 9.70.1 Detailed Description

KDDocument is an internally used enhanced QTextDocument.

Definition at line 47 of file KDDocument.h.

### Public Member Functions

- KDDocument (const QString &text, QObject \*parent=0)
- KDDocument (QObject \*parent=0)
- QSize minimumSizeHint ()
- QSize sizeHint ()
- ~KDDocument ()

### 9.70.2 Constructor & Destructor Documentation

#### 9.70.2.1 KDDocument::KDDocument (QObject \*parent = 0) [explicit]

Definition at line 38 of file KDDocument.cpp.

```

39     : QTextDocument ( p ),
40       mHintValid( false ),
41       mSizeHint (),
42       mMinimumSizeHint ()
43 {
44
45 }
```

### 9.70.2.2 **KDTextDocument::KDTextDocument** (const QString & *text*, [QObject](#) \* *parent* = 0) [explicit]

Definition at line 47 of file KDTextDocument.cpp.

```
48     : QTextDocument( text, p ),
49     mHintValid( false ),
50     mSizeHint(),
51     mMinimumSizeHint()
52 {
53
54 }
```

### 9.70.2.3 **KDTextDocument::~~KDTextDocument** ()

Definition at line 56 of file KDTextDocument.cpp.

```
56 {}
```

## 9.70.3 Member Function Documentation

### 9.70.3.1 **QSize KDTextDocument::minimumSizeHint** ()

Definition at line 66 of file KDTextDocument.cpp.

Referenced by sizeHint().

```
67 {
68     /*
69     QTextCursor cursor( this );
70     if( ! cursor.atEnd() )
71         cursor.movePosition( QTextCursor::NextBlock );
72     qDebug() << "KDTextDocument::minimumSizeHint() found:" << cursor.block().text();
73     QSizeF s( documentLayout()->blockBoundingRect( cursor.block() ).size() );
74     qDebug() << "KDTextDocument::minimumSizeHint() found rect" << documentLayout()->blockBoundingRect (
75     return QSize( static_cast<int>(s.width()),
76                 static_cast<int>(s.height()) );
77     */
78
79     if( mHintValid )
80         return mMinimumSizeHint;
81
82     mHintValid = true;
83     mSizeHint = sizeForWidth( -1 );
84     QSize sz(-1, -1);
85
86     // PENDING(kalle) Cache
87     sz.rwidth() = sizeForWidth( 0 ).width();
88     sz.rheight() = sizeForWidth( 32000 ).height();
89     if( mSizeHint.height() < sz.height() )
90         sz.rheight() = mSizeHint.height();
91
92     mMinimumSizeHint = sz;
93     return sz;
94 }
```

### 9.70.3.2 QSize KDTextDocument::sizeHint ()

Definition at line 59 of file KDTextDocument.cpp.

References [minimumSizeHint\(\)](#).

```
60 {  
61     if( !mHintValid )  
62         (void)minimumSizeHint();  
63     return mSizeHint;  
64 }
```

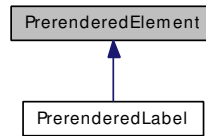
The documentation for this class was generated from the following files:

- [KDTextDocument.h](#)
- [KDTextDocument.cpp](#)

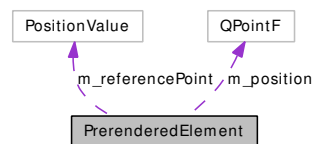
## 9.71 PrerenderedElement Class Reference

```
#include <KDChartTextLabelCache.h>
```

Inheritance diagram for PrerenderedElement:



Collaboration diagram for PrerenderedElement:



### 9.71.1 Detailed Description

base class for prerendered elements like labels, pixmaps, markers, etc.

Definition at line 40 of file `KDChartTextLabelCache.h`.

#### Public Member Functions

- virtual const QPixmap & [pixmap](#) () const=0  
*Returns the rendered element.*
- const QPointF & [position](#) () const  
*Get the position of the element.*
- [PrerenderedElement](#) ()
- [KDChartEnums::PositionValue](#) [referencePoint](#) () const  
*Get the reference point of the element.*
- virtual QPointF [referencePointLocation](#) ([KDChartEnums::PositionValue](#)) const=0  
*Return the location of the reference point relatively to the pixmap's origin.*
- void [setPosition](#) (const QPointF &position)  
*Set the position of the element.*
- void [setReferencePoint](#) ([KDChartEnums::PositionValue](#))  
*Set the reference point of the element.*
- virtual [~PrerenderedElement](#) ()



## Protected Member Functions

- virtual void [invalidate](#) () const=0  
*[invalidate\(\)](#) needs to be called if any of the properties that determine the visual appearance of the prerendered element change.*

### 9.71.2 Constructor & Destructor Documentation

#### 9.71.2.1 PrerenderedElement::PrerenderedElement ()

Definition at line 59 of file KDChartTextLabelCache.cpp.

```
60      : m_referencePoint( KDChartEnums::PositionNorthWest )
61  {
62  }
```

#### 9.71.2.2 virtual PrerenderedElement::~~PrerenderedElement () [virtual]

Definition at line 43 of file KDChartTextLabelCache.h.

```
43 {}
```

### 9.71.3 Member Function Documentation

#### 9.71.3.1 virtual void PrerenderedElement::invalidate () const [protected, pure virtual]

[invalidate\(\)](#) needs to be called if any of the properties that determine the visual appearance of the prerendered element change.

It can be called for a const object, as objects may need to force recalculation of the pixmap.

Implemented in [PrerenderedLabel](#).

#### 9.71.3.2 virtual const QPixmap& PrerenderedElement::pixmap () const [pure virtual]

Returns the rendered element.

If any of the properties have change, the element will be regenerated.

Implemented in [PrerenderedLabel](#).

#### 9.71.3.3 const QPointF & PrerenderedElement::position () const

Get the position of the element.

Definition at line 69 of file KDChartTextLabelCache.cpp.

Referenced by KDChart::TernaryAxis::paintCtx().

```
70 {
71     return m_position;
72 }
```

#### 9.71.3.4 [KDChartEnums::PositionValue](#) PrerenderedElement::referencePoint () const

Get the reference point of the element.

Definition at line 79 of file KDChartTextLabelCache.cpp.

Referenced by PrerenderedLabel::referencePointLocation().

```
80 {  
81     return m_referencePoint;  
82 }
```

#### 9.71.3.5 [virtual QPointF](#) PrerenderedElement::referencePointLocation ([KDChartEnums::PositionValue](#)) const [pure virtual]

Return the location of the reference point relatively to the pixmap's origin.

Implemented in [PrerenderedLabel](#).

#### 9.71.3.6 [void](#) PrerenderedElement::setPosition (const [QPointF](#) & *position*)

Set the position of the element.

Definition at line 64 of file KDChartTextLabelCache.cpp.

```
65 {    // this does not invalidate the element  
66     m_position = position;  
67 }
```

#### 9.71.3.7 [void](#) PrerenderedElement::setReferencePoint ([KDChartEnums::PositionValue](#))

Set the reference point of the element.

Every element has nine possible reference points (all compass directions, plus the center).

Definition at line 74 of file KDChartTextLabelCache.cpp.

```
75 {    // this does not invalidate the element  
76     m_referencePoint = point;  
77 }
```

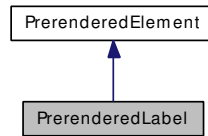
The documentation for this class was generated from the following files:

- [KDChartTextLabelCache.h](#)
- [KDChartTextLabelCache.cpp](#)

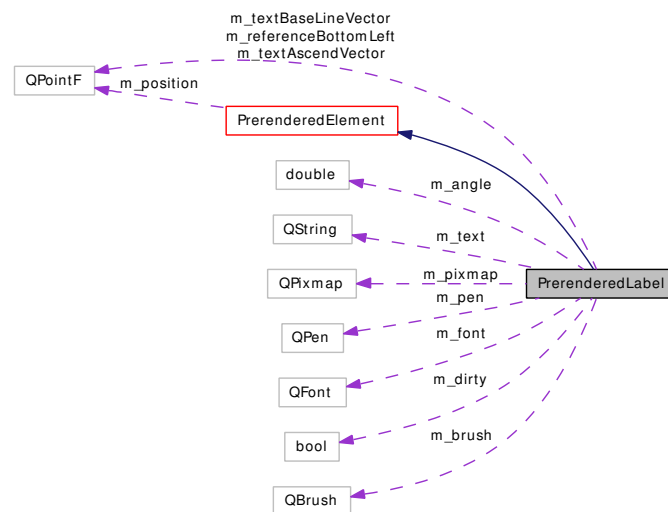
## 9.72 PrerenderedLabel Class Reference

```
#include <KDChartTextLabelCache.h>
```

Inheritance diagram for PrerenderedLabel:



Collaboration diagram for PrerenderedLabel:



### 9.72.1 Detailed Description

`PrerenderedLabel` is an internal `KDChart` class that simplifies creation and caching of cached text labels.

It provides reference points to anchor the text to other elements. Reference points use the positions defined in `KDChartEnums`.

Usage:

```
double angle = 90.0;
CachedLabel label;
label.paint( font, tr("Label"), angle );
```

Definition at line 101 of file `KDChartTextLabelCache.h`.

### Public Member Functions

- double `angle` () const

**Returns:**

*the label's angle in degrees*

- `const QBrush & brush () const`

**Returns:**

*the label's brush*

- `const QFont & font () const`

**Returns:**

*the label's font.*

- `const QPen & pen () const`
- `const QPixmap & pixmap () const`

*Returns the rendered element.*

- `const QPointF & position () const`

*Get the position of the element.*

- `PrerenderedLabel ()`
- `KDChartEnums::PositionValue referencePoint () const`

*Get the reference point of the element.*

- `QPointF referencePointLocation () const`
- `QPointF referencePointLocation (KDChartEnums::PositionValue position) const`

*Return the location of the reference point relatively to the pixmap's origin.*

- `void setAngle (double angle)`  
*Sets the angle of the label to angle degrees.*

- `void setBrush (const QBrush &brush)`  
*Sets the label's brush to brush.*

- `void setFont (const QFont &font)`  
*Sets the label's font to font.*

- `void setPen (const QPen &)`
- `void setPosition (const QPointF &position)`  
*Set the position of the element.*

- `void setReferencePoint (KDChartEnums::PositionValue)`  
*Set the reference point of the element.*

- `void setText (const QString &text)`  
*Sets the label's text to text.*

- `const QString & text () const`

**Returns:**

*the label's text*

- `~PrerenderedLabel ()`

## Protected Member Functions

- void [invalidate](#) () const

*Invalidates the prerendered data, forces re-rendering.*

## 9.72.2 Constructor & Destructor Documentation

### 9.72.2.1 PrerenderedLabel::PrerenderedLabel ()

Definition at line 84 of file KDChartTextLabelCache.cpp.

```
85     : PrerenderedElement()
86     , m_dirty( true )
87     , m_font( qApp->font() )
88     , m_brush( Qt::black )
89     , m_pen( Qt::black ) // do not use anything invisible
90     , m_angle( 0.0 )
91 {
92 }
```

### 9.72.2.2 PrerenderedLabel::~~PrerenderedLabel ()

Definition at line 94 of file KDChartTextLabelCache.cpp.

References `DUMP_CACHE_STATS`.

```
95 {
96     DUMP_CACHE_STATS;
97 }
```

## 9.72.3 Member Function Documentation

### 9.72.3.1 double PrerenderedLabel::angle () const

#### Returns:

the label's angle in degrees

Definition at line 170 of file KDChartTextLabelCache.cpp.

```
171 {
172     return m_angle;
173 }
```

### 9.72.3.2 const QBrush & PrerenderedLabel::brush () const

#### Returns:

the label's brush

Definition at line 153 of file KDChartTextLabelCache.cpp.

```
154 {  
155     return m_brush;  
156 }
```

### 9.72.3.3 const QFont & PrerenderedLabel::font () const

#### Returns:

the label's font.

Definition at line 119 of file KDChartTextLabelCache.cpp.

```
120 {  
121     return m_font;  
122 }
```

### 9.72.3.4 void PrerenderedLabel::invalidate () const [protected, virtual]

Invalidates the prerendered data, forces re-rendering.

Implements [PrerenderedElement](#).

Definition at line 102 of file KDChartTextLabelCache.cpp.

Referenced by [setAngle\(\)](#), [setBrush\(\)](#), [setFont\(\)](#), and [setText\(\)](#).

```
103 {  
104     m_dirty = true;  
105 }
```

### 9.72.3.5 const QPen& PrerenderedLabel::pen () const

### 9.72.3.6 const QPixmap & PrerenderedLabel::pixmap () const [virtual]

Returns the rendered element.

If any of the properties have change, the element will be regenerated.

Implements [PrerenderedElement](#).

Definition at line 175 of file KDChartTextLabelCache.cpp.

References [INC\\_HIT\\_COUNT](#), and [INC\\_MISS\\_COUNT](#).

Referenced by [KDChart::TernaryAxis::paintCtx\(\)](#), and [KDChart::TernaryAxis::requiredMargins\(\)](#).

```
176 {  
177     if ( m_dirty ) {  
178         INC_MISS_COUNT;  
179         paint();  
180     } else {  
181         INC_HIT_COUNT;  
182     }  
183     return m_pixmap;  
184 }
```

### 9.72.3.7 `const QPointF & PrerenderedElement::position () const` [inherited]

Get the position of the element.

Definition at line 69 of file `KDChartTextLabelCache.cpp`.

Referenced by `KDChart::TernaryAxis::paintCtx()`.

```
70 {  
71     return m_position;  
72 }
```

### 9.72.3.8 `KDChartEnums::PositionValue PrerenderedElement::referencePoint () const` [inherited]

Get the reference point of the element.

Definition at line 79 of file `KDChartTextLabelCache.cpp`.

Referenced by `referencePointLocation()`.

```
80 {  
81     return m_referencePoint;  
82 }
```

### 9.72.3.9 `QPointF PrerenderedLabel::referencePointLocation () const`

Definition at line 289 of file `KDChartTextLabelCache.cpp`.

References `PrerenderedElement::referencePoint()`.

```
290 {  
291     return referencePointLocation( referencePoint() );  
292 }
```

### 9.72.3.10 `QPointF PrerenderedLabel::referencePointLocation (KDChartEnums::PositionValue position) const` [virtual]

Return the location of the reference point relatively to the pixmap's origin.

Implements [PrerenderedElement](#).

Definition at line 294 of file `KDChartTextLabelCache.cpp`.

References `INC_HIT_COUNT`, `INC_MISS_COUNT`, `KDChartEnums::PositionCenter`, `KDChartEnums::PositionEast`, `KDChartEnums::PositionFloating`, `KDChartEnums::PositionNorth`, `KDChartEnums::PositionNorthEast`, `KDChartEnums::PositionNorthWest`, `KDChartEnums::PositionSouth`, `KDChartEnums::PositionSouthEast`, `KDChartEnums::PositionSouthWest`, and `KDChartEnums::PositionUnknown`.

Referenced by `KDChart::TernaryAxis::paintCtx()`, and `KDChart::TernaryAxis::requiredMargins()`.

```
295 {  
296     if ( m_dirty ) {  
297         INC_MISS_COUNT;  
298         paint();  
299     }
```

```

299     } else {
300         INC_HIT_COUNT;
301     }
302
303     switch( position ) {
304     case KDChartEnums::PositionCenter:
305         return m_referenceBottomLeft + 0.5 * m_textBaseLineVector + 0.5 * m_textAscendVector;
306     case KDChartEnums::PositionNorthWest:
307         return m_referenceBottomLeft + m_textAscendVector;
308     case KDChartEnums::PositionNorth:
309         return m_referenceBottomLeft + 0.5 * m_textBaseLineVector + m_textAscendVector;
310     case KDChartEnums::PositionNorthEast:
311         return m_referenceBottomLeft + m_textBaseLineVector + m_textAscendVector;
312     case KDChartEnums::PositionEast:
313         return m_referenceBottomLeft + 0.5 * m_textAscendVector;
314     case KDChartEnums::PositionSouthEast:
315         return m_referenceBottomLeft + m_textBaseLineVector;
316     case KDChartEnums::PositionSouth:
317         return m_referenceBottomLeft + 0.5 * m_textBaseLineVector;
318     case KDChartEnums::PositionSouthWest:
319         return m_referenceBottomLeft;
320     case KDChartEnums::PositionWest:
321         return m_referenceBottomLeft + m_textBaseLineVector + 0.5 * m_textAscendVector;
322
323     case KDChartEnums::PositionUnknown: // intentional fall-through
324     case KDChartEnums::PositionFloating: // intentional fall-through
325     default:
326         return QPointF();
327     }
328 }

```

### 9.72.3.11 void PrerenderedLabel::setAngle (double *angle*)

Sets the angle of the label to *angle* degrees.

Definition at line 161 of file KDChartTextLabelCache.cpp.

References invalidate().

```

162 {
163     m_angle = angle;
164     invalidate();
165 }

```

### 9.72.3.12 void PrerenderedLabel::setBrush (const QBrush & *brush*)

Sets the label's brush to *brush*.

Definition at line 144 of file KDChartTextLabelCache.cpp.

References invalidate().

```

145 {
146     m_brush = brush;
147     invalidate();
148 }

```



**9.72.3.13 void PrerenderedLabel::setFont (const QFont & *font*)**

Sets the label's font to *font*.

Definition at line 110 of file KDChartTextLabelCache.cpp.

References invalidate().

```
111 {  
112     m_font = font;  
113     invalidate();  
114 }
```

**9.72.3.14 void PrerenderedLabel::setPen (const QPen &)****9.72.3.15 void PrerenderedElement::setPosition (const QPointF & *position*)** [inherited]

Set the position of the element.

Definition at line 64 of file KDChartTextLabelCache.cpp.

```
65 {    // this does not invalidate the element  
66     m_position = position;  
67 }
```

**9.72.3.16 void PrerenderedElement::setReferencePoint ([KDChartEnums::PositionValue](#))**  
[inherited]

Set the reference point of the element.

Every element has nine possible reference points (all compass directions, plus the center.

Definition at line 74 of file KDChartTextLabelCache.cpp.

```
75 {    // this does not invalidate the element  
76     m_referencePoint = point;  
77 }
```

**9.72.3.17 void PrerenderedLabel::setText (const QString & *text*)**

Sets the label's text to *text*.

Definition at line 127 of file KDChartTextLabelCache.cpp.

References invalidate().

Referenced by KDChart::TernaryAxis::setPosition(), KDChart::TernaryAxis::setTitleText(), and KDChart::TernaryAxis::TernaryAxis().

```
128 {  
129     m_text = text;  
130     invalidate();  
131 }
```

**9.72.3.18 const QString & PrerenderedLabel::text () const****Returns:**

the label's text

Definition at line 136 of file KDChartTextLabelCache.cpp.

Referenced by KDChart::TernaryAxis::titleText().

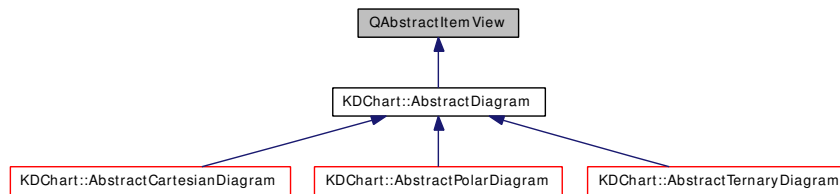
```
137 {  
138     return m_text;  
139 }
```

The documentation for this class was generated from the following files:

- [KDChartTextLabelCache.h](#)
- [KDChartTextLabelCache.cpp](#)

## 9.73 QAbstractItemView Class Reference

Inheritance diagram for QAbstractItemView:



### 9.73.1 Detailed Description

Class only listed here to document inheritance of some [KDChart](#) classes.

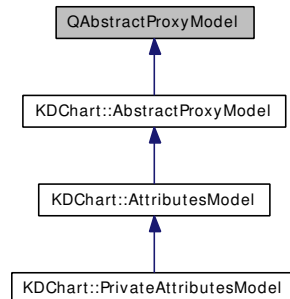
Please consult the respective Qt documentation for details: <http://doc.trolltech.com/>

The documentation for this class was generated from the following file:

- [KDChartChart.h](#)

## 9.74 QAbstractProxyModel Class Reference

Inheritance diagram for QAbstractProxyModel:



### 9.74.1 Detailed Description

Class only listed here to document inheritance of some [KDChart](#) classes.

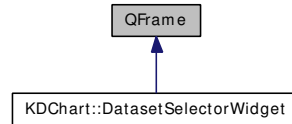
Please consult the respective Qt documentation for details: <http://doc.trolltech.com/>

The documentation for this class was generated from the following file:

- [KDChartChart.h](#)

## 9.75 QFrame Class Reference

Inheritance diagram for QFrame:



### 9.75.1 Detailed Description

Class only listed here to document inheritance of some [KDChart](#) classes.

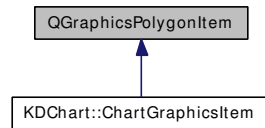
Please consult the respective Qt documentation for details: <http://doc.trolltech.com/>

The documentation for this class was generated from the following file:

- [KDChartChart.h](#)

## 9.76 QGraphicsPolygonItem Class Reference

Inheritance diagram for QGraphicsPolygonItem:



### 9.76.1 Detailed Description

Class only listed here to document inheritance of some [KDChart](#) classes.

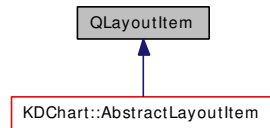
Please consult the respective Qt documentation for details: <http://doc.trolltech.com/>

The documentation for this class was generated from the following file:

- [KDChartChart.h](#)

## 9.77 QLayoutItem Class Reference

Inheritance diagram for QLayoutItem:



### 9.77.1 Detailed Description

Class only listed here to document inheritance of some [KDChart](#) classes.

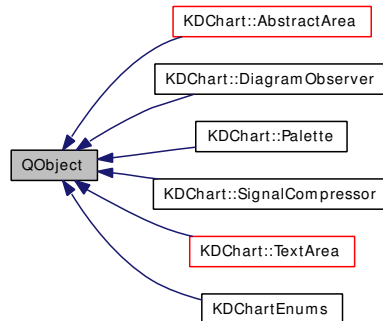
Please consult the respective Qt documentation for details: <http://doc.trolltech.com/>

The documentation for this class was generated from the following file:

- [KDChartChart.h](#)

## 9.78 QObject Class Reference

Inheritance diagram for QObject:



### 9.78.1 Detailed Description

Class only listed here to document inheritance of some [KDChart](#) classes.

Please consult the respective Qt documentation for details: <http://doc.trolltech.com/>

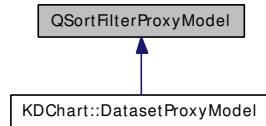
The documentation for this class was generated from the following file:

- [KDChartChart.h](#)



## 9.79 QSortFilterProxyModel Class Reference

Inheritance diagram for QSortFilterProxyModel:



### 9.79.1 Detailed Description

Class only listed here to document inheritance of some [KDChart](#) classes.

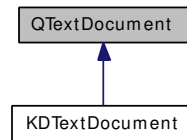
Please consult the respective Qt documentation for details: <http://doc.trolltech.com/>

The documentation for this class was generated from the following file:

- [KDChartChart.h](#)

## 9.80 QTextDocument Class Reference

Inheritance diagram for QTextDocument:



### 9.80.1 Detailed Description

Class only listed here to document inheritance of some [KDChart](#) classes.

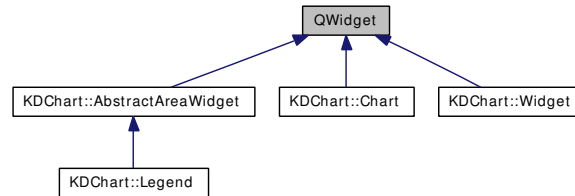
Please consult the respective Qt documentation for details: <http://doc.trolltech.com/>

The documentation for this class was generated from the following file:

- [KDChartChart.h](#)

## 9.81 QWidget Class Reference

Inheritance diagram for QWidget:



### 9.81.1 Detailed Description

Class only listed here to document inheritance of some [KDChart](#) classes.

Please consult the respective Qt documentation for details: <http://doc.trolltech.com/>

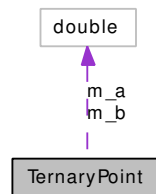
The documentation for this class was generated from the following file:

- [KDChartChart.h](#)

## 9.82 TernaryPoint Class Reference

```
#include <TernaryPoint.h>
```

Collaboration diagram for TernaryPoint:



### 9.82.1 Detailed Description

[TernaryPoint](#) defines a point within a ternary coordinate plane.

Definition at line 40 of file TernaryPoint.h.

### Public Member Functions

- double [a](#) () const
- double [b](#) () const
- double [c](#) () const
- bool [isValid](#) () const
- void [set](#) (double a, double b)
- [TernaryPoint](#) (double a, double b)
- [TernaryPoint](#) ()

### 9.82.2 Constructor & Destructor Documentation

#### 9.82.2.1 TernaryPoint::TernaryPoint ()

Definition at line 38 of file TernaryPoint.cpp.

References [isValid\(\)](#).

```

39      : m_a( -1.0 )
40      , m_b( -1.0 )
41  {
42      Q_ASSERT( !isValid() );
43  }
```

#### 9.82.2.2 TernaryPoint::TernaryPoint (double *a*, double *b*)

Definition at line 45 of file TernaryPoint.cpp.

References [set\(\)](#).

```
46      : m_a( -1.0 )  
47      , m_b( -1.0 )  
48 {  
49     set( a, b );  
50 }
```

### 9.82.3 Member Function Documentation

#### 9.82.3.1 double TernaryPoint::a () const

Definition at line 46 of file TernaryPoint.h.

Referenced by operator<<(), and translate().

```
46 { return m_a; }
```

#### 9.82.3.2 double TernaryPoint::b () const

Definition at line 47 of file TernaryPoint.h.

Referenced by operator<<(), and translate().

```
47 { return m_b; }
```

#### 9.82.3.3 double TernaryPoint::c () const

Definition at line 48 of file TernaryPoint.h.

Referenced by operator<<().

```
48 { return 1.0 - m_a - m_b; }
```

#### 9.82.3.4 bool TernaryPoint::isValid () const

Definition at line 67 of file TernaryPoint.cpp.

Referenced by operator<<(), set(), TernaryPoint(), and translate().

```
68 {  
69     return  
70         m_a >= 0.0 && m_a <= 1.0  
71         && m_b >= 0.0 && m_b <= 1.0  
72         && 1.0 - m_a + m_b >= - std::numeric_limits<double>::epsilon();  
73 }
```

#### 9.82.3.5 void TernaryPoint::set (double *a*, double *b*)

Definition at line 52 of file TernaryPoint.cpp.

References isValid().

Referenced by TernaryPoint().

```
53 {
54     if ( a >= 0.0 && a <= 1.0
55         && b >= 0.0 && b <= 1.0
56         && 1.0 - a - b >= -2.0 * std::numeric_limits<double>::epsilon() ) {
57         m_a = a;
58         m_b = b;
59         Q_ASSERT( isValid() ); // more a test for isValid
60     } else {
61         m_a = -1.0;
62         m_b = -1.0;
63         Q_ASSERT( ! isValid() );
64     }
65 }
```

The documentation for this class was generated from the following files:

- [TernaryPoint.h](#)
- [TernaryPoint.cpp](#)

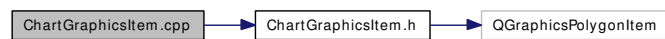
## Chapter 10

# KD Chart 2 File Documentation

### 10.1 ChartGraphicsItem.cpp File Reference

```
#include "ChartGraphicsItem.h"
```

Include dependency graph for ChartGraphicsItem.cpp:



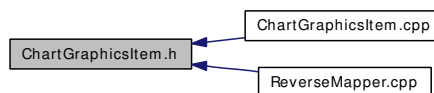
## 10.2 ChartGraphicsItem.h File Reference

```
#include <QGraphicsPolygonItem>
```

Include dependency graph for ChartGraphicsItem.h:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [KDChart](#)

### Classes

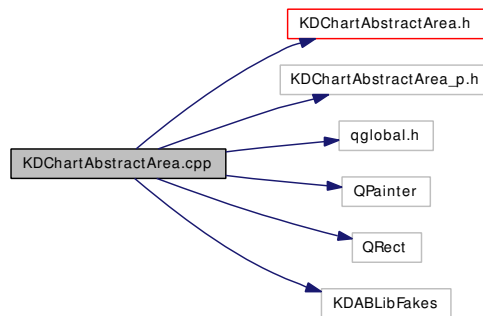
- class [KDChart::ChartGraphicsItem](#)  
*Graphics item used inside of the [ReverseMapper](#).*



## 10.3 KDChartAbstractArea.cpp File Reference

```
#include "KDChartAbstractArea.h"
#include "KDChartAbstractArea_p.h"
#include <qglobal.h>
#include <QPainter>
#include <QRect>
#include <KDABLibFakes>
```

Include dependency graph for KDChartAbstractArea.cpp:



### Defines

- `#define d(d_func())`

### 10.3.1 Define Documentation

#### 10.3.1.1 `#define d(d_func())`

Definition at line 39 of file KDChartAbstractArea.cpp.

Referenced by KDChart::AbstractCoordinatePlane::AbstractCoordinatePlane(), KDChart::AbstractCartesianDiagram::addAxis(), KDChart::Palette::addBrush(), KDChart::Chart::addCoordinatePlane(), KDChart::Legend::addDiagram(), KDChart::AbstractCoordinatePlane::addDiagram(), KDChart::Widget::addHeaderFooter(), KDChart::Chart::addHeaderFooter(), KDChart::Widget::addLegend(), KDChart::Chart::addLegend(), KDChart::CartesianCoordinatePlane::adjustHorizontalRangeToData(), KDChart::GridAttributes::adjustLowerBoundToGrid(), KDChart::CartesianCoordinatePlane::adjustRangesToData(), KDChart::GridAttributes::adjustUpperBoundToGrid(), KDChart::CartesianCoordinatePlane::adjustVerticalRangeToData(), KDChart::RelativePosition::alignment(), KDChart::Legend::alignment(), KDChart::Widget::allHeadersFooters(), KDChart::Widget::allLegends(), KDChart::AbstractDiagram::allowOverlappingDataValueTexts(), KDChart::ThreeDBarAttributes::angle(), KDChart::PolarCoordinatePlane::angleUnit(), KDChart::AbstractDiagram::antiAliasing(), KDChart::ValueTrackerAttributes::areaBrush(), KDChart::AbstractDiagram::attributesModel(), KDChart::AbstractDiagram::attributesModelIndex(), KDChart::CartesianCoordinatePlane::autoAdjustGridToZoom(), KDChart::CartesianCoordinatePlane::autoAdjustHorizontalRangeToData(), KDChart::CartesianCoordinatePlane::autoAdjustVerticalRangeToData(), KDChart::TextAttributes::autoRotate(), KDChart::TextAttributes::autoShrink(), KDChart::AbstractCartesianDiagram::axes(), KDChart::CartesianCoordinatePlane::axes-

CalcModeX(), KDChart::CartesianCoordinatePlane::axesCalcModeY(), KDChart::DataValueAttributes::backgroundAttributes(), KDChart::Chart::backgroundAttributes(), KDChart::AbstractAreaBase::backgroundAttributes(), KDChart::BarDiagram::barAttributes(), KDChart::BarAttributes::barGapFactor(), KDChart::AbstractArea::bottomOverlap(), KDChart::Legend::brush(), KDChart::BackgroundAttributes::brush(), KDChart::Legend::brushes(), KDChart::Plotter::calculateDataBoundaries(), KDChart::LineDiagram::calculateDataBoundaries(), KDChart::BarDiagram::calculateDataBoundaries(), KDChart::TextAttributes::calculatedFont(), KDChart::CartesianCoordinatePlane::calculateRawDataBoundingRect(), KDChart::RingDiagram::clone(), KDChart::PolarDiagram::clone(), KDChart::Plotter::clone(), KDChart::PieDiagram::clone(), KDChart::LineDiagram::clone(), KDChart::Legend::clone(), KDChart::HeaderFooter::clone(), KDChart::BarDiagram::clone(), KDChart::PolarDiagram::closeDatasets(), KDChart::AbstractAxis::connectSignals(), KDChart::Legend::constDiagrams(), KDChart::Widget::coordinatePlane(), KDChart::PaintContext::coordinatePlane(), KDChart::Chart::coordinatePlane(), KDChart::AbstractDiagram::coordinatePlane(), KDChart::AbstractAxis::coordinatePlane(), KDChart::Chart::coordinatePlaneLayout(), KDChart::Chart::coordinatePlanes(), KDChart::AbstractAxis::createObserver(), KDChart::AbstractDiagram::dataBoundaries(), KDChart::DataValueAttributes::dataLabel(), KDChart::Legend::datasetCount(), KDChart::AbstractDiagram::datasetDimension(), KDChart::DataValueAttributes::decimalDigits(), KDChart::AbstractAxis::delayedInit(), KDChart::AbstractAxis::deleteObserver(), KDChart::AbstractThreeDAttributes::depth(), KDChart::Legend::diagram(), KDChart::AbstractCoordinatePlane::diagram(), KDChart::AbstractAxis::diagram(), KDChart::Legend::diagrams(), KDChart::AbstractCoordinatePlane::diagrams(), KDChart::LineAttributes::displayArea(), KDChart::CartesianCoordinatePlane::doesIsometricScaling(), KDChart::AbstractDiagram::doItemsLayout(), KDChart::CartesianCoordinatePlane::doneSetZoomCenter(), KDChart::CartesianCoordinatePlane::doneSetZoomFactorX(), KDChart::CartesianCoordinatePlane::doneSetZoomFactorY(), KDChart::BarAttributes::drawSolidExcessArrows(), KDChart::PieAttributes::explode(), KDChart::PieAttributes::explodeFactor(), KDChart::Widget::firstHeaderFooter(), KDChart::BarAttributes::fixedBarWidth(), KDChart::BarAttributes::fixedDataValueGap(), KDChart::BarAttributes::fixedValueBlockGap(), KDChart::Legend::floatingPosition(), KDChart::TextAttributes::font(), KDChart::TextAttributes::fontSize(), KDChart::DataValueAttributes::frameAttributes(), KDChart::Chart::frameAttributes(), KDChart::AbstractAreaBase::frameAttributes(), KDChart::CartesianAxis::geometry(), KDChart::AbstractCoordinatePlane::geometry(), KDChart::Palette::getBrush(), KDChart::AbstractAreaBase::getFrameLeadings(), KDChart::AbstractCoordinatePlane::globalGridAttributes(), KDChart::Widget::globalLeadingBottom(), KDChart::Widget::globalLeadingLeft(), KDChart::Widget::globalLeadingRight(), KDChart::Widget::globalLeadingTop(), KDChart::AbstractPieDiagram::granularity(), KDChart::PolarCoordinatePlane::gridAttributes(), KDChart::CartesianCoordinatePlane::gridAttributes(), KDChart::AbstractCoordinatePlane::gridDimensionsList(), KDChart::GridAttributes::gridGranularitySequence(), KDChart::GridAttributes::gridPen(), KDChart::GridAttributes::gridStepWidth(), KDChart::GridAttributes::gridSubStepWidth(), KDChart::BarAttributes::groupGapFactor(), KDChart::CartesianCoordinatePlane::handleFixedDataCoordinateSpaceRelation(), KDChart::TextAttributes::hasAbsoluteFontSize(), KDChart::CartesianAxis::hasDefaultTitleTextAttributes(), KDChart::CartesianCoordinatePlane::hasFixedDataCoordinateSpaceRelation(), KDChart::PolarCoordinatePlane::hasOwnGridAttributes(), KDChart::CartesianCoordinatePlane::hasOwnGridAttributes(), KDChart::Chart::headerFooter(), KDChart::Chart::headerFooters(), KDChart::RelativePosition::horizontalPadding(), KDChart::CartesianCoordinatePlane::horizontalRange(), KDChart::AbstractDiagram::indexAt(), KDChart::AbstractDiagram::indexesAt(), KDChart::ValueTrackerAttributes::isEnabled(), KDChart::AbstractThreeDAttributes::isEnabled(), KDChart::GridAttributes::isGridVisible(), KDChart::CartesianCoordinatePlane::isHorizontalRangeReversed(), KDChart::AbstractCoordinatePlane::isRubberBandZoomingEnabled(), KDChart::GridAttributes::isSubGridVisible(), KDChart::Palette::isValid(), KDChart::CartesianCoordinatePlane::isVerticalRangeReversed(), KDChart::TextAttributes::isVisible(), KDChart::MarkerAttributes::isVisible(), KDChart::FrameAttributes::isVisible(), KDChart::DataValueAttributes::isVisible(), KDChart::BackgroundAttributes::isVisible(), KDChart::AbstractCoordinatePlane::isVisiblePoint(), KDChart::AbstractAxis::labels(), KDChart::TernaryCoordinatePlane::layoutDiagrams(), KDChart::PolarCoordinatePlane::layoutDiagrams(), KDChart::CartesianCoordinate

Plane::layoutDiagrams(), KDChart::CartesianAxis::layoutPlanes(), KDChart::AbstractArea::left-  
 Overlap(), KDChart::Widget::legend(), KDChart::Legend::Legend(), KDChart::Chart::legend(),  
 KDChart::Chart::legends(), KDChart::Legend::legendStyle(), KDChart::Plotter::lineAttributes(),  
 KDChart::LineDiagram::lineAttributes(), KDChart::ThreeDLineAttributes::lineXRotation(),  
 KDChart::ThreeDLineAttributes::lineYRotation(), KDChart::Legend::markerAttributes(),  
 KDChart::DataValueAttributes::markerAttributes(), KDChart::MarkerAttributes::markerColor(),  
 KDChart::ValueTrackerAttributes::markerSize(), KDChart::MarkerAttributes::markerSize(),  
 KDChart::MarkerAttributes::markerStyle(), KDChart::MarkerAttributes::markerStylesMap(),  
 KDChart::CartesianAxis::maximumSize(), KDChart::TextAttributes::minimalFontSize(), KDChart::Line-  
 Attributes::missingValuesPolicy(), KDChart::Chart::mouseDoubleClickEvent(), KDChart::Abstract-  
 CoordinatePlane::mouseDoubleClickEvent(), KDChart::Chart::mouseMoveEvent(), KDChart::Abstract-  
 CoordinatePlane::mouseMoveEvent(), KDChart::Chart::mousePressEvent(), KDChart::Abstract-  
 CoordinatePlane::mousePressEvent(), KDChart::Chart::mouseReleaseEvent(), KDChart::Abstract-  
 CoordinatePlane::mouseReleaseEvent(), KDChart::DataValueAttributes::negativePosition(),  
 KDChart::Plotter::numberOfAbscissaSegments(), KDChart::LineDiagram::numberOfAbscissa-  
 Segments(), KDChart::BarDiagram::numberOfAbscissaSegments(), KDChart::Plotter::number-  
 OfOrdinateSegments(), KDChart::LineDiagram::numberOfOrdinateSegments(), KDChart::Bar-  
 Diagram::numberOfOrdinateSegments(), KDChart::AbstractAxis::observedBy(), KDChart::Value-  
 TrackerAttributes::operator=(), KDChart::ThreeDPieAttributes::operator=(), KDChart::Three-  
 DLineAttributes::operator=(), KDChart::ThreeDBarAttributes::operator=(), KDChart::Text-  
 Attributes::operator=(), KDChart::PieAttributes::operator=(), KDChart::LineAttributes::operator=(),  
 KDChart::GridAttributes::operator=(), KDChart::FrameAttributes::operator=(), KDChart::Data-  
 ValueAttributes::operator=(), KDChart::BarAttributes::operator=(), KDChart::Background-  
 Attributes::operator=(), KDChart::AbstractThreeDAttributes::operator=(), KDChart::Relative-  
 Position::operator==(), KDChart::Legend::orientation(), KDChart::FrameAttributes::padding(),  
 KDChart::TernaryPointDiagram::paint(), KDChart::TernaryLineDiagram::paint(), KDChart::Ternary-  
 CoordinatePlane::paint(), KDChart::PolarCoordinatePlane::paint(), KDChart::Plotter::paint(),  
 KDChart::PieDiagram::paint(), KDChart::LineDiagram::paint(), KDChart::Legend::paint(), KD-  
 Chart::Chart::paint(), KDChart::CartesianCoordinatePlane::paint(), KDChart::CartesianAxis::paint(),  
 KDChart::BarDiagram::paint(), KDChart::AbstractArea::paintAll(), KDChart::AbstractAreaBase::paint-  
 Background(), KDChart::CartesianAxis::paintCtx(), KDChart::AbstractDiagram::paintDataValue-  
 Text(), KDChart::PaintContext::painter(), KDChart::Chart::paintEvent(), KDChart::AbstractArea-  
 Widget::paintEvent(), KDChart::AbstractAreaBase::paintFrame(), KDChart::AbstractAreaWidget::paint-  
 IntoRect(), KDChart::AbstractDiagram::paintMarker(), KDChart::AbstractCoordinatePlane::parent(),  
 KDChart::ValueTrackerAttributes::pen(), KDChart::TextAttributes::pen(), KDChart::Marker-  
 Attributes::pen(), KDChart::Legend::pen(), KDChart::FrameAttributes::pen(), KDChart::Legend::pens(),  
 KDChart::AbstractDiagram::percentMode(), KDChart::AbstractPieDiagram::pieAttributes(),  
 KDChart::BackgroundAttributes::pixmap(), KDChart::BackgroundAttributes::pixmapMode(), KD-  
 Chart::Legend::position(), KDChart::HeaderFooter::position(), KDChart::CartesianAxis::position(),  
 KDChart::DataValueAttributes::positivePosition(), KDChart::DataValueAttributes::prefix(),  
 KDChart::PolarCoordinatePlane::radiusUnit(), KDChart::PaintContext::rectangle(), KDChart::Relative-  
 Position::referenceArea(), KDChart::Legend::referenceArea(), KDChart::AbstractCoordinate-  
 Plane::referenceCoordinatePlane(), KDChart::AbstractCartesianDiagram::referenceDiagram(),  
 KDChart::AbstractCartesianDiagram::referenceDiagramOffset(), KDChart::RelativePosition::reference-  
 Point(), KDChart::RelativePosition::referencePoints(), KDChart::RelativePosition::reference-  
 Position(), KDChart::RingDiagram::relativeThickness(), KDChart::Chart::reLayoutFloatingLegends(),  
 KDChart::Palette::removeBrush(), KDChart::Legend::removeDiagram(), KDChart::Legend::remove-  
 Diagrams(), KDChart::Chart::replaceCoordinatePlane(), KDChart::Legend::replaceDiagram(),  
 KDChart::AbstractCoordinatePlane::replaceDiagram(), KDChart::Widget::replaceHeader-  
 Footer(), KDChart::Chart::replaceHeaderFooter(), KDChart::Widget::replaceLegend(),  
 KDChart::Chart::replaceLegend(), KDChart::Widget::resetData(), KDChart::Plotter::resetLine-  
 Attributes(), KDChart::LineDiagram::resetLineAttributes(), KDChart::RelativePosition::reset-  
 ReferencePosition(), KDChart::Legend::resetTexts(), KDChart::CartesianAxis::resetTitleText-  
 Attributes(), KDChart::Plotter::resize(), KDChart::LineDiagram::resize(), KDChart::Bar-

Diagram::resize(), KDChart::PolarCoordinatePlane::resizeEvent(), KDChart::Chart::resizeEvent(),  
 KDChart::Legend::resizeLayout(), KDChart::AbstractArea::rightOverlap(), KDChart::Polar-  
 Diagram::rotateCircularLabels(), KDChart::TextAttributes::rotation(), KDChart::Relative-  
 Position::rotation(), KDChart::GridAttributes::setAdjustBoundsToGrid(), KDChart::RelativePosition::set-  
 Alignment(), KDChart::Legend::setAlignment(), KDChart::AbstractDiagram::setAllowOverlapping-  
 DataValueTexts(), KDChart::ThreeDBarAttributes::setAngle(), KDChart::AbstractDiagram::setAnti-  
 Aliasing(), KDChart::ValueTrackerAttributes::setAreaBrush(), KDChart::AbstractDiagram::setAttributes-  
 Model(), KDChart::AbstractCartesianDiagram::setAttributesModel(), KDChart::AbstractDiagram::set-  
 AttributesModelRootIndex(), KDChart::CartesianCoordinatePlane::setAutoAdjustGridToZoom(),  
 KDChart::CartesianCoordinatePlane::setAutoAdjustHorizontalRangeToData(), KDChart::Cartesian-  
 CoordinatePlane::setAutoAdjustVerticalRangeToData(), KDChart::TextAttributes::setAutoRotate(),  
 KDChart::TextAttributes::setAutoShrink(), KDChart::CartesianCoordinatePlane::setAxesCalcModes(),  
 KDChart::CartesianCoordinatePlane::setAxesCalcModeX(), KDChart::CartesianCoordinatePlane::set-  
 AxesCalcModeY(), KDChart::DataValueAttributes::setBackgroundAttributes(), KDChart::Chart::set-  
 BackgroundAttributes(), KDChart::AbstractAreaBase::setBackgroundAttributes(), KDChart::Bar-  
 Diagram::setBarAttributes(), KDChart::BarAttributes::setBarGapFactor(), KDChart::Legend::set-  
 Brush(), KDChart::BackgroundAttributes::setBrush(), KDChart::Legend::setBrushesFromDiagram(),  
 KDChart::PolarDiagram::setCloseDatasets(), KDChart::Legend::setColor(), KDChart::PaintContext::set-  
 CoordinatePlane(), KDChart::AbstractDiagram::setCoordinatePlane(), KDChart::AbstractDiagram::set-  
 DataBoundariesDirty(), KDChart::Widget::setDataCell(), KDChart::DataValueAttributes::set-  
 DataLabel(), KDChart::Widget::setDataset(), KDChart::AbstractDiagram::setDatasetDimension(),  
 KDChart::AbstractDiagram::setDataValueAttributes(), KDChart::DataValueAttributes::setDecimal-  
 Digits(), KDChart::AbstractThreeDAttributes::setDepth(), KDChart::LineAttributes::setDisplayArea(),  
 KDChart::BarAttributes::setDrawSolidExcessArrows(), KDChart::ValueTrackerAttributes::setEnabled(),  
 KDChart::AbstractThreeDAttributes::setEnabled(), KDChart::PieAttributes::setExplode(), KDChart::Pie-  
 Attributes::setExplodeFactor(), KDChart::BarAttributes::setFixedBarWidth(), KDChart::Cartesian-  
 CoordinatePlane::setFixedDataCoordinateSpaceRelation(), KDChart::BarAttributes::setFixedData-  
 ValueGap(), KDChart::BarAttributes::setFixedValueBlockGap(), KDChart::Legend::setFloating-  
 Position(), KDChart::TextAttributes::setFont(), KDChart::TextAttributes::setFontSize(), KDChart::Data-  
 ValueAttributes::setFrameAttributes(), KDChart::Chart::setFrameAttributes(), KDChart::Abstract-  
 AreaBase::setFrameAttributes(), KDChart::CartesianAxis::setGeometry(), KDChart::Abstract-  
 CoordinatePlane::setGeometry(), KDChart::AbstractCoordinatePlane::setGlobalGridAttributes(),  
 KDChart::Widget::setGlobalLeading(), KDChart::Chart::setGlobalLeading(), KDChart::Widget::set-  
 GlobalLeadingBottom(), KDChart::Chart::setGlobalLeadingBottom(), KDChart::Widget::set-  
 GlobalLeadingLeft(), KDChart::Chart::setGlobalLeadingLeft(), KDChart::Widget::setGlobal-  
 LeadingRight(), KDChart::Chart::setGlobalLeadingRight(), KDChart::Widget::setGlobalLeading-  
 Top(), KDChart::Chart::setGlobalLeadingTop(), KDChart::AbstractPieDiagram::setGranularity(),  
 KDChart::PolarCoordinatePlane::setGridAttributes(), KDChart::CartesianCoordinatePlane::setGrid-  
 Attributes(), KDChart::GridAttributes::setGridGranularitySequence(), KDChart::AbstractCoordinate-  
 Plane::setGridNeedsRecalculate(), KDChart::GridAttributes::setGridPen(), KDChart::GridAttributes::set-  
 GridStepWidth(), KDChart::GridAttributes::setGridSubStepWidth(), KDChart::GridAttributes::setGrid-  
 Visible(), KDChart::BarAttributes::setGroupGapFactor(), KDChart::AbstractDiagram::setHidden(),  
 KDChart::RelativePosition::setHorizontalPadding(), KDChart::CartesianCoordinatePlane::setHorizontal-  
 Range(), KDChart::CartesianCoordinatePlane::setHorizontalRangeReversed(), KDChart::Cartesian-  
 CoordinatePlane::setIsometricScaling(), KDChart::AbstractAxis::setLabels(), KDChart::Legend::set-  
 LegendStyle(), KDChart::Plotter::setLineAttributes(), KDChart::LineDiagram::setLineAttributes(),  
 KDChart::ThreeDLineAttributes::setLineXRotation(), KDChart::ThreeDLineAttributes::setLine-  
 YRotation(), KDChart::Legend::setMarkerAttributes(), KDChart::DataValueAttributes::setMarker-  
 Attributes(), KDChart::MarkerAttributes::setMarkerColor(), KDChart::ValueTrackerAttributes::set-  
 MarkerSize(), KDChart::MarkerAttributes::setMarkerSize(), KDChart::MarkerAttributes::setMarker-  
 Style(), KDChart::MarkerAttributes::setMarkerStylesMap(), KDChart::TextAttributes::setMinimal-  
 FontSize(), KDChart::LineAttributes::setMissingValuesPolicy(), KDChart::AbstractDiagram::set-  
 Model(), KDChart::AbstractCartesianDiagram::setModel(), KDChart::DataValueAttributes::set-  
 NegativePosition(), KDChart::Legend::setOrientation(), KDChart::FrameAttributes::setPadding(),

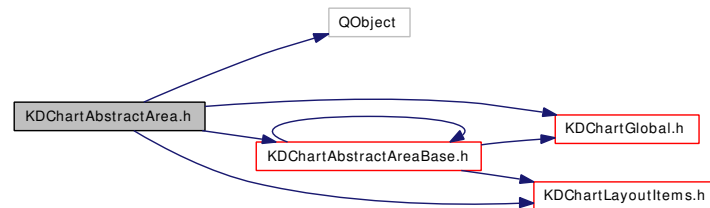
KDChart::PaintContext::setPainter(), KDChart::AbstractCoordinatePlane::setParent(), KDChart::ValueTrackerAttributes::setPen(), KDChart::TextAttributes::setPen(), KDChart::MarkerAttributes::setPen(), KDChart::Legend::setPen(), KDChart::FrameAttributes::setPen(), KDChart::AbstractDiagram::setPercentMode(), KDChart::AbstractPieDiagram::setPieAttributes(), KDChart::BackgroundAttributes::setPixmap(), KDChart::BackgroundAttributes::setPixmapMode(), KDChart::Legend::setPosition(), KDChart::HeaderFooter::setPosition(), KDChart::CartesianAxis::setPosition(), KDChart::DataValueAttributes::setPositivePosition(), KDChart::DataValueAttributes::setPrefix(), KDChart::PaintContext::setRectangle(), KDChart::RelativePosition::setReferenceArea(), KDChart::Legend::setReferenceArea(), KDChart::AbstractCoordinatePlane::setReferenceCoordinatePlane(), KDChart::AbstractCartesianDiagram::setReferenceDiagram(), KDChart::RelativePosition::setReferencePoints(), KDChart::RelativePosition::setReferencePosition(), KDChart::RingDiagram::setRelativeThickness(), KDChart::AbstractDiagram::setRootIndex(), KDChart::AbstractCartesianDiagram::setRootIndex(), KDChart::PolarDiagram::setRotateCircularLabels(), KDChart::TextAttributes::setRotation(), KDChart::RelativePosition::setRotation(), KDChart::AbstractCoordinatePlane::setRubberBandZoomingEnabled(), KDChart::AbstractDiagram::setSelection(), KDChart::AbstractAxis::setShortLabels(), KDChart::PolarDiagram::setShowDelimitersAtPosition(), KDChart::PolarDiagram::setShowLabelsAtPosition(), KDChart::Legend::setShowLines(), KDChart::Legend::setSpacing(), KDChart::PolarCoordinatePlane::setStartPosition(), KDChart::GridAttributes::setSubGridPen(), KDChart::GridAttributes::setSubGridVisible(), KDChart::DataValueAttributes::setSuffix(), KDChart::Legend::setText(), KDChart::Legend::setTextAttributes(), KDChart::DataValueAttributes::setTextAttributes(), KDChart::AbstractAxis::setTextAttributes(), KDChart::BarDiagram::setThreeDBarAttributes(), KDChart::Plotter::setThreeDLineAttributes(), KDChart::LineDiagram::setThreeDLineAttributes(), KDChart::AbstractPieDiagram::setThreeDPieAttributes(), KDChart::Legend::setTitleText(), KDChart::CartesianAxis::setTitleText(), KDChart::Legend::setTitleTextAttributes(), KDChart::CartesianAxis::setTitleTextAttributes(), KDChart::LineAttributes::setTransparency(), KDChart::Widget::setType(), KDChart::Plotter::setType(), KDChart::LineDiagram::setType(), KDChart::HeaderFooter::setType(), KDChart::BarDiagram::setType(), KDChart::AbstractDiagram::setUnitPrefix(), KDChart::AbstractDiagram::setUnitSuffix(), KDChart::Legend::setUseAutomaticMarkerSize(), KDChart::BarAttributes::setUseFixedBarWidth(), KDChart::BarAttributes::setUseFixedDataValueGap(), KDChart::BarAttributes::setUseFixedValueBlockGap(), KDChart::ThreeDPieAttributes::setUseShadowColors(), KDChart::ThreeDBarAttributes::setUseShadowColors(), KDChart::Plotter::setValueTrackerAttributes(), KDChart::LineDiagram::setValueTrackerAttributes(), KDChart::RelativePosition::setVerticalPadding(), KDChart::CartesianCoordinatePlane::setVerticalRange(), KDChart::CartesianCoordinatePlane::setVerticalRangeReversed(), KDChart::TextAttributes::setVisible(), KDChart::MarkerAttributes::setVisible(), KDChart::FrameAttributes::setVisible(), KDChart::DataValueAttributes::setVisible(), KDChart::BackgroundAttributes::setVisible(), KDChart::GridAttributes::setZeroLinePen(), KDChart::PolarCoordinatePlane::setZoomCenter(), KDChart::PolarCoordinatePlane::setZoomFactorX(), KDChart::PolarCoordinatePlane::setZoomFactorY(), KDChart::AbstractAxis::shortLabels(), KDChart::PolarDiagram::showDelimitersAtPosition(), KDChart::PolarDiagram::showLabelsAtPosition(), KDChart::Legend::showLines(), KDChart::Palette::size(), KDChart::Legend::sizeHint(), KDChart::PolarCoordinatePlane::slotLayoutChanged(), KDChart::Legend::spacing(), KDChart::PolarCoordinatePlane::startPosition(), KDChart::GridAttributes::subGridPen(), KDChart::DataValueAttributes::suffix(), KDChart::AbstractCartesianDiagram::takeAxis(), KDChart::Chart::takeCoordinatePlane(), KDChart::AbstractCoordinatePlane::takeDiagram(), KDChart::Widget::takeHeaderFooter(), KDChart::Chart::takeHeaderFooter(), KDChart::Widget::takeLegend(), KDChart::Chart::takeLegend(), KDChart::Legend::text(), KDChart::Legend::textAttributes(), KDChart::DataValueAttributes::textAttributes(), KDChart::AbstractAxis::textAttributes(), KDChart::Legend::texts(), KDChart::BarDiagram::threeDBarAttributes(), KDChart::Plotter::threeDItemDepth(), KDChart::LineDiagram::threeDItemDepth(), KDChart::BarDiagram::threeDItemDepth(), KDChart::Plotter::threeDLineAttributes(), KDChart::LineDiagram::threeDLineAttributes(), KDChart::AbstractPieDiagram::threeDPieAttributes(), KDChart::Legend::titleText(), KDChart::CartesianAxis::titleText(), KDChart::Legend::titleTextAttributes(), KDChart::CartesianAxis::titleTextAttributes(), KDChart::AbstractArea::topOverlap(), KDChart::TernaryCoordinatePlane::translate(), KDChart::PolarCoordinatePlane::translate(), KDChart::CartesianCoordinatePlane::translate(), KDChart::CartesianCoordinatePlane::translate-

Back(), KDChart::PolarCoordinatePlane::translatePolar(), KDChart::LineAttributes::transparency(), KDChart::Plotter::type(), KDChart::LineDiagram::type(), KDChart::HeaderFooter::type(), KDChart::BarDiagram::type(), KDChart::AbstractDiagram::unitPrefix(), KDChart::AbstractDiagram::unitSuffix(), KDChart::AbstractDiagram::update(), KDChart::AbstractAxis::update(), KDChart::Legend::useAutomaticMarkerSize(), KDChart::AbstractDiagram::useDefaultColors(), KDChart::BarAttributes::useFixedBarWidth(), KDChart::BarAttributes::useFixedDataValueGap(), KDChart::BarAttributes::useFixedValueBlockGap(), KDChart::AbstractDiagram::useRainbowColors(), KDChart::AbstractDiagram::usesExternalAttributesModel(), KDChart::ThreeDPieAttributes::useShadowColors(), KDChart::ThreeDBarAttributes::useShadowColors(), KDChart::AbstractDiagram::useSubduedColors(), KDChart::AbstractThreeDAttributes::validDepth(), KDChart::AbstractDiagram::valueForCell(), KDChart::LineDiagram::valueForCellTesting(), KDChart::Plotter::valueTrackerAttributes(), KDChart::LineDiagram::valueTrackerAttributes(), KDChart::RelativePosition::verticalPadding(), KDChart::CartesianCoordinatePlane::verticalRange(), KDChart::GridAttributes::zeroLinePen(), KDChart::PolarCoordinatePlane::zoomCenter(), KDChart::CartesianCoordinatePlane::zoomCenter(), KDChart::PolarCoordinatePlane::zoomFactorX(), KDChart::CartesianCoordinatePlane::zoomFactorX(), KDChart::PolarCoordinatePlane::zoomFactorY(), KDChart::CartesianCoordinatePlane::zoomFactorY(), KDChart::AbstractAxis::~~AbstractAxis(), KDChart::AbstractCartesianDiagram::~~AbstractCartesianDiagram(), and KDChart::CartesianAxis::~~CartesianAxis().

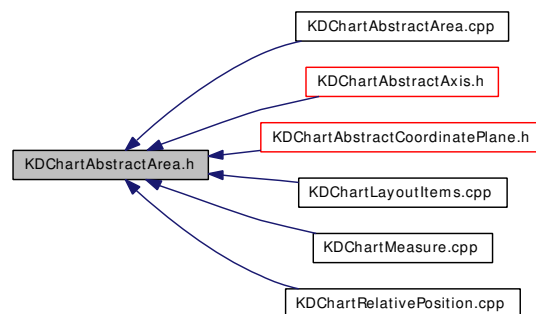
## 10.4 KDChartAbstractArea.h File Reference

```
#include <QObject>
#include "KDChartGlobal.h"
#include "KDChartAbstractAreaBase.h"
#include "KDChartLayoutItems.h"
```

Include dependency graph for KDChartAbstractArea.h:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [KDChart](#)

### Classes

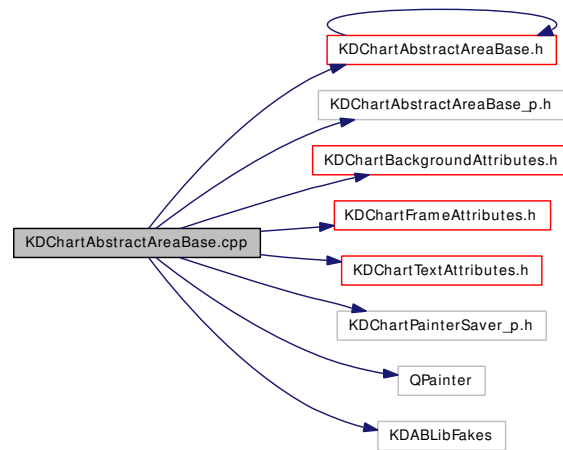
- class [KDChart::AbstractArea](#)

*An area in the chart with a background, a frame, etc.*

## 10.5 KDChartAbstractAreaBase.cpp File Reference

```
#include "KDChartAbstractAreaBase.h"
#include "KDChartAbstractAreaBase_p.h"
#include <KDChartBackgroundAttributes.h>
#include <KDChartFrameAttributes.h>
#include <KDChartTextAttributes.h>
#include "KDChartPainterSaver_p.h"
#include <QPainter>
#include <KDABLibFakes>
```

Include dependency graph for KDChartAbstractAreaBase.cpp:



### Defines

- #define `d_func()`

### 10.5.1 Define Documentation

#### 10.5.1.1 #define `d_func()`

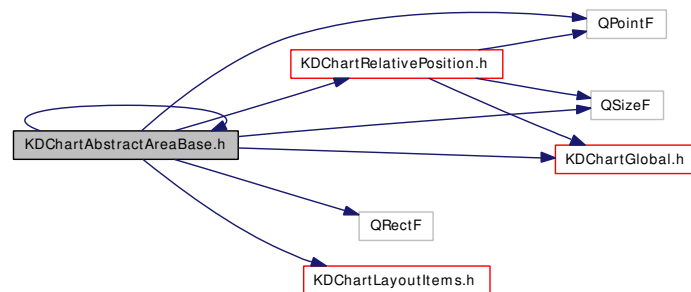
Definition at line 73 of file KDChartAbstractAreaBase.cpp.



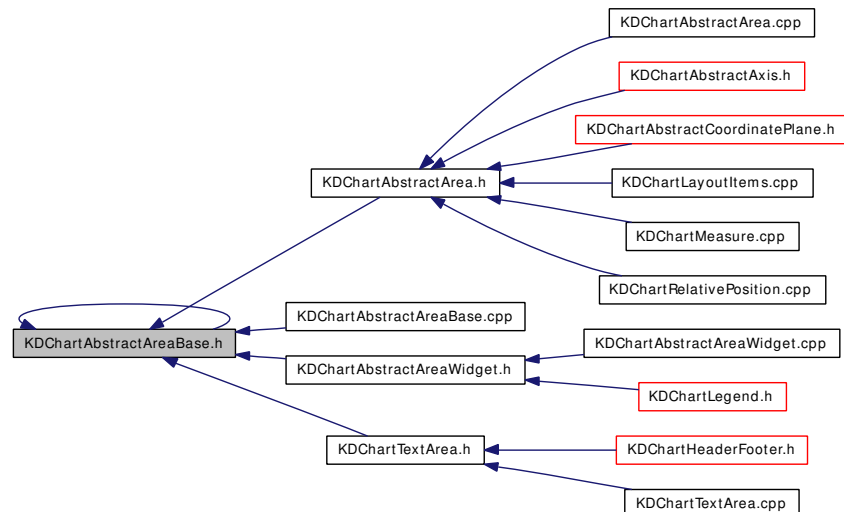
## 10.6 KDChartAbstractAreaBase.h File Reference

```
#include <QPointF>
#include <QSizeF>
#include <QRectF>
#include "KDChartGlobal.h"
#include "KDChartLayoutItems.h"
#include "KDChartRelativePosition.h"
#include "KDChartAbstractAreaBase.h"
```

Include dependency graph for KDChartAbstractAreaBase.h:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace [KDChart](#)

## Classes

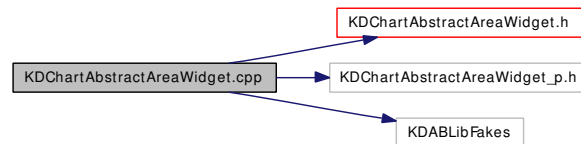
- class [KDChart::AbstractAreaBase](#)

*Base class for [AbstractArea](#) and [AbstractAreaWidget](#): An area in the chart with a background, a frame, etc.*

## 10.7 KDChartAbstractAreaWidget.cpp File Reference

```
#include "KDChartAbstractAreaWidget.h"  
#include "KDChartAbstractAreaWidget_p.h"  
#include <KDABLibFakes>
```

Include dependency graph for KDChartAbstractAreaWidget.cpp:



### Defines

- #define `d_func()`

#### 10.7.1 Define Documentation

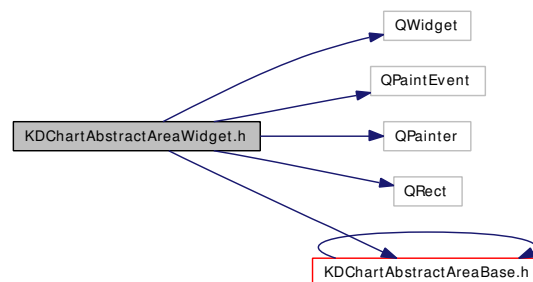
##### 10.7.1.1 #define `d_func()`

Definition at line 91 of file KDChartAbstractAreaWidget.cpp.

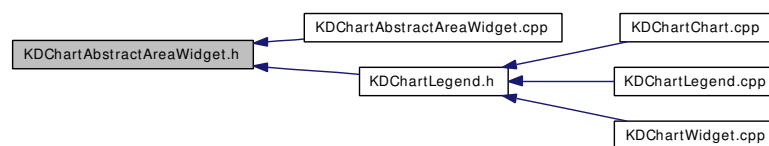
## 10.8 KDChartAbstractAreaWidget.h File Reference

```
#include <QWidget>
#include <QPaintEvent>
#include <QPainter>
#include <QRect>
#include "KDChartAbstractAreaBase.h"
```

Include dependency graph for KDChartAbstractAreaWidget.h:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [KDChart](#)

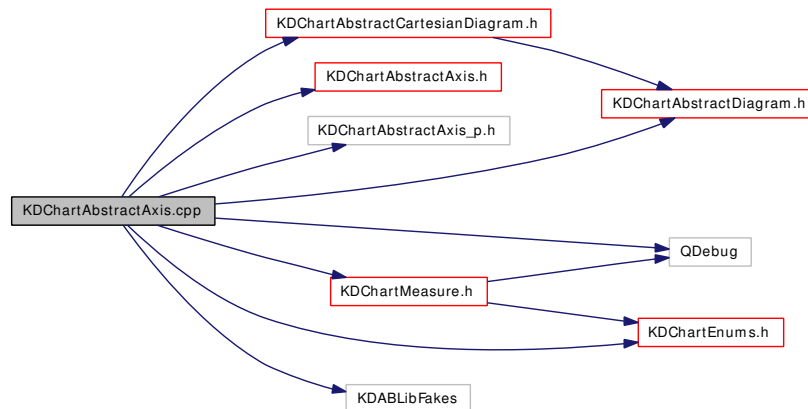
### Classes

- class [KDChart::AbstractAreaWidget](#)  
*An area in the chart with a background, a frame, etc.*

## 10.9 KDChartAbstractAxis.cpp File Reference

```
#include <QDebug>
#include "KDChartAbstractAxis.h"
#include "KDChartAbstractAxis_p.h"
#include "KDChartAbstractDiagram.h"
#include "KDChartAbstractCartesianDiagram.h"
#include "KDChartEnums.h"
#include "KDChartMeasure.h"
#include <KDABLibFakes>
```

Include dependency graph for KDChartAbstractAxis.cpp:



### Defines

- #define [d](#) d\_func()

### 10.9.1 Define Documentation

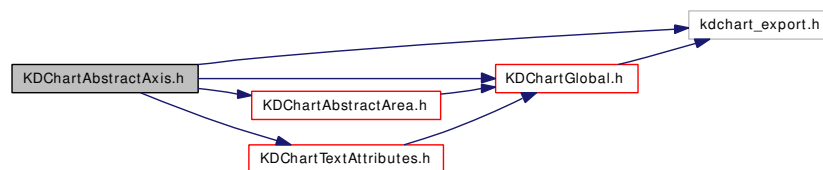
#### 10.9.1.1 #define [d](#) d\_func()

Definition at line 39 of file KDChartAbstractAxis.cpp.

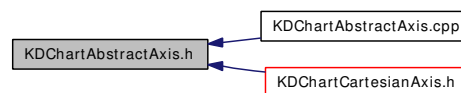
## 10.10 KDChartAbstractAxis.h File Reference

```
#include "kdchart_export.h"
#include "KDChartGlobal.h"
#include "KDChartAbstractArea.h"
#include "KDChartTextAttributes.h"
```

Include dependency graph for KDChartAbstractAxis.h:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [KDChart](#)

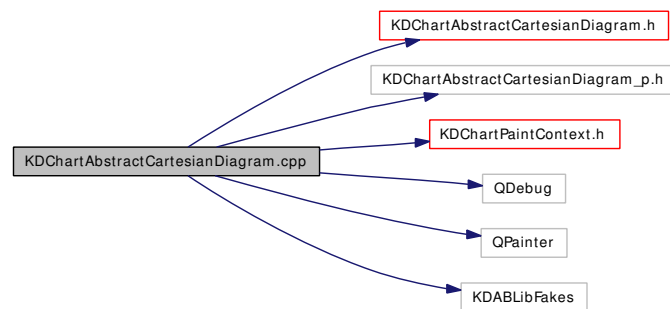
### Classes

- class [KDChart::AbstractAxis](#)  
*The base class for axes.*

## 10.11 KDChartAbstractCartesianDiagram.cpp File Reference

```
#include "KDChartAbstractCartesianDiagram.h"
#include "KDChartAbstractCartesianDiagram_p.h"
#include "KDChartPaintContext.h"
#include <QDebug>
#include <QPainter>
#include <KDABLibFakes>
```

Include dependency graph for KDChartAbstractCartesianDiagram.cpp:



### Defines

- #define [d](#) d\_func()

#### 10.11.1 Define Documentation

##### 10.11.1.1 #define d d\_func()

Definition at line 68 of file KDChartAbstractCartesianDiagram.cpp.

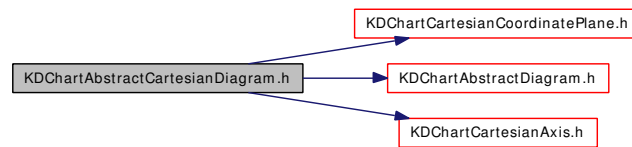
## 10.12 KDChartAbstractCartesianDiagram.h File Reference

```
#include "KDChartCartesianCoordinatePlane.h"
```

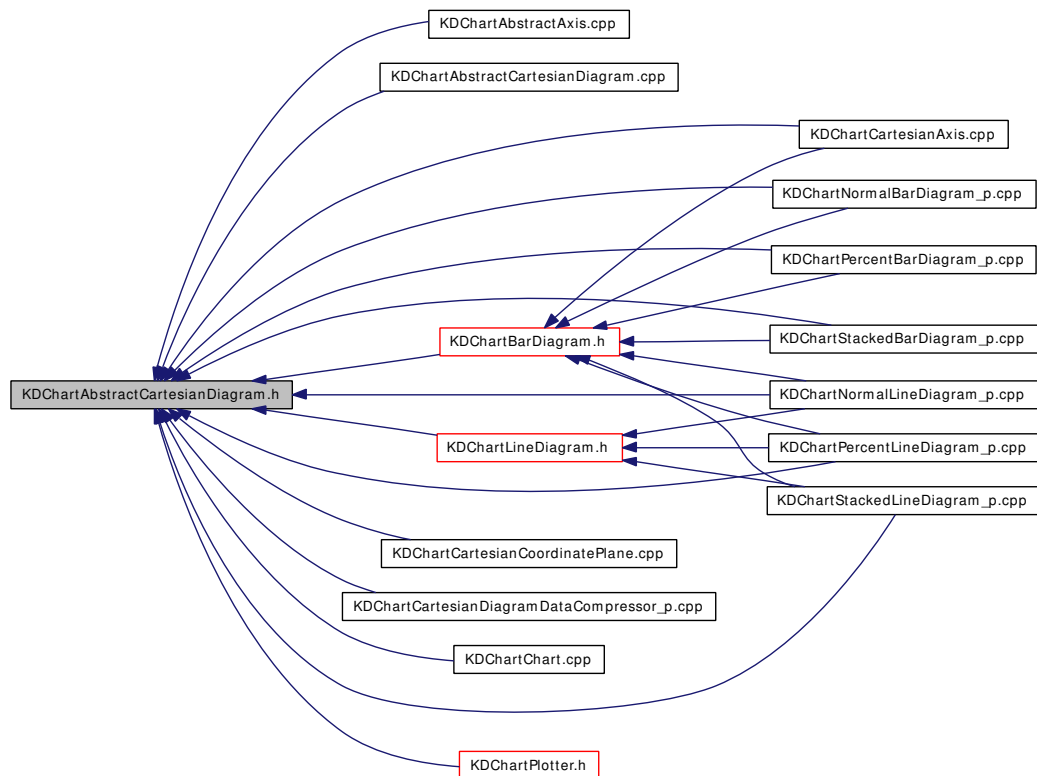
```
#include "KDChartAbstractDiagram.h"
```

```
#include "KDChartCartesianAxis.h"
```

Include dependency graph for KDChartAbstractCartesianDiagram.h:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [KDChart](#)

### Classes

- class [KDChart::AbstractCartesianDiagram](#)  
*Base class for diagrams based on a cartesian coordinate system.*

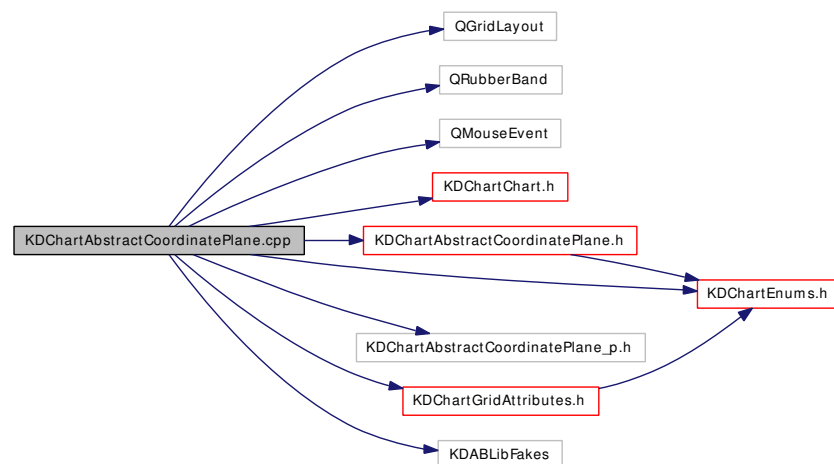




## 10.13 KDChartAbstractCoordinatePlane.cpp File Reference

```
#include <QGridLayout>
#include <QRubberBand>
#include <QMouseEvent>
#include "KDChartChart.h"
#include "KDChartAbstractCoordinatePlane.h"
#include "KDChartAbstractCoordinatePlane_p.h"
#include "KDChartGridAttributes.h"
#include <KDABLibFakes>
#include "KDChartEnums.h"
```

Include dependency graph for KDChartAbstractCoordinatePlane.cpp:



### Defines

- #define [d\\_func\(\)](#)

### Functions

- QDebug [KDChart::operator<<](#) (QDebug stream, const [DataDimension](#) &r)

### 10.13.1 Define Documentation

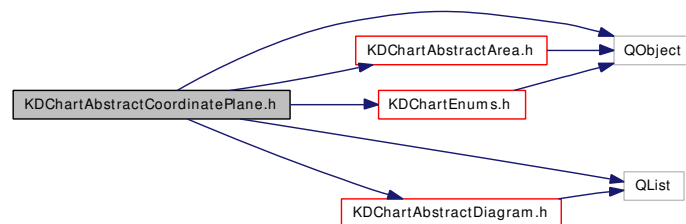
#### 10.13.1.1 #define d\_func()

Definition at line 41 of file KDChartAbstractCoordinatePlane.cpp.

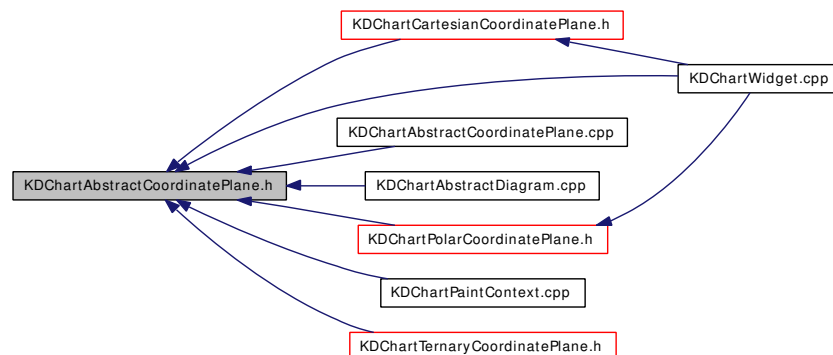
## 10.14 KDChartAbstractCoordinatePlane.h File Reference

```
#include <QObject>
#include <QList>
#include "KDChartAbstractArea.h"
#include "KDChartAbstractDiagram.h"
#include "KDChartEnums.h"
```

Include dependency graph for KDChartAbstractCoordinatePlane.h:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [KDChart](#)

### Classes

- class [KDChart::AbstractCoordinatePlane](#)  
Base class common for all coordinate planes, [CartesianCoordinatePlane](#), [PolarCoordinatePlane](#), [TernaryCoordinatePlane](#).
- class [KDChart::DataDimension](#)  
Helper class for one dimension of data, e.g.

## Typedefs

- typedef QList< DataDimension > [KDChart::DataDimensionsList](#)

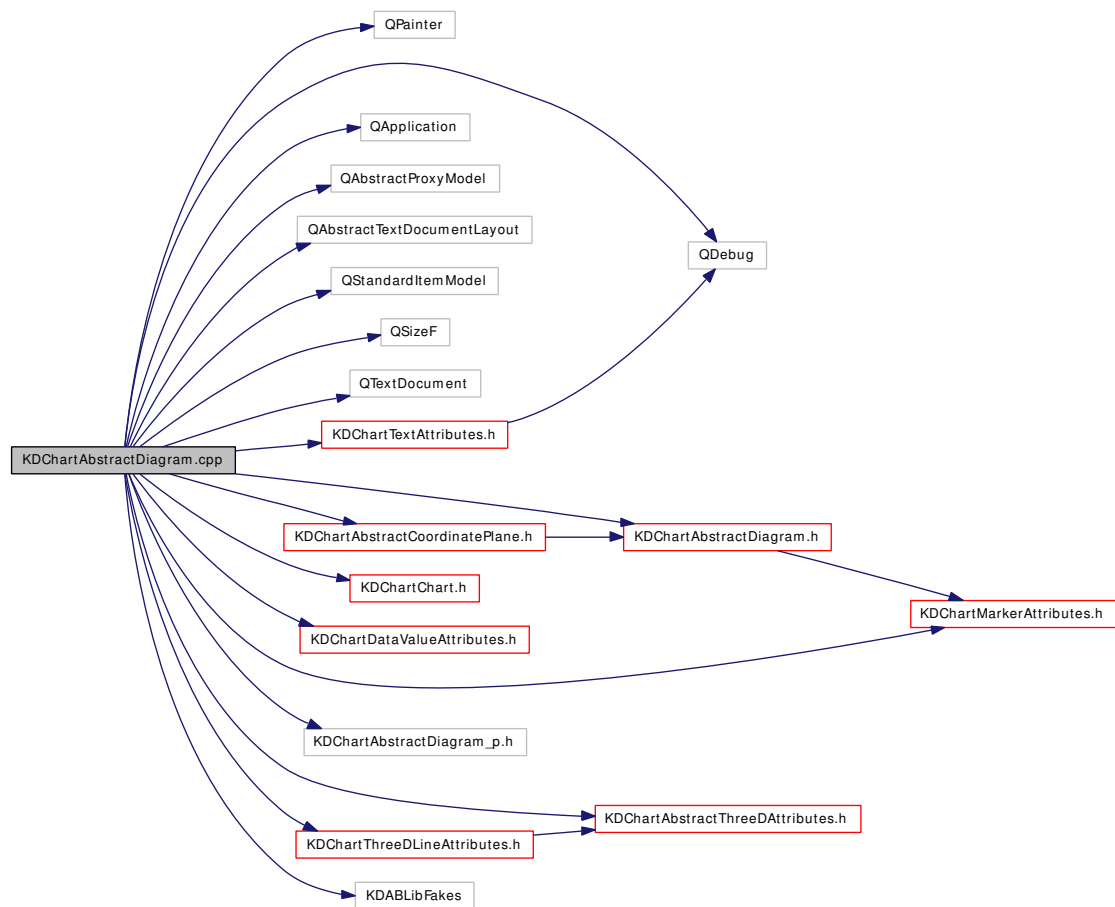
## Functions

- QDebug [KDChart::operator<<](#) (QDebug stream, const [DataDimension](#) &r)

## 10.15 KDChartAbstractDiagram.cpp File Reference

```
#include <QPainter>
#include <QDebug>
#include <QApplication>
#include <QAbstractProxyModel>
#include <QAbstractTextDocumentLayout>
#include <QStandardItemModel>
#include <QSizeF>
#include <QTextDocument>
#include "KDChartAbstractCoordinatePlane.h"
#include "KDChartChart.h"
#include "KDChartDataValueAttributes.h"
#include "KDChartTextAttributes.h"
#include "KDChartMarkerAttributes.h"
#include "KDChartAbstractDiagram.h"
#include "KDChartAbstractDiagram_p.h"
#include "KDChartAbstractThreeDAttributes.h"
#include "KDChartThreeDLineAttributes.h"
#include <KDABLibFakes>
```

Include dependency graph for KDChartAbstractDiagram.cpp:



## Defines

- `#define d d_func()`

### 10.15.1 Define Documentation

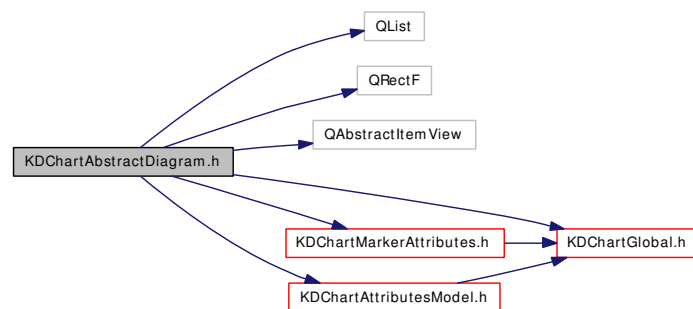
#### 10.15.1.1 `#define d d_func()`

Definition at line 109 of file `KDChartAbstractDiagram.cpp`.

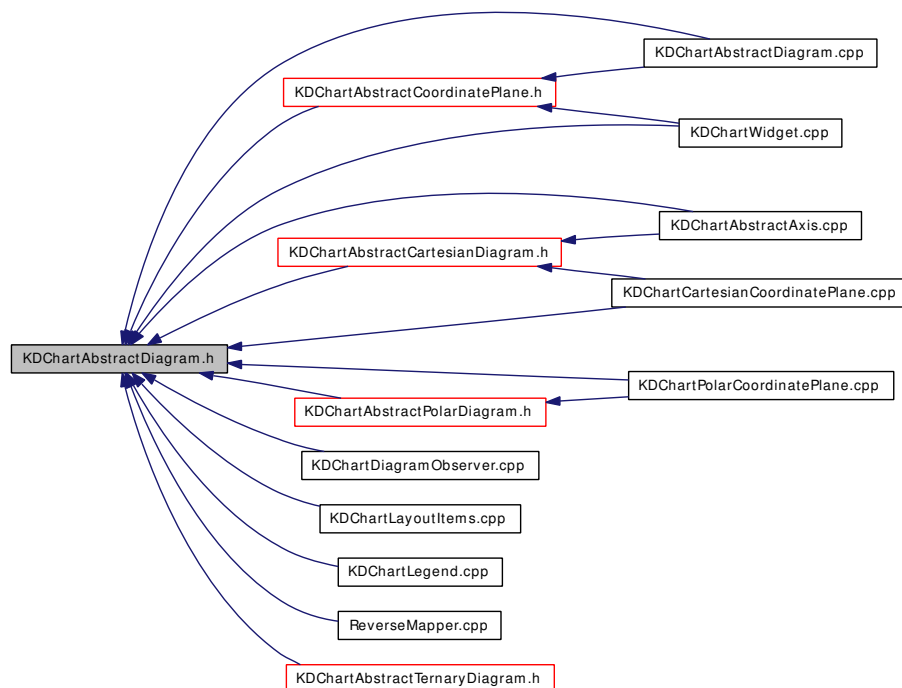
## 10.16 KDChartAbstractDiagram.h File Reference

```
#include <QList>
#include <QRectF>
#include <QAbstractItemView>
#include "KDChartGlobal.h"
#include "KDChartMarkerAttributes.h"
#include "KDChartAttributesModel.h"
```

Include dependency graph for KDChartAbstractDiagram.h:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace [KDChart](#)

## Classes

- class [KDChart::AbstractDiagram](#)  
*[AbstractDiagram](#) defines the interface for diagram classes.*
- class [KDChart::PrivateAttributesModel](#)  
*Internally used class just adding a special constructor used by [AbstractDiagram](#).*

## Typedefs

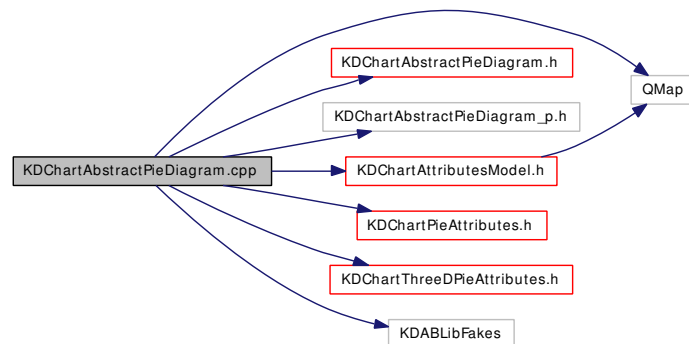
- typedef QList< AbstractDiagram \* > [KDChart::AbstractDiagramList](#)
- typedef QList< const AbstractDiagram \* > [KDChart::ConstAbstractDiagramList](#)



## 10.17 KDChartAbstractPieDiagram.cpp File Reference

```
#include <QMap>
#include "KDChartAbstractPieDiagram.h"
#include "KDChartAbstractPieDiagram_p.h"
#include "KDChartAttributesModel.h"
#include "KDChartPieAttributes.h"
#include "KDChartThreeDPieAttributes.h"
#include <KDABLibFakes>
```

Include dependency graph for KDChartAbstractPieDiagram.cpp:



### Defines

- #define [d\\_d\\_func\(\)](#)

#### 10.17.1 Define Documentation

##### 10.17.1.1 #define d\_d\_func()

Definition at line 62 of file KDChartAbstractPieDiagram.cpp.

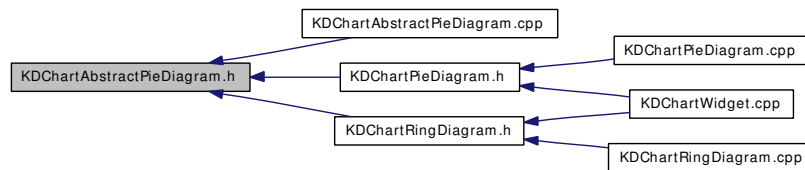
## 10.18 KDChartAbstractPieDiagram.h File Reference

```
#include "KDChartAbstractPolarDiagram.h"
```

Include dependency graph for KDChartAbstractPieDiagram.h:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [KDChart](#)

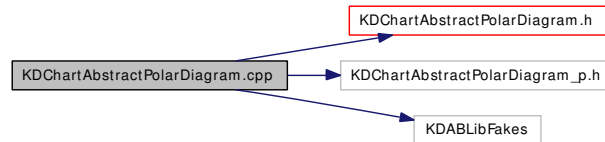
### Classes

- class [KDChart::AbstractPieDiagram](#)  
*Base class for any diagram type.*

## 10.19 KDChartAbstractPolarDiagram.cpp File Reference

```
#include "KDChartAbstractPolarDiagram.h"
#include "KDChartAbstractPolarDiagram_p.h"
#include <KDABLibFakes>
```

Include dependency graph for KDChartAbstractPolarDiagram.cpp:



### Defines

- `#define d d_func()`

#### 10.19.1 Define Documentation

##### 10.19.1.1 `#define d d_func()`

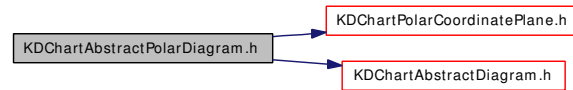
Definition at line 46 of file KDChartAbstractPolarDiagram.cpp.

## 10.20 KDChartAbstractPolarDiagram.h File Reference

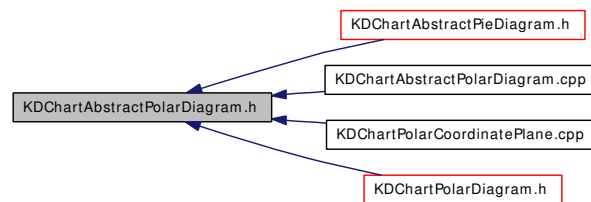
```
#include "KDChartPolarCoordinatePlane.h"
```

```
#include "KDChartAbstractDiagram.h"
```

Include dependency graph for KDChartAbstractPolarDiagram.h:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [KDChart](#)

### Classes

- class [KDChart::AbstractPolarDiagram](#)  
*Base class for diagrams based on a polar coordinate system.*

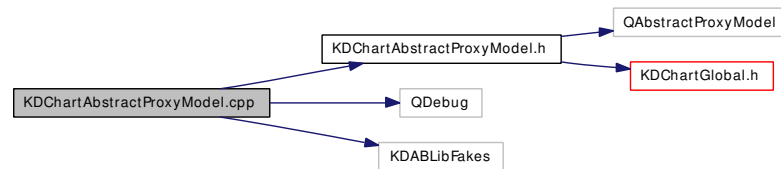
## 10.21 KDChartAbstractProxyModel.cpp File Reference

```
#include "KDChartAbstractProxyModel.h"
```

```
#include <QDebug>
```

```
#include <KDABLibFakes>
```

Include dependency graph for KDChartAbstractProxyModel.cpp:



### Namespaces

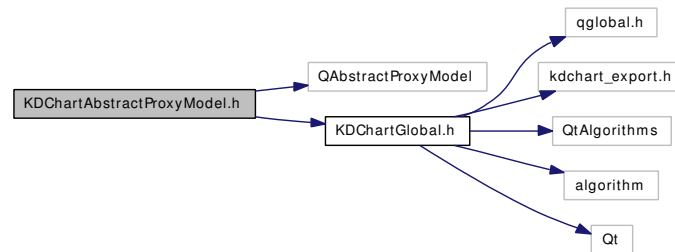
- namespace [KDChart](#)

## 10.22 KDChartAbstractProxyModel.h File Reference

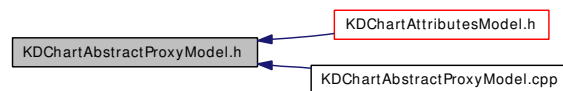
```
#include <QAbstractProxyModel>
```

```
#include "KDChartGlobal.h"
```

Include dependency graph for KDChartAbstractProxyModel.h:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [KDChart](#)

### Classes

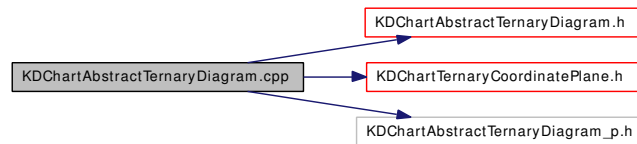
- class [KDChart::AbstractProxyModel](#)

*Base class for all proxy models used inside KD Chart.*

## 10.23 KDChartAbstractTernaryDiagram.cpp File Reference

```
#include "KDChartAbstractTernaryDiagram.h"  
#include "KDChartTernaryCoordinatePlane.h"  
#include "KDChartAbstractTernaryDiagram_p.h"
```

Include dependency graph for KDChartAbstractTernaryDiagram.cpp:



### Defines

- #define `d_d_func()`

#### 10.23.1 Define Documentation

##### 10.23.1.1 #define `d_d_func()`

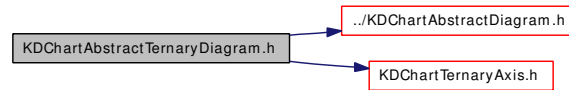
Definition at line 46 of file KDChartAbstractTernaryDiagram.cpp.

## 10.24 KDChartAbstractTernaryDiagram.h File Reference

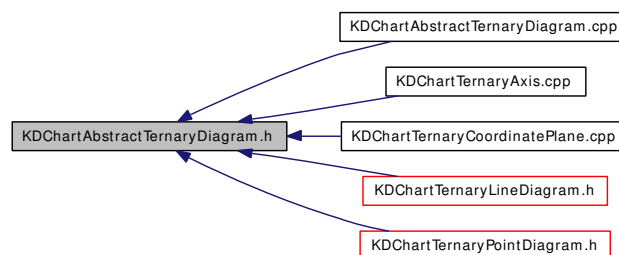
```
#include "../KDChartAbstractDiagram.h"
```

```
#include "KDChartTernaryAxis.h"
```

Include dependency graph for KDChartAbstractTernaryDiagram.h:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [KDChart](#)

### Classes

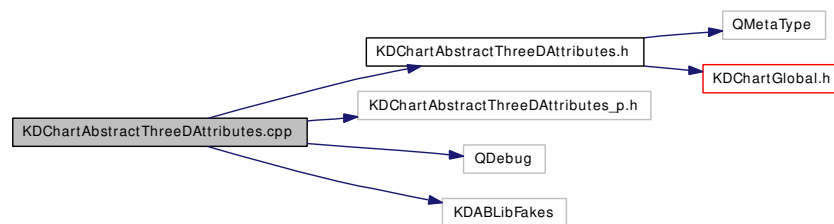
- class [KDChart::AbstractTernaryDiagram](#)  
*Base class for diagrams based on a ternary coordinate plane.*



## 10.25 KDChartAbstractThreeDAttributes.cpp File Reference

```
#include "KDChartAbstractThreeDAttributes.h"
#include "KDChartAbstractThreeDAttributes_p.h"
#include <QDebug>
#include <KDABLibFakes>
```

Include dependency graph for KDChartAbstractThreeDAttributes.cpp:



### Defines

- #define `d_func()`

### Functions

- QDebug `operator<<` (QDebug *dbg*, const `KDChart::AbstractThreeDAttributes` &*a*)

#### 10.25.1 Define Documentation

##### 10.25.1.1 #define `d_func()`

Definition at line 33 of file KDChartAbstractThreeDAttributes.cpp.

#### 10.25.2 Function Documentation

##### 10.25.2.1 QDebug `operator<<` (QDebug *dbg*, const `KDChart::AbstractThreeDAttributes` &*a*)

Definition at line 116 of file KDChartAbstractThreeDAttributes.cpp.

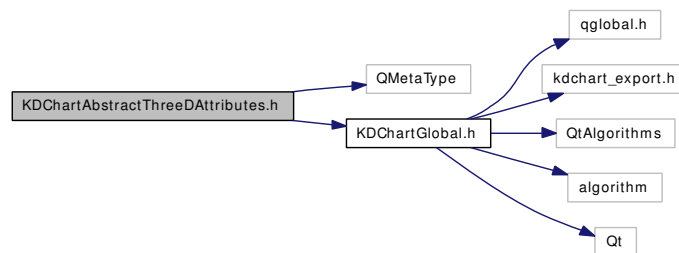
```
117 {
118     dbg << "enabled=" << a.isEnabled()
119         << "depth=" << a.depth();
120     return dbg;
121 }
```

## 10.26 KDChartAbstractThreeDAttributes.h File Reference

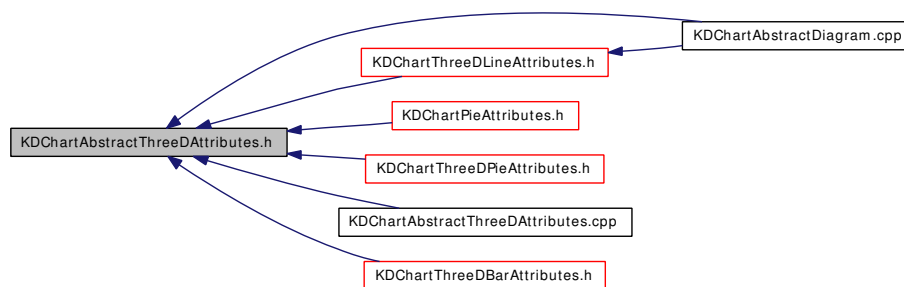
```
#include <QMetaType>
```

```
#include "KDChartGlobal.h"
```

Include dependency graph for KDChartAbstractThreeDAttributes.h:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [KDChart](#)

### Classes

- class [KDChart::AbstractThreeDAttributes](#)

*Base class for 3D attributes.*

### Functions

- KDCHART\_EXPORT QDebug [operator<<](#) (QDebug, const [KDChart::AbstractThreeDAttributes](#) &)

## 10.26.1 Function Documentation

### 10.26.1.1 KDCHART\_EXPORT QDebug operator<< (QDebug, const [KDChart::AbstractThreeDAttributes](#) &)

Definition at line 116 of file KDChartAbstractThreeDAttributes.cpp.

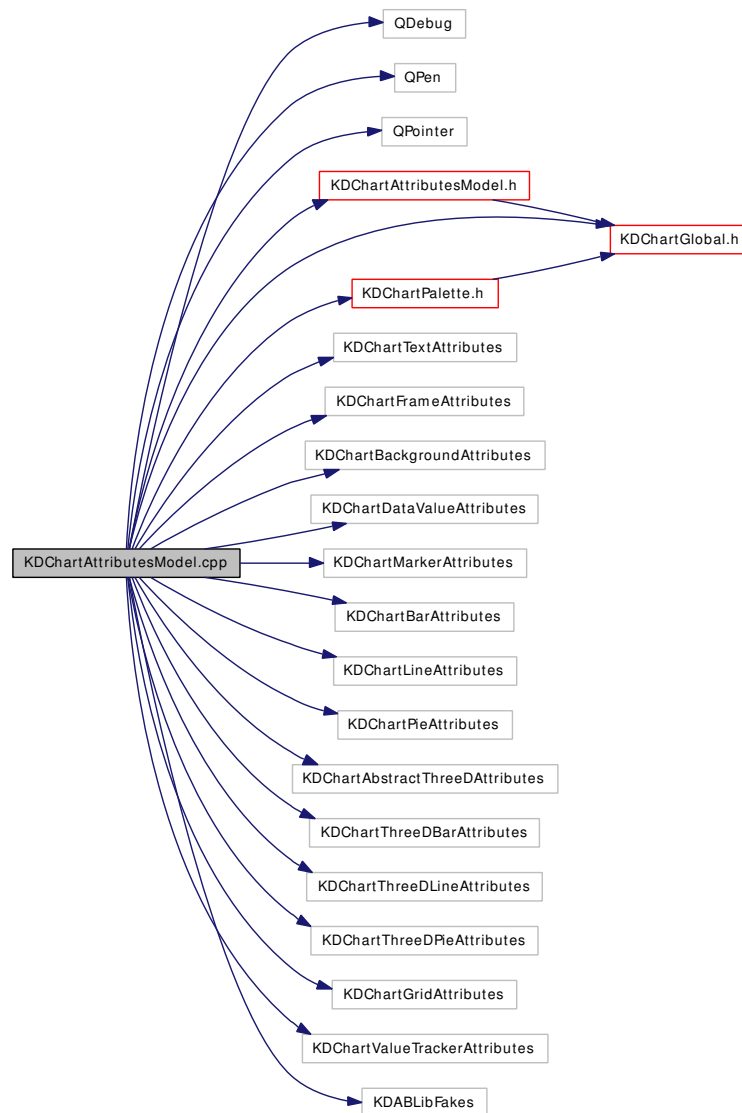
References [KDChart::AbstractThreeDAttributes::depth\(\)](#), and [KDChart::AbstractThreeDAttributes::isEnabled\(\)](#).

```
117 {  
118     dbg << "enabled=" << a.isEnabled()  
119         << "depth=" << a.depth();  
120     return dbg;  
121 }
```

## 10.27 KDChartAttributesModel.cpp File Reference

```
#include <QDebug>
#include <QPen>
#include <QPointer>
#include "KDChartAttributesModel.h"
#include "KDChartPalette.h"
#include "KDChartGlobal.h"
#include <KDChartTextAttributes>
#include <KDChartFrameAttributes>
#include <KDChartBackgroundAttributes>
#include <KDChartDataValueAttributes>
#include <KDChartMarkerAttributes>
#include <KDChartBarAttributes>
#include <KDChartLineAttributes>
#include <KDChartPieAttributes>
#include <KDChartAbstractThreeDAttributes>
#include <KDChartThreeDBarAttributes>
#include <KDChartThreeDLineAttributes>
#include <KDChartThreeDPieAttributes>
#include <KDChartGridAttributes>
#include <KDChartValueTrackerAttributes>
#include <KDABLibFakes>
```

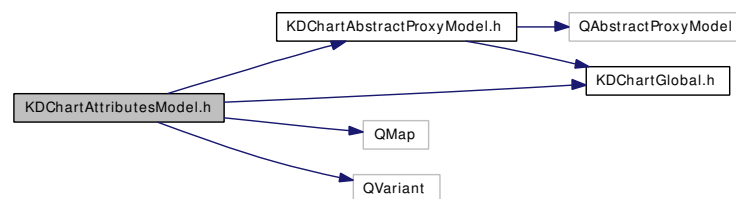
Include dependency graph for KDChartAttributesModel.cpp:



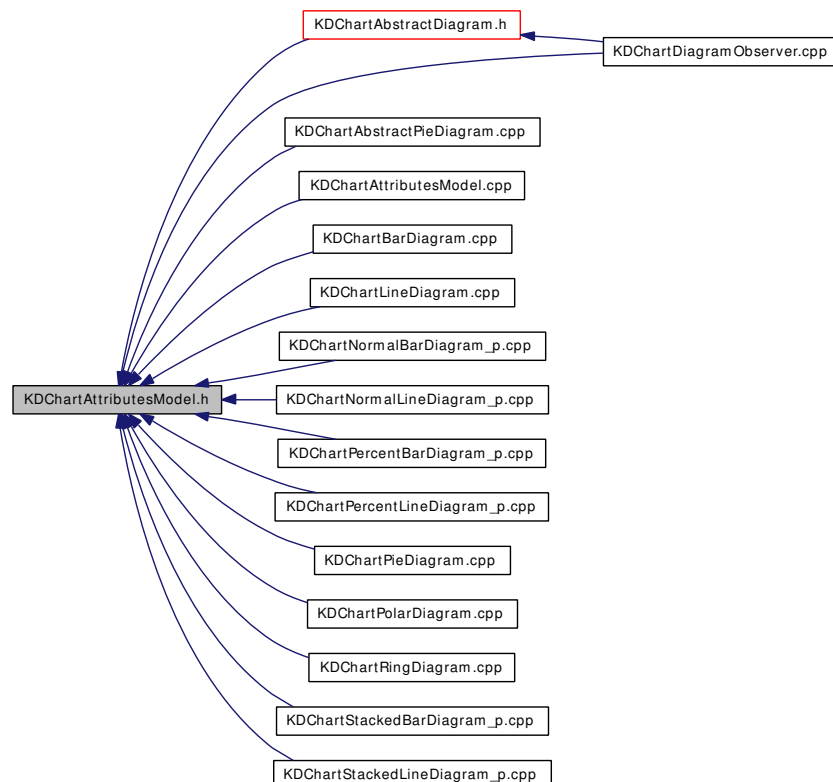
## 10.28 KDChartAttributesModel.h File Reference

```
#include "KDChartAbstractProxyModel.h"
#include <QMap>
#include <QVariant>
#include "KDChartGlobal.h"
```

Include dependency graph for KDChartAttributesModel.h:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace [KDChart](#)

## Classes

- class [KDChart::AttributesModel](#)  
*A proxy model used for storing attributes.*

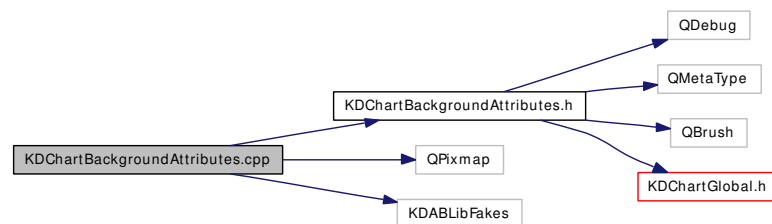
## 10.29 KDChartBackgroundAttributes.cpp File Reference

```
#include "KDChartBackgroundAttributes.h"
```

```
#include <QPixmap>
```

```
#include <KDABLibFakes>
```

Include dependency graph for KDChartBackgroundAttributes.cpp:



### Defines

- #define `d_func()`

### Functions

- QDebug `operator<<` (QDebug dbg, const `KDChart::BackgroundAttributes` &ba)

#### 10.29.1 Define Documentation

##### 10.29.1.1 #define `d_func()`

Definition at line 31 of file KDChartBackgroundAttributes.cpp.

#### 10.29.2 Function Documentation

##### 10.29.2.1 QDebug `operator<<` (QDebug *dbg*, const `KDChart::BackgroundAttributes` & *ba*)

Definition at line 150 of file KDChartBackgroundAttributes.cpp.

```

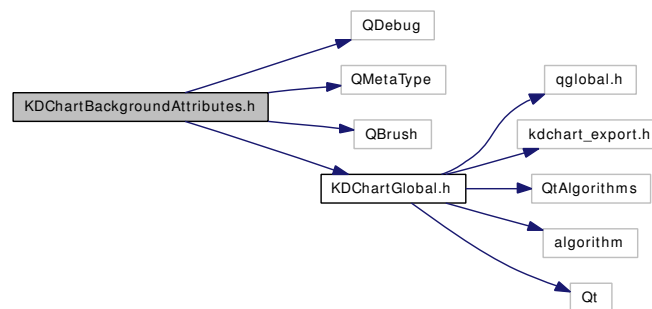
151 {
152     dbg << "KDChart::BackgroundAttributes ("
153         << "visible=" << ba.isVisible()
154         << "brush=" << ba.brush()
155         << "pixmapmode=" << ba.pixmapMode()
156         << "pixmap=" << ba.pixmap()
157         << ") ";
158     return dbg;
159 }
```



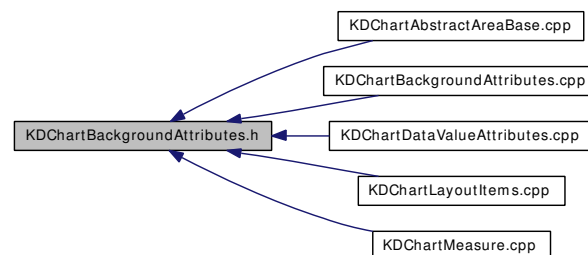
## 10.30 KDChartBackgroundAttributes.h File Reference

```
#include <QDebug>
#include <QMetaType>
#include <QBrush>
#include "KDChartGlobal.h"
```

Include dependency graph for KDChartBackgroundAttributes.h:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [KDChart](#)

### Classes

- class [KDChart::BackgroundAttributes](#)  
*Set of attributes usable for background pixmaps.*

### Functions

- KDCHART\_EXPORT QDebug [operator<<](#) (QDebug, const [KDChart::BackgroundAttributes](#) &)
- [Q\\_DECLARE\\_TYPEINFO](#) ([KDChart::BackgroundAttributes](#), Q\_MOVABLE\_TYPE)

## 10.30.1 Function Documentation

### 10.30.1.1 KDCHART\_EXPORT QDebug operator<< (QDebug, const [KDChart::BackgroundAttributes](#) &)

Definition at line 150 of file KDChartBackgroundAttributes.cpp.

References [KDChart::BackgroundAttributes::brush\(\)](#), [KDChart::BackgroundAttributes::isVisible\(\)](#), [KDChart::BackgroundAttributes::pixmap\(\)](#), and [KDChart::BackgroundAttributes::pixmapMode\(\)](#).

```
151 {
152     dbg << "KDChart::BackgroundAttributes ("
153         << "visible=" << ba.isVisible()
154         << "brush=" << ba.brush()
155         << "pixmapmode=" << ba.pixmapMode()
156         << "pixmap=" << ba.pixmap()
157         << ") ";
158     return dbg;
159 }
```

### 10.30.1.2 Q\_DECLARE\_TYPEINFO ([KDChart::BackgroundAttributes](#), Q\_MOVABLE\_TYPE)

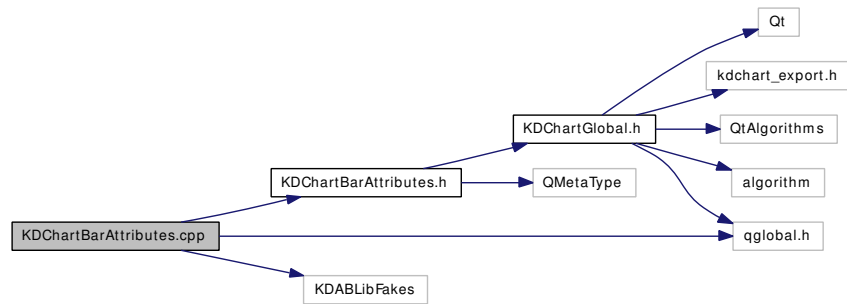
## 10.31 KDChartBarAttributes.cpp File Reference

```
#include "KDChartBarAttributes.h"
```

```
#include <qglobal.h>
```

```
#include <KDABLibFakes>
```

Include dependency graph for KDChartBarAttributes.cpp:



### Defines

- #define [d\\_func\(\)](#)

#### 10.31.1 Define Documentation

##### 10.31.1.1 #define d\_func()

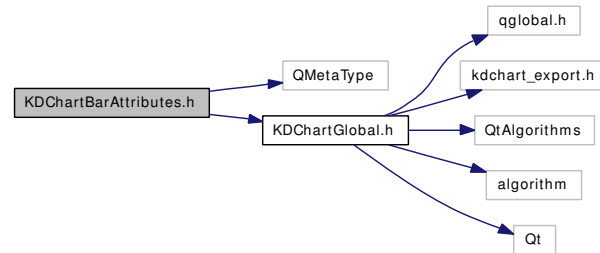
Definition at line 31 of file KDChartBarAttributes.cpp.

## 10.32 KDChartBarAttributes.h File Reference

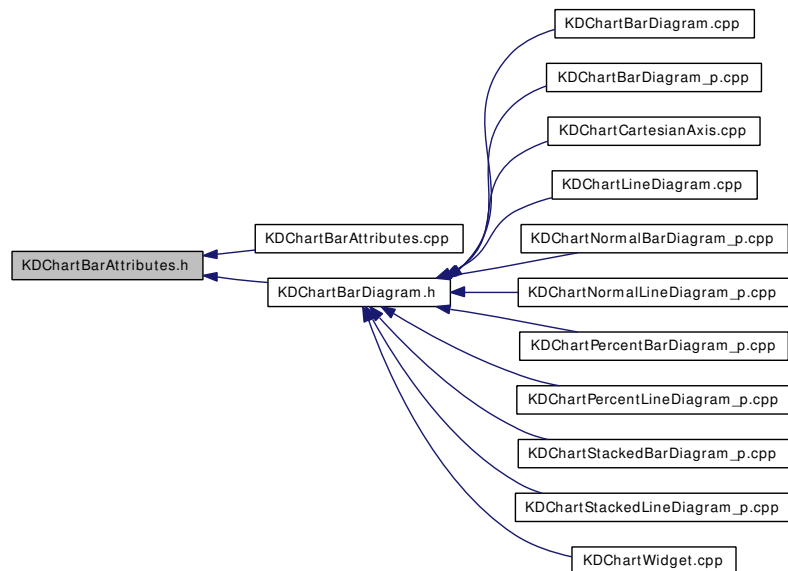
```
#include <QMetaType>
```

```
#include "KDChartGlobal.h"
```

Include dependency graph for KDChartBarAttributes.h:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [KDChart](#)

### Classes

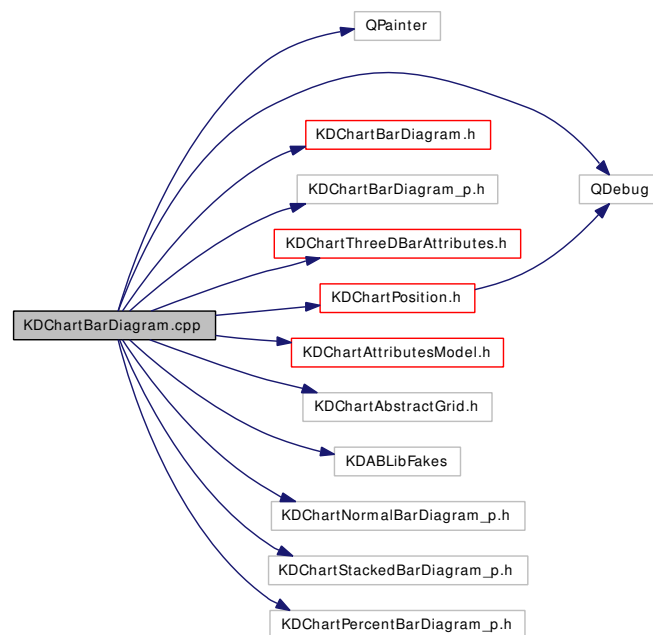
- class [KDChart::BarAttributes](#)

*Set of attributes for changing the appearance of bar charts.*

## 10.33 KDChartBarDiagram.cpp File Reference

```
#include <QPainter>
#include <QDebug>
#include "KDChartBarDiagram.h"
#include "KDChartBarDiagram_p.h"
#include "KDChartThreeDBarAttributes.h"
#include "KDChartPosition.h"
#include "KDChartAttributesModel.h"
#include "KDChartAbstractGrid.h"
#include <KDABLibFakes>
#include "KDChartNormalBarDiagram_p.h"
#include "KDChartStackedBarDiagram_p.h"
#include "KDChartPercentBarDiagram_p.h"
```

Include dependency graph for KDChartBarDiagram.cpp:



## Defines

- #define `d_func()`

### 10.33.1 Define Documentation

#### 10.33.1.1 `#define d d_func()`

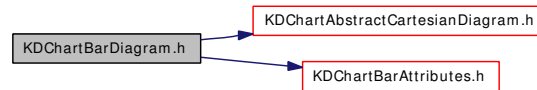
Definition at line 52 of file KDChartBarDiagram.cpp.

## 10.34 KDChartBarDiagram.h File Reference

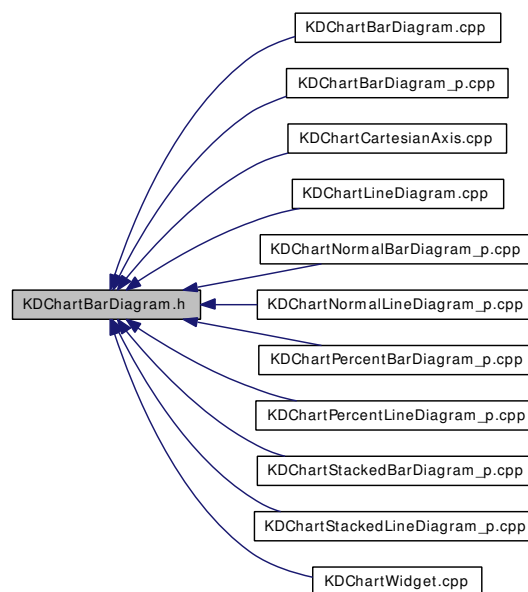
```
#include "KDChartAbstractCartesianDiagram.h"
```

```
#include "KDChartBarAttributes.h"
```

Include dependency graph for KDChartBarDiagram.h:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [KDChart](#)

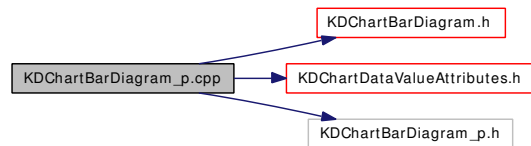
### Classes

- class [KDChart::BarDiagram](#)  
*BarDiagram* defines a common bar diagram.

## 10.35 KDChartBarDiagram\_p.cpp File Reference

```
#include "KDChartBarDiagram.h"  
#include "KDChartDataValueAttributes.h"  
#include "KDChartBarDiagram_p.h"
```

Include dependency graph for KDChartBarDiagram\_p.cpp:

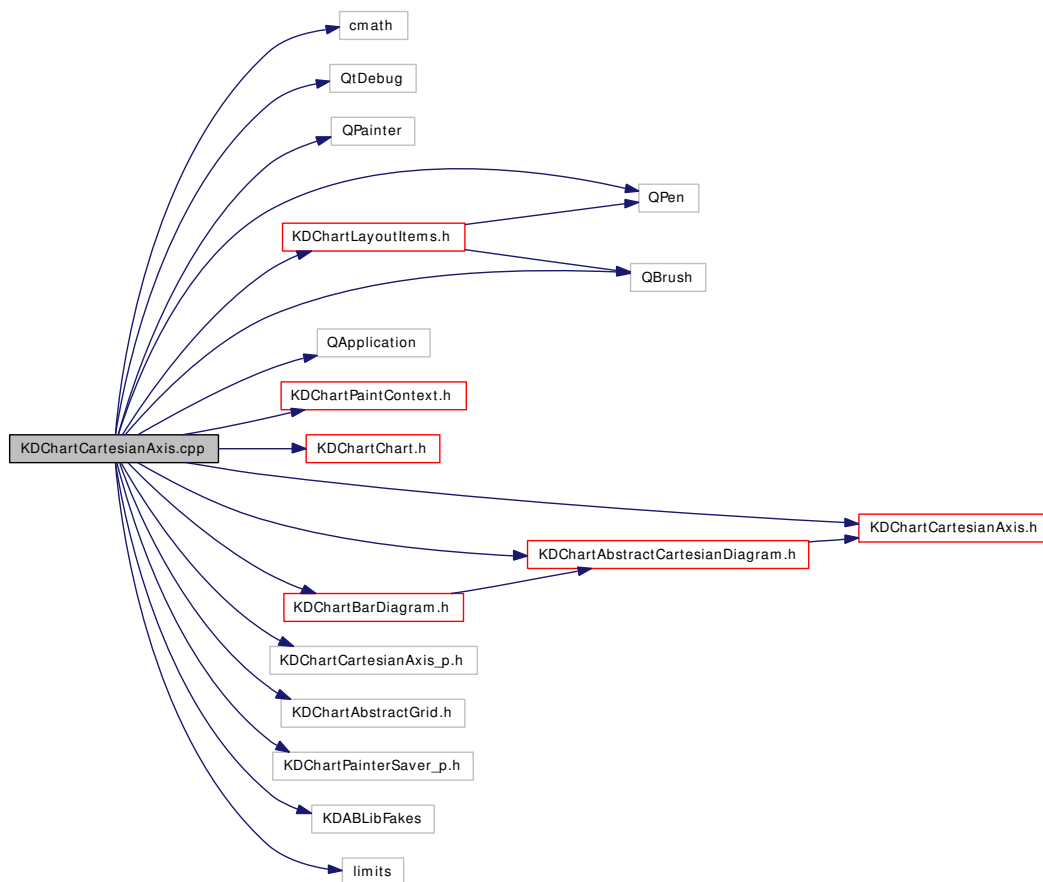




## 10.36 KDChartCartesianAxis.cpp File Reference

```
#include <cmath>
#include <QtDebug>
#include <QPainter>
#include <QPen>
#include <QBrush>
#include <QApplication>
#include "KDChartPaintContext.h"
#include "KDChartChart.h"
#include "KDChartCartesianAxis.h"
#include "KDChartCartesianAxis_p.h"
#include "KDChartAbstractCartesianDiagram.h"
#include "KDChartAbstractGrid.h"
#include "KDChartPainterSaver_p.h"
#include "KDChartLayoutItems.h"
#include "KDChartBarDiagram.h"
#include <KDABLibFakes>
#include <limits>
```

Include dependency graph for KDChartCartesianAxis.cpp:



## Defines

- `#define d (d_func())`

## Functions

- static void `calculateNextLabel` (qreal &labelValue, qreal step, bool isLogarithmic)
- static void `calculateOverlap` (int i, int first, int last, int measure, bool isBarDiagram, int &firstOverlap, int &lastOverlap)
- static bool `referenceDiagramIsBarDiagram` (const `AbstractDiagram` \*diagram)

### 10.36.1 Define Documentation

#### 10.36.1.1 `#define d (d_func())`

Definition at line 50 of file KDChartCartesianAxis.cpp.

## 10.36.2 Function Documentation

### 10.36.2.1 static void calculateNextLabel (qreal & *labelValue*, qreal *step*, bool *isLogarithmic*) [static]

Definition at line 349 of file KDChartCartesianAxis.cpp.

Referenced by KDChart::CartesianAxis::paintCtx().

```

350 {
351     if ( isLogarithmic ){
352         labelValue *= 10.0;
353         if( labelValue == 0.0 )
354             labelValue = 1.0; //std::numeric_limits< double >::epsilon();
355     }else{
356         //qDebug() << "new axis label:" << labelValue << "+" << step << "=" << labelValue+step;
357         labelValue += step;
358     }
359 /*     if( qAbs(labelValue) < 1.0e-15 )
360         labelValue = 0.0; */
361 }
```

### 10.36.2.2 static void calculateOverlap (int *i*, int *first*, int *last*, int *measure*, bool *isBarDiagram*, int & *firstOverlap*, int & *lastOverlap*) [static]

Definition at line 1012 of file KDChartCartesianAxis.cpp.

Referenced by KDChart::CartesianAxis::maximumSize().

```

1016 {
1017     if( i == first ){
1018         if( isBarDiagram ){
1019             //TODO(khz): Calculate the amount of left overlap
1020             //             for bar diagrams.
1021         }else{
1022             firstOverlap = measure / 2;
1023         }
1024     }
1025     // we test both bounds in on go: first and last might be equal
1026     if( i == last ){
1027         if( isBarDiagram ){
1028             //TODO(khz): Calculate the amount of right overlap
1029             //             for bar diagrams.
1030         }else{
1031             lastOverlap = measure / 2;
1032         }
1033     }
1034 }
```

### 10.36.2.3 static bool referenceDiagramIsBarDiagram (const [AbstractDiagram](#) \* *diagram*) [static]

Definition at line 364 of file KDChartCartesianAxis.cpp.

References KDChart::AbstractCartesianDiagram::referenceDiagram().

Referenced by KDChart::CartesianAxis::maximumSize(), and KDChart::CartesianAxis::paintCtx().

```

365 {
```

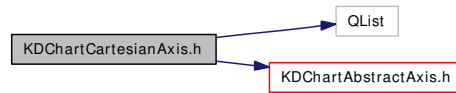
```
366     const AbstractCartesianDiagram * dia =
367         qobject_cast< const AbstractCartesianDiagram * >( diagram );
368     if( dia && dia->referenceDiagram() )
369         dia = dia->referenceDiagram();
370     return qobject_cast< const BarDiagram* >( dia ) != 0;
371 }
```

## 10.37 KDChartCartesianAxis.h File Reference

```
#include <QList>
```

```
#include "KDChartAbstractAxis.h"
```

Include dependency graph for KDChartCartesianAxis.h:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [KDChart](#)

### Classes

- class [KDChart::CartesianAxis](#)  
*The class for cartesian axes.*

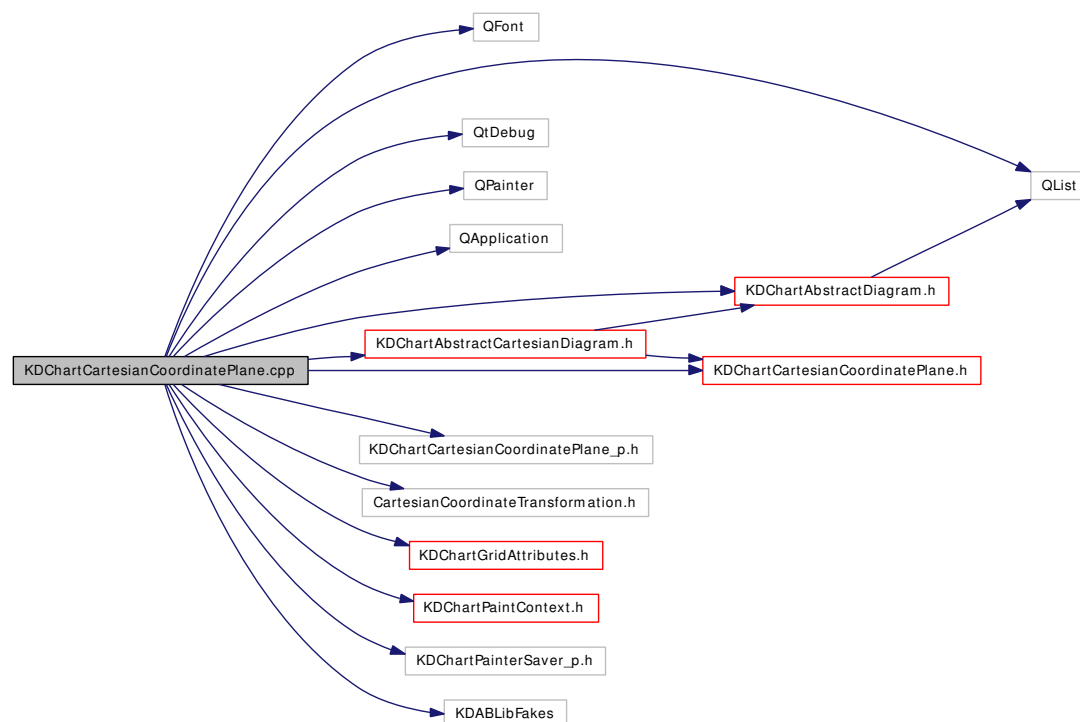
### Typedefs

- typedef `QList< CartesianAxis * >` [KDChart::CartesianAxisList](#)

## 10.38 KDChartCartesianCoordinatePlane.cpp File Reference

```
#include <QFont>
#include <QList>
#include <QtDebug>
#include <QPainter>
#include <QApplication>
#include "KDChartAbstractDiagram.h"
#include "KDChartAbstractCartesianDiagram.h"
#include "KDChartCartesianCoordinatePlane.h"
#include "KDChartCartesianCoordinatePlane_p.h"
#include "CartesianCoordinateTransformation.h"
#include "KDChartGridAttributes.h"
#include "KDChartPaintContext.h"
#include "KDChartPainterSaver_p.h"
#include <KDABLibFakes>
```

Include dependency graph for KDChartCartesianCoordinatePlane.cpp:



### Defines

- #define `d_func()`

## 10.38.1 Define Documentation

### 10.38.1.1 `#define d_d_func()`

Definition at line 46 of file KDChartCartesianCoordinatePlane.cpp.

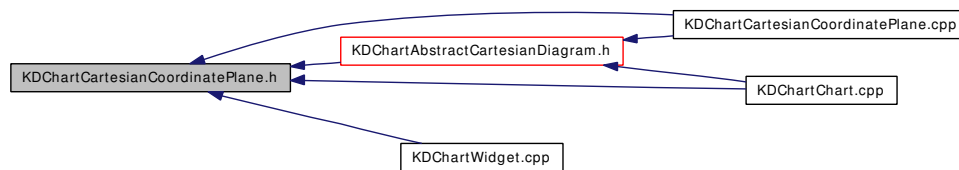
## 10.39 KDChartCartesianCoordinatePlane.h File Reference

```
#include "KDChartAbstractCoordinatePlane.h"
```

Include dependency graph for KDChartCartesianCoordinatePlane.h:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [KDChart](#)

### Classes

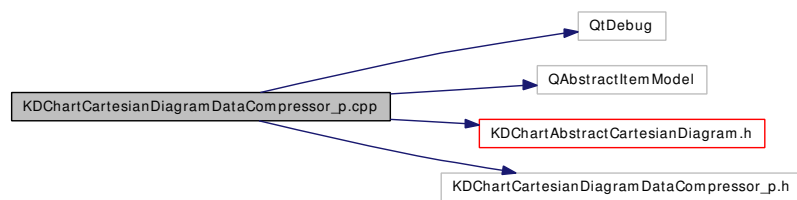
- class [KDChart::CartesianCoordinatePlane](#)  
*Cartesian coordinate plane.*



## 10.40 KDChartCartesianDiagramDataCompressor\_p.cpp File Reference

```
#include <QtDebug>
#include <QAbstractItemModel>
#include "KDChartAbstractCartesianDiagram.h"
#include "KDChartCartesianDiagramDataCompressor_p.h"
```

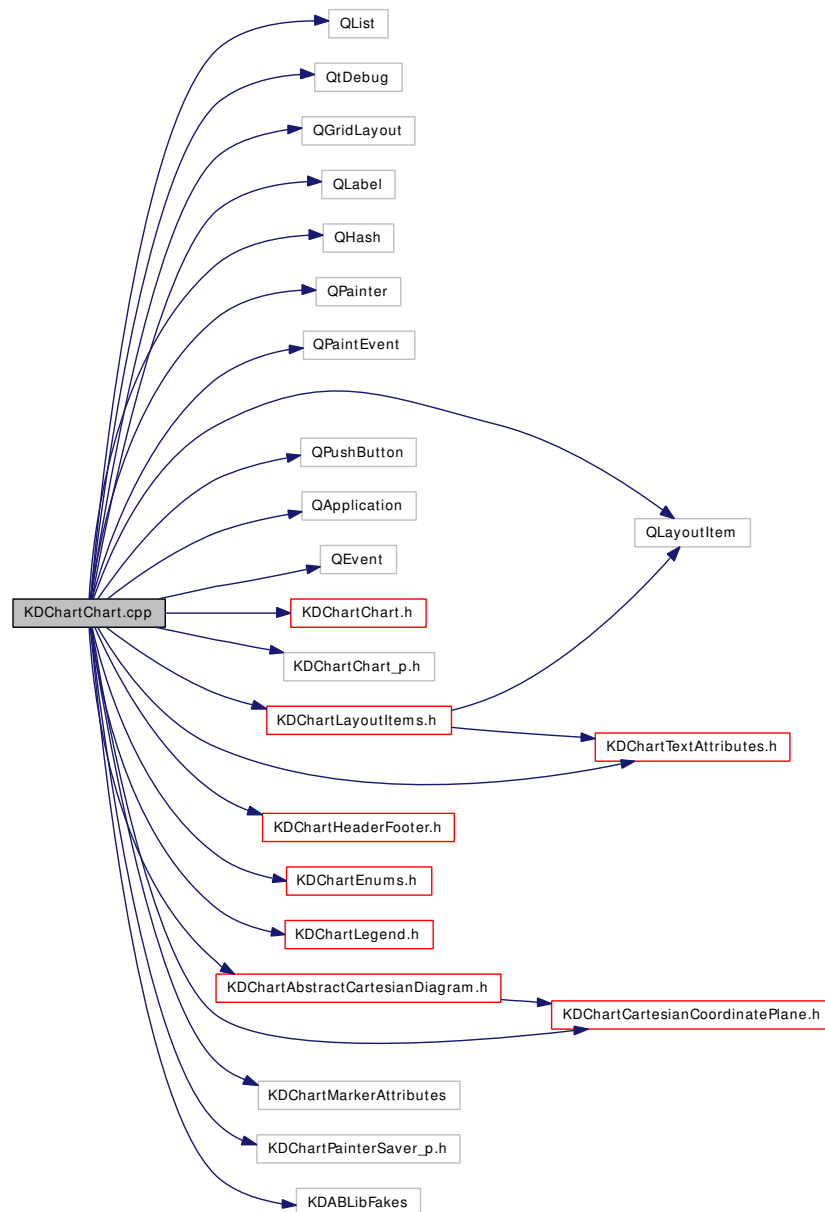
Include dependency graph for KDChartCartesianDiagramDataCompressor\_p.cpp:



## 10.41 KDChartChart.cpp File Reference

```
#include <QList>
#include <QtDebug>
#include <QGridLayout>
#include <QLabel>
#include <QHash>
#include <QPainter>
#include <QPaintEvent>
#include <QLayoutItem>
#include <QPushButton>
#include <QApplication>
#include <QEvent>
#include "KDChartChart.h"
#include "KDChartChart_p.h"
#include "KDChartCartesianCoordinatePlane.h"
#include "KDChartAbstractCartesianDiagram.h"
#include "KDChartHeaderFooter.h"
#include "KDChartEnums.h"
#include "KDChartLegend.h"
#include "KDChartLayoutItems.h"
#include <KDChartTextAttributes.h>
#include <KDChartMarkerAttributes>
#include "KDChartPainterSaver_p.h"
#include <KDABLibFakes>
```

Include dependency graph for KDChartChart.cpp:



## Defines

- #define [ADD\\_AUTO\\_SPACER\\_IF\\_NEEDED](#)(spacerRow, spacerColumn, hLayoutIsAtTop, hLayout, vLayoutIsAtLeft, vLayout)
- #define [ADD\\_VBOX\\_WITH\\_LEGENDS](#)(row, column, align)
- #define [d\\_func\(\)](#)
- #define [SET\\_ALL\\_MARGINS\\_TO\\_ZERO](#)

## Functions

- template<typename T> static T \* [findOrCreateLayoutByObjectName](#) (QLayout \*parentLayout, const char \*name)

## 10.41.1 Define Documentation

### 10.41.1.1 `#define ADD_AUTO_SPACER_IF_NEEDED(spacerRow, spacerColumn, hLayoutIsAtTop, hLayout, vLayoutIsAtLeft, vLayout)`

**Value:**

```
{ \
    if( hLayout || vLayout ) { \
        AutoSpacerLayoutItem * spacer \
        = new AutoSpacerLayoutItem( hLayoutIsAtTop, hLayout, vLayoutIsAtLeft, vLayout ); \
        planeLayout->addItem( spacer, spacerRow, spacerColumn, 1, 1 ); \
        spacer->setParentLayout( planeLayout ); \
        planeLayoutItems << spacer; \
    } \
}
```

### 10.41.1.2 `#define ADD_VBOX_WITH_LEGENDS(row, column, align)`

**Value:**

```
{ \
    QVBoxLayout* innerLayout = new QVBoxLayout(); \
    for (int i = 0; i < count; ++i) { \
        legend = list.at(i); \
        if( legend->alignment() == ( align ) ) \
            innerLayout->addItem( new MyWidgetItem(legend, Qt::AlignLeft) ); \
    } \
    gridLayout->addLayout( innerLayout, row, column, ( align ) ); \
}
```

### 10.41.1.3 `#define d d_func()`

Definition at line 805 of file KDChartChart.cpp.

### 10.41.1.4 `#define SET_ALL_MARGINS_TO_ZERO`

Definition at line 57 of file KDChartChart.cpp.

## 10.41.2 Function Documentation

### 10.41.2.1 `template<typename T> static T* findOrCreateLayoutByObjectName (QLayout * parentLayout, const char * name) [static]`

Definition at line 423 of file KDChartChart.cpp.

```
424 {
425     T *box = qFindChild<T*>( parentLayout, QString::fromLatin1( name ) );
426     if ( !box ) {
427         box = new T();
428         // TESTING(khz): set the margin of all of the layouts to Zero
429 #if defined SET_ALL_MARGINS_TO_ZERO
430         box->setMargin(0);
431 #endif
432     }
```

```
432         box->setObjectName( QString::fromLatin1( name ) );
433         box->setSizeConstraint( QLayout::SetFixedSize );
434     }
435     return box;
436 }
```

## 10.42 KDChartChart.h File Reference

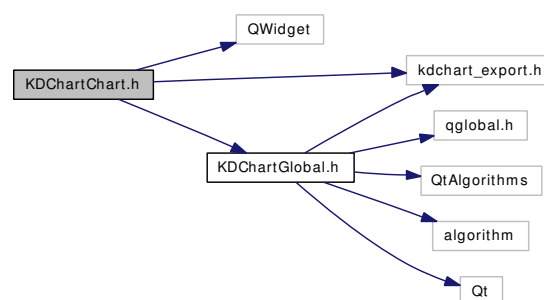
### 10.42.1 Detailed Description

Declaring the class [KDChart::Chart](#).

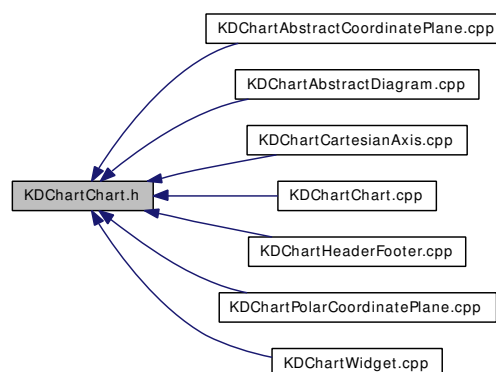
Definition in file [KDChartChart.h](#).

```
#include <QWidget>
#include "kdchart_export.h"
#include "KDChartGlobal.h"
```

Include dependency graph for KDChartChart.h:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace [KDChart](#)

## Classes

- class [KDChart::Chart](#)

*A chart with one or more diagrams.*

## Typedefs

- typedef QList< AbstractCoordinatePlane \* > [KDChart::CoordinatePlaneList](#)
- typedef QList< HeaderComponent \* > [KDChart::HeaderFooterList](#)
- typedef QList< Legend \* > [KDChart::LegendList](#)

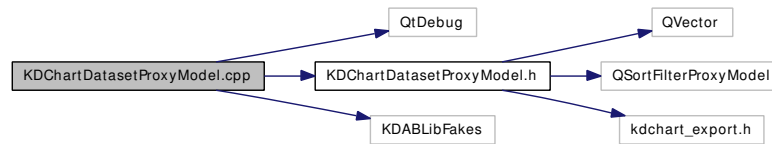
## 10.43 KDChartDatasetProxyModel.cpp File Reference

```
#include <QtDebug>
```

```
#include "KDChartDatasetProxyModel.h"
```

```
#include <KDABLibFakes>
```

Include dependency graph for KDChartDatasetProxyModel.cpp:

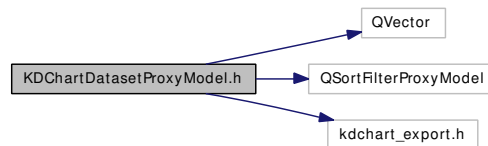




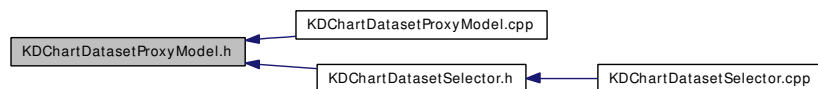
## 10.44 KDChartDatasetProxyModel.h File Reference

```
#include <QVector>
#include <QSortFilterProxyModel>
#include "kdchart_export.h"
```

Include dependency graph for KDChartDatasetProxyModel.h:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [KDChart](#)

### Classes

- class [KDChart::DatasetProxyModel](#)  
*[DatasetProxyModel](#) takes a [KDChart](#) dataset configuration and translates it into a filtering proxy model.*

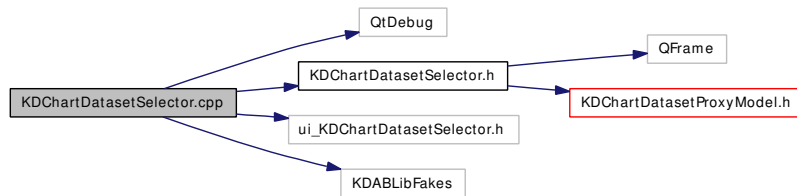
### Typedefs

- typedef `QVector< int >` [KDChart::DatasetDescriptionVector](#)

## 10.45 KDChartDatasetSelector.cpp File Reference

```
#include <QtDebug>
#include "KDChartDatasetSelector.h"
#include "ui_KDChartDatasetSelector.h"
#include <KDABLibFakes>
```

Include dependency graph for KDChartDatasetSelector.cpp:

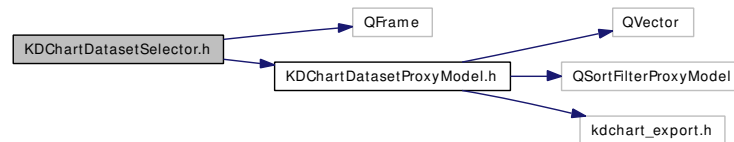


## 10.46 KDChartDatasetSelector.h File Reference

```
#include <QFrame>
```

```
#include "KDChartDatasetProxyModel.h"
```

Include dependency graph for KDChartDatasetSelector.h:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [KDChart](#)

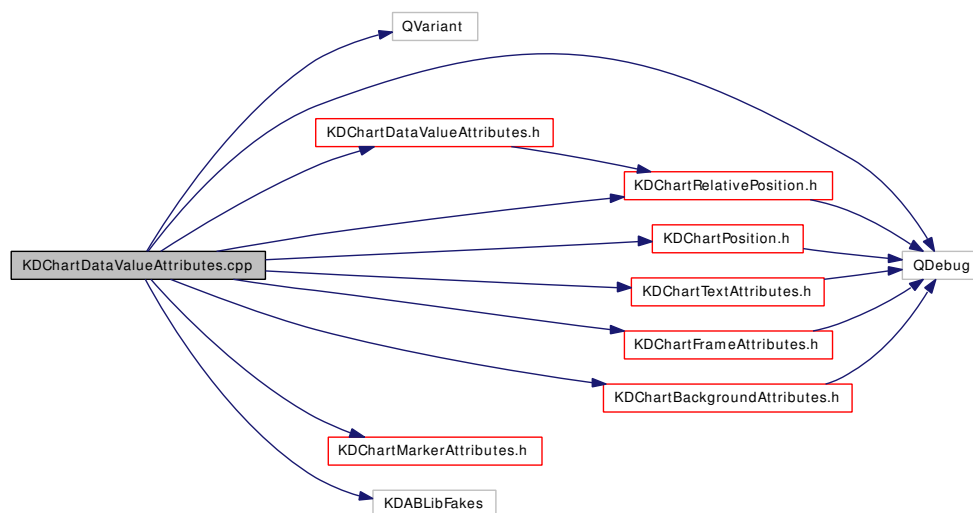
### Classes

- class [KDChart::DatasetSelectorWidget](#)

## 10.47 KDChartDataValueAttributes.cpp File Reference

```
#include <QVariant>
#include <QDebug>
#include "KDChartDataValueAttributes.h"
#include "KDChartRelativePosition.h"
#include "KDChartPosition.h"
#include <KDChartTextAttributes.h>
#include <KDChartFrameAttributes.h>
#include <KDChartBackgroundAttributes.h>
#include <KDChartMarkerAttributes.h>
#include <KDABLibFakes>
```

Include dependency graph for KDChartDataValueAttributes.cpp:



### Defines

- #define `d_func()`
- #define `KDCHART_DATA_VALUE_AUTO_DIGITS` 4

### Functions

- QDebug `operator<<` (QDebug dbg, const KDChart::DataValueAttributes &val)

#### 10.47.1 Define Documentation

##### 10.47.1.1 #define `d_func()`

Definition at line 43 of file KDChartDataValueAttributes.cpp.

### 10.47.1.2 #define KDCHART\_DATA\_VALUE\_AUTO\_DIGITS 4

Definition at line 40 of file KDChartDataValueAttributes.cpp.

## 10.47.2 Function Documentation

### 10.47.2.1 QDebug operator<< (QDebug *dbg*, const [KDChart::DataValueAttributes](#) & *val*)

Definition at line 324 of file KDChartDataValueAttributes.cpp.

```
325 {
326     dbg << "RelativePosition DataValueAttributes ("
327         << "visible=" << val.isVisible()
328         << "textattributes=" << val.textAttributes()
329         << "frameattributes=" << val.frameAttributes()
330         << "backgroundattributes=" << val.backgroundAttributes()
331         << "decimaldigits=" << val.decimalDigits()
332         << "poweroftendivisor=" << val.powerOfTenDivisor()
333         << "showinfinite=" << val.showInfinite()
334         << "negativerelativeposition=" << val.negativePosition()
335         << "positiverelativeposition=" << val.positivePosition()
336         << "showRepetitiveDataLabels=" << val.showRepetitiveDataLabels()
337         << ") ";
338     return dbg;
339 }
```

## 10.48 KDChartDataValueAttributes.h File Reference

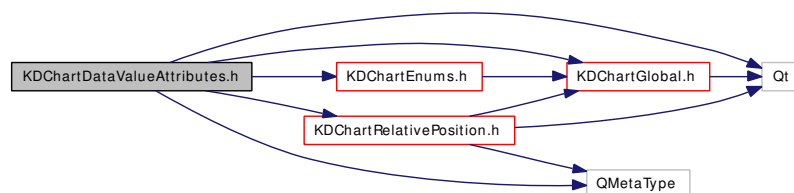
### 10.48.1 Detailed Description

Declaring the class [KDChart::DataValueAttributes](#).

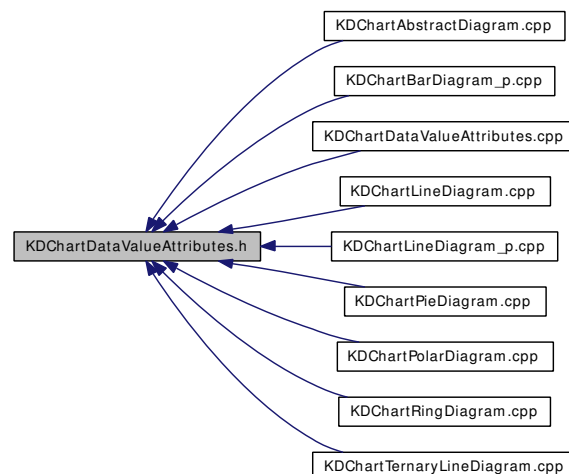
Definition in file [KDChartDataValueAttributes.h](#).

```
#include <Qt>
#include <QMetaType>
#include "KDChartGlobal.h"
#include "KDChartEnums.h"
#include "KDChartRelativePosition.h"
```

Include dependency graph for KDChartDataValueAttributes.h:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace [KDChart](#)

## Classes

- class [KDChart::DataValueAttributes](#)  
*Diagram attributes dealing with data value labels.*

## Functions

- KDCHART\_EXPORT QDebug [operator<<](#) (QDebug, const [KDChart::DataValueAttributes](#) &)
- [Q\\_DECLARE\\_TYPEINFO](#) ([KDChart::DataValueAttributes](#), Q\_MOVABLE\_TYPE)

### 10.48.2 Function Documentation

#### 10.48.2.1 KDCHART\_EXPORT QDebug operator<< (QDebug, const [KDChart::DataValueAttributes](#) &)

Definition at line 324 of file `KDChartDataValueAttributes.cpp`.

References [KDChart::DataValueAttributes::backgroundAttributes\(\)](#), [KDChart::DataValueAttributes::decimalDigits\(\)](#), [KDChart::DataValueAttributes::frameAttributes\(\)](#), [KDChart::DataValueAttributes::isVisible\(\)](#), [KDChart::DataValueAttributes::negativePosition\(\)](#), [KDChart::DataValueAttributes::positivePosition\(\)](#), and [KDChart::DataValueAttributes::textAttributes\(\)](#).

```

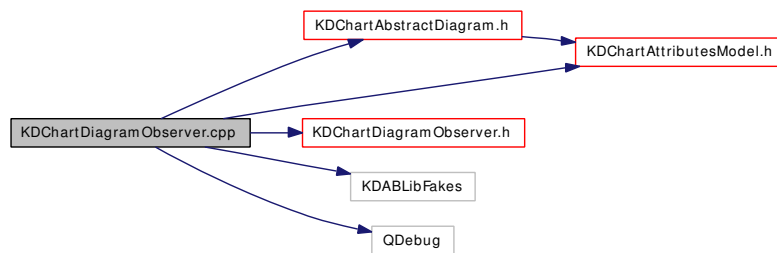
325 {
326     dbg << "RelativePosition DataValueAttributes ("
327         << "visible=" << val.isVisible()
328         << "textattributes=" << val.textAttributes()
329         << "frameattributes=" << val.frameAttributes()
330         << "backgroundattributes=" << val.backgroundAttributes()
331         << "decimaldigits=" << val.decimalDigits()
332         << "poweroftendivisor=" << val.powerOfTenDivisor()
333         << "showinfinite=" << val.showInfinite()
334         << "negativerelativeposition=" << val.negativePosition()
335         << "positiverelativeposition=" << val.positivePosition()
336         << "showRepetitiveDataLabels=" << val.showRepetitiveDataLabels()
337         << ") ";
338     return dbg;
339 }
```

#### 10.48.2.2 Q\_DECLARE\_TYPEINFO ([KDChart::DataValueAttributes](#), Q\_MOVABLE\_TYPE)

## 10.49 KDChartDiagramObserver.cpp File Reference

```
#include <KDChartAbstractDiagram.h>
#include <KDChartDiagramObserver.h>
#include <KDChartAttributesModel.h>
#include <KDABLibFakes>
#include <QDebug>
```

Include dependency graph for KDChartDiagramObserver.cpp:

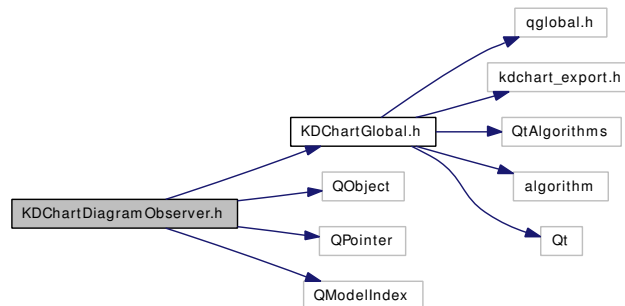




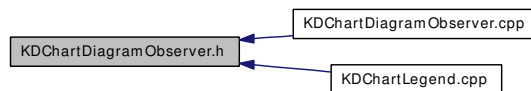
## 10.50 KDChartDiagramObserver.h File Reference

```
#include "KDChartGlobal.h"  
#include <QObject>  
#include <QPointer>  
#include <QModelIndex>
```

Include dependency graph for KDChartDiagramObserver.h:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [KDChart](#)

### Classes

- class [KDChart::DiagramObserver](#)

A [DiagramObserver](#) watches the associated diagram for changes and deletion and emits corresponding signals.

## 10.51 KDChartEnums.h File Reference

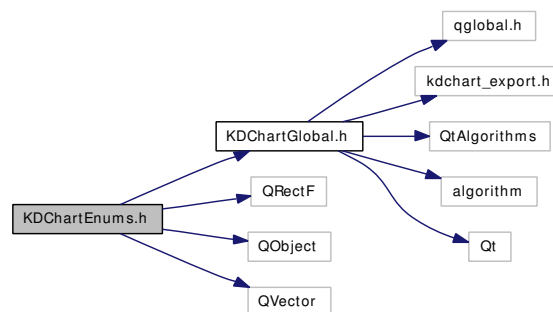
### 10.51.1 Detailed Description

Definition of global enums.

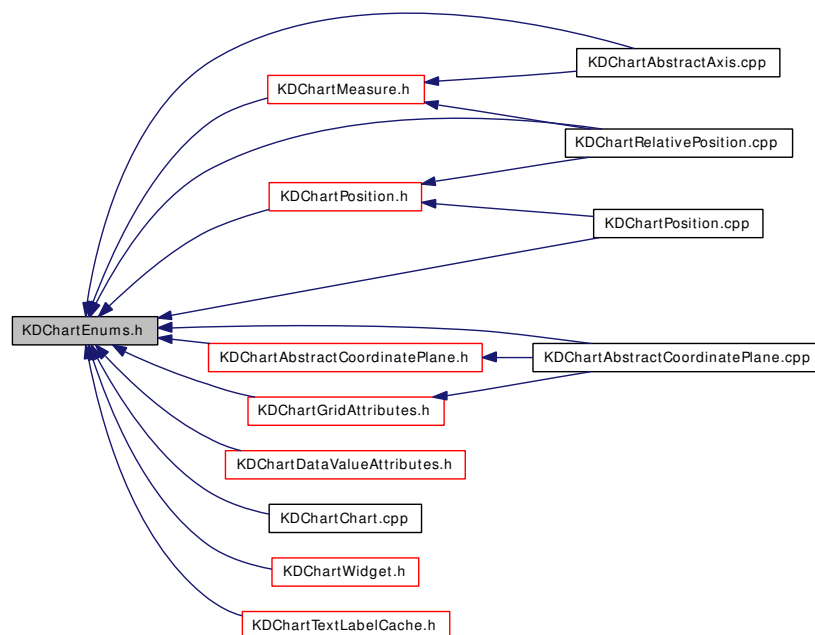
Definition in file [KDChartEnums.h](#).

```
#include "KDChartGlobal.h"
#include <QRectF>
#include <QObject>
#include <QVector>
```

Include dependency graph for KDChartEnums.h:



This graph shows which files directly or indirectly include this file:



## Classes

- class [KDChartEnums](#)

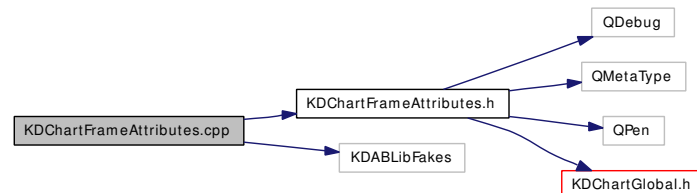
*Project global class providing some enums needed both by KDChartParams and by KDChartCustomBox.*

## 10.52 KDChartFrameAttributes.cpp File Reference

```
#include "KDChartFrameAttributes.h"
```

```
#include <KDABLibFakes>
```

Include dependency graph for KDChartFrameAttributes.cpp:



### Defines

- #define [d\\_d\\_func\(\)](#)

### Functions

- QDebug [operator<<](#) (QDebug dbg, const [KDChart::FrameAttributes](#) &fa)

#### 10.52.1 Define Documentation

##### 10.52.1.1 #define d\_d\_func()

Definition at line 30 of file KDChartFrameAttributes.cpp.

#### 10.52.2 Function Documentation

##### 10.52.2.1 QDebug operator<< (QDebug *dbg*, const [KDChart::FrameAttributes](#) &*fa*)

Definition at line 124 of file KDChartFrameAttributes.cpp.

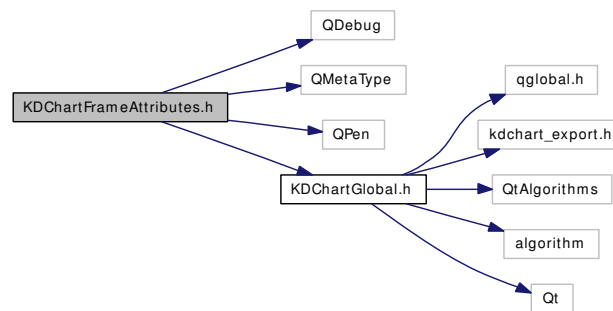
```

125 {
126     dbg << "KDChart::FrameAttributes ("
127         << "visible=" << fa.isVisible()
128         << "pen=" << fa.pen()
129         << "padding=" << fa.padding()
130         << ") ";
131     return dbg;
132 }
```

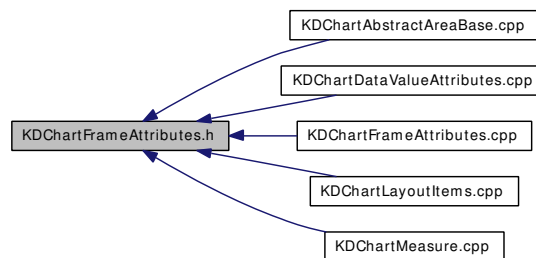
## 10.53 KDChartFrameAttributes.h File Reference

```
#include <QDebug>
#include <QMetaType>
#include <QPen>
#include "KDChartGlobal.h"
```

Include dependency graph for KDChartFrameAttributes.h:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace [KDChart](#)

## Classes

- class [KDChart::FrameAttributes](#)  
*A set of attributes for frames around items.*

## Functions

- KDCHART\_EXPORT QDebug [operator<<](#) (QDebug, const [KDChart::FrameAttributes](#) &)
- [Q\\_DECLARE\\_TYPEINFO](#) ([KDChart::FrameAttributes](#), Q\_MOVABLE\_TYPE)

## 10.53.1 Function Documentation

### 10.53.1.1 KDCHART\_EXPORT QDebug operator<< (QDebug, const [KDChart::FrameAttributes](#) &)

Definition at line 124 of file KDChartFrameAttributes.cpp.

References [KDChart::FrameAttributes::isVisible\(\)](#), [KDChart::FrameAttributes::padding\(\)](#), and [KDChart::FrameAttributes::pen\(\)](#).

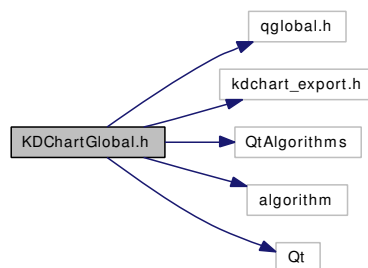
```
125 {  
126     dbg << "KDChart::FrameAttributes ("  
127         << "visible=" << fa.isVisible()  
128         << "pen=" << fa.pen()  
129         << "padding=" << fa.padding()  
130         << ") ";  
131     return dbg;  
132 }
```

### 10.53.1.2 Q\_DECLARE\_TYPEINFO ([KDChart::FrameAttributes](#), Q\_MOVABLE\_TYPE)

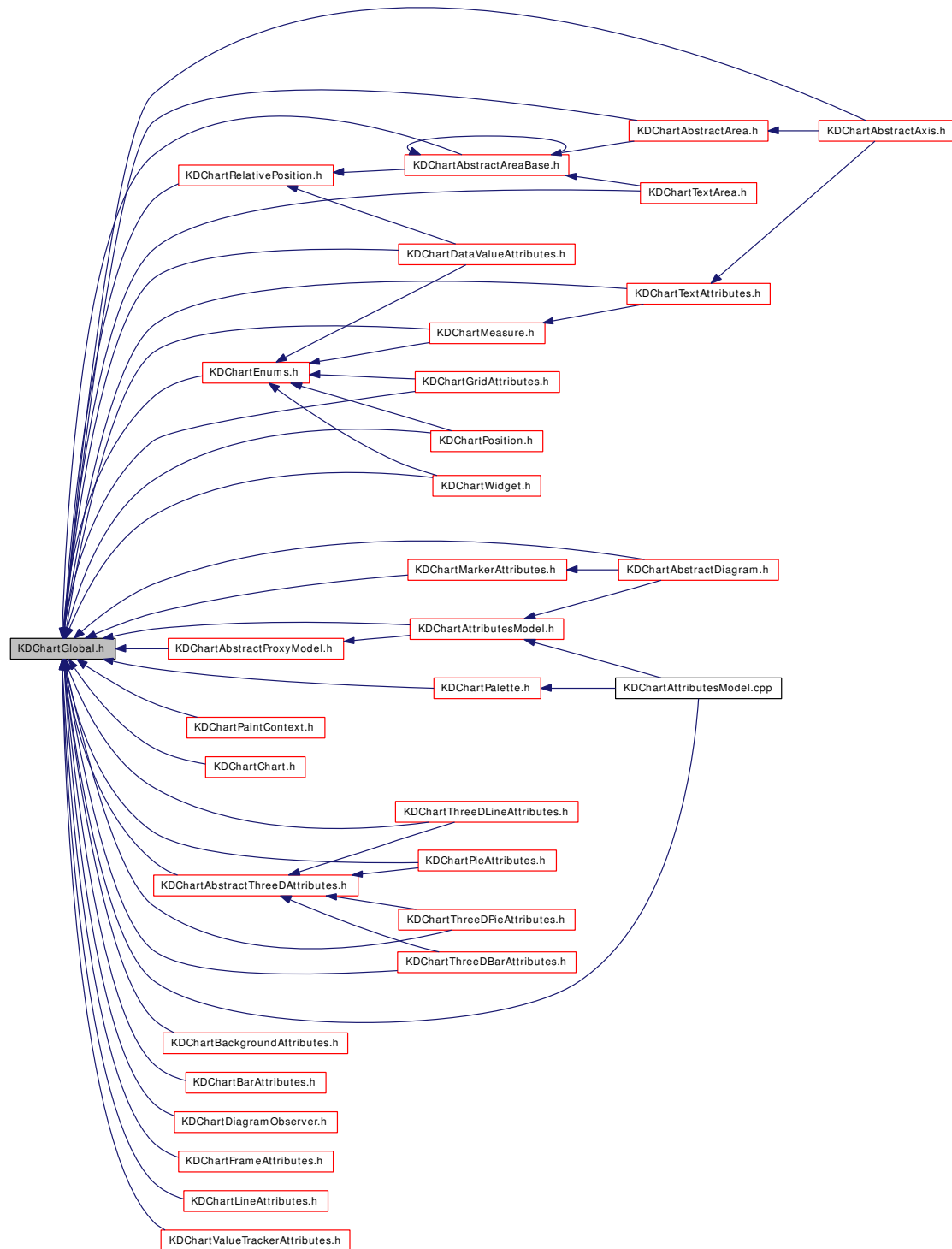
## 10.54 KDChartGlobal.h File Reference

```
#include <qglobal.h>
#include "kdchart_export.h"
#include <QtAlgorithms>
#include <algorithm>
#include <Qt>
```

Include dependency graph for KDChartGlobal.h:



This graph shows which files directly or indirectly include this file:





## Namespaces

- namespace [KDChart](#)

## Defines

- #define [KDAB\\_SET\\_OBJECT\\_NAME](#)(x) `__kdab__dereference_for_methodcall( x ).setObjectName( QLatin1String( #x ) )`
- #define [KDCHART\\_DECLARE\\_DERIVED\\_DIAGRAM](#)(X, PLANE)
- #define [KDCHART\\_DECLARE\\_PRIVATE\\_BASE\\_POLYMORPHIC](#)(X)
- #define [KDCHART\\_DECLARE\\_PRIVATE\\_BASE\\_POLYMORPHIC\\_QWIDGET](#)(X)
- #define [KDCHART\\_DECLARE\\_PRIVATE\\_BASE\\_VALUE](#)(X)
- #define [KDCHART\\_DECLARE\\_PRIVATE\\_DERIVED](#)(X)
- #define [KDCHART\\_DECLARE\\_PRIVATE\\_DERIVED\\_PARENT](#)(X, ParentType)
- #define [KDCHART\\_DECLARE\\_PRIVATE\\_DERIVED\\_QWIDGET](#)(X) `KDCHART_DECLARE_PRIVATE_DERIVED_PARENT( X, QWidget* )`
- #define [KDCHART\\_DECLARE\\_SWAP\\_BASE](#)(X)
- #define [KDCHART\\_DECLARE\\_SWAP\\_DERIVED](#)(X) `void swap( X& other ) { doSwap( other ); }`
- #define [KDCHART\\_DECLARE\\_SWAP\\_SPECIALISATION](#)(X)
- #define [KDCHART\\_DECLARE\\_SWAP\\_SPECIALISATION\\_DERIVED](#)(X) `KDCHART_DECLARE_SWAP_SPECIALISATION( X )`
- #define [KDCHART\\_DERIVED\\_PRIVATE\\_FOOTER](#)(CLASS, PARENT)
- #define [KDCHART\\_IMPL\\_DERIVED\\_DIAGRAM](#)(CLASS, PARENT, PLANE)
- #define [KDCHART\\_IMPL\\_DERIVED\\_PLANE](#)(CLASS, BASEPLANE)

## Enumerations

- enum [KDChart::DisplayRoles](#) {  
[KDChart::DatasetPenRole](#) = 0x0A79EF95,  
[KDChart::DatasetBrushRole](#),  
[KDChart::DataValueLabelAttributesRole](#),  
[KDChart::ThreeDAttributesRole](#),  
[KDChart::LineAttributesRole](#),  
[KDChart::ThreeDLineAttributesRole](#),  
[KDChart::BarAttributesRole](#),  
[KDChart::ThreeDBarAttributesRole](#),  
[KDChart::PieAttributesRole](#),  
[KDChart::ThreeDPieAttributesRole](#),  
[KDChart::DataHiddenRole](#),  
[KDChart::ValueTrackerAttributesRole](#) }

## Functions

- template<typename T> T & [\\_\\_kdab\\_\\_dereference\\_for\\_methodcall](#) (T \*o)
- template<typename T> T & [\\_\\_kdab\\_\\_dereference\\_for\\_methodcall](#) (T &o)

### 10.54.1 Define Documentation

#### 10.54.1.1 `#define KDAB_SET_OBJECT_NAME(x) __kdab__dereference_for_methodcall( x ).setObjectName( QLatin1String( #x ) )`

Definition at line 46 of file KDChartGlobal.h.

#### 10.54.1.2 `#define KDCHART_DECLARE_DERIVED_DIAGRAM(X, PLANE)`

**Value:**

```
protected:                                     \
    class Private;                             \
    inline Private * d_func();                  \
    inline const Private * d_func() const;      \
    explicit inline X( Private * );             \
    explicit inline X( Private *, QWidget *, PLANE * ); \
private:                                       \
    void init();
```

Definition at line 173 of file KDChartGlobal.h.

#### 10.54.1.3 `#define KDCHART_DECLARE_PRIVATE_BASE_POLYMORPHIC(X)`

**Value:**

```
protected:                                     \
    class Private;                             \
    Private * d_func() { return _d; }           \
    const Private * d_func() const { return _d; } \
    explicit inline X( Private * );             \
private:                                       \
    void init();                               \
    Private * _d;
```

Definition at line 117 of file KDChartGlobal.h.

#### 10.54.1.4 `#define KDCHART_DECLARE_PRIVATE_BASE_POLYMORPHIC_QWIDGET(X)`

**Value:**

```
protected:                                     \
    class Private;                             \
    Private * d_func() { return _d; }           \
    const Private * d_func() const { return _d; } \
    explicit inline X( Private *, QWidget* );    \
private:                                       \
    void init();                               \
    Private * _d;
```

Definition at line 140 of file KDChartGlobal.h.

**10.54.1.5 #define KDCHART\_DECLARE\_PRIVATE\_BASE\_VALUE(X)****Value:**

```

public:
    inline void swap( X & other ) { qSwap( _d, other._d ); } \
protected:
    class Private;
    Private * d_func() { return _d; } \
    const Private * d_func() const { return _d; } \
private:
    void init(); \
    Private * _d;

```

Definition at line 94 of file KDChartGlobal.h.

**10.54.1.6 #define KDCHART\_DECLARE\_PRIVATE\_DERIVED(X)****Value:**

```

protected:
    class Private; \
    inline Private * d_func(); \
    inline const Private * d_func() const; \
    explicit inline X( Private * ); \
private:
    void init(); \

```

Definition at line 60 of file KDChartGlobal.h.

**10.54.1.7 #define KDCHART\_DECLARE\_PRIVATE\_DERIVED\_PARENT(X, ParentType)****Value:**

```

protected:
    class Private; \
    inline Private * d_func(); \
    inline const Private * d_func() const; \
    explicit inline X( Private *, ParentType ); \
private:
    void init(); \

```

Definition at line 81 of file KDChartGlobal.h.

**10.54.1.8 #define KDCHART\_DECLARE\_PRIVATE\_DERIVED\_QWIDGET(X) KDCHART\_DECLARE\_PRIVATE\_DERIVED\_PARENT( X, [QWidget\\*](#) )**

Definition at line 91 of file KDChartGlobal.h.

**10.54.1.9 #define KDCHART\_DECLARE\_SWAP\_BASE(X)****Value:**

```
protected: \
    void doSwap( X& other ) \
    { qSwap( _d, other._d); }
```

Definition at line 224 of file KDChartGlobal.h.

**10.54.1.10** **#define KDCHART\_DECLARE\_SWAP\_DERIVED(X) void swap( X& other ) { doSwap( other ); }**

Definition at line 229 of file KDChartGlobal.h.

**10.54.1.11** **#define KDCHART\_DECLARE\_SWAP\_SPECIALISATION(X)**

**Value:**

```
template <> inline void qSwap<X>( X & lhs, X & rhs ) \
{ lhs.swap( rhs ); } \
namespace std { \
    template <> inline void swap<X>( X & lhs, X & rhs ) \
    { lhs.swap( rhs ); } \
}
```

Definition at line 208 of file KDChartGlobal.h.

**10.54.1.12** **#define KDCHART\_DECLARE\_SWAP\_SPECIALISATION\_DERIVED(X) KDCHART\_DECLARE\_SWAP\_SPECIALISATION( X )**

Definition at line 221 of file KDChartGlobal.h.

**10.54.1.13** **#define KDCHART\_DERIVED\_PRIVATE\_FOOTER(CLASS, PARENT)**

**Value:**

```
inline CLASS::CLASS( Private * p ) \
: PARENT( p ) { init(); } \
inline CLASS::Private * CLASS::d_func() \
{ return static_cast<Private*>( PARENT::d_func() ); } \
inline const CLASS::Private * CLASS::d_func() const \
{ return static_cast<const Private*>( PARENT::d_func() ); }
```

Definition at line 152 of file KDChartGlobal.h.

**10.54.1.14** **#define KDCHART\_IMPL\_DERIVED\_DIAGRAM(CLASS, PARENT, PLANE)**

**Value:**

```
inline CLASS::CLASS( Private * p ) \
: PARENT( p ) { init(); } \
inline CLASS::CLASS( \
    Private * p, QWidget* parent, PLANE * plane ) \
: PARENT( p, parent, plane ) { init(); } \
inline CLASS::Private * CLASS::d_func() \
```

```

    { return static_cast<Private *>( PARENT::d_func() ); } \
inline const CLASS::Private * CLASS::d_func() const \
    { return static_cast<const Private *>( PARENT::d_func() ); }

```

Definition at line 184 of file KDChartGlobal.h.

#### 10.54.1.15 #define KDCHART\_IMPL\_DERIVED\_PLANE(CLASS, BASEPLANE)

**Value:**

```

inline CLASS::CLASS( Private * p, Chart* parent ) \
    : BASEPLANE( p, parent ) { init(); } \
inline CLASS::Private * CLASS::d_func() \
    { return static_cast<Private *>( BASEPLANE::d_func() ); } \
inline const CLASS::Private * CLASS::d_func() const \
    { return static_cast<const Private *>( BASEPLANE::d_func() ); }

```

Definition at line 196 of file KDChartGlobal.h.

### 10.54.2 Function Documentation

#### 10.54.2.1 template<typename T> T& \_\_kdab\_\_dereference\_for\_methodcall (T \* o)

Definition at line 42 of file KDChartGlobal.h.

```

42                                     {
43     return *o;
44 }

```

#### 10.54.2.2 template<typename T> T& \_\_kdab\_\_dereference\_for\_methodcall (T & o)

Definition at line 37 of file KDChartGlobal.h.

```

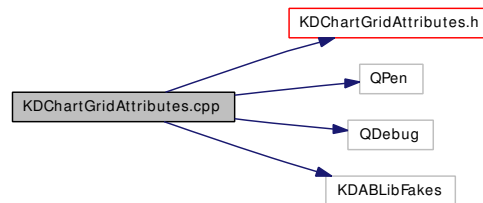
37                                     {
38     return o;
39 }

```

## 10.55 KDChartGridAttributes.cpp File Reference

```
#include "KDChartGridAttributes.h"
#include <QPen>
#include <QDebug>
#include <KDABLibFakes>
```

Include dependency graph for KDChartGridAttributes.cpp:



### Defines

- #define `d_func()`

### Functions

- `QDebug operator<< (QDebug dbg, const KDChart::GridAttributes &a)`

#### 10.55.1 Define Documentation

##### 10.55.1.1 #define `d_func()`

Definition at line 33 of file KDChartGridAttributes.cpp.

#### 10.55.2 Function Documentation

##### 10.55.2.1 `QDebug operator<< (QDebug dbg, const KDChart::GridAttributes &a)`

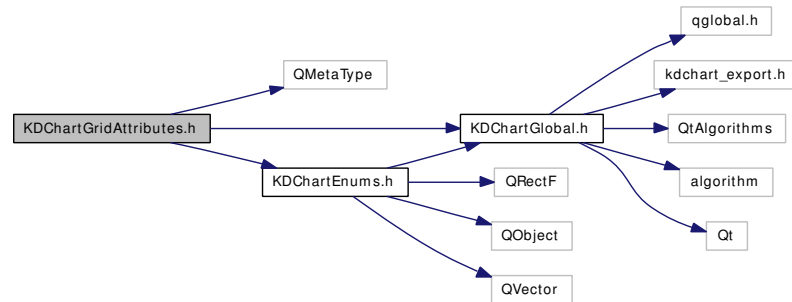
Definition at line 279 of file KDChartGridAttributes.cpp.

```
280 {
281     dbg << "KDChart::GridAttributes ("
282         << "visible=" << a.isGridVisible()
283         << "subVisible=" << a.isSubGridVisible()
284         // KDChartEnums::GranularitySequence sequence;
285         << "stepWidth=" << a.gridStepWidth()
286         << "subStepWidth=" << a.gridSubStepWidth()
287         << "pen=" << a.gridPen()
288         << "subPen=" << a.subGridPen()
289         << "zeroPen=" << a.zeroLinePen()
290         << ") ";
291     return dbg;
292 }
```

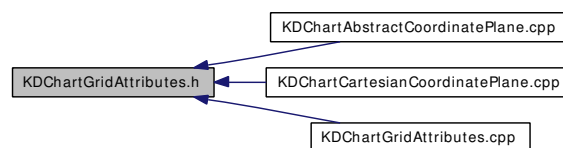
## 10.56 KDChartGridAttributes.h File Reference

```
#include <QMetaType>
#include "KDChartGlobal.h"
#include "KDChartEnums.h"
```

Include dependency graph for KDChartGridAttributes.h:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [KDChart](#)

### Classes

- class [KDChart::GridAttributes](#)  
A set of attributes controlling the appearance of grids.

### Functions

- `KDCHART_EXPORT QDebug operator<< (QDebug, const KDChart::GridAttributes &)`
- `Q_DECLARE_TYPEINFO (KDChart::GridAttributes, Q_MOVABLE_TYPE)`

#### 10.56.1 Function Documentation

##### 10.56.1.1 KDCHART\_EXPORT QDebug operator<< (QDebug, const [KDChart::GridAttributes](#) &)

Definition at line 279 of file KDChartGridAttributes.cpp.

References      [KDChart::GridAttributes::gridPen\(\)](#),      [KDChart::GridAttributes::gridStepWidth\(\)](#),  
[KDChart::GridAttributes::gridSubStepWidth\(\)](#),      [KDChart::GridAttributes::isGridVisible\(\)](#),  
[KDChart::GridAttributes::isSubGridVisible\(\)](#),      [KDChart::GridAttributes::subGridPen\(\)](#),      and  
[KDChart::GridAttributes::zeroLinePen\(\)](#).

```

280 {
281     dbg << "KDChart::GridAttributes("
282         << "visible=" << a.isGridVisible()
283         << "subVisible=" << a.isSubGridVisible()
284         // KDChartEnums::GranularitySequence sequence;
285         << "stepWidth=" << a.gridStepWidth()
286         << "subStepWidth=" << a.gridSubStepWidth()
287         << "pen=" << a.gridPen()
288         << "subPen=" << a.subGridPen()
289         << "zeroPen=" << a.zeroLinePen()
290         << ") ";
291     return dbg;
292 }
```

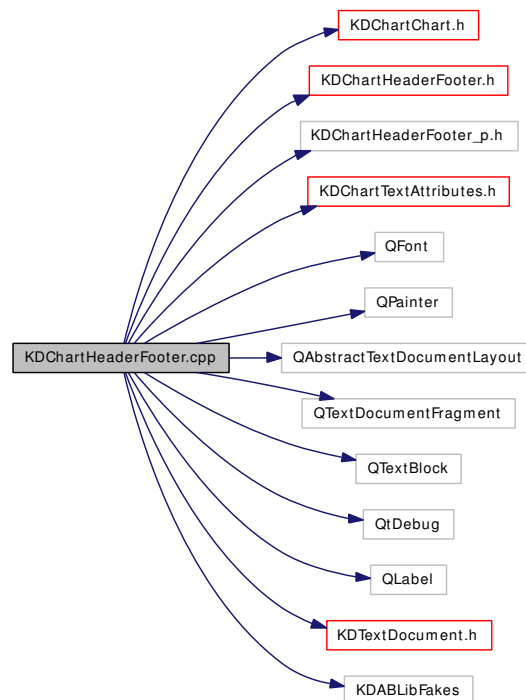
#### 10.56.1.2 Q\_DECLARE\_TYPEINFO ([KDChart::GridAttributes](#), Q\_MOVABLE\_TYPE)



## 10.57 KDChartHeaderFooter.cpp File Reference

```
#include "KDChartChart.h"
#include "KDChartHeaderFooter.h"
#include "KDChartHeaderFooter_p.h"
#include <KDChartTextAttributes.h>
#include <QFont>
#include <QPainter>
#include <QAbstractTextDocumentLayout>
#include <QTextDocumentFragment>
#include <QTextBlock>
#include <QtDebug>
#include <QLabel>
#include "KDTextDocument.h"
#include <KDABLibFakes>
```

Include dependency graph for KDChartHeaderFooter.cpp:



### Defines

- #define `d_func()`

## 10.57.1 Define Documentation

### 10.57.1.1 `#define d d_func()`

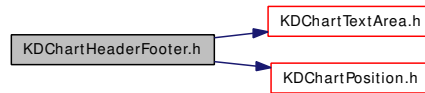
Definition at line 53 of file KDChartHeaderFooter.cpp.

## 10.58 KDChartHeaderFooter.h File Reference

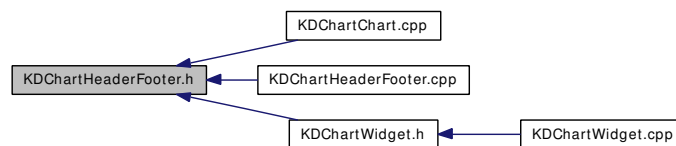
```
#include "KDChartTextArea.h"
```

```
#include "KDChartPosition.h"
```

Include dependency graph for KDChartHeaderFooter.h:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [KDChart](#)

### Classes

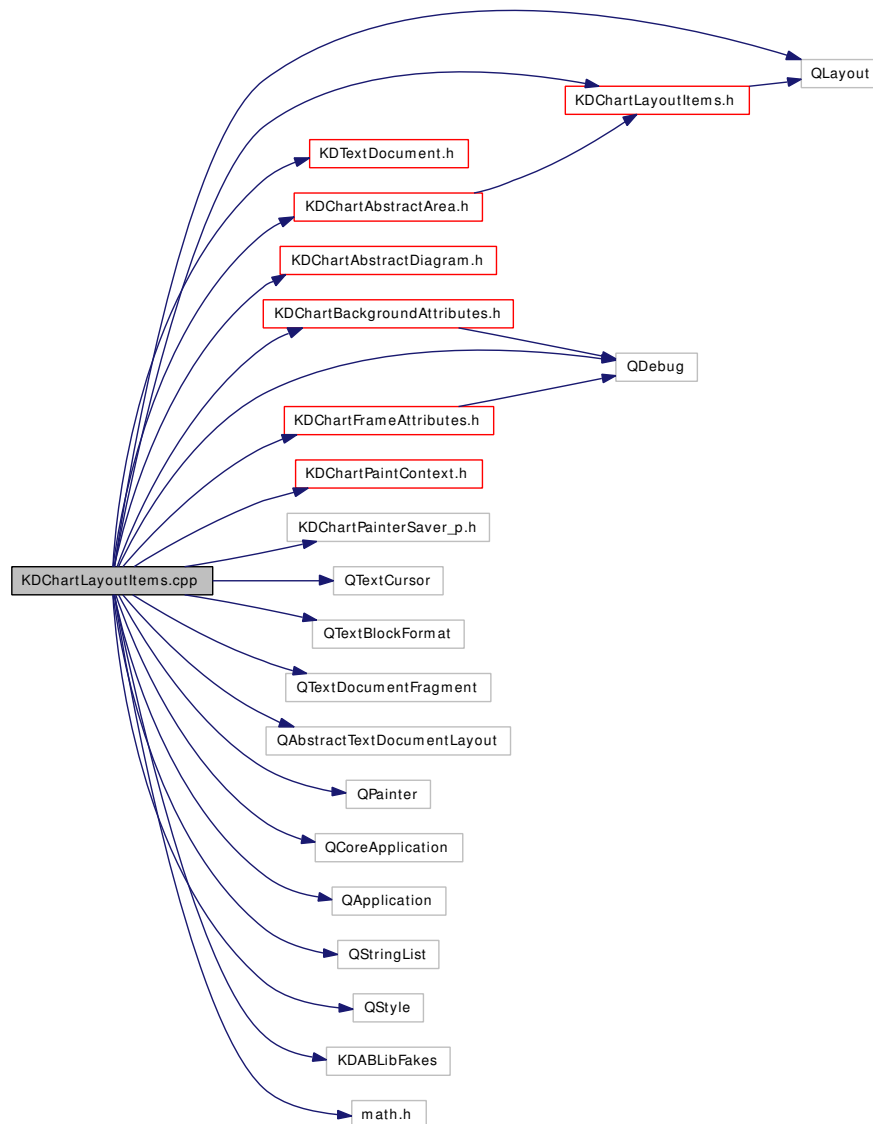
- class [KDChart::HeaderFooter](#)

*A header or even footer displaying text above or below charts.*

## 10.59 KDChartLayoutItems.cpp File Reference

```
#include "KDChartLayoutItems.h"
#include "KDTextDocument.h"
#include "KDChartAbstractArea.h"
#include "KDChartAbstractDiagram.h"
#include "KDChartBackgroundAttributes.h"
#include "KDChartFrameAttributes.h"
#include "KDChartPaintContext.h"
#include "KDChartPainterSaver_p.h"
#include <QTextCursor>
#include <QTextBlockFormat>
#include <QTextDocumentFragment>
#include <QAbstractTextDocumentLayout>
#include <QLayout>
#include <QPainter>
#include <QDebug>
#include <QCoreApplication>
#include <QApplication>
#include <QStringList>
#include <QStyle>
#include <KDABLibFakes>
#include <math.h>
```

Include dependency graph for KDChartLayoutItems.cpp:



## Defines

- #define [PI](#) 3.141592653589793

## Functions

- static QPointF [rotatedPoint](#) (const QPointF &pt, qreal rotation)
- static QRectF [rotatedRect](#) (const QRectF &rect, qreal angle)
- static void [updateCommonBrush](#) (QBrush &commonBrush, bool &bStart, const [KDChart::AbstractArea](#) &area)

## 10.59.1 Define Documentation

### 10.59.1.1 #define PI 3.141592653589793

Definition at line 50 of file KDChartLayoutItems.cpp.

Referenced by KDChart::TextLayoutItem::intersects(), and rotatedPoint().

## 10.59.2 Function Documentation

### 10.59.2.1 static QPointF rotatedPoint (const QPointF & *pt*, qreal *rotation*) [static]

Definition at line 361 of file KDChartLayoutItems.cpp.

References PI.

Referenced by rotatedRect().

```

362 {
363     const qreal angle = PI * rotation / 180.0;
364     const qreal cosAngle = cos( angle );
365     const qreal sinAngle = sin( angle );
366     return QPointF(
367         (cosAngle * pt.x() + sinAngle * pt.y() ),
368         (cosAngle * pt.y() + sinAngle * pt.x() ) );
369 }
```

### 10.59.2.2 static QRectF rotatedRect (const QRectF & *rect*, qreal *angle*) [static]

Definition at line 371 of file KDChartLayoutItems.cpp.

References rotatedPoint().

Referenced by KDChart::TextLayoutItem::paint().

```

372 {
373     const QPointF topLeft( rotatedPoint( rect.topLeft(), angle ) );
374     //const QPointF topRight( rotatedPoint( rect.topRight(), angle ) );
375     //const QPointF bottomLeft( rotatedPoint( rect.bottomLeft(), angle ) );
376     //const QPointF bottomRight( rotatedPoint( rect.bottomRight(), angle ) );
377     const QPointF siz( rotatedPoint( QPointF( rect.size().width(), rect.size().height() ), angle ) );
378     const QRectF result(
379         topLeft,
380         QSizeF( siz.x(), //bottomRight.x() - topLeft.x(),
381                 siz.y() ) ); //bottomRight.y() - topLeft.y() );
382     //qDebug() << "angle" << angle << "\nbefore:" << rect << "\n after:" << result;
383     return result;
384 }
```

### 10.59.2.3 static void updateCommonBrush (QBrush & *commonBrush*, bool & *bStart*, const KDChart::AbstractArea & *area*) [static]

Definition at line 802 of file KDChartLayoutItems.cpp.

References KDChart::AbstractAreaBase::backgroundAttributes(), KDChart::BackgroundAttributes::BackgroundPixmapModeNone, KDChart::AbstractAreaBase::frameAttributes(), and KDChart::FrameAttributes::isVisible().

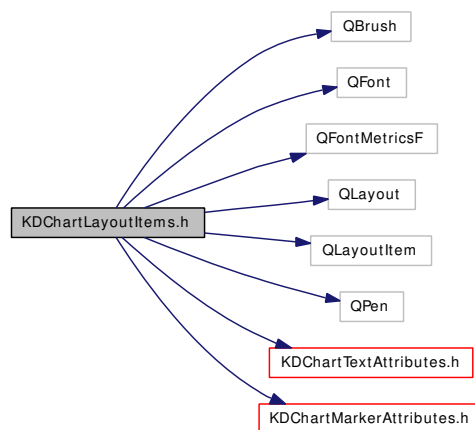
Referenced by KDChart::AutoSpacerLayoutItem::sizeHint().

```
803 {
804     const KDChart::BackgroundAttributes ba( area.backgroundAttributes() );
805     const bool hasSimpleBrush = (
806         ! area.frameAttributes().isVisible() &&
807         ba.isVisible() &&
808         ba.pixmapMode() == KDChart::BackgroundAttributes::BackgroundPixmapModeNone &&
809         ba.brush().gradient() == 0 );
810     if( bStart ){
811         bStart = false;
812         commonBrush = hasSimpleBrush ? ba.brush() : QBrush();
813     }else{
814         if( ! hasSimpleBrush || ba.brush() != commonBrush )
815             {
816                 commonBrush = QBrush();
817             }
818     }
819 }
```

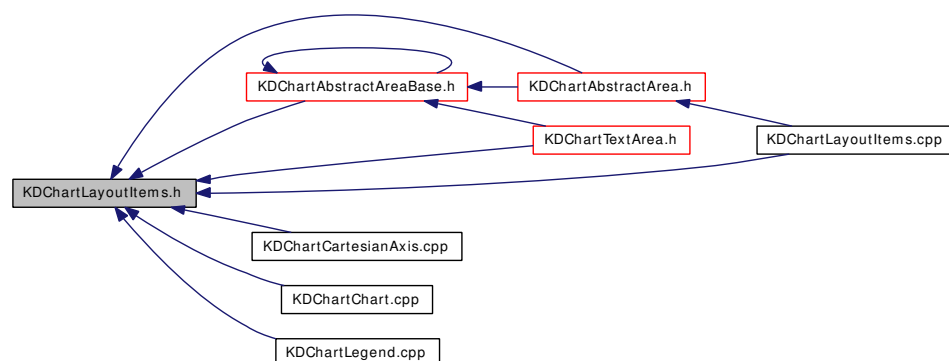
## 10.60 KDChartLayoutItems.h File Reference

```
#include <QBrush>
#include <QFont>
#include <QFontMetricsF>
#include <QLayout>
#include <QLayoutItem>
#include <QPen>
#include "KDChartTextAttributes.h"
#include "KDChartMarkerAttributes.h"
```

Include dependency graph for KDChartLayoutItems.h:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace [KDChart](#)



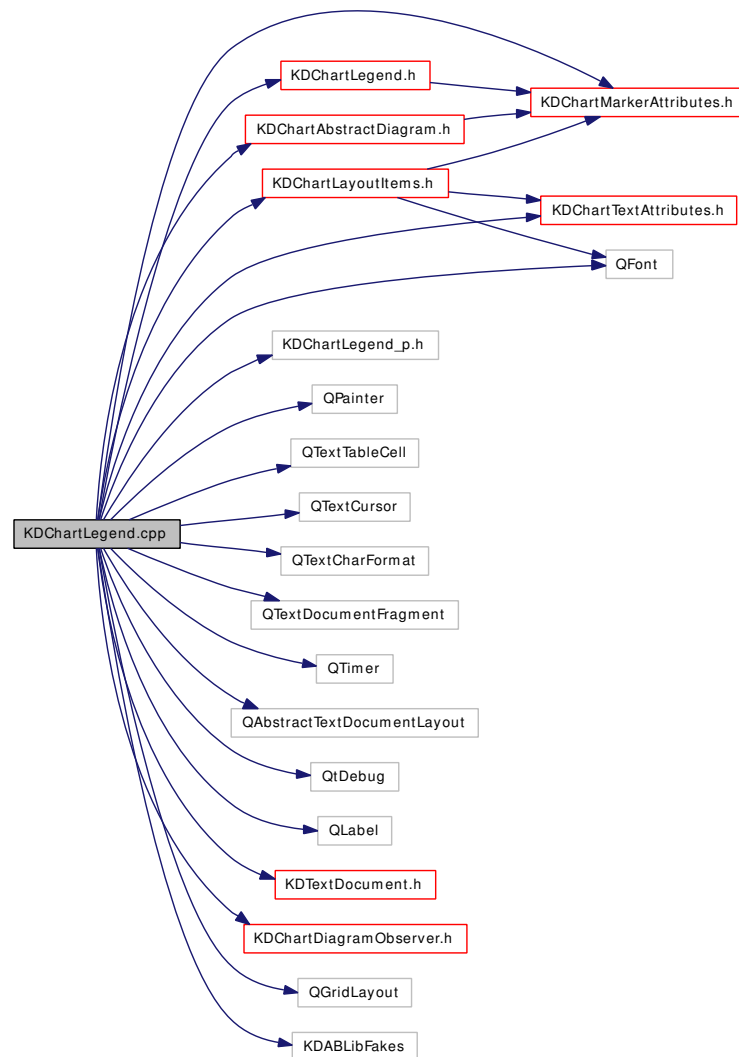
## Classes

- class [KDChart::AbstractLayoutItem](#)  
*Base class for all layout items of KD Chart.*
- class [KDChart::AutoSpacerLayoutItem](#)  
*An empty layout item.*
- class [KDChart::HorizontalLineLayoutItem](#)  
*Layout item showing a horizontal line.*
- class [KDChart::LineLayoutItem](#)  
*Layout item showing a coloured line.*
- class [KDChart::LineWithMarkerLayoutItem](#)  
*Layout item showing a coloured line and a data point marker.*
- class [KDChart::MarkerLayoutItem](#)  
*Layout item showing a data point marker.*
- class [KDChart::TextLayoutItem](#)  
*Layout item showing a text.*
- class [KDChart::VerticalLineLayoutItem](#)  
*Layout item showing a vertical line.*

## 10.61 KDChartLegend.cpp File Reference

```
#include "KDChartLegend.h"
#include "KDChartLegend_p.h"
#include <KDChartTextAttributes.h>
#include <KDChartMarkerAttributes.h>
#include <QFont>
#include <QPainter>
#include <QTextTableCell>
#include <QTextCursor>
#include <QTextCharFormat>
#include <QTextDocumentFragment>
#include <QTimer>
#include <QAbstractTextDocumentLayout>
#include <QtDebug>
#include <QLabel>
#include <KDChartAbstractDiagram.h>
#include "KDTextDocument.h"
#include <KDChartDiagramObserver.h>
#include <QGridLayout>
#include "KDChartLayoutItems.h"
#include <KDABLibFakes>
```

Include dependency graph for KDChartLegend.cpp:



## Defines

- `#define d d_func()`

### 10.61.1 Define Documentation

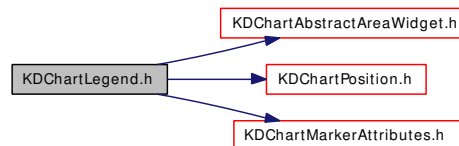
#### 10.61.1.1 `#define d d_func()`

Definition at line 82 of file KDChartLegend.cpp.

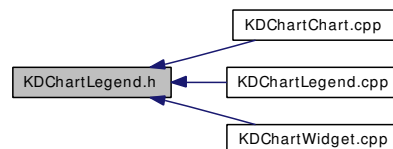
## 10.62 KDChartLegend.h File Reference

```
#include "KDChartAbstractAreaWidget.h"
#include "KDChartPosition.h"
#include "KDChartMarkerAttributes.h"
```

Include dependency graph for KDChartLegend.h:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [KDChart](#)

### Classes

- class [KDChart::Legend](#)  
*[Legend](#) defines the interface for the legend drawing class.*

### Typedefs

- typedef `QList< const AbstractDiagram * >` [KDChart::ConstDiagramList](#)
- typedef `QList< AbstractDiagram * >` [KDChart::DiagramList](#)

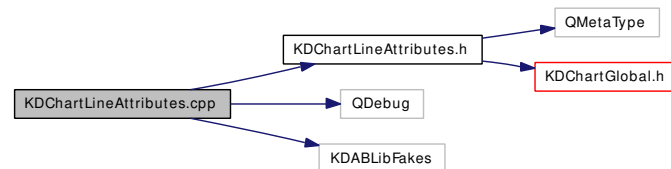
## 10.63 KDChartLineAttributes.cpp File Reference

```
#include "KDChartLineAttributes.h"
```

```
#include <QDebug>
```

```
#include <KDABLibFakes>
```

Include dependency graph for KDChartLineAttributes.cpp:



### Defines

- #define `d_func()`

### Functions

- QDebug `operator<<` (QDebug *dbg*, const `KDChart::LineAttributes` &*a*)

#### 10.63.1 Define Documentation

##### 10.63.1.1 #define `d_func()`

Definition at line 31 of file KDChartLineAttributes.cpp.

#### 10.63.2 Function Documentation

##### 10.63.2.1 QDebug `operator<<` (QDebug *dbg*, const `KDChart::LineAttributes` &*a*)

Definition at line 123 of file KDChartLineAttributes.cpp.

```

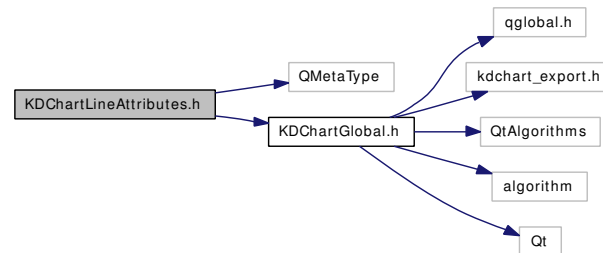
124 {
125     dbg << "KDChart::LineAttributes("
126         //      MissingValuesPolicy missingValuesPolicy;
127         << "bool=" << a.displayArea()
128         << "transparency=" << a.transparency()
129         << ") ";
130     return dbg;
131 }
132 }
```

## 10.64 KDChartLineAttributes.h File Reference

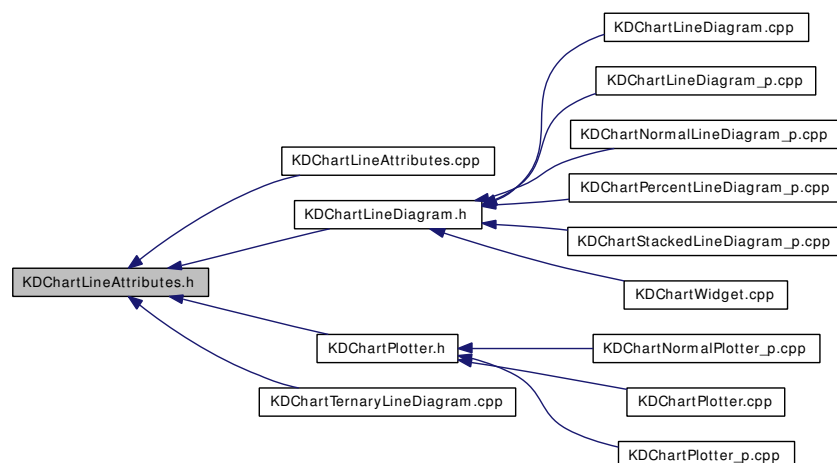
```
#include <QMetaType>
```

```
#include "KDChartGlobal.h"
```

Include dependency graph for KDChartLineAttributes.h:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [KDChart](#)

### Classes

- class [KDChart::LineAttributes](#)

*Set of attributes for changing the appearance of line charts.*

### Functions

- `KDCHART_EXPORT QDebug operator<< (QDebug, const KDChart::LineAttributes &)`
- `Q_DECLARE_TYPEINFO (KDChart::LineAttributes, Q_MOVABLE_TYPE)`

## 10.64.1 Function Documentation

### 10.64.1.1 KDCHART\_EXPORT QDebug operator<< (QDebug, const [KDChart::LineAttributes](#) &)

Definition at line 123 of file KDChartLineAttributes.cpp.

References [KDChart::LineAttributes::displayArea\(\)](#), and [KDChart::LineAttributes::transparency\(\)](#).

```
124 {  
125     dbg << "KDChart::LineAttributes ("  
126         //      MissingValuesPolicy missingValuesPolicy;  
127         << "bool="<<a.displayArea()  
128         << "transparency="<<a.transparency()  
129         << ") ";  
130     return dbg;  
131  
132 }
```

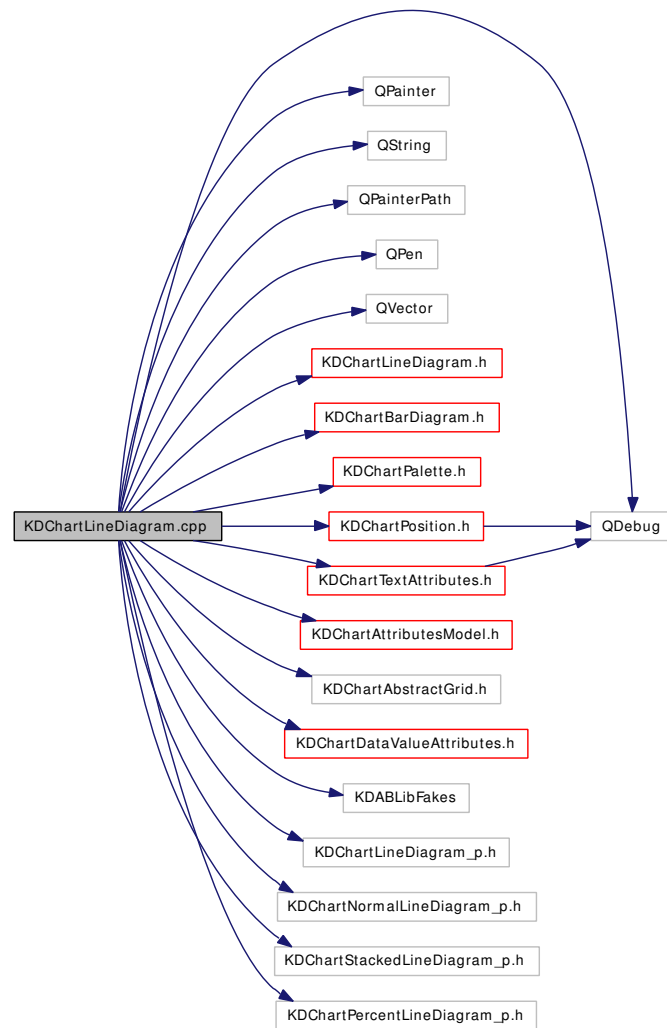
### 10.64.1.2 Q\_DECLARE\_TYPEINFO ([KDChart::LineAttributes](#), Q\_MOVABLE\_TYPE)

## 10.65 KDChartLineDiagram.cpp File Reference

```
#include <QDebug>
#include <QPainter>
#include <QString>
#include <QPainterPath>
#include <QPen>
#include <QVector>
#include "KDChartLineDiagram.h"
#include "KDChartBarDiagram.h"
#include "KDChartPalette.h"
#include "KDChartPosition.h"
#include "KDChartTextAttributes.h"
#include "KDChartAttributesModel.h"
#include "KDChartAbstractGrid.h"
#include "KDChartDataValueAttributes.h"
#include <KDABLibFakes>
#include "KDChartLineDiagram_p.h"
#include "KDChartNormalLineDiagram_p.h"
#include "KDChartStackedLineDiagram_p.h"
#include "KDChartPercentLineDiagram_p.h"
```

Include dependency graph for KDChartLineDiagram.cpp:





## Defines

- `#define d_func()`

### 10.65.1 Define Documentation

#### 10.65.1.1 `#define d_func()`

Definition at line 58 of file KDChartLineDiagram.cpp.

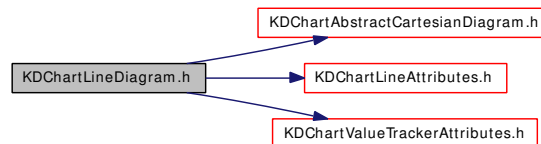
## 10.66 KDChartLineDiagram.h File Reference

```
#include "KDChartAbstractCartesianDiagram.h"
```

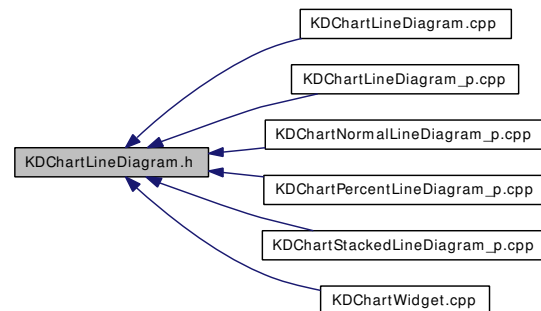
```
#include "KDChartLineAttributes.h"
```

```
#include "KDChartValueTrackerAttributes.h"
```

Include dependency graph for KDChartLineDiagram.h:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [KDChart](#)

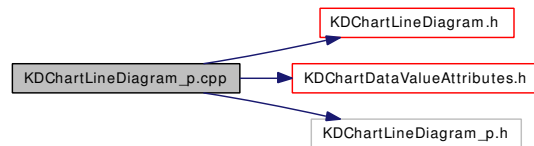
### Classes

- class [KDChart::LineDiagram](#)  
*LineDiagram* defines a common line diagram.

## 10.67 KDChartLineDiagram\_p.cpp File Reference

```
#include "KDChartLineDiagram.h"  
#include "KDChartDataValueAttributes.h"  
#include "KDChartLineDiagram_p.h"
```

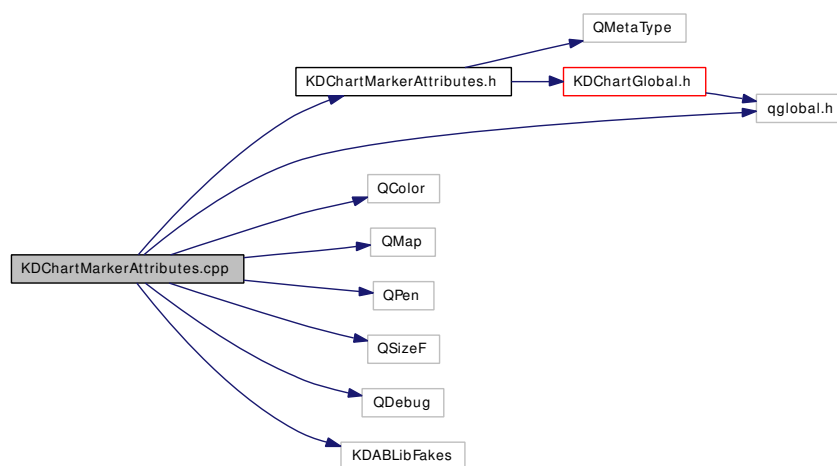
Include dependency graph for KDChartLineDiagram\_p.cpp:



## 10.68 KDChartMarkerAttributes.cpp File Reference

```
#include "KDChartMarkerAttributes.h"
#include <QColor>
#include <QMap>
#include <QPen>
#include <QSizeF>
#include <QDebug>
#include <qglobal.h>
#include <KDABLibFakes>
```

Include dependency graph for KDChartMarkerAttributes.cpp:



### Defines

- #define [d\\_func\(\)](#)

### Functions

- QDebug [operator<<](#) (QDebug dbg, const [MarkerAttributes](#) &ma)

#### 10.68.1 Define Documentation

##### 10.68.1.1 #define [d\\_func\(\)](#)

Definition at line 85 of file KDChartMarkerAttributes.cpp.

## 10.68.2 Function Documentation

### 10.68.2.1 QDebug operator<< (QDebug *dbg*, const [MarkerAttributes](#) & *ma*)

Definition at line 172 of file KDChartMarkerAttributes.cpp.

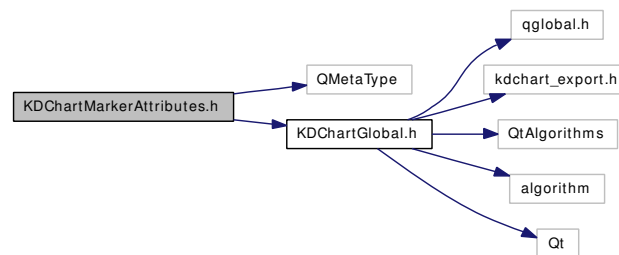
```
172                                     {
173     return dbg << "KDChart::MarkerAttributes("
174                << "visible=" << ma.isVisible()
175                << "markerStylesMap=" << ma.markerStylesMap()
176                << "markerStyle=" << ma.markerStyle()
177                << "markerColor=" << ma.markerColor()
178                << "pen=" << ma.pen()
179                << ") ";
180 }
```

## 10.69 KDChartMarkerAttributes.h File Reference

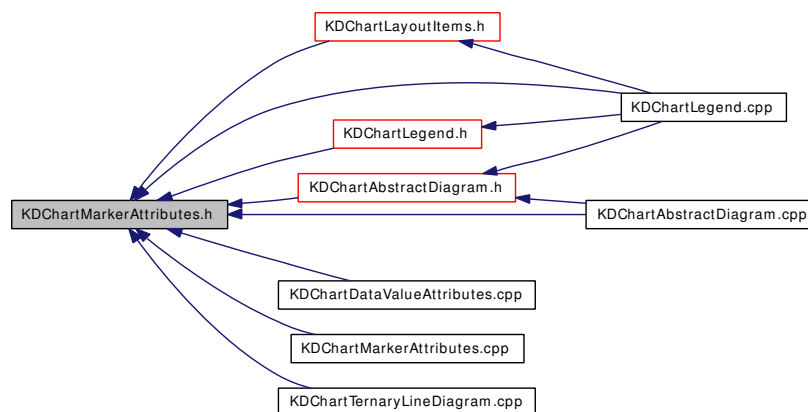
```
#include <QMetaType>
```

```
#include "KDChartGlobal.h"
```

Include dependency graph for KDChartMarkerAttributes.h:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [KDChart](#)

### Classes

- class [KDChart::MarkerAttributes](#)

*A set of ottributes controlling the appearance of data set markers.*

### Functions

- KDCHART\_EXPORT QDebug [operator<<](#) (QDebug, const [KDChart::MarkerAttributes](#) &)
- [Q\\_DECLARE\\_TYPEINFO](#) ([KDChart::MarkerAttributes](#), Q\_MOVABLE\_TYPE)

## 10.69.1 Function Documentation

### 10.69.1.1 KDCHART\_EXPORT QDebug operator<< (QDebug, const KDChart::MarkerAttributes &)

Definition at line 172 of file KDChartMarkerAttributes.cpp.

References KDChart::MarkerAttributes::isVisible(), KDChart::MarkerAttributes::markerColor(), KDChart::MarkerAttributes::markerStyle(), KDChart::MarkerAttributes::markerStylesMap(), and KDChart::MarkerAttributes::pen().

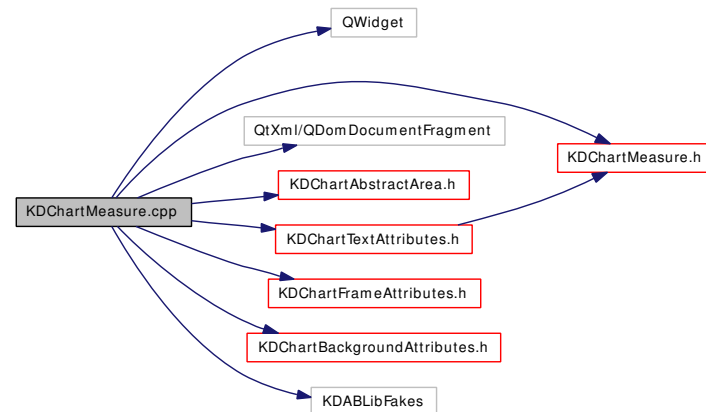
```
172                                     {
173     return dbg << "KDChart::MarkerAttributes("
174                << "visible=" << ma.isVisible()
175                << "markerStylesMap=" << ma.markerStylesMap()
176                << "markerStyle=" << ma.markerStyle()
177                << "markerColor=" << ma.markerColor()
178                << "pen=" << ma.pen()
179                << ") ";
180 }
```

### 10.69.1.2 Q\_DECLARE\_TYPEINFO (KDChart::MarkerAttributes, Q\_MOVABLE\_TYPE)

## 10.70 KDChartMeasure.cpp File Reference

```
#include <QWidget>
#include "KDChartMeasure.h"
#include <QtXml/QDomDocumentFragment>
#include <KDChartAbstractArea.h>
#include <KDChartTextAttributes.h>
#include <KDChartFrameAttributes.h>
#include <KDChartBackgroundAttributes.h>
#include <KDABLibFakes>
```

Include dependency graph for KDChartMeasure.cpp:



### Namespaces

- namespace [KDChart](#)

### Functions

- QDebug [operator<<](#) (QDebug dbg, const [KDChart::Measure](#) &m)

#### 10.70.1 Function Documentation

##### 10.70.1.1 QDebug operator<< (QDebug *dbg*, const [KDChart::Measure](#) &*m*)

Definition at line 233 of file KDChartMeasure.cpp.

```
234 {
235     dbg << "KDChart::Measure("
236         << "value=" << m.value()
237         << "calculationmode=" << m.calculationMode()
238         << "referencearea=" << m.referenceArea()
239         << "referenceorientation=" << m.referenceOrientation()
240         << ") ";
```



```
241     return dbg;
242 }
```

## 10.71 KDChartMeasure.h File Reference

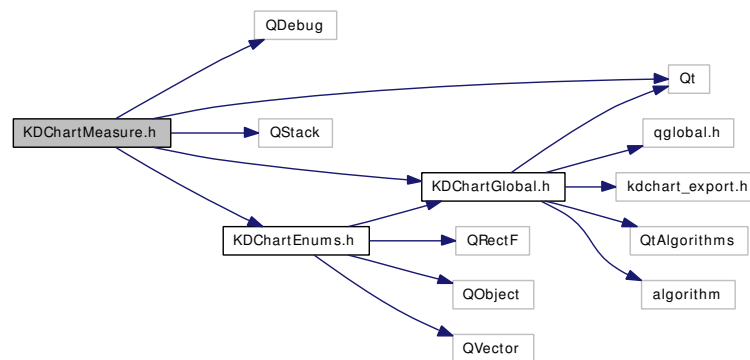
### 10.71.1 Detailed Description

Declaring the class [KDChart::Measure](#).

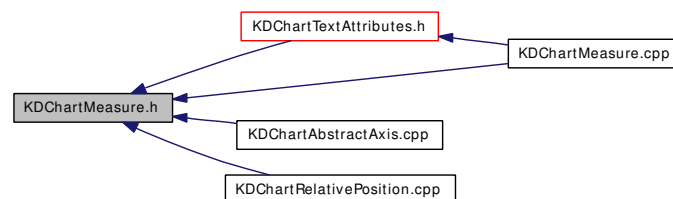
Definition in file [KDChartMeasure.h](#).

```
#include <QDebug>
#include <Qt>
#include <QStack>
#include "KDChartGlobal.h"
#include "KDChartEnums.h"
```

Include dependency graph for KDChartMeasure.h:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [KDChart](#)

### Classes

- class [KDChart::GlobalMeasureScaling](#)  
*Auxiliary class used by the [KDChart::Measure](#) and [KDChart::Chart](#) class.*
- class [KDChart::Measure](#)  
*Measure is used to specify all relative and/or absolute measures in [KDChart](#), e.g.*

## Functions

- KDCHART\_EXPORT QDebug [operator<<](#) (QDebug, const [KDChart::Measure](#) &)

### 10.71.2 Function Documentation

#### 10.71.2.1 KDCHART\_EXPORT QDebug [operator<<](#) (QDebug, const [KDChart::Measure](#) &)

Definition at line 233 of file KDChartMeasure.cpp.

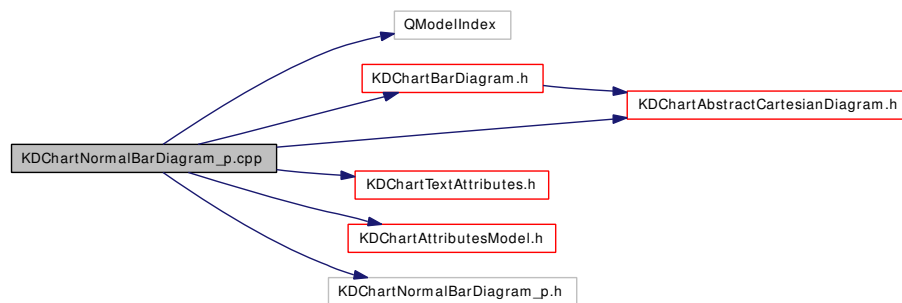
References [KDChart::Measure::calculationMode\(\)](#), [KDChart::Measure::referenceArea\(\)](#), [KDChart::Measure::referenceOrientation\(\)](#), and [KDChart::Measure::value\(\)](#).

```
234 {  
235     dbg << "KDChart::Measure ("  
236         << "value=" << m.value()  
237         << "calculationmode=" << m.calculationMode()  
238         << "referencearea=" << m.referenceArea()  
239         << "referenceorientation=" << m.referenceOrientation()  
240         << ")";  
241     return dbg;  
242 }
```

## 10.72 KDChartNormalBarDiagram\_p.cpp File Reference

```
#include <QModelIndex>
#include "KDChartBarDiagram.h"
#include "KDChartTextAttributes.h"
#include "KDChartAttributesModel.h"
#include "KDChartAbstractCartesianDiagram.h"
#include "KDChartNormalBarDiagram_p.h"
```

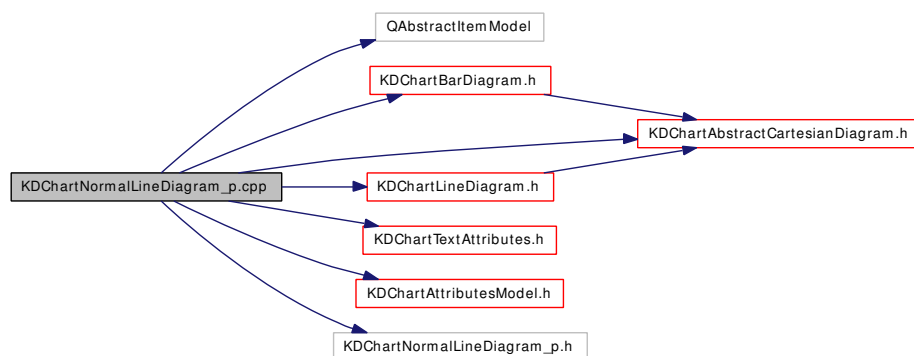
Include dependency graph for KDChartNormalBarDiagram\_p.cpp:



## 10.73 KDChartNormalLineDiagram\_p.cpp File Reference

```
#include <QAbstractItemModel>
#include "KDChartBarDiagram.h"
#include "KDChartLineDiagram.h"
#include "KDChartTextAttributes.h"
#include "KDChartAttributesModel.h"
#include "KDChartAbstractCartesianDiagram.h"
#include "KDChartNormalLineDiagram_p.h"
```

Include dependency graph for KDChartNormalLineDiagram\_p.cpp:

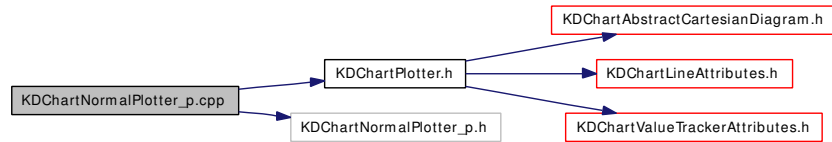


## 10.74 KDChartNormalPlotter\_p.cpp File Reference

```
#include "KDChartPlotter.h"
```

```
#include "KDChartNormalPlotter_p.h"
```

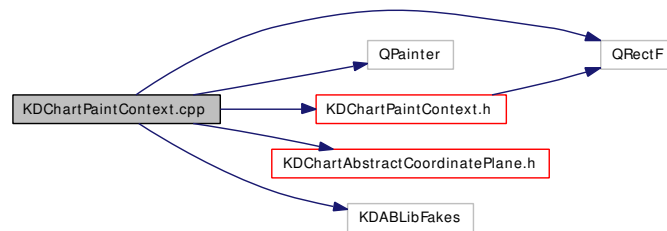
Include dependency graph for KDChartNormalPlotter\_p.cpp:



## 10.75 KDChartPaintContext.cpp File Reference

```
#include <QRectF>
#include <QPainter>
#include "KDChartPaintContext.h"
#include "KDChartAbstractCoordinatePlane.h"
#include <KDABLibFakes>
```

Include dependency graph for KDChartPaintContext.cpp:



### Defines

- #define `d` (`d_func()`)

#### 10.75.1 Define Documentation

##### 10.75.1.1 #define `d` (`d_func()`)

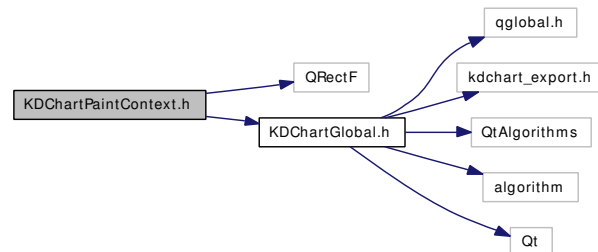
Definition at line 36 of file KDChartPaintContext.cpp.

## 10.76 KDChartPaintContext.h File Reference

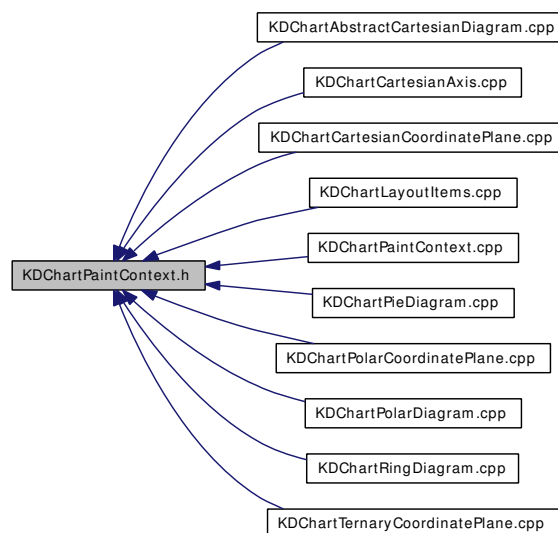
```
#include <QRectF>
```

```
#include "KDChartGlobal.h"
```

Include dependency graph for KDChartPaintContext.h:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [KDChart](#)

### Classes

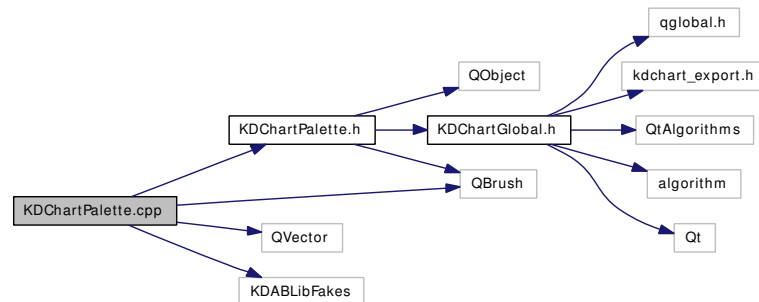
- class [KDChart::PaintContext](#)  
*Stores information about painting diagrams.*



## 10.77 KDChartPalette.cpp File Reference

```
#include "KDChartPalette.h"
#include <QBrush>
#include <QVector>
#include <KDABLibFakes>
```

Include dependency graph for KDChartPalette.cpp:



### Defines

- #define [d\\_d\\_func\(\)](#)

### Functions

- static [Palette makeDefaultPalette \(\)](#)
- static [Palette makeRainbowPalette \(\)](#)
- static [Palette makeSubduedPalette \(\)](#)

#### 10.77.1 Define Documentation

##### 10.77.1.1 #define [d\\_d\\_func\(\)](#)

Definition at line 103 of file KDChartPalette.cpp.

#### 10.77.2 Function Documentation

##### 10.77.2.1 static [Palette @72::makeDefaultPalette \(\)](#) [static]

Definition at line 40 of file KDChartPalette.cpp.

References [KDChart::Palette::addBrush\(\)](#).

Referenced by [KDChart::Palette::defaultPalette\(\)](#).

```

40                                     {
41     Palette p;
42
43     p.addBrush( Qt::red );
```

```

44         p.addBrush( Qt::green );
45         p.addBrush( Qt::blue );
46         p.addBrush( Qt::cyan );
47         p.addBrush( Qt::magenta );
48         p.addBrush( Qt::yellow );
49         p.addBrush( Qt::darkRed );
50         p.addBrush( Qt::darkGreen );
51         p.addBrush( Qt::darkBlue );
52         p.addBrush( Qt::darkCyan );
53         p.addBrush( Qt::darkMagenta );
54         p.addBrush( Qt::darkYellow );
55
56         return p;
57     }

```

### 10.77.2.2 static **Palette** @72::makeRainbowPalette() [static]

Definition at line 84 of file KDChartPalette.cpp.

References KDChart::Palette::addBrush(), and KDChart::Palette::getBrush().

Referenced by KDChart::Palette::rainbowPalette().

```

84                                     {
85         Palette p;
86
87         p.addBrush( QColor(255, 0,196) );
88         p.addBrush( QColor(255, 0, 96) );
89         p.addBrush( QColor(255, 128,64) );
90         p.addBrush( Qt::yellow );
91         p.addBrush( Qt::green );
92         p.addBrush( Qt::cyan );
93         p.addBrush( QColor( 96, 96,255) );
94         p.addBrush( QColor(160, 0,255) );
95         for( int i = 8 ; i < 16 ; ++i )
96             p.addBrush( p.getBrush(i-8).color().light(), i );
97
98         return p;
99     }

```

### 10.77.2.3 static **Palette** @72::makeSubduedPalette() [static]

Definition at line 59 of file KDChartPalette.cpp.

References KDChart::Palette::addBrush().

Referenced by KDChart::Palette::subduedPalette().

```

59                                     {
60         Palette p;
61
62         p.addBrush( QColor( 0xe0,0x7f,0x70 ) );
63         p.addBrush( QColor( 0xe2,0xa5,0x6f ) );
64         p.addBrush( QColor( 0xe0,0xc9,0x70 ) );
65         p.addBrush( QColor( 0xd1,0xe0,0x70 ) );
66         p.addBrush( QColor( 0xac,0xe0,0x70 ) );
67         p.addBrush( QColor( 0x86,0xe0,0x70 ) );
68         p.addBrush( QColor( 0x70,0xe0,0x7f ) );
69         p.addBrush( QColor( 0x70,0xe0,0xa4 ) );
70         p.addBrush( QColor( 0x70,0xe0,0xc9 ) );
71         p.addBrush( QColor( 0x70,0xd1,0xe0 ) );

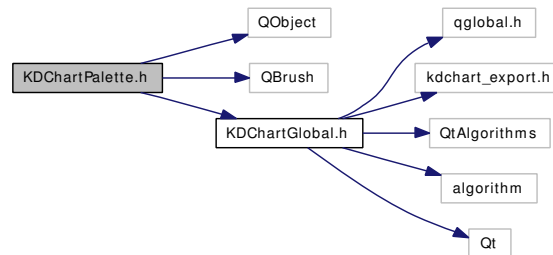
```

```
72         p.addBrush( QColor( 0x70,0xac,0xe0 ) );
73         p.addBrush( QColor( 0x70,0x86,0xe0 ) );
74         p.addBrush( QColor( 0x7f,0x70,0xe0 ) );
75         p.addBrush( QColor( 0xa4,0x70,0xe0 ) );
76         p.addBrush( QColor( 0xc9,0x70,0xe0 ) );
77         p.addBrush( QColor( 0xe0,0x70,0xd1 ) );
78         p.addBrush( QColor( 0xe0,0x70,0xac ) );
79         p.addBrush( QColor( 0xe0,0x70,0x86 ) );
80
81         return p;
82     }
```

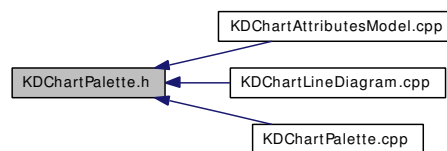
## 10.78 KDChartPalette.h File Reference

```
#include <QObject>
#include <QBrush>
#include "KDChartGlobal.h"
```

Include dependency graph for KDChartPalette.h:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [KDChart](#)

### Classes

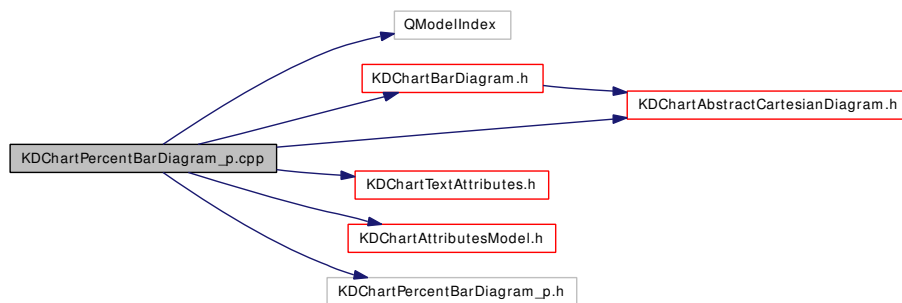
- class [KDChart::Palette](#)

A *Palette* is a set of brushes (or colors) to be used for painting data sets.

## 10.79 KDChartPercentBarDiagram\_p.cpp File Reference

```
#include <QModelIndex>
#include "KDChartBarDiagram.h"
#include "KDChartTextAttributes.h"
#include "KDChartAttributesModel.h"
#include "KDChartAbstractCartesianDiagram.h"
#include "KDChartPercentBarDiagram_p.h"
```

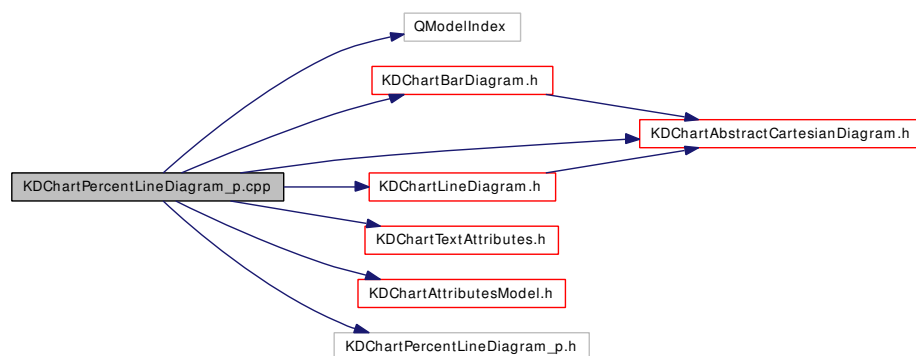
Include dependency graph for KDChartPercentBarDiagram\_p.cpp:



## 10.80 KDChartPercentLineDiagram\_p.cpp File Reference

```
#include <QModelIndex>
#include "KDChartBarDiagram.h"
#include "KDChartLineDiagram.h"
#include "KDChartTextAttributes.h"
#include "KDChartAttributesModel.h"
#include "KDChartAbstractCartesianDiagram.h"
#include "KDChartPercentLineDiagram_p.h"
```

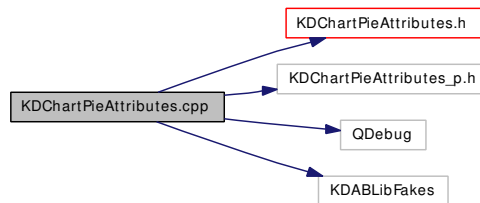
Include dependency graph for KDChartPercentLineDiagram\_p.cpp:



## 10.81 KDChartPieAttributes.cpp File Reference

```
#include "KDChartPieAttributes.h"
#include "KDChartPieAttributes_p.h"
#include <QDebug>
#include <KDABLibFakes>
```

Include dependency graph for KDChartPieAttributes.cpp:



### Defines

- #define [d\\_d\\_func\(\)](#)

### Functions

- QDebug [operator<<](#) (QDebug dbg, const [KDChart::PieAttributes](#) &a)

#### 10.81.1 Define Documentation

##### 10.81.1.1 #define [d\\_d\\_func\(\)](#)

Definition at line 33 of file KDChartPieAttributes.cpp.

#### 10.81.2 Function Documentation

##### 10.81.2.1 QDebug [operator<<](#) (QDebug *dbg*, const [KDChart::PieAttributes](#) &*a*)

Definition at line 106 of file KDChartPieAttributes.cpp.

```
107 {
108     dbg << "KDChart::PieAttributes(";
109     dbg << "explodeFactor=" << a.explodeFactor() << " ";
110     return dbg;
111 }
```

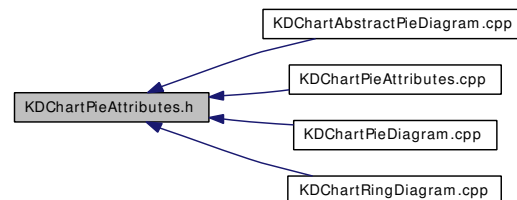
## 10.82 KDChartPieAttributes.h File Reference

```
#include <QMetaType>
#include "KDChartAbstractThreeDAttributes.h"
#include "KDChartGlobal.h"
```

Include dependency graph for KDChartPieAttributes.h:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [KDChart](#)

### Classes

- class [KDChart::PieAttributes](#)

*A set of attributes controlling the appearance of pie charts.*

### Functions

- `KDCHART_EXPORT QDebug operator<< (QDebug, const KDChart::PieAttributes &)`
- `Q_DECLARE_TYPEINFO (KDChart::PieAttributes, Q_MOVABLE_TYPE)`

#### 10.82.1 Function Documentation

##### 10.82.1.1 KDCHART\_EXPORT QDebug operator<< (QDebug, const [KDChart::PieAttributes](#) &)

Definition at line 106 of file KDChartPieAttributes.cpp.

References [KDChart::PieAttributes::explodeFactor\(\)](#).



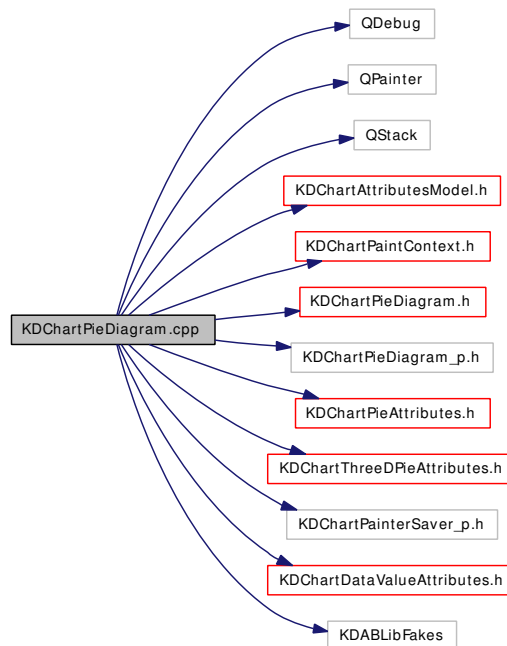
```
107 {  
108     dbg << "KDChart::PieAttributes(";  
109     dbg << "explodeFactor=" << a.explodeFactor() << ")";  
110     return dbg;  
111 }
```

#### 10.82.1.2 Q\_DECLARE\_TYPEINFO (KDChart::PieAttributes, Q\_MOVABLE\_TYPE)

## 10.83 KDChartPieDiagram.cpp File Reference

```
#include <QDebug>
#include <QPainter>
#include <QStack>
#include "KDChartAttributesModel.h"
#include "KDChartPaintContext.h"
#include "KDChartPieDiagram.h"
#include "KDChartPieDiagram_p.h"
#include "KDChartPieAttributes.h"
#include "KDChartThreeDPieAttributes.h"
#include "KDChartPainterSaver_p.h"
#include "KDChartDataValueAttributes.h"
#include <KDABLibFakes>
```

Include dependency graph for KDChartPieDiagram.cpp:



### Defines

- #define `d_func()`

### Functions

- static QRectF `buildReferenceRect` (const `PolarCoordinatePlane` \*plane)

## 10.83.1 Define Documentation

### 10.83.1.1 #define d\_d\_func()

Definition at line 50 of file KDChartPieDiagram.cpp.

## 10.83.2 Function Documentation

### 10.83.2.1 static QRectF buildReferenceRect (const PolarCoordinatePlane \**plane*) [static]

Definition at line 116 of file KDChartPieDiagram.cpp.

References KDChart::PolarCoordinatePlane::translate().

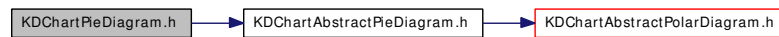
Referenced by KDChart::PieDiagram::paint().

```
117 {
118     QRectF contentsRect;
119     //qDebug() << ".....";
120     QPointF referencePointAtTop = plane->translate( QPointF( 1, 0 ) );
121     QPointF temp = plane->translate( QPointF( 0, 0 ) ) - referencePointAtTop;
122     const double offset = temp.y();
123     referencePointAtTop.setX( referencePointAtTop.x() - offset );
124     contentsRect.setTopLeft( referencePointAtTop );
125     contentsRect.setBottomRight( referencePointAtTop + QPointF( 2*offset, 2*offset ) );
126     //qDebug() << contentsRect;
127     return contentsRect;
128 }
```

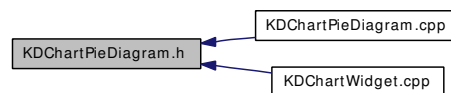
## 10.84 KDChartPieDiagram.h File Reference

```
#include "KDChartAbstractPieDiagram.h"
```

Include dependency graph for KDChartPieDiagram.h:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [KDChart](#)

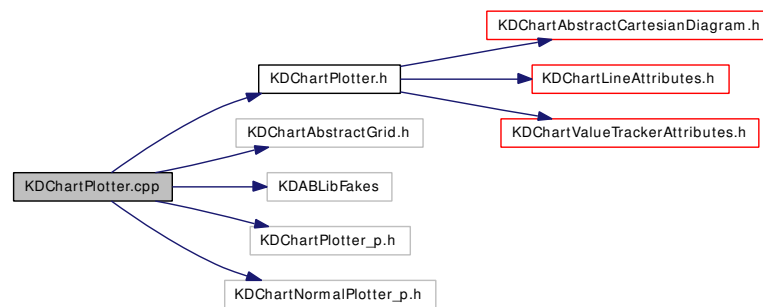
### Classes

- class [KDChart::PieDiagram](#)  
*PieDiagram* defines a common pie diagram.

## 10.85 KDChartPlotter.cpp File Reference

```
#include "KDChartPlotter.h"  
#include "KDChartAbstractGrid.h"  
#include <KDABLibFakes>  
#include "KDChartPlotter_p.h"  
#include "KDChartNormalPlotter_p.h"
```

Include dependency graph for KDChartPlotter.cpp:



### Defines

- #define `d_func()`

#### 10.85.1 Define Documentation

##### 10.85.1.1 #define `d_func()`

Definition at line 44 of file KDChartPlotter.cpp.

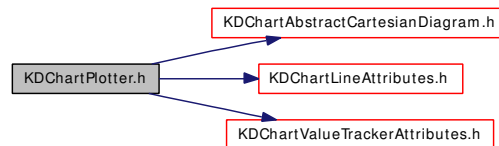
## 10.86 KDChartPlotter.h File Reference

```
#include "KDChartAbstractCartesianDiagram.h"
```

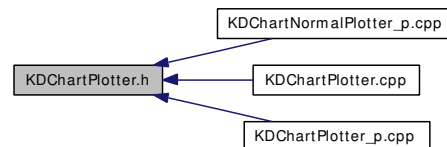
```
#include "KDChartLineAttributes.h"
```

```
#include "KDChartValueTrackerAttributes.h"
```

Include dependency graph for KDChartPlotter.h:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [KDChart](#)

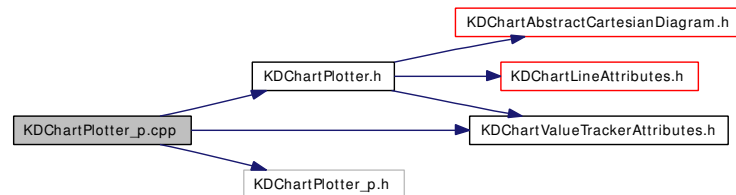
### Classes

- class [KDChart::Plotter](#)  
*Plotter* defines a diagram type plotting two-dimensional data.

## 10.87 KDChartPlotter\_p.cpp File Reference

```
#include "KDChartPlotter.h"  
#include "KDChartPlotter_p.h"  
#include "KDChartValueTrackerAttributes.h"
```

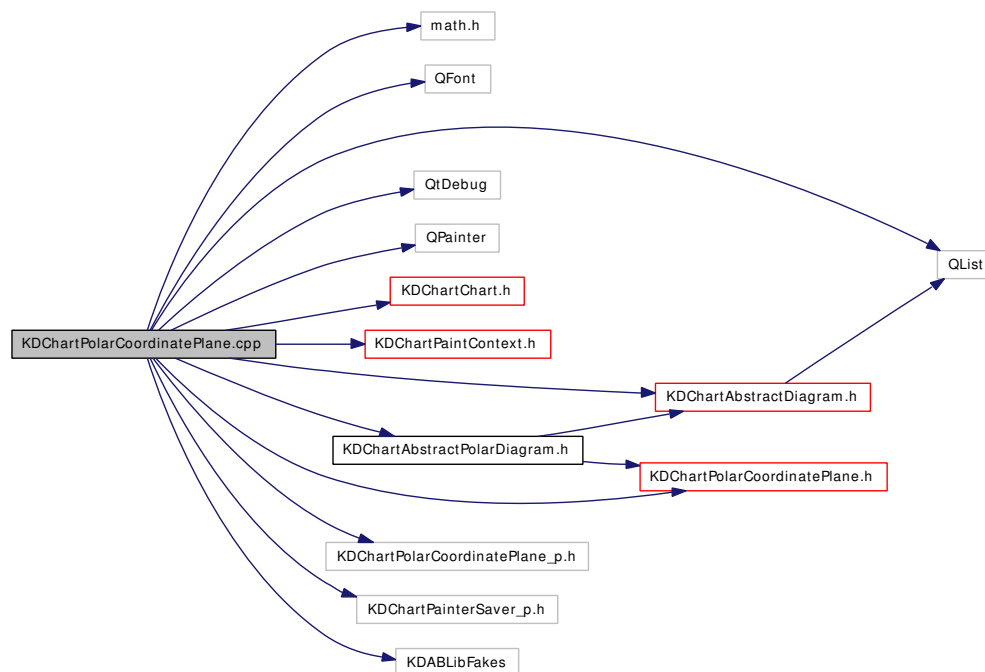
Include dependency graph for KDChartPlotter\_p.cpp:



## 10.88 KDChartPolarCoordinatePlane.cpp File Reference

```
#include <math.h>
#include <QFont>
#include <QList>
#include <QtDebug>
#include <QPainter>
#include "KDChartChart.h"
#include "KDChartPaintContext.h"
#include "KDChartAbstractDiagram.h"
#include "KDChartAbstractPolarDiagram.h"
#include "KDChartPolarCoordinatePlane.h"
#include "KDChartPolarCoordinatePlane_p.h"
#include "KDChartPainterSaver_p.h"
#include <KDABLibFakes>
```

Include dependency graph for KDChartPolarCoordinatePlane.cpp:



## Defines

- #define `d_func()`



## 10.88.1 Define Documentation

### 10.88.1.1 `#define d_d_func()`

Definition at line 45 of file KDChartPolarCoordinatePlane.cpp.

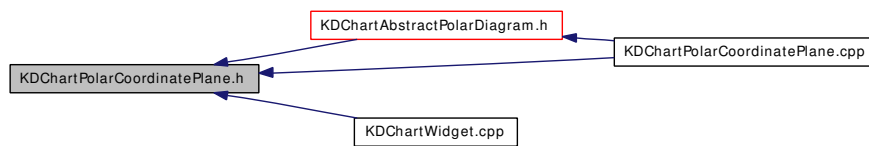
## 10.89 KDChartPolarCoordinatePlane.h File Reference

```
#include "KDChartAbstractCoordinatePlane.h"
```

Include dependency graph for KDChartPolarCoordinatePlane.h:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [KDChart](#)

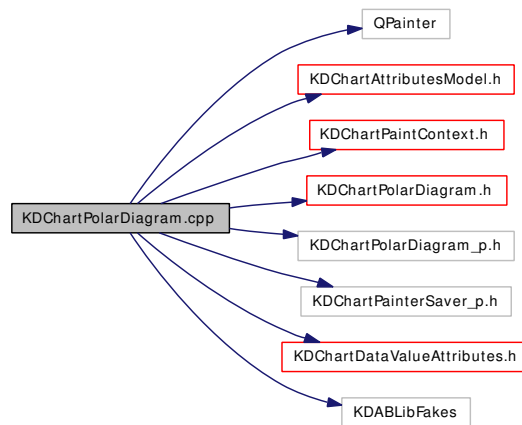
### Classes

- class [KDChart::PolarCoordinatePlane](#)  
*Polar coordinate plane.*

## 10.90 KDChartPolarDiagram.cpp File Reference

```
#include <QPainter>
#include "KDChartAttributesModel.h"
#include "KDChartPaintContext.h"
#include "KDChartPolarDiagram.h"
#include "KDChartPolarDiagram_p.h"
#include "KDChartPainterSaver_p.h"
#include "KDChartDataValueAttributes.h"
#include <KDABLibFakes>
```

Include dependency graph for KDChartPolarDiagram.cpp:



### Defines

- `#define d d_func()`

#### 10.90.1 Define Documentation

##### 10.90.1.1 `#define d d_func()`

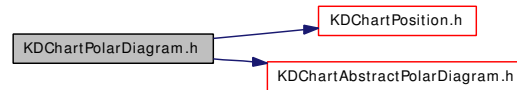
Definition at line 47 of file KDChartPolarDiagram.cpp.

## 10.91 KDChartPolarDiagram.h File Reference

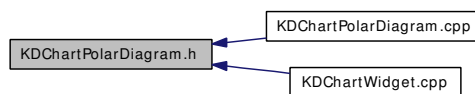
```
#include "KDChartPosition.h"
```

```
#include "KDChartAbstractPolarDiagram.h"
```

Include dependency graph for KDChartPolarDiagram.h:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [KDChart](#)

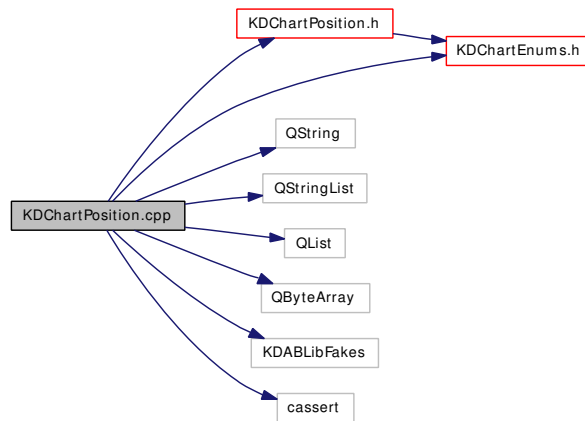
### Classes

- class [KDChart::PolarDiagram](#)  
*PolarDiagram* defines a common polar diagram.

## 10.92 KDChartPosition.cpp File Reference

```
#include <KDChartPosition.h>
#include <KDChartEnums.h>
#include <QString>
#include <QStringList>
#include <QList>
#include <QByteArray>
#include <KDABLibFakes>
#include <cassert>
```

Include dependency graph for KDChartPosition.cpp:



### Functions

- QDebug [operator<<](#) (QDebug dbg, const [KDChart::Position](#) &p)

### Variables

- static int [maxPositionValue](#) = 10
- static [Position](#) [staticPositionCenter](#) = [Position](#)( [KDChartEnums::PositionCenter](#) )
- static [Position](#) [staticPositionEast](#) = [Position](#)( [KDChartEnums::PositionEast](#) )
- static [Position](#) [staticPositionFloating](#) = [Position](#)( [KDChartEnums::PositionFloating](#) )
- static const char \* [staticPositionNames](#) [ ]
- static [Position](#) [staticPositionNorth](#) = [Position](#)( [KDChartEnums::PositionNorth](#) )
- static [Position](#) [staticPositionNorthEast](#) = [Position](#)( [KDChartEnums::PositionNorthEast](#) )
- static [Position](#) [staticPositionNorthWest](#) = [Position](#)( [KDChartEnums::PositionNorthWest](#) )
- static [Position](#) [staticPositionSouth](#) = [Position](#)( [KDChartEnums::PositionSouth](#) )
- static [Position](#) [staticPositionSouthEast](#) = [Position](#)( [KDChartEnums::PositionSouthEast](#) )
- static [Position](#) [staticPositionSouthWest](#) = [Position](#)( [KDChartEnums::PositionSouthWest](#) )
- static [Position](#) [staticPositionUnknown](#) = [Position](#)( [KDChartEnums::PositionUnknown](#) )
- static [Position](#) [staticPositionWest](#) = [Position](#)( [KDChartEnums::PositionWest](#) )

## 10.92.1 Function Documentation

### 10.92.1.1 QDebug operator<< (QDebug *dbg*, const [KDChart::Position](#) & *p*)

Definition at line 260 of file KDChartPosition.cpp.

```
261 {
262     dbg << "KDChart::Position("
263         << p.name() << ")";
264     return dbg;
265 }
```

## 10.92.2 Variable Documentation

### 10.92.2.1 int [maxPositionValue](#) = 10 [static]

Definition at line 80 of file KDChartPosition.cpp.

Referenced by [KDChart::Position::fromName\(\)](#), [KDChart::Position::names\(\)](#), and [KDChart::Position::printableNames\(\)](#).

### 10.92.2.2 [Position](#) [staticPositionCenter](#) = [Position](#)( [KDChartEnums::PositionCenter](#) ) [static]

Definition at line 69 of file KDChartPosition.cpp.

### 10.92.2.3 [Position](#) [staticPositionEast](#) = [Position](#)( [KDChartEnums::PositionEast](#) ) [static]

Definition at line 73 of file KDChartPosition.cpp.

### 10.92.2.4 [Position](#) [staticPositionFloating](#) = [Position](#)( [KDChartEnums::PositionFloating](#) ) [static]

Definition at line 78 of file KDChartPosition.cpp.

### 10.92.2.5 const char\* [staticPositionNames](#)[ ] [static]

**Initial value:**

```
{
    QT_TRANSLATE_NOOP("Position", "Unknown Position"),
    QT_TRANSLATE_NOOP("Position", "Center"),
    QT_TRANSLATE_NOOP("Position", "NorthWest"),
    QT_TRANSLATE_NOOP("Position", "North"),
    QT_TRANSLATE_NOOP("Position", "NorthEast"),
    QT_TRANSLATE_NOOP("Position", "East"),
    QT_TRANSLATE_NOOP("Position", "SouthEast"),
    QT_TRANSLATE_NOOP("Position", "South"),
    QT_TRANSLATE_NOOP("Position", "SouthWest"),
    QT_TRANSLATE_NOOP("Position", "West"),
}
```

Definition at line 49 of file KDChartPosition.cpp.

Referenced by KDChart::Position::fromName(), KDChart::Position::name(), KDChart::Position::names(), and KDChart::Position::printableName().

**10.92.2.6** `Position staticPositionNorth = Position( KDChartEnums::PositionNorth )` `[static]`

Definition at line 71 of file KDChartPosition.cpp.

**10.92.2.7** `Position staticPositionNorthEast = Position( KDChartEnums::PositionNorthEast )`  
`[static]`

Definition at line 72 of file KDChartPosition.cpp.

**10.92.2.8** `Position staticPositionNorthWest = Position( KDChartEnums::PositionNorthWest )`  
`[static]`

Definition at line 70 of file KDChartPosition.cpp.

**10.92.2.9** `Position staticPositionSouth = Position( KDChartEnums::PositionSouth )` `[static]`

Definition at line 75 of file KDChartPosition.cpp.

**10.92.2.10** `Position staticPositionSouthEast = Position( KDChartEnums::PositionSouthEast )`  
`[static]`

Definition at line 74 of file KDChartPosition.cpp.

**10.92.2.11** `Position staticPositionSouthWest = Position( KDChartEnums::PositionSouthWest )`  
`[static]`

Definition at line 76 of file KDChartPosition.cpp.

**10.92.2.12** `Position staticPositionUnknown = Position( KDChartEnums::PositionUnknown )`  
`[static]`

Definition at line 68 of file KDChartPosition.cpp.

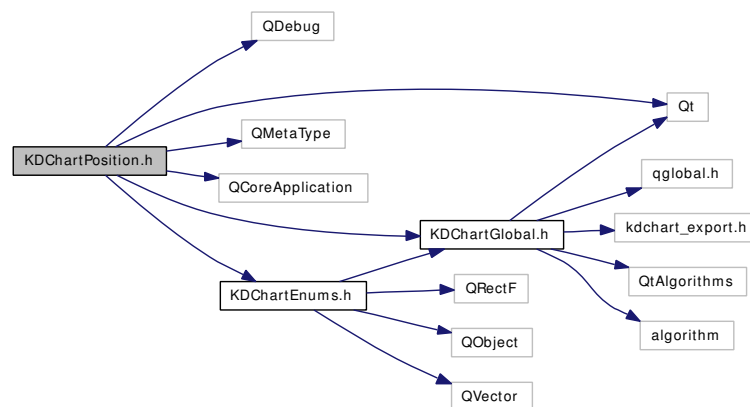
**10.92.2.13** `Position staticPositionWest = Position( KDChartEnums::PositionWest )` `[static]`

Definition at line 77 of file KDChartPosition.cpp.

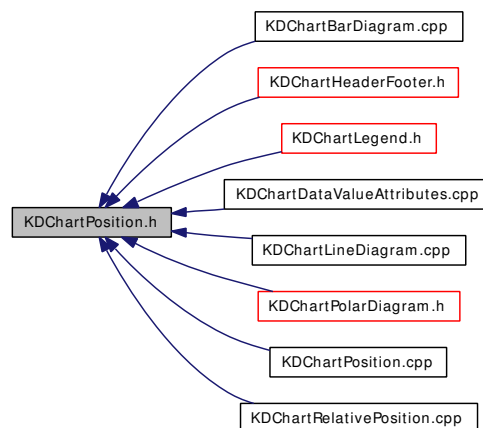
## 10.93 KDChartPosition.h File Reference

```
#include <QDebug>
#include <Qt>
#include <QMetaType>
#include <QCoreApplication>
#include "KDChartGlobal.h"
#include "KDChartEnums.h"
```

Include dependency graph for KDChartPosition.h:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [KDChart](#)

### Classes

- class [KDChart::Position](#)



*Defines a position, using compass terminology.*

- class [KDChart::PositionPoints](#)  
*Stores the absolute target points of a [Position](#).*

## Functions

- KDCHART\_EXPORT QDebug [operator<<](#) (QDebug, const [KDChart::Position](#) &)
- [Q\\_DECLARE\\_TYPEINFO](#) ([KDChart::Position](#), Q\_MOVABLE\_TYPE)

### 10.93.1 Function Documentation

#### 10.93.1.1 KDCHART\_EXPORT QDebug operator<< (QDebug, const [KDChart::Position](#) &)

Definition at line 260 of file KDChartPosition.cpp.

References [KDChart::Position::name\(\)](#).

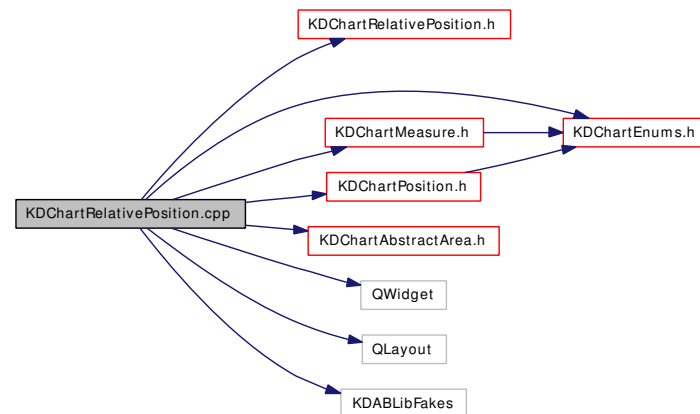
```
261 {  
262     dbg << "KDChart::Position ("  
263         << p.name() << ") "  
264     return dbg;  
265 }
```

#### 10.93.1.2 Q\_DECLARE\_TYPEINFO ([KDChart::Position](#), Q\_MOVABLE\_TYPE)

## 10.94 KDChartRelativePosition.cpp File Reference

```
#include "KDChartRelativePosition.h"
#include "KDChartEnums.h"
#include "KDChartMeasure.h"
#include "KDChartPosition.h"
#include "KDChartAbstractArea.h"
#include <QWidget>
#include <QLayout>
#include <KDABLibFakes>
```

Include dependency graph for KDChartRelativePosition.cpp:



### Defines

- #define `d_func()`

### Functions

- QDebug `operator<<` (QDebug *dbg*, const `KDChart::RelativePosition` &*rp*)

#### 10.94.1 Define Documentation

##### 10.94.1.1 #define `d_func()`

Definition at line 93 of file KDChartRelativePosition.cpp.

#### 10.94.2 Function Documentation

##### 10.94.2.1 QDebug `operator<<` (QDebug *dbg*, const `KDChart::RelativePosition` &*rp*)

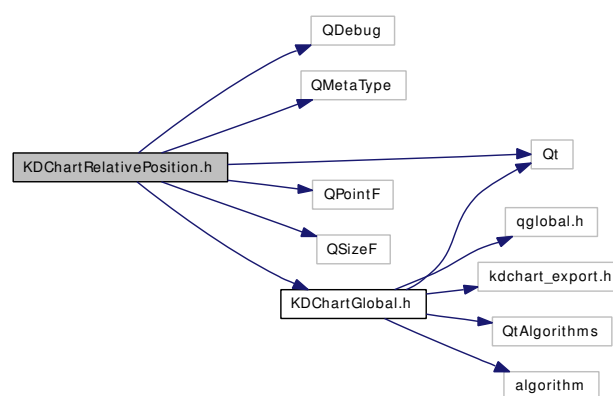
Definition at line 210 of file KDChartRelativePosition.cpp.

```
211 {
212     dbg << "KDChart::RelativePosition("
213         << "referencearea="<<rp.referenceArea()
214         << "referenceposition="<<rp.referencePosition()
215         << "alignment="<<rp.alignment()
216         << "horizontalpadding="<<rp.horizontalPadding()
217         << "verticalpadding="<<rp.verticalPadding()
218         << "rotation="<<rp.rotation()
219         << ")";
220     return dbg;
221 }
```

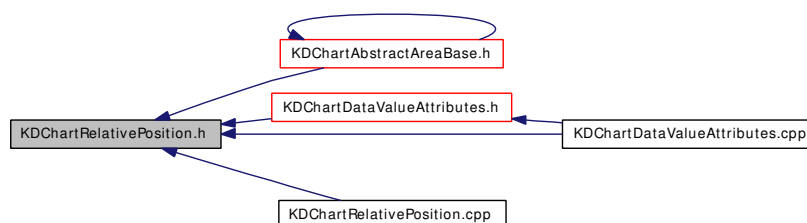
## 10.95 KDChartRelativePosition.h File Reference

```
#include <QDebug>
#include <QMetaType>
#include <Qt>
#include <QPointF>
#include <QSizeF>
#include "KDChartGlobal.h"
```

Include dependency graph for KDChartRelativePosition.h:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace [KDChart](#)

## Classes

- class [KDChart::RelativePosition](#)

*Defines relative position information: reference area, position in this area, horizontal / vertical padding, and rotating.*

## Functions

- KDCHART\_EXPORT QDebug operator<< (QDebug, const [KDChart::RelativePosition](#) &)
- [Q\\_DECLARE\\_TYPEINFO](#) ([KDChart::RelativePosition](#), Q\_MOVABLE\_TYPE)

### 10.95.1 Function Documentation

#### 10.95.1.1 KDCHART\_EXPORT QDebug operator<< (QDebug, const [KDChart::RelativePosition](#) &)

Definition at line 210 of file KDChartRelativePosition.cpp.

References [KDChart::RelativePosition::alignment\(\)](#), [KDChart::RelativePosition::horizontalPadding\(\)](#), [KDChart::RelativePosition::referenceArea\(\)](#), [KDChart::RelativePosition::referencePosition\(\)](#), [KDChart::RelativePosition::rotation\(\)](#), and [KDChart::RelativePosition::verticalPadding\(\)](#).

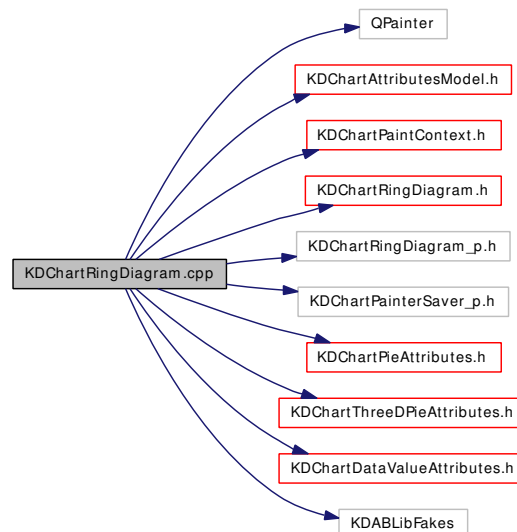
```
211 {
212     dbg << "KDChart::RelativePosition("
213         << "referencearea="<<rp.referenceArea()
214         << "referenceposition="<<rp.referencePosition()
215         << "alignment="<<rp.alignment()
216         << "horizontalpadding="<<rp.horizontalPadding()
217         << "verticalpadding="<<rp.verticalPadding()
218         << "rotation="<<rp.rotation()
219         << ") ";
220     return dbg;
221 }
```

#### 10.95.1.2 [Q\\_DECLARE\\_TYPEINFO](#) ([KDChart::RelativePosition](#), Q\_MOVABLE\_TYPE)

## 10.96 KDChartRingDiagram.cpp File Reference

```
#include <QPainter>
#include "KDChartAttributesModel.h"
#include "KDChartPaintContext.h"
#include "KDChartRingDiagram.h"
#include "KDChartRingDiagram_p.h"
#include "KDChartPainterSaver_p.h"
#include "KDChartPieAttributes.h"
#include "KDChartThreeDPieAttributes.h"
#include "KDChartDataValueAttributes.h"
#include <KDABLibFakes>
```

Include dependency graph for KDChartRingDiagram.cpp:



### Defines

- #define [d\\_func\(\)](#)

### 10.96.1 Define Documentation

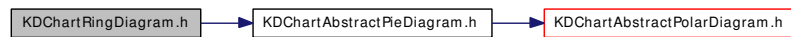
#### 10.96.1.1 #define d\_func()

Definition at line 48 of file KDChartRingDiagram.cpp.

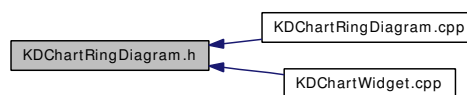
## 10.97 KDChartRingDiagram.h File Reference

```
#include "KDChartAbstractPieDiagram.h"
```

Include dependency graph for KDChartRingDiagram.h:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [KDChart](#)

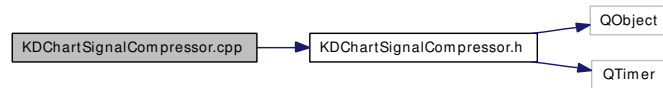
### Classes

- class [KDChart::RingDiagram](#)  
*RingDiagram* defines a common ring diagram.

## 10.98 KDChartSignalCompressor.cpp File Reference

```
#include "KDChartSignalCompressor.h"
```

Include dependency graph for KDChartSignalCompressor.cpp:





## 10.99 KDChartSignalCompressor.h File Reference

```
#include <QObject>
```

```
#include <QTimer>
```

Include dependency graph for KDChartSignalCompressor.h:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [KDChart](#)

### Classes

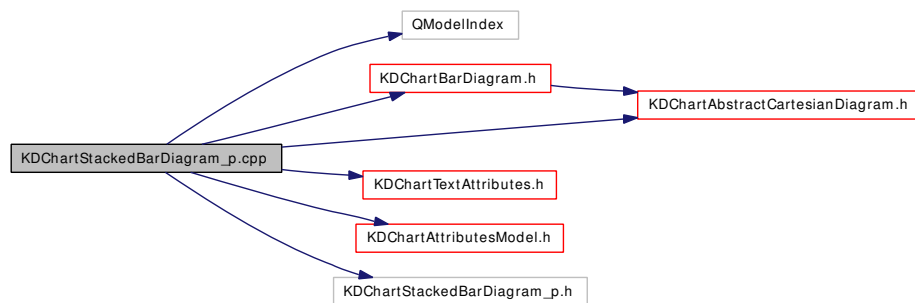
- class [KDChart::SignalCompressor](#)

*[SignalCompressor](#) compresses signals where the same signal needs to be emitted by several pieces of the code, but only one of the signals should be received at the end.*

## 10.100 KDChartStackedBarDiagram\_p.cpp File Reference

```
#include <QModelIndex>
#include "KDChartBarDiagram.h"
#include "KDChartTextAttributes.h"
#include "KDChartAttributesModel.h"
#include "KDChartAbstractCartesianDiagram.h"
#include "KDChartStackedBarDiagram_p.h"
```

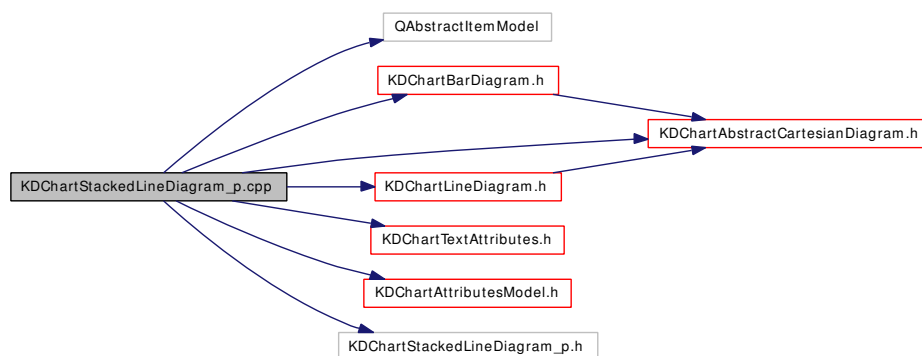
Include dependency graph for KDChartStackedBarDiagram\_p.cpp:



## 10.101 KDChartStackedLineDiagram\_p.cpp File Reference

```
#include <QAbstractItemModel>
#include "KDChartBarDiagram.h"
#include "KDChartLineDiagram.h"
#include "KDChartTextAttributes.h"
#include "KDChartAttributesModel.h"
#include "KDChartAbstractCartesianDiagram.h"
#include "KDChartStackedLineDiagram_p.h"
```

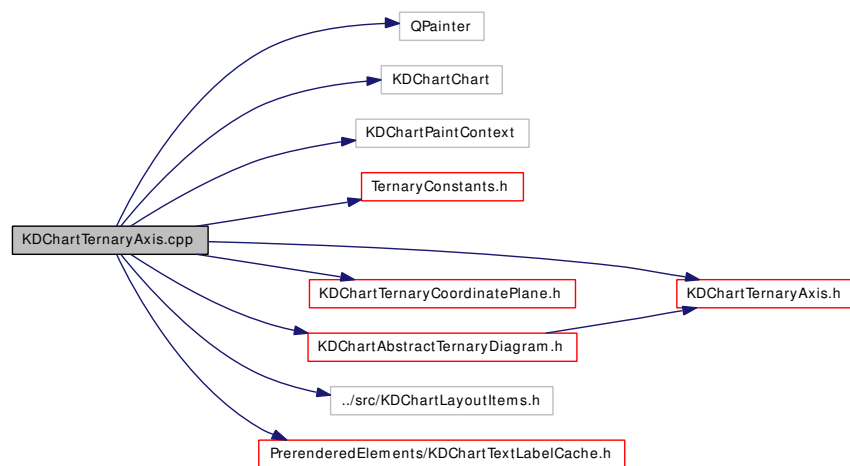
Include dependency graph for KDChartStackedLineDiagram\_p.cpp:



## 10.102 KDChartTernaryAxis.cpp File Reference

```
#include <QPainter>
#include <KDChartChart>
#include <KDChartPaintContext>
#include "TernaryConstants.h"
#include "KDChartTernaryAxis.h"
#include "KDChartTernaryCoordinatePlane.h"
#include "KDChartAbstractTernaryDiagram.h"
#include "../src/KDChartLayoutItems.h"
#include "PrerenderedElements/KDChartTextLabelCache.h"
```

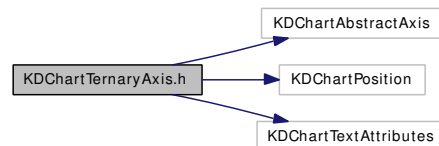
Include dependency graph for KDChartTernaryAxis.cpp:



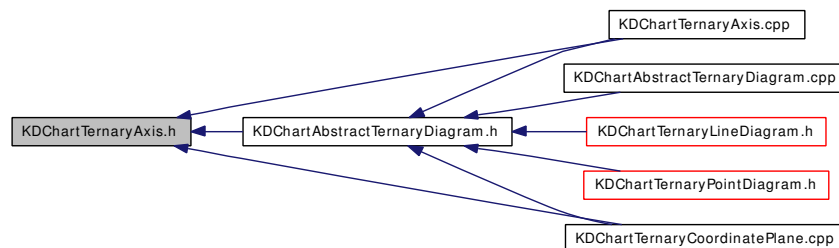
## 10.103 KDChartTernaryAxis.h File Reference

```
#include <KDChartAbstractAxis>
#include <KDChartPosition>
#include <KDChartTextAttributes>
```

Include dependency graph for KDChartTernaryAxis.h:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [KDChart](#)

### Classes

- class [KDChart::TernaryAxis](#)  
The class for ternary axes.

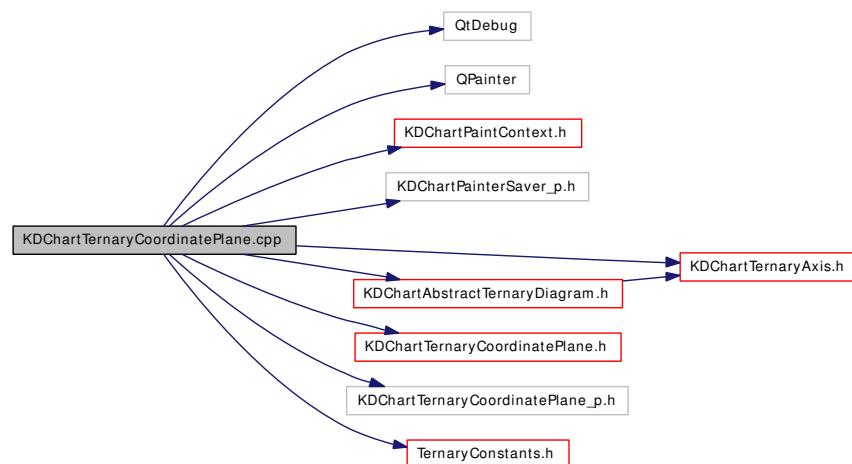
### Typedefs

- typedef `QList< TernaryAxis * >` [KDChart::TernaryAxisList](#)

## 10.104 KDChartTernaryCoordinatePlane.cpp File Reference

```
#include <QtDebug>
#include <QPainter>
#include "KDChartPaintContext.h"
#include "KDChartPainterSaver_p.h"
#include "KDChartTernaryAxis.h"
#include "KDChartAbstractTernaryDiagram.h"
#include "KDChartTernaryCoordinatePlane.h"
#include "KDChartTernaryCoordinatePlane_p.h"
#include "TernaryConstants.h"
```

Include dependency graph for KDChartTernaryCoordinatePlane.cpp:



### Defines

- #define [d\\_func\(\)](#)

#### 10.104.1 Define Documentation

##### 10.104.1.1 #define [d\\_func\(\)](#)

Definition at line 44 of file KDChartTernaryCoordinatePlane.cpp.

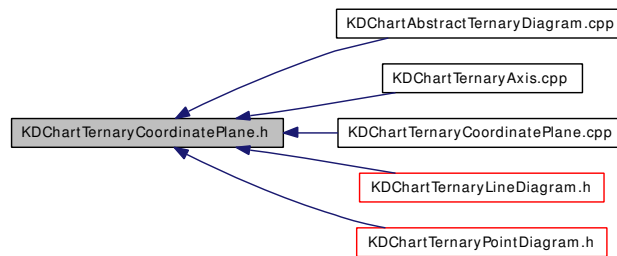
## 10.105 KDChartTernaryCoordinatePlane.h File Reference

```
#include "../KDChartAbstractCoordinatePlane.h"
```

Include dependency graph for KDChartTernaryCoordinatePlane.h:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [KDChart](#)

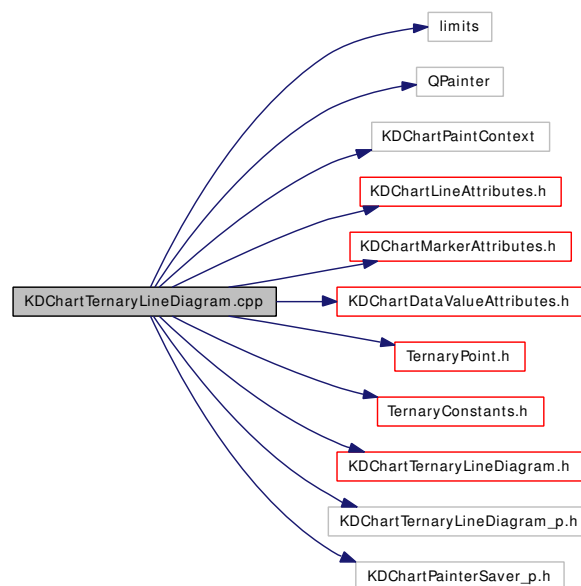
### Classes

- class [KDChart::TernaryCoordinatePlane](#)  
*Ternary coordinate plane.*

## 10.106 KDChartTernaryLineDiagram.cpp File Reference

```
#include <limits>
#include <QPainter>
#include <KDChartPaintContext>
#include "KDChartLineAttributes.h"
#include "KDChartMarkerAttributes.h"
#include "KDChartDataValueAttributes.h"
#include "TernaryPoint.h"
#include "TernaryConstants.h"
#include "KDChartTernaryLineDiagram.h"
#include "KDChartTernaryLineDiagram_p.h"
#include "KDChartPainterSaver_p.h"
```

Include dependency graph for KDChartTernaryLineDiagram.cpp:



### Defines

- #define [d\\_func\(\)](#)

### 10.106.1 Define Documentation

#### 10.106.1.1 #define d\_func()

Definition at line 48 of file KDChartTernaryLineDiagram.cpp.

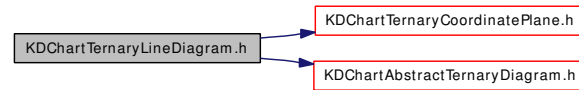


## 10.107 KDChartTernaryLineDiagram.h File Reference

```
#include "KDChartTernaryCoordinatePlane.h"
```

```
#include "KDChartAbstractTernaryDiagram.h"
```

Include dependency graph for KDChartTernaryLineDiagram.h:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [KDChart](#)

### Classes

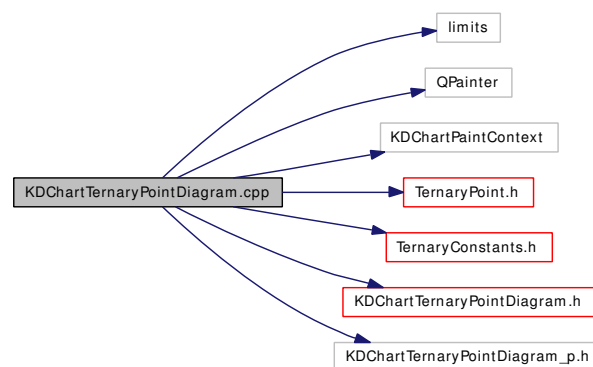
- class [KDChart::TernaryLineDiagram](#)

A [TernaryLineDiagram](#) is a line diagram with a ternary coordinate plane.

## 10.108 KDChartTernaryPointDiagram.cpp File Reference

```
#include <limits>
#include <QPainter>
#include <KDChartPaintContext>
#include "TernaryPoint.h"
#include "TernaryConstants.h"
#include "KDChartTernaryPointDiagram.h"
#include "KDChartTernaryPointDiagram_p.h"
```

Include dependency graph for KDChartTernaryPointDiagram.cpp:



### Defines

- #define `d_func()`

#### 10.108.1 Define Documentation

##### 10.108.1.1 #define `d_func()`

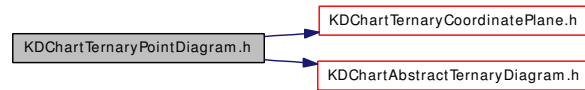
Definition at line 43 of file KDChartTernaryPointDiagram.cpp.

## 10.109 KDChartTernaryPointDiagram.h File Reference

```
#include "KDChartTernaryCoordinatePlane.h"
```

```
#include "KDChartAbstractTernaryDiagram.h"
```

Include dependency graph for KDChartTernaryPointDiagram.h:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [KDChart](#)

### Classes

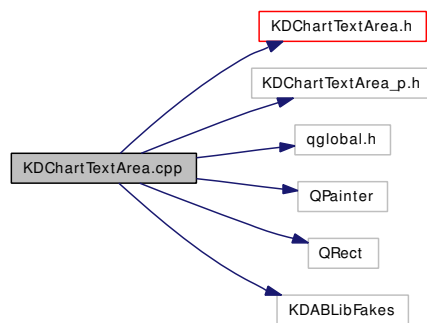
- class [KDChart::TernaryPointDiagram](#)

A *[TernaryPointDiagram](#)* is a point diagram within a ternary coordinate plane.

## 10.110 KDChartTextArea.cpp File Reference

```
#include "KDChartTextArea.h"  
#include "KDChartTextArea_p.h"  
#include <qglobal.h>  
#include <QPainter>  
#include <QRect>  
#include <KDABLibFakes>
```

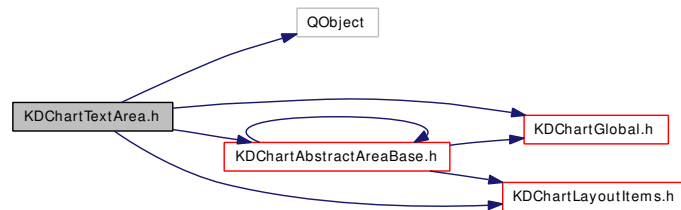
Include dependency graph for KDChartTextArea.cpp:



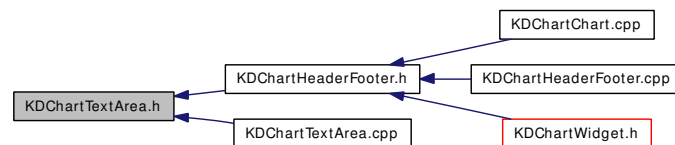
## 10.111 KDChartTextArea.h File Reference

```
#include <QObject>
#include "KDChartGlobal.h"
#include "KDChartAbstractAreaBase.h"
#include "KDChartLayoutItems.h"
```

Include dependency graph for KDChartTextArea.h:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace [KDChart](#)

## Classes

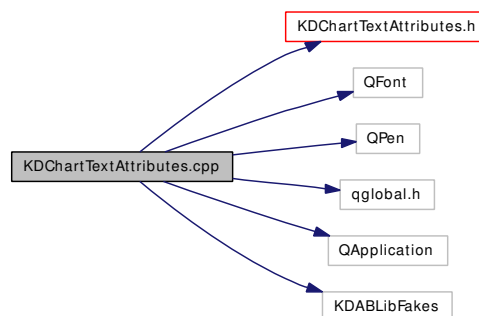
- class [KDChart::TextArea](#)

*A text area in the chart with a background, a frame, etc.*

## 10.112 KDChartTextAttributes.cpp File Reference

```
#include "KDChartTextAttributes.h"
#include <QFont>
#include <QPen>
#include <qglobal.h>
#include <QApplication>
#include <KDABLibFakes>
```

Include dependency graph for KDChartTextAttributes.cpp:



### Defines

- #define `d_func()`

### Functions

- QDebug `operator<<` (QDebug dbg, const `KDChart::TextAttributes` &ta)

#### 10.112.1 Define Documentation

##### 10.112.1.1 #define `d_func()`

Definition at line 34 of file KDChartTextAttributes.cpp.

#### 10.112.2 Function Documentation

##### 10.112.2.1 QDebug `operator<<` (QDebug *dbg*, const `KDChart::TextAttributes` & *ta*)

Definition at line 233 of file KDChartTextAttributes.cpp.

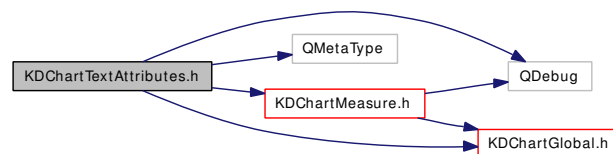
```
234 {
235     dbg << "KDChart::TextAttributes ("
236         << "visible="<<ta.isVisible()
237         << "font="<<ta.font().toString() /* What? No QDebug for QFont? */
238         << "fontsize="<<ta.fontSize()
239         << "minimalFontSize="<<ta.minimalFontSize()
```

```
240         << "autorotate="<<ta.autoRotate()  
241         << "autoshrink="<<ta.autoShrink()  
242         << "rotation="<<ta.rotation()  
243         << "pen="<<ta.pen()  
244         << " ) ";  
245     return dbg;  
246 }
```

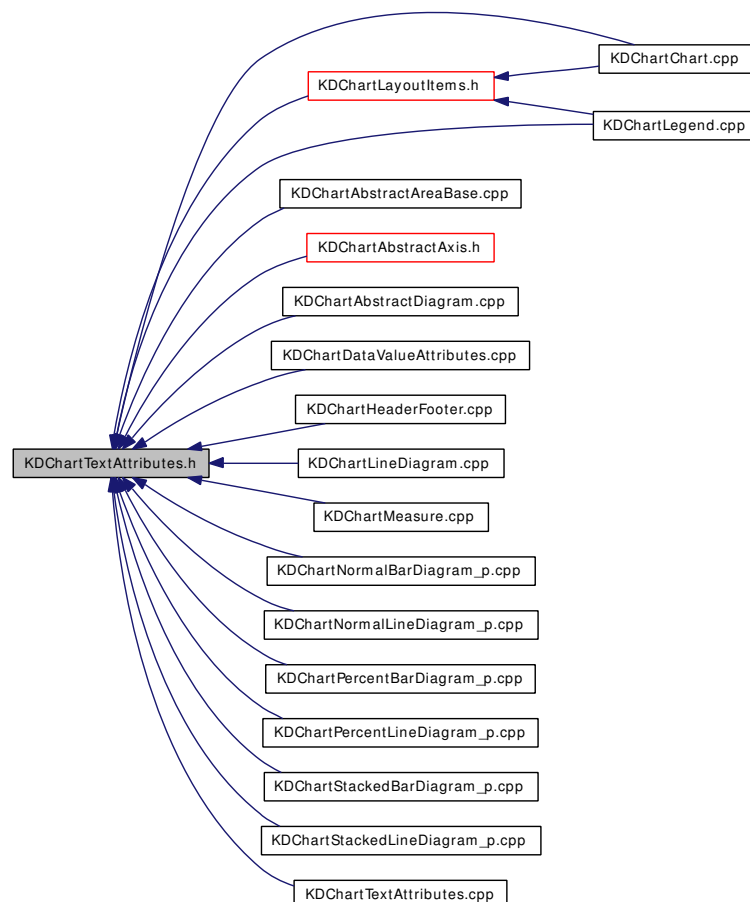
## 10.113 KDChartTextAttributes.h File Reference

```
#include <QDebug>
#include <QMetaType>
#include "KDChartGlobal.h"
#include "KDChartMeasure.h"
```

Include dependency graph for KDChartTextAttributes.h:



This graph shows which files directly or indirectly include this file:





## Namespaces

- namespace [KDChart](#)

## Classes

- class [KDChart::TextAttributes](#)

*A set of text attributes.*

## Functions

- `KDCHART_EXPORT QDebug operator<< (QDebug, const KDChart::TextAttributes &)`
- `Q_DECLARE_TYPEINFO (KDChart::TextAttributes, Q_MOVABLE_TYPE)`

### 10.113.1 Function Documentation

#### 10.113.1.1 `KDCHART_EXPORT QDebug operator<< (QDebug, const KDChart::TextAttributes &)`

Definition at line 233 of file `KDChartTextAttributes.cpp`.

References `KDChart::TextAttributes::autoRotate()`, `KDChart::TextAttributes::autoShrink()`, `KDChart::TextAttributes::font()`, `KDChart::TextAttributes::fontSize()`, `KDChart::TextAttributes::isVisible()`, `KDChart::TextAttributes::minimalFontSize()`, `KDChart::TextAttributes::pen()`, and `KDChart::TextAttributes::rotation()`.

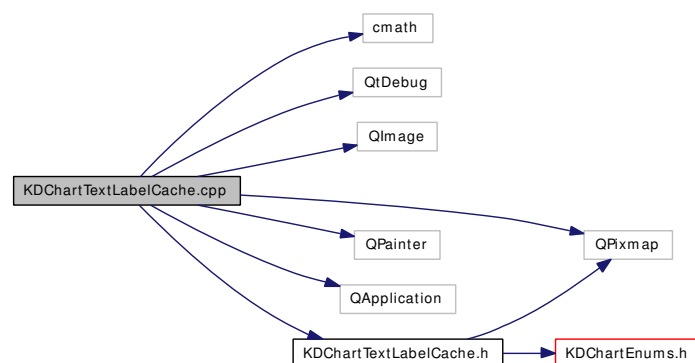
```
234 {
235     dbg << "KDChart::TextAttributes("
236         << "visible="<<ta.isVisible()
237         << "font="<<ta.font().toString() /* What? No QDebug for QFont? */
238         << "fontsize="<<ta.fontSize()
239         << "minimalfontsize="<<ta.minimalFontSize()
240         << "autorotate="<<ta.autoRotate()
241         << "autoshrink="<<ta.autoShrink()
242         << "rotation="<<ta.rotation()
243         << "pen="<<ta.pen()
244         << ")";
245     return dbg;
246 }
```

#### 10.113.1.2 `Q_DECLARE_TYPEINFO (KDChart::TextAttributes, Q_MOVABLE_TYPE)`

## 10.114 KDChartTextLabelCache.cpp File Reference

```
#include <cmath>
#include <QtDebug>
#include <QImage>
#include <QPixmap>
#include <QPainter>
#include <QApplication>
#include "KDChartTextLabelCache.h"
```

Include dependency graph for KDChartTextLabelCache.cpp:



### Defines

- #define [DUMP\\_CACHE\\_STATS](#)
- #define [INC\\_HIT\\_COUNT](#) { ++HitCount; }
- #define [INC\\_MISS\\_COUNT](#) { ++MissCount; }

### Variables

- int [HitCount](#) = 0
- int [MissCount](#) = 0

### 10.114.1 Define Documentation

#### 10.114.1.1 #define DUMP\_CACHE\_STATS

##### Value:

```
if ( HitCount != 0 && MissCount != 0 ) { \
    int total = HitCount + MissCount; \
    double hitQuote = ( 1.0 * HitCount ) / total; \
    qDebug() << "PrerenderedLabel dtor: hits/misses/total:" \
    << HitCount << "/" << MissCount << "/" << total \
    << "(" << 100 * hitQuote << "% hits)"; \
}
```

Definition at line 45 of file KDChartTextLabelCache.cpp.

Referenced by PrerenderedLabel::~~PrerenderedLabel().

#### 10.114.1.2 `#define INC_HIT_COUNT { ++HitCount; }`

Definition at line 43 of file KDChartTextLabelCache.cpp.

Referenced by PrerenderedLabel::pixmap(), and PrerenderedLabel::referencePointLocation().

#### 10.114.1.3 `#define INC_MISS_COUNT { ++MissCount; }`

Definition at line 44 of file KDChartTextLabelCache.cpp.

Referenced by PrerenderedLabel::pixmap(), and PrerenderedLabel::referencePointLocation().

### 10.114.2 Variable Documentation

#### 10.114.2.1 `int HitCount = 0`

Definition at line 41 of file KDChartTextLabelCache.cpp.

#### 10.114.2.2 `int MissCount = 0`

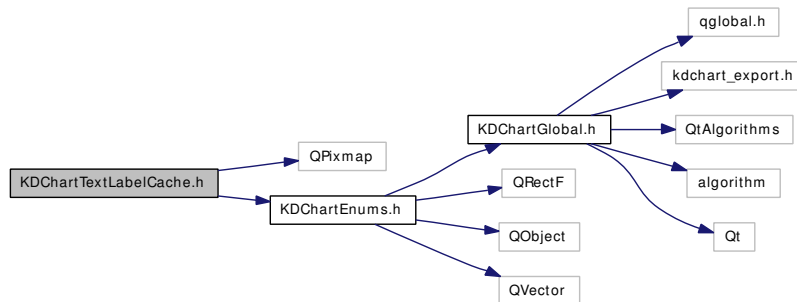
Definition at line 42 of file KDChartTextLabelCache.cpp.

## 10.115 KDChartTextLabelCache.h File Reference

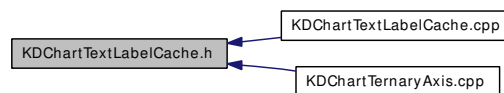
```
#include <QPixmap>
```

```
#include "KDChartEnums.h"
```

Include dependency graph for KDChartTextLabelCache.h:



This graph shows which files directly or indirectly include this file:



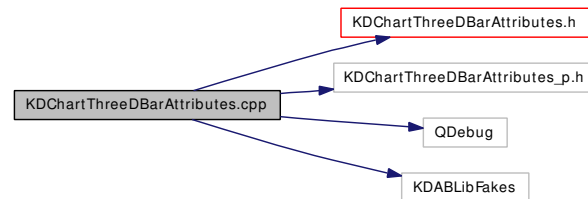
## Classes

- class [PrerenderedElement](#)  
*base class for prerendered elements like labels, pixmaps, markers, etc.*
- class [PrerenderedLabel](#)  
*PrerenderedLabel is an internal [KDChart](#) class that simplifies creation and caching of cached text labels.*

## 10.116 KDChartThreeDBarAttributes.cpp File Reference

```
#include "KDChartThreeDBarAttributes.h"
#include "KDChartThreeDBarAttributes_p.h"
#include <QDebug>
#include <KDABLibFakes>
```

Include dependency graph for KDChartThreeDBarAttributes.cpp:



### Defines

- #define [d d\\_func\(\)](#)

### Functions

- QDebug [operator<<](#) (QDebug dbg, const [KDChart::ThreeDBarAttributes](#) &a)

#### 10.116.1 Define Documentation

##### 10.116.1.1 #define [d d\\_func\(\)](#)

Definition at line 33 of file KDChartThreeDBarAttributes.cpp.

#### 10.116.2 Function Documentation

##### 10.116.2.1 QDebug [operator<<](#) (QDebug *dbg*, const [KDChart::ThreeDBarAttributes](#) &*a*)

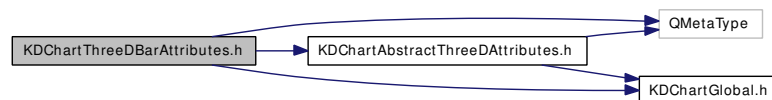
Definition at line 105 of file KDChartThreeDBarAttributes.cpp.

```
106 {
107     dbg << "KDChart::ThreeDBarAttributes(";
108     dbg = operator <<( dbg, static_cast<const AbstractThreeDAttributes&>(a) );
109     dbg << "useShadowColors=" << a.useShadowColors()
110         << "angle=" << a.angle() << ")";
111     return dbg;
112 }
```

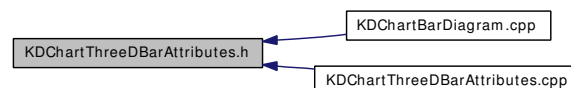
## 10.117 KDChartThreeDBarAttributes.h File Reference

```
#include <QMetaType>
#include "KDChartAbstractThreeDAttributes.h"
#include "KDChartGlobal.h"
```

Include dependency graph for KDChartThreeDBarAttributes.h:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [KDChart](#)

### Classes

- class [KDChart::ThreeDBarAttributes](#)

*A set of 3D bar attributes.*

### Functions

- `KDCHART_EXPORT QDebug operator<< (QDebug, const KDChart::ThreeDBarAttributes &)`
- `Q_DECLARE_TYPEINFO (KDChart::ThreeDBarAttributes, Q_MOVABLE_TYPE)`

#### 10.117.1 Function Documentation

##### 10.117.1.1 `KDCHART_EXPORT QDebug operator<< (QDebug, const KDChart::ThreeDBarAttributes &)`

Definition at line 105 of file KDChartThreeDBarAttributes.cpp.

References `KDChart::ThreeDBarAttributes::angle()`, and `KDChart::ThreeDBarAttributes::useShadowColors()`.

```

106 {
107     dbg << "KDChart::ThreeDBarAttributes(";
108     dbg = operator <<( dbg, static_cast<const AbstractThreeDAttributes>(a) );

```

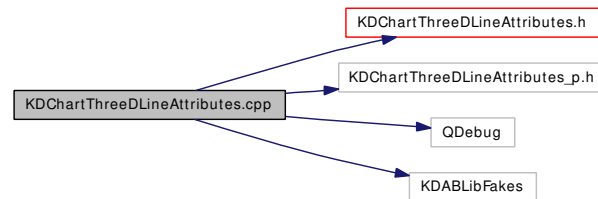
```
109     dbg << "useShadowColors=" << a.useShadowColors()
110         << "angle=" << a.angle() << " ";
111     return dbg;
112 }
```

#### 10.117.1.2 Q\_DECLARE\_TYPEINFO (KDChart::ThreeDBarAttributes, Q\_MOVABLE\_TYPE)

## 10.118 KDChartThreeDLineAttributes.cpp File Reference

```
#include "KDChartThreeDLineAttributes.h"
#include "KDChartThreeDLineAttributes_p.h"
#include <QDebug>
#include <KDABLibFakes>
```

Include dependency graph for KDChartThreeDLineAttributes.cpp:



### Defines

- #define [d d\\_func\(\)](#)

### Functions

- QDebug [operator<<](#) (QDebug dbg, const [KDChart::ThreeDLineAttributes](#) &a)

#### 10.118.1 Define Documentation

##### 10.118.1.1 #define [d d\\_func\(\)](#)

Definition at line 33 of file KDChartThreeDLineAttributes.cpp.

#### 10.118.2 Function Documentation

##### 10.118.2.1 QDebug [operator<<](#) (QDebug *dbg*, const [KDChart::ThreeDLineAttributes](#) &*a*)

Definition at line 106 of file KDChartThreeDLineAttributes.cpp.

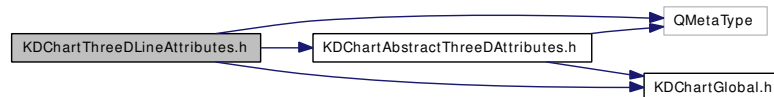
```
107 {
108     dbg << "KDChart::ThreeDLineAttributes(";
109     dbg = operator <<( dbg, static_cast<const AbstractThreeDAttributes&>(a) );
110     dbg << " lineXRotation=" << a.lineXRotation()
111         << " lineYRotation=" << a.lineYRotation()
112         << ")";
113     return dbg;
114 }
```



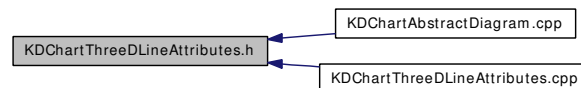
## 10.119 KDChartThreeDLineAttributes.h File Reference

```
#include <QMetaType>
#include "KDChartAbstractThreeDAttributes.h"
#include "KDChartGlobal.h"
```

Include dependency graph for KDChartThreeDLineAttributes.h:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [KDChart](#)

### Classes

- class [KDChart::ThreeDLineAttributes](#)

*A set of 3D line attributes.*

### Functions

- `KDCHART_EXPORT QDebug operator<< (QDebug, const KDChart::ThreeDLineAttributes &)`
- `Q_DECLARE_TYPEINFO (KDChart::ThreeDLineAttributes, Q_MOVABLE_TYPE)`

#### 10.119.1 Function Documentation

##### 10.119.1.1 `KDCHART_EXPORT QDebug operator<< (QDebug, const KDChart::ThreeDLineAttributes &)`

Definition at line 106 of file KDChartThreeDLineAttributes.cpp.

References `KDChart::ThreeDLineAttributes::lineXRotation()`, and `KDChart::ThreeDLineAttributes::lineYRotation()`.

```
107 {
108     dbg << "KDChart::ThreeDLineAttributes(";
109     dbg = operator <<( dbg, static_cast<const AbstractThreeDAttributes>(a) );
```

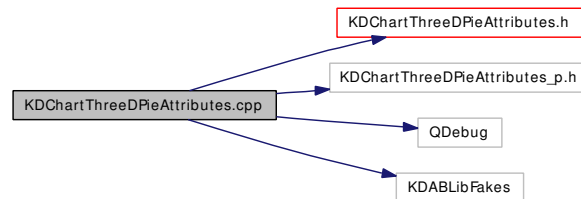
```
110     dbg << " lineXRotation="<< a.lineXRotation()  
111         << " lineYRotation="<< a.lineYRotation()  
112         << ") ";  
113     return dbg;  
114 }
```

#### 10.119.1.2 Q\_DECLARE\_TYPEINFO ([KDChart::ThreeDLineAttributes](#), Q\_MOVABLE\_TYPE)

## 10.120 KDChartThreeDPieAttributes.cpp File Reference

```
#include "KDChartThreeDPieAttributes.h"
#include "KDChartThreeDPieAttributes_p.h"
#include <QDebug>
#include <KDABLibFakes>
```

Include dependency graph for KDChartThreeDPieAttributes.cpp:



### Defines

- #define [d d\\_func\(\)](#)

### Functions

- QDebug [operator<<](#) (QDebug dbg, const [KDChart::ThreeDPieAttributes](#) &a)

#### 10.120.1 Define Documentation

##### 10.120.1.1 #define [d d\\_func\(\)](#)

Definition at line 33 of file KDChartThreeDPieAttributes.cpp.

#### 10.120.2 Function Documentation

##### 10.120.2.1 QDebug [operator<<](#) (QDebug *dbg*, const [KDChart::ThreeDPieAttributes](#) &*a*)

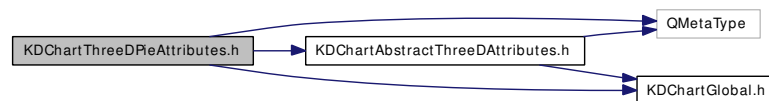
Definition at line 92 of file KDChartThreeDPieAttributes.cpp.

```
93 {
94     dbg << "KDChart::ThreeDPieAttributes(";
95     dbg = operator <<( dbg, static_cast<const AbstractThreeDAttributes>(a) );
96     dbg << "useShadowColors=" << a.useShadowColors() << ")";
97     return dbg;
98 }
```

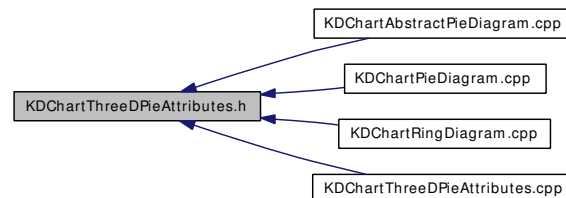
## 10.121 KDChartThreeDPieAttributes.h File Reference

```
#include <QMetaType>
#include "KDChartAbstractThreeDAttributes.h"
#include "KDChartGlobal.h"
```

Include dependency graph for KDChartThreeDPieAttributes.h:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [KDChart](#)

### Classes

- class [KDChart::ThreeDPieAttributes](#)

*A set of 3D pie attributes.*

### Functions

- `KDCHART_EXPORT QDebug operator<< (QDebug, const KDChart::ThreeDPieAttributes &)`
- `Q_DECLARE_TYPEINFO (KDChart::ThreeDPieAttributes, Q_MOVABLE_TYPE)`

#### 10.121.1 Function Documentation

##### 10.121.1.1 `KDCHART_EXPORT QDebug operator<< (QDebug, const KDChart::ThreeDPieAttributes &)`

Definition at line 92 of file KDChartThreeDPieAttributes.cpp.

References [KDChart::ThreeDPieAttributes::useShadowColors\(\)](#).

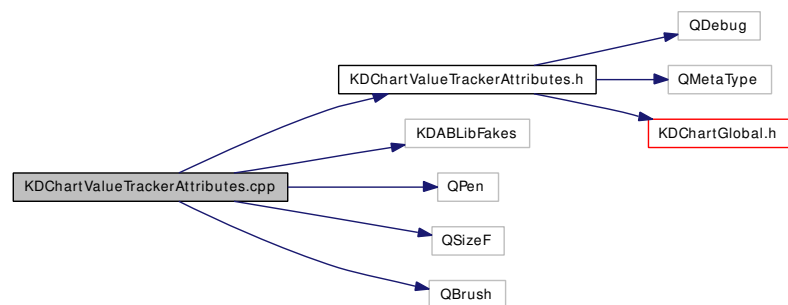
```
93 {
94     dbg << "KDChart::ThreeDPieAttributes(";
95     dbg = operator <<( dbg, static_cast<const AbstractThreeDAttributes&>(a) );
96     dbg << "useShadowColors="<< a.useShadowColors() << ")";
97     return dbg;
98 }
```

#### 10.121.1.2 Q\_DECLARE\_TYPEINFO (KDChart::ThreeDPieAttributes, Q\_MOVABLE\_TYPE)

## 10.122 KDChartValueTrackerAttributes.cpp File Reference

```
#include "KDChartValueTrackerAttributes.h"
#include <KDABLibFakes>
#include <QPen>
#include <QSizeF>
#include <QBrush>
```

Include dependency graph for KDChartValueTrackerAttributes.cpp:



### Defines

- #define `d_func()`

### Functions

- QDebug `operator<<` (QDebug *dbg*, const `KDChart::ValueTrackerAttributes` &*va*)

#### 10.122.1 Define Documentation

##### 10.122.1.1 #define `d_func()`

Definition at line 33 of file KDChartValueTrackerAttributes.cpp.

#### 10.122.2 Function Documentation

##### 10.122.2.1 QDebug `operator<<` (QDebug *dbg*, const `KDChart::ValueTrackerAttributes` &*va*)

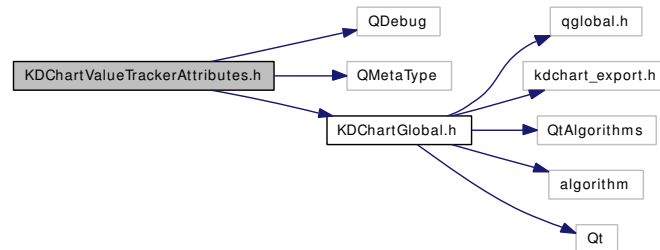
Definition at line 133 of file KDChartValueTrackerAttributes.cpp.

```
134 {
135     dbg << "KDChart::ValueTrackerAttributes ("
136         << "pen=" << va.pen ()
137         << "markerSize=" << va.markerSize ()
138         << "enabled=" << va.isEnabled ()
139         << ") ";
140     return dbg;
141 }
```

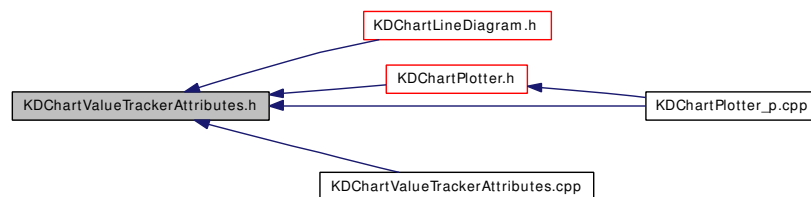
## 10.123 KDChartValueTrackerAttributes.h File Reference

```
#include <QDebug>
#include <QMetaType>
#include "KDChartGlobal.h"
```

Include dependency graph for KDChartValueTrackerAttributes.h:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [KDChart](#)

### Classes

- class [KDChart::ValueTrackerAttributes](#)  
*Cell-specific attributes regarding value tracking.*

### Functions

- `KDCHART_EXPORT QDebug operator<< (QDebug, const KDChart::ValueTrackerAttributes &)`
- `Q_DECLARE_TYPEINFO (KDChart::ValueTrackerAttributes, Q_MOVABLE_TYPE)`

#### 10.123.1 Function Documentation

##### 10.123.1.1 KDCHART\_EXPORT QDebug operator<< (QDebug, const [KDChart::ValueTrackerAttributes](#) &)

Definition at line 133 of file KDChartValueTrackerAttributes.cpp.

References `KDChart::ValueTrackerAttributes::isEnabled()`, `KDChart::ValueTrackerAttributes::markerSize()`, and `KDChart::ValueTrackerAttributes::pen()`.

```
134 {
135     dbg << "KDChart::ValueTrackerAttributes ("
136           << "pen=" << va.pen()
137           << "markerSize=" << va.markerSize()
138           << "enabled=" << va.isEnabled()
139           << ") ";
140     return dbg;
141 }
```

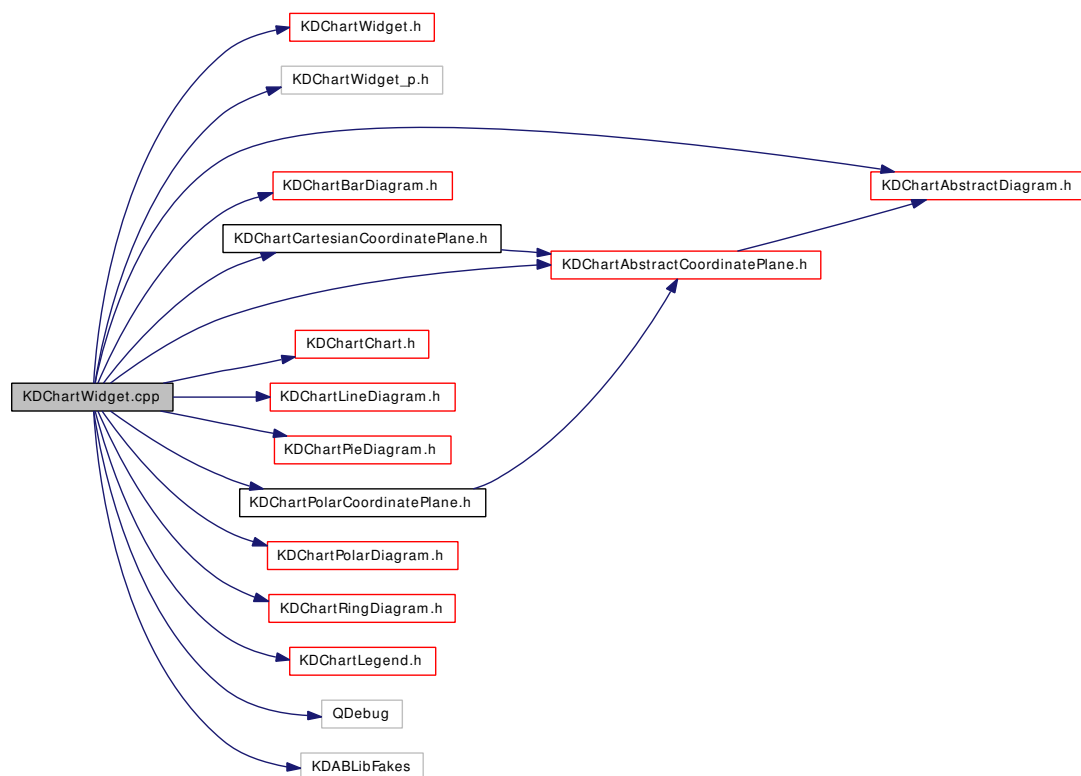
#### 10.123.1.2 `Q_DECLARE_TYPEINFO` (`KDChart::ValueTrackerAttributes`, `Q_MOVABLE_TYPE`)



## 10.124 KDChartWidget.cpp File Reference

```
#include <KDChartWidget.h>
#include <KDChartWidget_p.h>
#include <KDChartAbstractDiagram.h>
#include <KDChartBarDiagram.h>
#include <KDChartCartesianCoordinatePlane.h>
#include <KDChartChart.h>
#include <KDChartAbstractCoordinatePlane.h>
#include <KDChartLineDiagram.h>
#include <KDChartPieDiagram.h>
#include <KDChartPolarCoordinatePlane.h>
#include <KDChartPolarDiagram.h>
#include <KDChartRingDiagram.h>
#include <KDChartLegend.h>
#include <QDebug>
#include <KDABLibFakes>
```

Include dependency graph for KDChartWidget.cpp:



## Defines

- `#define d_func()`
- `#define SET_SUB_TYPE(DIAGRAM, SUBTYPE)`
- `#define TEST_SUB_TYPE(DIAGRAM, INTERNALSUBTYPE, SUBTYPE)`

## Functions

- static bool `isCartesian (KDChart::Widget::ChartType type)`
- static bool `isPolar (KDChart::Widget::ChartType type)`

### 10.124.1 Define Documentation

#### 10.124.1.1 `#define d_func()`

Definition at line 49 of file KDChartWidget.cpp.

#### 10.124.1.2 `#define SET_SUB_TYPE(DIAGRAM, SUBTYPE)`

##### Value:

```
{ \
    if( DIAGRAM ) \
        DIAGRAM->setType( SUBTYPE ); \
}
```

Referenced by `KDChart::Widget::setSubType()`.

#### 10.124.1.3 `#define TEST_SUB_TYPE(DIAGRAM, INTERNALSUBTYPE, SUBTYPE)`

##### Value:

```
{ \
    if( DIAGRAM && DIAGRAM->type() == INTERNALSUBTYPE ) \
        retVal = SUBTYPE; \
}
```

Referenced by `KDChart::Widget::subType()`.

### 10.124.2 Function Documentation

#### 10.124.2.1 static bool `isCartesian (KDChart::Widget::ChartType type)` [static]

Definition at line 385 of file KDChartWidget.cpp.

References `KDChart::Widget::Bar`, and `KDChart::Widget::Line`.

Referenced by `KDChart::Widget::setType()`.

```
386 {
387     return (type == KDChart::Widget::Bar || type == KDChart::Widget::Line);
388 }
```

**10.124.2.2 static bool isPolar (KDChart::Widget::ChartType *type*)** [static]

Definition at line 390 of file KDChartWidget.cpp.

References KDChart::Widget::Pie, KDChart::Widget::Polar, and KDChart::Widget::Ring.

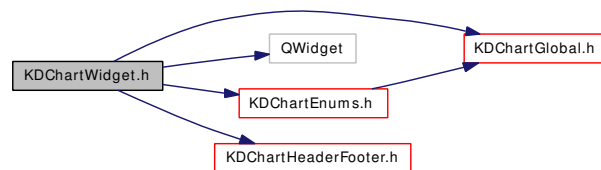
Referenced by KDChart::Widget::setType().

```
391 {  
392     return (type == KDChart::Widget::Pie  
393             || type == KDChart::Widget::Ring  
394             || type == KDChart::Widget::Polar );  
395 }
```

## 10.125 KDChartWidget.h File Reference

```
#include "KDChartGlobal.h"  
#include <QWidget>  
#include "KDChartEnums.h"  
#include "KDChartHeaderFooter.h"
```

Include dependency graph for KDChartWidget.h:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace [KDChart](#)

## Classes

- class [KDChart::Widget](#)

*The [KD Chart](#) widget for usage without Model/View.*

## 10.126 KDChartZoomParameters.h File Reference

### Namespaces

- namespace [KDChart](#)

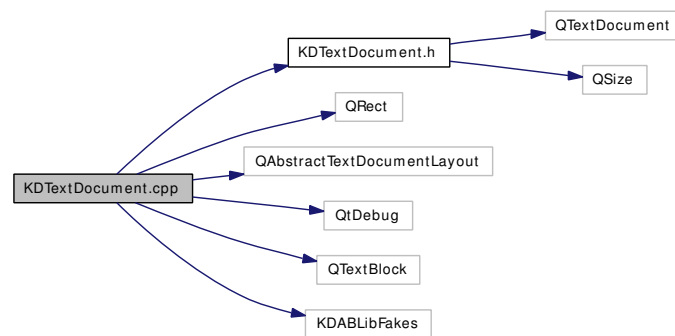
### Classes

- class [KDChart::ZoomParameters](#)  
*[ZoomParameters](#) stores the center and the factor of zooming internally.*

## 10.127 KDDocument.cpp File Reference

```
#include "KDDocument.h"  
#include <QRect>  
#include <QAbstractTextDocumentLayout>  
#include <QtDebug>  
#include <QTextBlock>  
#include <KDABLibFakes>
```

Include dependency graph for KDDocument.cpp:

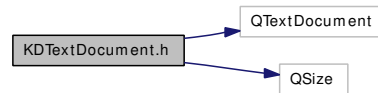


## 10.128 KDDocument.h File Reference

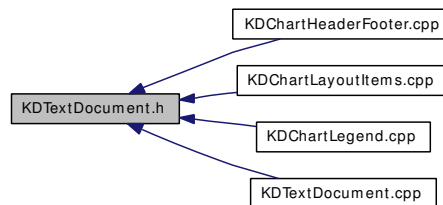
```
#include <QTextDocument>
```

```
#include <QSize>
```

Include dependency graph for KDDocument.h:



This graph shows which files directly or indirectly include this file:



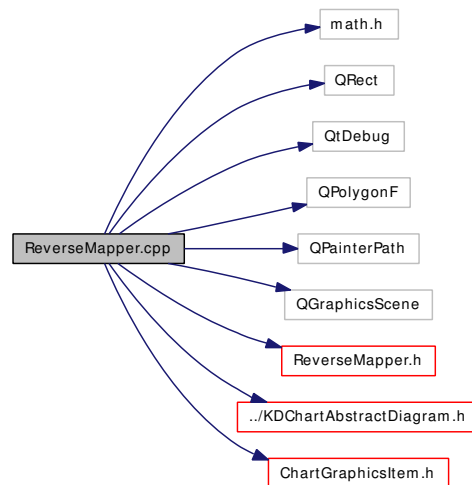
### Classes

- class [KDDocument](#)  
*KDDocument* is an internally used enhanced *QTextDocument*.

## 10.129 ReverseMapper.cpp File Reference

```
#include <math.h>
#include <QRect>
#include <QtDebug>
#include <QPolygonF>
#include <QPainterPath>
#include <QGraphicsScene>
#include "ReverseMapper.h"
#include "../KDChartAbstractDiagram.h"
#include "ChartGraphicsItem.h"
```

Include dependency graph for ReverseMapper.cpp:





## 10.130 ReverseMapper.h File Reference

```
#include <QModelIndex>
```

Include dependency graph for ReverseMapper.h:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [KDChart](#)

### Classes

- class [KDChart::ReverseMapper](#)

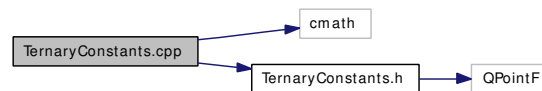
The [ReverseMapper](#) stores information about objects on a chart and their respective model indexes.

## 10.131 TernaryConstants.cpp File Reference

```
#include <cmath>
```

```
#include "TernaryConstants.h"
```

Include dependency graph for TernaryConstants.cpp:



### Functions

- const QPointF [AxisVector\\_B\\_A](#) (TriangleTop)
- const QPointF [AxisVector\\_B\\_C](#) (TriangleBottomRight)
- const QPointF [AxisVector\\_C\\_A](#) (TriangleTop-TriangleBottomRight)
- const QPointF [FullMarkerDistanceAC](#) (-[RelMarkerLength](#) \*Norm\_C\_A)
- const QPointF [FullMarkerDistanceBA](#) ([RelMarkerLength](#) \*Norm\_B\_A)
- const QPointF [FullMarkerDistanceBC](#) ([RelMarkerLength](#) \*Norm\_B\_C)
- const QPointF [Norm\\_B\\_A](#) (-AxisVector\_B\_A.y(), AxisVector\_B\_A.x())
- const QPointF [Norm\\_B\\_C](#) (-AxisVector\_B\_C.y(), AxisVector\_B\_C.x())
- const QPointF [Norm\\_C\\_A](#) (-AxisVector\_C\_A.y(), AxisVector\_C\_A.x())
- const QPointF [TriangleBottomLeft](#) (0.0, 0.0)
- const QPointF [TriangleBottomRight](#) (1.0, 0.0)
- const QPointF [TriangleTop](#) (0.5, [TriangleHeight](#))

### Variables

- const double [RelMarkerLength](#) = 0.03 \* [TriangleWidth](#)
- const double [Sqrt3](#) = sqrt( 3.0 )
- const double [TriangleHeight](#) = 0.5 \* [Sqrt3](#)
- const double [TriangleWidth](#) = 1.0

### 10.131.1 Function Documentation

- 10.131.1.1 `const QPointF AxisVector_B_A (TriangleTop)`
- 10.131.1.2 `const QPointF AxisVector_B_C (TriangleBottomRight)`
- 10.131.1.3 `const QPointF AxisVector_C_A (TriangleTop- TriangleBottomRight)`
- 10.131.1.4 `const QPointF FullMarkerDistanceAC (RelMarkerLength * Norm_C_A)`
- 10.131.1.5 `const QPointF FullMarkerDistanceBA (RelMarkerLength * Norm_B_A)`
- 10.131.1.6 `const QPointF FullMarkerDistanceBC (RelMarkerLength * Norm_B_C)`
- 10.131.1.7 `const QPointF Norm_B_A (-AxisVector_B_A.y(), AxisVector_B_A.x())`
- 10.131.1.8 `const QPointF Norm_B_C (-AxisVector_B_C.y(), AxisVector_B_C.x())`
- 10.131.1.9 `const QPointF Norm_C_A (-AxisVector_C_A.y(), AxisVector_C_A.x())`
- 10.131.1.10 `const QPointF TriangleBottomLeft (0. 0, 0. 0)`
- 10.131.1.11 `const QPointF TriangleBottomRight (1. 0, 0. 0)`
- 10.131.1.12 `const QPointF TriangleTop (0. 5, TriangleHeight)`

### 10.131.2 Variable Documentation

- 10.131.2.1 `const double RelMarkerLength = 0.03 * TriangleWidth`
- 10.131.2.2 `const double Sqrt3 = sqrt( 3.0 )`
- 10.131.2.3 `const double TriangleHeight = 0.5 * Sqrt3`

Referenced by `KDChart::TernaryPointDiagram::calculateDataBoundaries()`, `KDChart::TernaryLineDiagram::calculateDataBoundaries()`, and `KDChart::TernaryCoordinatePlane::layoutDiagrams()`.

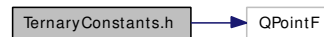
- 10.131.2.4 `const double TriangleWidth = 1.0`

Referenced by `KDChart::TernaryCoordinatePlane::layoutDiagrams()`.

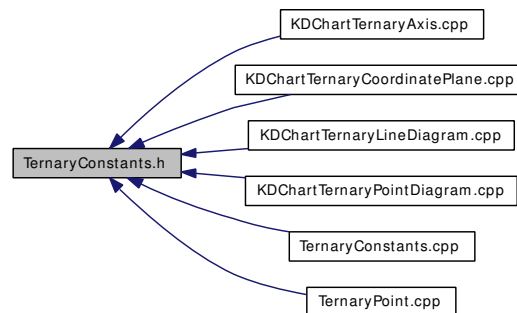
## 10.132 TernaryConstants.h File Reference

```
#include <QPointF>
```

Include dependency graph for TernaryConstants.h:



This graph shows which files directly or indirectly include this file:



### Variables

- const QPointF [AxisVector\\_B\\_A](#)
- const QPointF [AxisVector\\_B\\_C](#)
- const QPointF [AxisVector\\_C\\_A](#)
- const QPointF [FullMarkerDistanceAC](#)
- const QPointF [FullMarkerDistanceBA](#)
- const QPointF [FullMarkerDistanceBC](#)
- const QPointF [Norm\\_B\\_A](#)
- const QPointF [Norm\\_B\\_C](#)
- const QPointF [Norm\\_C\\_A](#)
- const double [RelMarkerLength](#)
- const double [Sqrt3](#)
- const QPointF [TriangleBottomLeft](#)
- const QPointF [TriangleBottomRight](#)
- const double [TriangleHeight](#)
- const QPointF [TriangleTop](#)
- const double [TriangleWidth](#)

### 10.132.1 Variable Documentation

#### 10.132.1.1 const QPointF AxisVector\_B\_A

#### 10.132.1.2 const QPointF AxisVector\_B\_C

#### 10.132.1.3 const QPointF AxisVector\_C\_A

Referenced by `translate()`.

**10.132.1.4**   `const QPointF FullMarkerDistanceAC`

**10.132.1.5**   `const QPointF FullMarkerDistanceBA`

**10.132.1.6**   `const QPointF FullMarkerDistanceBC`

**10.132.1.7**   `const QPointF Norm_B_A`

**10.132.1.8**   `const QPointF Norm_B_C`

**10.132.1.9**   `const QPointF Norm_C_A`

**10.132.1.10**   `const double RelMarkerLength`

**10.132.1.11**   `const double Sqrt3`

**10.132.1.12**   `const QPointF TriangleBottomLeft`

Referenced by `KDChart::TernaryPointDiagram::calculateDataBoundaries()`, and `KDChart::TernaryLineDiagram::calculateDataBoundaries()`.

**10.132.1.13**   `const QPointF TriangleBottomRight`

Referenced by `KDChart::TernaryPointDiagram::calculateDataBoundaries()`, and `KDChart::TernaryLineDiagram::calculateDataBoundaries()`.

**10.132.1.14**   `const double TriangleHeight`

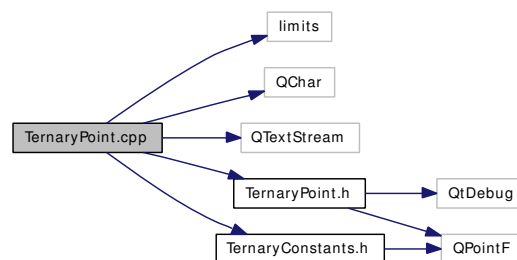
**10.132.1.15**   `const QPointF TriangleTop`

**10.132.1.16**   `const double TriangleWidth`

## 10.133 TernaryPoint.cpp File Reference

```
#include <limits>
#include <QChar>
#include <QTextStream>
#include "TernaryPoint.h"
#include "TernaryConstants.h"
```

Include dependency graph for TernaryPoint.cpp:



### Functions

- QDebug [operator<<](#) (QDebug stream, const [TernaryPoint](#) &point)
- QPointF [translate](#) (const [TernaryPoint](#) &point)

### 10.133.1 Function Documentation

#### 10.133.1.1 QDebug operator<< (QDebug stream, const [TernaryPoint](#) &point)

Definition at line 75 of file TernaryPoint.cpp.

References [TernaryPoint::a\(\)](#), [TernaryPoint::b\(\)](#), [TernaryPoint::c\(\)](#), and [TernaryPoint::isValid\(\)](#).

```
76 {
77     QString string;
78     QTextStream text( &string );
79     text << "[TernaryPoint: ";
80     if ( point.isValid() ) {
81         text.setFieldWidth( 2 );
82         text.setPadChar( QLatin1Char( '0' ) );
83         text << ( int ) ( point.a() * 100.0 ) << "%|"
84             << ( int ) ( point.b() * 100.0 ) << "%|"
85             << ( int ) ( point.c() * 100.0 ) << "%]";
86     } else {
87         text << "a=" << point.a() << " - b=" << point.b() << " - INVALID]";
88     }
89     stream << string;
90     return stream;
91 }
```

#### 10.133.1.2 QPointF translate (const [TernaryPoint](#) &point)

Definition at line 93 of file TernaryPoint.cpp.

References TernaryPoint::a(), AxisVector\_C\_A, TernaryPoint::b(), and TernaryPoint::isValid().

Referenced by KDChart::TernaryPointDiagram::paint(), and KDChart::TernaryLineDiagram::paint().

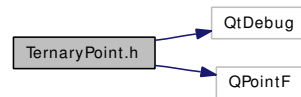
```
94 {
95     if ( point.isValid() ) {
96         // the position is calculated by
97         // - first moving along the B-C line to the function that b
98         //   selects
99         // - then traversing the selected function until we meet with
100        //   the function that A selects (which is a parallel of the B-C
101        //   line)
102        QPointF bPosition( 1.0 - point.b(), 0.0 );
103        QPointF aPosition( point.a() * AxisVector_C_A );
104        QPointF result( bPosition + aPosition );
105        return result;
106    } else {
107        qWarning() << "TernaryPoint::translate(TernaryPoint): cannot translate invalid ternary points:
108        << point;
109        return QPointF();
110    }
111 }
```

## 10.134 TernaryPoint.h File Reference

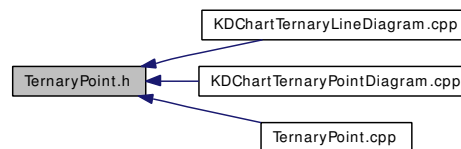
```
#include <QtDebug>
```

```
#include <QPointF>
```

Include dependency graph for TernaryPoint.h:



This graph shows which files directly or indirectly include this file:



### Classes

- class [TernaryPoint](#)

*[TernaryPoint](#) defines a point within a ternary coordinate plane.*

### Functions

- QDebug [operator<<](#) (QDebug stream, const [TernaryPoint](#) &point)
- QPointF [translate](#) (const [TernaryPoint](#) &)

### 10.134.1 Function Documentation

#### 10.134.1.1 QDebug operator<< (QDebug stream, const [TernaryPoint](#) & point)

Definition at line 75 of file TernaryPoint.cpp.

References [TernaryPoint::a\(\)](#), [TernaryPoint::b\(\)](#), [TernaryPoint::c\(\)](#), and [TernaryPoint::isValid\(\)](#).

```

76 {
77     QString string;
78     QTextStream text( &string );
79     text << "[TernaryPoint: ";
80     if ( point.isValid() ) {
81         text.setFieldWidth( 2 );
82         text.setPadChar( QLatin1Char( '0' ) );
83         text << ( int ) ( point.a() * 100.0 ) << "%| "
84             << ( int ) ( point.b() * 100.0 ) << "%| "
85             << ( int ) ( point.c() * 100.0 ) << "%]";
86     } else {
87         text << "a=" << point.a() << " - b=" << point.b() << " - INVALID]";
88     }
  
```



```
89     stream << string;
90     return stream;
91 }
```

### 10.134.1.2 QPointF translate (const TernaryPoint &)

Definition at line 93 of file TernaryPoint.cpp.

References TernaryPoint::a(), AxisVector\_C\_A, TernaryPoint::b(), and TernaryPoint::isValid().

Referenced by KDChart::TernaryLineDiagram::paint(), and KDChart::TernaryPointDiagram::paint().

```
94 {
95     if ( point.isValid() ) {
96         // the position is calculated by
97         // - first moving along the B-C line to the function that b
98         //   selects
99         // - then traversing the selected function until we meet with
100        //   the function that A selects (which is a parallel of the B-C
101        //   line)
102        QPointF bPosition( 1.0 - point.b(), 0.0 );
103        QPointF aPosition( point.a() * AxisVector_C_A );
104        QPointF result( bPosition + aPosition );
105        return result;
106    } else {
107        qWarning() << "TernaryPoint::translate(TernaryPoint): cannot translate invalid ternary points:
108        << point;
109        return QPointF();
110    }
111 }
```



# Chapter 11

## KD Chart 2 Page Documentation

### 11.1 Deprecated List

Member	<a href="#">KDChart::AbstractDiagram::valueForCell</a>	(int row, int column) const		
Member	<a href="#">KDChart::AbstractPieDiagram::setStartPosition</a>	(int degrees) Use	PolarCoordinate-	
	Plane::setStartPosition( qreal degrees ) instead.			
Member	<a href="#">KDChart::AbstractPieDiagram::startPosition</a>	() const Use	qreal	PolarCoordinate-
	Plane::startPosition instead.			
Member	<a href="#">KDChart::PolarDiagram::setZeroDegreePosition</a>	(int degrees) Use	PolarCoordinate-	
	Plane::setStartPosition( qreal degrees ) instead.			
Member	<a href="#">KDChart::PolarDiagram::zeroDegreePosition</a>	() const Use	qreal	PolarCoordinate-
	Plane::startPosition instead.			

# Index

- ~AbstractArea
  - KDChart::AbstractArea, [37](#)
- ~AbstractAreaBase
  - KDChart::AbstractAreaBase, [49](#)
- ~AbstractAreaWidget
  - KDChart::AbstractAreaWidget, [56](#)
- ~AbstractAxis
  - KDChart::AbstractAxis, [69](#)
- ~AbstractCartesianDiagram
  - KDChart::AbstractCartesianDiagram, [92](#)
- ~AbstractCoordinatePlane
  - KDChart::AbstractCoordinatePlane, [139](#)
- ~AbstractDiagram
  - KDChart::AbstractDiagram, [173](#)
- ~AbstractPieDiagram
  - KDChart::AbstractPieDiagram, [219](#)
- ~AbstractPolarDiagram
  - KDChart::AbstractPolarDiagram, [267](#)
- ~AbstractThreeDAttributes
  - KDChart::AbstractThreeDAttributes, [350](#)
- ~AttributesModel
  - KDChart::AttributesModel, [356](#)
- ~BackgroundAttributes
  - KDChart::BackgroundAttributes, [379](#)
- ~BarAttributes
  - KDChart::BarAttributes, [384](#)
- ~BarDiagram
  - KDChart::BarDiagram, [398](#)
- ~CartesianAxis
  - KDChart::CartesianAxis, [452](#)
- ~CartesianCoordinatePlane
  - KDChart::CartesianCoordinatePlane, [495](#)
- ~Chart
  - KDChart::Chart, [546](#)
- ~DataValueAttributes
  - KDChart::DataValueAttributes, [583](#)
- ~DiagramObserver
  - KDChart::DiagramObserver, [594](#)
- ~FrameAttributes
  - KDChart::FrameAttributes, [596](#)
- ~GlobalMeasureScaling
  - KDChart::GlobalMeasureScaling, [601](#)
- ~GridAttributes
  - KDChart::GridAttributes, [604](#)
- ~HeaderFooter
  - KDChart::HeaderFooter, [614](#)
- ~KDTextDocument
  - KDTextDocument, [1320](#)
- ~Legend
  - KDChart::Legend, [642](#)
- ~LineAttributes
  - KDChart::LineAttributes, [671](#)
- ~LineDiagram
  - KDChart::LineDiagram, [682](#)
- ~MarkerAttributes
  - KDChart::MarkerAttributes, [746](#)
- ~PaintContext
  - KDChart::PaintContext, [765](#)
- ~Palette
  - KDChart::Palette, [769](#)
- ~PieAttributes
  - KDChart::PieAttributes, [773](#)
- ~PieDiagram
  - KDChart::PieDiagram, [783](#)
- ~Plotter
  - KDChart::Plotter, [838](#)
- ~PolarCoordinatePlane
  - KDChart::PolarCoordinatePlane, [893](#)
- ~PolarDiagram
  - KDChart::PolarDiagram, [933](#)
- ~PrerenderedElement
  - PrerenderedElement, [1323](#)
- ~PrerenderedLabel
  - PrerenderedLabel, [1327](#)
- ~RelativePosition
  - KDChart::RelativePosition, [1009](#)
- ~ReverseMapper
  - KDChart::ReverseMapper, [1018](#)
- ~RingDiagram
  - KDChart::RingDiagram, [1028](#)
- ~TernaryAxis
  - KDChart::TernaryAxis, [1078](#)
- ~TernaryCoordinatePlane
  - KDChart::TernaryCoordinatePlane, [1104](#)
- ~TernaryLineDiagram
  - KDChart::TernaryLineDiagram, [1140](#)
- ~TernaryPointDiagram
  - KDChart::TernaryPointDiagram, [1185](#)
- ~TextArea
  - KDChart::TextArea, [1225](#)

- ~TextAttributes
  - KDChart::TextAttributes, [1242](#)
- ~ThreeDBarAttributes
  - KDChart::ThreeDBarAttributes, [1262](#)
- ~ThreeDLineAttributes
  - KDChart::ThreeDLineAttributes, [1267](#)
- ~ThreeDPieAttributes
  - KDChart::ThreeDPieAttributes, [1272](#)
- ~ValueTrackerAttributes
  - KDChart::ValueTrackerAttributes, [1277](#)
- ~Widget
  - KDChart::Widget, [1291](#)
- \_\_kdab\_\_dereference\_for\_methodcall
  - KDChartGlobal.h, [1431](#)
- a
  - TernaryPoint, [1343](#)
- AbstractArea
  - KDChart::AbstractArea, [37](#)
- AbstractAreaBase
  - KDChart::AbstractAreaBase, [49](#)
- AbstractAreaWidget
  - KDChart::AbstractAreaWidget, [56](#)
- AbstractAxis
  - KDChart::AbstractAxis, [69](#)
- AbstractCartesianDiagram
  - KDChart::AbstractCartesianDiagram, [92](#)
- AbstractCoordinatePlane
  - KDChart::AbstractCoordinatePlane, [139](#)
- AbstractDiagram
  - KDChart::AbstractDiagram, [172](#)
- AbstractDiagramList
  - KDChart, [32](#)
- AbstractLayoutItem
  - KDChart::AbstractLayoutItem, [210](#)
- AbstractPieDiagram
  - KDChart::AbstractPieDiagram, [219](#)
- AbstractPolarDiagram
  - KDChart::AbstractPolarDiagram, [267](#)
- AbstractProxyModel
  - KDChart::AbstractProxyModel, [305](#)
- AbstractThreeDAttributes
  - KDChart::AbstractThreeDAttributes, [350](#)
- ADD\_AUTO\_SPACER\_IF\_NEEDED
  - KDChartChart.cpp, [1406](#)
- ADD\_VBOX\_WITH\_LEGENDS
  - KDChartChart.cpp, [1406](#)
- addAxis
  - KDChart::AbstractCartesianDiagram, [93](#)
  - KDChart::BarDiagram, [399](#)
  - KDChart::LineDiagram, [683](#)
  - KDChart::Plotter, [838](#)
- addBrush
  - KDChart::Palette, [770](#)
- addCircle
  - KDChart::ReverseMapper, [1018](#)
- addCoordinatePlane
  - KDChart::Chart, [546](#)
- addDiagram
  - KDChart::AbstractCoordinatePlane, [140](#)
  - KDChart::CartesianCoordinatePlane, [495](#)
  - KDChart::Legend, [642](#)
  - KDChart::PolarCoordinatePlane, [893](#)
  - KDChart::TernaryCoordinatePlane, [1104](#)
- addHeaderFooter
  - KDChart::Chart, [547](#)
  - KDChart::Widget, [1291](#)
- addItem
  - KDChart::ReverseMapper, [1018](#)
- addLegend
  - KDChart::Chart, [547](#)
  - KDChart::Widget, [1292](#)
- addLine
  - KDChart::ReverseMapper, [1018](#)
- addPolygon
  - KDChart::ReverseMapper, [1019](#)
- addRect
  - KDChart::ReverseMapper, [1019](#)
- adjustedToMaxEmptyInnerPercentage
  - KDChart::CartesianCoordinatePlane, [495](#)
- adjustHorizontalRangeToData
  - KDChart::CartesianCoordinatePlane, [496](#)
- adjustLowerBoundToGrid
  - KDChart::GridAttributes, [604](#)
- adjustRangesToData
  - KDChart::CartesianCoordinatePlane, [496](#)
- adjustUpperBoundToGrid
  - KDChart::GridAttributes, [604](#)
- adjustVerticalRangeToData
  - KDChart::CartesianCoordinatePlane, [497](#)
- alignment
  - KDChart::Legend, [643](#)
  - KDChart::RelativePosition, [1010](#)
- alignToReferencePoint
  - KDChart::AbstractArea, [38](#)
  - KDChart::AbstractAreaBase, [49](#)
  - KDChart::AbstractAreaWidget, [57](#)
  - KDChart::AbstractAxis, [69](#)
  - KDChart::AbstractCoordinatePlane, [140](#)
  - KDChart::CartesianAxis, [452](#)
  - KDChart::CartesianCoordinatePlane, [497](#)
  - KDChart::HeaderFooter, [614](#)
  - KDChart::Legend, [643](#)
  - KDChart::PolarCoordinatePlane, [894](#)
  - KDChart::TernaryAxis, [1079](#)
  - KDChart::TernaryCoordinatePlane, [1105](#)
  - KDChart::TextArea, [1226](#)
- allHeadersFooters

- KDChart::Widget, 1292
- allLegends
  - KDChart::Widget, 1292
- allowOverlappingDataValueTexts
  - KDChart::AbstractCartesianDiagram, 93
  - KDChart::AbstractDiagram, 173
  - KDChart::AbstractPieDiagram, 219
  - KDChart::AbstractPolarDiagram, 267
  - KDChart::AbstractTernaryDiagram, 313
  - KDChart::BarDiagram, 399
  - KDChart::LineDiagram, 683
  - KDChart::PieDiagram, 784
  - KDChart::Plotter, 839
  - KDChart::PolarDiagram, 933
  - KDChart::RingDiagram, 1029
  - KDChart::TernaryLineDiagram, 1140
  - KDChart::TernaryPointDiagram, 1185
- angle
  - KDChart::ThreeDBarAttributes, 1262
  - PrerenderedLabel, 1327
- angleUnit
  - KDChart::PolarCoordinatePlane, 894
- antiAliasing
  - KDChart::AbstractCartesianDiagram, 93
  - KDChart::AbstractDiagram, 173
  - KDChart::AbstractPieDiagram, 220
  - KDChart::AbstractPolarDiagram, 267
  - KDChart::AbstractTernaryDiagram, 313
  - KDChart::BarDiagram, 399
  - KDChart::LineDiagram, 683
  - KDChart::PieDiagram, 784
  - KDChart::Plotter, 839
  - KDChart::PolarDiagram, 934
  - KDChart::RingDiagram, 1029
  - KDChart::TernaryLineDiagram, 1140
  - KDChart::TernaryPointDiagram, 1185
- areaBrush
  - KDChart::ValueTrackerAttributes, 1277
- areaGeometry
  - KDChart::AbstractArea, 38
  - KDChart::AbstractAreaBase, 49
  - KDChart::AbstractAreaWidget, 57
  - KDChart::AbstractAxis, 69
  - KDChart::AbstractCoordinatePlane, 140
  - KDChart::CartesianAxis, 452
  - KDChart::CartesianCoordinatePlane, 497
  - KDChart::HeaderFooter, 614
  - KDChart::Legend, 643
  - KDChart::PolarCoordinatePlane, 894
  - KDChart::TernaryAxis, 1079
  - KDChart::TernaryCoordinatePlane, 1105
  - KDChart::TextArea, 1226
- attributesChanged
  - KDChart::AttributesModel, 356
  - KDChart::PrivateAttributesModel, 993
- AttributesModel
  - KDChart::AttributesModel, 356
- attributesModel
  - KDChart::AbstractCartesianDiagram, 93
  - KDChart::AbstractDiagram, 173
  - KDChart::AbstractPieDiagram, 220
  - KDChart::AbstractPolarDiagram, 267
  - KDChart::AbstractTernaryDiagram, 314
  - KDChart::BarDiagram, 399
  - KDChart::LineDiagram, 684
  - KDChart::PieDiagram, 784
  - KDChart::Plotter, 839
  - KDChart::PolarDiagram, 934
  - KDChart::RingDiagram, 1029
  - KDChart::TernaryLineDiagram, 1140
  - KDChart::TernaryPointDiagram, 1185
- attributesModelRootIndex
  - KDChart::AbstractCartesianDiagram, 94
  - KDChart::AbstractDiagram, 174
  - KDChart::AbstractPieDiagram, 220
  - KDChart::AbstractPolarDiagram, 268
  - KDChart::AbstractTernaryDiagram, 314
  - KDChart::BarDiagram, 400
  - KDChart::LineDiagram, 684
  - KDChart::PieDiagram, 785
  - KDChart::Plotter, 840
  - KDChart::PolarDiagram, 934
  - KDChart::RingDiagram, 1030
  - KDChart::TernaryLineDiagram, 1141
  - KDChart::TernaryPointDiagram, 1186
- autoAdjustGridToZoom
  - KDChart::CartesianCoordinatePlane, 498
- autoAdjustHorizontalRangeToData
  - KDChart::CartesianCoordinatePlane, 498
- autoAdjustVerticalRangeToData
  - KDChart::CartesianCoordinatePlane, 498
- autoReferenceArea
  - KDChart::HeaderFooter, 615
  - KDChart::TextArea, 1226
  - KDChart::TextLayoutItem, 1252
- autoRotate
  - KDChart::TextAttributes, 1242
- autoShrink
  - KDChart::TextAttributes, 1242
- AutoSpacerLayoutItem
  - KDChart::AutoSpacerLayoutItem, 372
- axes
  - KDChart::AbstractCartesianDiagram, 94
  - KDChart::BarDiagram, 400
  - KDChart::LineDiagram, 685
  - KDChart::Plotter, 840
- AxesCalcMode
  - KDChart::AbstractCoordinatePlane, 139

- KDChart::CartesianCoordinatePlane, 494
  - KDChart::PolarCoordinatePlane, 893
  - KDChart::TernaryCoordinatePlane, 1104
- axesCalcModeX
  - KDChart::CartesianCoordinatePlane, 499
- axesCalcModeY
  - KDChart::CartesianCoordinatePlane, 499
- AxisVector\_B\_A
  - TernaryConstants.cpp, 1541
  - TernaryConstants.h, 1542
- AxisVector\_B\_C
  - TernaryConstants.cpp, 1541
  - TernaryConstants.h, 1542
- AxisVector\_C\_A
  - TernaryConstants.cpp, 1541
  - TernaryConstants.h, 1542
- b
  - TernaryPoint, 1343
- BackgroundAttributes
  - KDChart::BackgroundAttributes, 379
- backgroundAttributes
  - KDChart::AbstractArea, 38
  - KDChart::AbstractAreaBase, 49
  - KDChart::AbstractAreaWidget, 57
  - KDChart::AbstractAxis, 70
  - KDChart::AbstractCoordinatePlane, 141
  - KDChart::CartesianAxis, 452
  - KDChart::CartesianCoordinatePlane, 499
  - KDChart::Chart, 548
  - KDChart::DataValueAttributes, 583
  - KDChart::HeaderFooter, 615
  - KDChart::Legend, 643
  - KDChart::PolarCoordinatePlane, 894
  - KDChart::TernaryAxis, 1079
  - KDChart::TernaryCoordinatePlane, 1105
  - KDChart::TextArea, 1226
- BackgroundPixmapMode
  - KDChart::BackgroundAttributes, 378
- BackgroundPixmapModeCentered
  - KDChart::BackgroundAttributes, 378
- BackgroundPixmapModeNone
  - KDChart::BackgroundAttributes, 378
- BackgroundPixmapModeScaled
  - KDChart::BackgroundAttributes, 378
- BackgroundPixmapModeStretched
  - KDChart::BackgroundAttributes, 378
- Bar
  - KDChart::Widget, 1290
- BarAttributes
  - KDChart::BarAttributes, 384
- barAttributes
  - KDChart::BarDiagram, 401
- BarAttributesRole
  - KDChart, 33
- BarDiagram
  - KDChart::BarDiagram, 398
- barDiagram
  - KDChart::Widget, 1293
- barGapFactor
  - KDChart::BarAttributes, 384
- BarType
  - KDChart::BarDiagram, 398
- Bottom
  - KDChart::CartesianAxis, 451
- bottomOverlap
  - KDChart::AbstractArea, 38
  - KDChart::AbstractAxis, 70
  - KDChart::AbstractCoordinatePlane, 141
  - KDChart::CartesianAxis, 453
  - KDChart::CartesianCoordinatePlane, 500
  - KDChart::PolarCoordinatePlane, 895
  - KDChart::TernaryAxis, 1079
  - KDChart::TernaryCoordinatePlane, 1106
- brush
  - KDChart::AbstractCartesianDiagram, 95, 96
  - KDChart::AbstractDiagram, 174, 175
  - KDChart::AbstractPieDiagram, 221, 222
  - KDChart::AbstractPolarDiagram, 268, 269
  - KDChart::AbstractTernaryDiagram, 315, 316
  - KDChart::BackgroundAttributes, 379
  - KDChart::BarDiagram, 402, 403
  - KDChart::Legend, 643
  - KDChart::LineDiagram, 685, 686
  - KDChart::PieDiagram, 785, 786
  - KDChart::Plotter, 841, 842
  - KDChart::PolarDiagram, 935, 936
  - KDChart::RingDiagram, 1030, 1031
  - KDChart::TernaryLineDiagram, 1141, 1142
  - KDChart::TernaryPointDiagram, 1186, 1187
  - PrerenderedLabel, 1327
- brushes
  - KDChart::Legend, 644
- buddy
  - KDChart::DatasetProxyModel, 573
- buildReferenceRect
  - KDChartPieDiagram.cpp, 1477
- c
  - TernaryPoint, 1343
- calcMode
  - KDChart::DataDimension, 569
- calculateDataBoundaries
  - KDChart::AbstractCartesianDiagram, 96
  - KDChart::AbstractDiagram, 176
  - KDChart::AbstractPieDiagram, 222
  - KDChart::AbstractPolarDiagram, 270
  - KDChart::AbstractTernaryDiagram, 316

- KDChart::BarDiagram, 403
- KDChart::LineDiagram, 686
- KDChart::PieDiagram, 787
- KDChart::Plotter, 842
- KDChart::PolarDiagram, 936
- KDChart::RingDiagram, 1032
- KDChart::TernaryLineDiagram, 1143
- KDChart::TernaryPointDiagram, 1188
- calculatedFont
  - KDChart::TextAttributes, 1243
- calculatedFontSize
  - KDChart::TextAttributes, 1243
- calculatedPoint
  - KDChart::RelativePosition, 1010
- calculatedValue
  - KDChart::Measure, 759, 760
- calculateNextLabel
  - KDChartCartesianAxis.cpp, 1397
- calculateOverlap
  - KDChartCartesianAxis.cpp, 1397
- calculateRawDataBoundingRect
  - KDChart::CartesianCoordinatePlane, 500
- calculationMode
  - KDChart::Measure, 761
- CartesianAxis
  - KDChart::CartesianAxis, 451
- CartesianAxisList
  - KDChart, 32
- CartesianCoordinatePlane
  - KDChart::CartesianCoordinatePlane, 494
- Center
  - KDChart::Position, 983
- center
  - KDChart::ZoomParameters, 1306
- changed
  - KDChart::Palette, 770
- Chart
  - KDChart::Chart, 546
- ChartGraphicsItem
  - KDChart::ChartGraphicsItem, 566
- ChartGraphicsItem.cpp, 1345
- ChartGraphicsItem.h, 1346
- ChartType
  - KDChart::Widget, 1290
- checkInvariants
  - KDChart::AbstractCartesianDiagram, 96
  - KDChart::AbstractDiagram, 176
  - KDChart::AbstractPieDiagram, 222
  - KDChart::AbstractPolarDiagram, 270
  - KDChart::AbstractTernaryDiagram, 316
  - KDChart::BarDiagram, 403
  - KDChart::LineDiagram, 687
  - KDChart::PieDiagram, 787
  - KDChart::Plotter, 842
  - KDChart::PolarDiagram, 937
  - KDChart::RingDiagram, 1032
  - KDChart::TernaryLineDiagram, 1143
  - KDChart::TernaryPointDiagram, 1188
- clear
  - KDChart::ReverseMapper, 1019
- clone
  - KDChart::BarDiagram, 404
  - KDChart::HeaderFooter, 615
  - KDChart::Legend, 644
  - KDChart::LineDiagram, 687
  - KDChart::PieDiagram, 788
  - KDChart::Plotter, 843
  - KDChart::PolarDiagram, 937
  - KDChart::RingDiagram, 1032
- closeDatasets
  - KDChart::PolarDiagram, 938
- column
  - KDChart::ChartGraphicsItem, 566
- columnCount
  - KDChart::AbstractPieDiagram, 223
  - KDChart::AbstractPolarDiagram, 270
  - KDChart::AttributesModel, 356
  - KDChart::PieDiagram, 788
  - KDChart::PolarDiagram, 938
  - KDChart::PrivateAttributesModel, 993
  - KDChart::RingDiagram, 1033
- columnToIndex
  - KDChart::AbstractCartesianDiagram, 97
  - KDChart::AbstractDiagram, 176
  - KDChart::AbstractPieDiagram, 223
  - KDChart::AbstractPolarDiagram, 271
  - KDChart::AbstractTernaryDiagram, 317
  - KDChart::BarDiagram, 404
  - KDChart::LineDiagram, 687
  - KDChart::PieDiagram, 788
  - KDChart::Plotter, 843
  - KDChart::PolarDiagram, 938
  - KDChart::RingDiagram, 1033
  - KDChart::TernaryLineDiagram, 1144
  - KDChart::TernaryPointDiagram, 1188
- compare
  - KDChart::AbstractArea, 39
  - KDChart::AbstractAreaBase, 50
  - KDChart::AbstractAreaWidget, 57
  - KDChart::AbstractAxis, 70, 71
  - KDChart::AbstractCartesianDiagram, 97, 99
  - KDChart::AbstractCoordinatePlane, 141
  - KDChart::AbstractDiagram, 176
  - KDChart::AbstractPieDiagram, 223
  - KDChart::AbstractPolarDiagram, 271
  - KDChart::AbstractTernaryDiagram, 317
  - KDChart::AttributesModel, 356
  - KDChart::BarDiagram, 404, 406, 407



- KDChart::CartesianAxis, [453](#), [454](#)
- KDChart::CartesianCoordinatePlane, [501](#)
- KDChart::HeaderFooter, [615](#), [616](#)
- KDChart::Legend, [644](#), [645](#)
- KDChart::LineDiagram, [688–690](#)
- KDChart::PieDiagram, [788](#)
- KDChart::Plotter, [843](#), [845](#), [846](#)
- KDChart::PolarCoordinatePlane, [895](#)
- KDChart::PolarDiagram, [938](#)
- KDChart::PrivateAttributesModel, [993](#)
- KDChart::RingDiagram, [1033](#)
- KDChart::TernaryAxis, [1080](#)
- KDChart::TernaryCoordinatePlane, [1106](#)
- KDChart::TernaryLineDiagram, [1144](#)
- KDChart::TernaryPointDiagram, [1189](#)
- KDChart::TextArea, [1227](#)
- compareAttributes
  - KDChart::AttributesModel, [359](#)
  - KDChart::PrivateAttributesModel, [995](#)
- configureDatasetProxyModel
  - KDChart::DatasetSelectorWidget, [580](#)
- connectSignals
  - KDChart::AbstractAxis, [71](#)
  - KDChart::CartesianAxis, [454](#)
  - KDChart::TernaryAxis, [1081](#)
- ConstAbstractDiagramList
  - KDChart, [32](#)
- ConstDiagramList
  - KDChart, [32](#)
- constDiagrams
  - KDChart::Legend, [646](#)
- coordinatePlane
  - KDChart::AbstractAxis, [72](#)
  - KDChart::AbstractCartesianDiagram, [99](#)
  - KDChart::AbstractDiagram, [178](#)
  - KDChart::AbstractPieDiagram, [225](#)
  - KDChart::AbstractPolarDiagram, [273](#)
  - KDChart::AbstractTernaryDiagram, [319](#)
  - KDChart::BarDiagram, [407](#)
  - KDChart::CartesianAxis, [455](#)
  - KDChart::Chart, [549](#)
  - KDChart::LineDiagram, [690](#)
  - KDChart::PaintContext, [766](#)
  - KDChart::PieDiagram, [790](#)
  - KDChart::Plotter, [846](#)
  - KDChart::PolarDiagram, [940](#)
  - KDChart::RingDiagram, [1035](#)
  - KDChart::TernaryAxis, [1081](#)
  - KDChart::TernaryLineDiagram, [1146](#)
  - KDChart::TernaryPointDiagram, [1190](#)
  - KDChart::Widget, [1293](#)
- coordinatePlaneLayout
  - KDChart::Chart, [549](#)
- CoordinatePlaneList
  - KDChart, [32](#)
- coordinatePlanes
  - KDChart::Chart, [549](#)
- CoordinateTransformationList
  - KDChart::PolarCoordinatePlane, [893](#)
- createObserver
  - KDChart::AbstractAxis, [72](#)
  - KDChart::CartesianAxis, [455](#)
  - KDChart::TernaryAxis, [1081](#)
- currentFactors
  - KDChart::GlobalMeasureScaling, [601](#)
- customizedLabel
  - KDChart::AbstractAxis, [72](#)
  - KDChart::CartesianAxis, [455](#)
  - KDChart::TernaryAxis, [1082](#)
- d
  - KDChartAbstractArea.cpp, [1347](#)
  - KDChartAbstractAreaBase.cpp, [1354](#)
  - KDChartAbstractAreaWidget.cpp, [1357](#)
  - KDChartAbstractAxis.cpp, [1359](#)
  - KDChartAbstractCartesianDiagram.cpp, [1361](#)
  - KDChartAbstractCoordinatePlane.cpp, [1364](#)
  - KDChartAbstractDiagram.cpp, [1368](#)
  - KDChartAbstractPieDiagram.cpp, [1371](#)
  - KDChartAbstractPolarDiagram.cpp, [1373](#)
  - KDChartAbstractTernaryDiagram.cpp, [1377](#)
  - KDChartAbstractThreeDAttributes.cpp, [1379](#)
  - KDChartBackgroundAttributes.cpp, [1386](#)
  - KDChartBarAttributes.cpp, [1389](#)
  - KDChartBarDiagram.cpp, [1392](#)
  - KDChartCartesianAxis.cpp, [1396](#)
  - KDChartCartesianCoordinatePlane.cpp, [1401](#)
  - KDChartChart.cpp, [1406](#)
  - KDChartDataValueAttributes.cpp, [1414](#)
  - KDChartFrameAttributes.cpp, [1422](#)
  - KDChartGridAttributes.cpp, [1432](#)
  - KDChartHeaderFooter.cpp, [1436](#)
  - KDChartLegend.cpp, [1445](#)
  - KDChartLineAttributes.cpp, [1447](#)
  - KDChartLineDiagram.cpp, [1451](#)
  - KDChartMarkerAttributes.cpp, [1454](#)
  - KDChartPaintContext.cpp, [1465](#)
  - KDChartPalette.cpp, [1467](#)
  - KDChartPieAttributes.cpp, [1473](#)
  - KDChartPieDiagram.cpp, [1477](#)
  - KDChartPlotter.cpp, [1479](#)
  - KDChartPolarCoordinatePlane.cpp, [1483](#)
  - KDChartPolarDiagram.cpp, [1485](#)
  - KDChartRelativePosition.cpp, [1492](#)
  - KDChartRingDiagram.cpp, [1496](#)
  - KDChartTernaryCoordinatePlane.cpp, [1504](#)
  - KDChartTernaryLineDiagram.cpp, [1506](#)
  - KDChartTernaryPointDiagram.cpp, [1508](#)

- KDChartTextAttributes.cpp, 1512
- KDChartThreeDBarAttributes.cpp, 1519
- KDChartThreeDLineAttributes.cpp, 1522
- KDChartThreeDPieAttributes.cpp, 1525
- KDChartValueTrackerAttributes.cpp, 1528
- KDChartWidget.cpp, 1532
- data
  - KDChart::AttributesModel, 360, 361
  - KDChart::DatasetProxyModel, 573
  - KDChart::PrivateAttributesModel, 996–998
- dataBoundaries
  - KDChart::AbstractCartesianDiagram, 100
  - KDChart::AbstractDiagram, 178
  - KDChart::AbstractPieDiagram, 225
  - KDChart::AbstractPolarDiagram, 273
  - KDChart::AbstractTernaryDiagram, 319
  - KDChart::BarDiagram, 407
  - KDChart::LineDiagram, 691
  - KDChart::PieDiagram, 790
  - KDChart::Plotter, 846
  - KDChart::PolarDiagram, 940
  - KDChart::RingDiagram, 1035
  - KDChart::TernaryLineDiagram, 1146
  - KDChart::TernaryPointDiagram, 1191
- dataChanged
  - KDChart::AbstractCartesianDiagram, 100
  - KDChart::AbstractDiagram, 179
  - KDChart::AbstractPieDiagram, 226
  - KDChart::AbstractPolarDiagram, 274
  - KDChart::AbstractTernaryDiagram, 320
  - KDChart::BarDiagram, 408
  - KDChart::LineDiagram, 691
  - KDChart::PieDiagram, 791
  - KDChart::Plotter, 847
  - KDChart::PolarDiagram, 941
  - KDChart::RingDiagram, 1036
  - KDChart::TernaryLineDiagram, 1146
  - KDChart::TernaryPointDiagram, 1191
- DataDimension
  - KDChart::DataDimension, 568
- DataDimensionsList
  - KDChart, 32
- dataHidden
  - KDChart::AbstractCartesianDiagram, 100
  - KDChart::AbstractDiagram, 179
  - KDChart::AbstractPieDiagram, 226
  - KDChart::AbstractPolarDiagram, 274
  - KDChart::AbstractTernaryDiagram, 320
  - KDChart::BarDiagram, 408
  - KDChart::LineDiagram, 691
  - KDChart::PieDiagram, 791
  - KDChart::Plotter, 847
  - KDChart::PolarDiagram, 941
  - KDChart::RingDiagram, 1036
  - KDChart::TernaryLineDiagram, 1147
  - KDChart::TernaryPointDiagram, 1192
- DataHiddenRole
  - KDChart, 33
- dataLabel
  - KDChart::DataValueAttributes, 583
- dataMap
  - KDChart::AttributesModel, 361
  - KDChart::PrivateAttributesModel, 998
- datasetBrushes
  - KDChart::AbstractCartesianDiagram, 100
  - KDChart::AbstractDiagram, 179
  - KDChart::AbstractPieDiagram, 226
  - KDChart::AbstractPolarDiagram, 274
  - KDChart::AbstractTernaryDiagram, 320
  - KDChart::BarDiagram, 408
  - KDChart::LineDiagram, 692
  - KDChart::PieDiagram, 791
  - KDChart::Plotter, 847
  - KDChart::PolarDiagram, 941
  - KDChart::RingDiagram, 1036
  - KDChart::TernaryLineDiagram, 1147
  - KDChart::TernaryPointDiagram, 1192
- DatasetBrushRole
  - KDChart, 33
- datasetCount
  - KDChart::Legend, 646
- DatasetDescriptionVector
  - KDChart, 32
- datasetDimension
  - KDChart::AbstractCartesianDiagram, 101
  - KDChart::AbstractDiagram, 180
  - KDChart::AbstractPieDiagram, 227
  - KDChart::AbstractPolarDiagram, 275
  - KDChart::AbstractTernaryDiagram, 321
  - KDChart::BarDiagram, 409
  - KDChart::LineDiagram, 692
  - KDChart::PieDiagram, 792
  - KDChart::Plotter, 848
  - KDChart::PolarDiagram, 942
  - KDChart::RingDiagram, 1037
  - KDChart::TernaryLineDiagram, 1147
  - KDChart::TernaryPointDiagram, 1192
- datasetLabels
  - KDChart::AbstractCartesianDiagram, 102
  - KDChart::AbstractDiagram, 180
  - KDChart::AbstractPieDiagram, 228
  - KDChart::AbstractPolarDiagram, 275
  - KDChart::AbstractTernaryDiagram, 321
  - KDChart::BarDiagram, 409
  - KDChart::LineDiagram, 693
  - KDChart::PieDiagram, 792
  - KDChart::Plotter, 848
  - KDChart::PolarDiagram, 942

- KDChart::RingDiagram, 1037
- KDChart::TernaryLineDiagram, 1148
- KDChart::TernaryPointDiagram, 1193
- datasetMarkers
  - KDChart::AbstractCartesianDiagram, 102
  - KDChart::AbstractDiagram, 181
  - KDChart::AbstractPieDiagram, 228
  - KDChart::AbstractPolarDiagram, 276
  - KDChart::AbstractTernaryDiagram, 322
  - KDChart::BarDiagram, 410
  - KDChart::LineDiagram, 693
  - KDChart::PieDiagram, 793
  - KDChart::Plotter, 849
  - KDChart::PolarDiagram, 943
  - KDChart::RingDiagram, 1038
  - KDChart::TernaryLineDiagram, 1148
  - KDChart::TernaryPointDiagram, 1193
- DatasetPenRole
  - KDChart, 33
- datasetPens
  - KDChart::AbstractCartesianDiagram, 103
  - KDChart::AbstractDiagram, 181
  - KDChart::AbstractPieDiagram, 229
  - KDChart::AbstractPolarDiagram, 276
  - KDChart::AbstractTernaryDiagram, 322
  - KDChart::BarDiagram, 410
  - KDChart::LineDiagram, 694
  - KDChart::PieDiagram, 793
  - KDChart::Plotter, 849
  - KDChart::PolarDiagram, 943
  - KDChart::RingDiagram, 1038
  - KDChart::TernaryLineDiagram, 1149
  - KDChart::TernaryPointDiagram, 1194
- DatasetProxyModel
  - KDChart::DatasetProxyModel, 572
- DatasetSelectorWidget
  - KDChart::DatasetSelectorWidget, 579
- DataValueAttributes
  - KDChart::DataValueAttributes, 583
- dataValueAttributes
  - KDChart::AbstractCartesianDiagram, 103, 104
  - KDChart::AbstractDiagram, 182, 183
  - KDChart::AbstractPieDiagram, 229, 230
  - KDChart::AbstractPolarDiagram, 277, 278
  - KDChart::AbstractTernaryDiagram, 323, 324
  - KDChart::BarDiagram, 411, 412
  - KDChart::LineDiagram, 694, 695
  - KDChart::PieDiagram, 794, 795
  - KDChart::Plotter, 850, 851
  - KDChart::PolarDiagram, 944, 945
  - KDChart::RingDiagram, 1039, 1040
  - KDChart::TernaryLineDiagram, 1150
  - KDChart::TernaryPointDiagram, 1194, 1195
- DataValueLabelAttributesRole
  - KDChart, 33
- decimalDigits
  - KDChart::DataValueAttributes, 584
- defaultAttributes
  - KDChart::DataValueAttributes, 584
- defaultAttributesAsVariant
  - KDChart::DataValueAttributes, 584
- defaultPalette
  - KDChart::Palette, 770
- delayedInit
  - KDChart::AbstractAxis, 73
  - KDChart::CartesianAxis, 456
  - KDChart::TernaryAxis, 1082
- deleteObserver
  - KDChart::AbstractAxis, 73
  - KDChart::CartesianAxis, 456
  - KDChart::TernaryAxis, 1082
- depth
  - KDChart::AbstractThreeDAttributes, 351
  - KDChart::ThreeDBarAttributes, 1262
  - KDChart::ThreeDLineAttributes, 1267
  - KDChart::ThreeDPieAttributes, 1272
- destroyedCoordinatePlane
  - KDChart::AbstractCoordinatePlane, 142
  - KDChart::CartesianCoordinatePlane, 501
  - KDChart::PolarCoordinatePlane, 896
  - KDChart::TernaryCoordinatePlane, 1106
- destroyedHeaderFooter
  - KDChart::HeaderFooter, 616
- destroyedLegend
  - KDChart::Legend, 646
- diagram
  - KDChart::AbstractAxis, 73
  - KDChart::AbstractCoordinatePlane, 142
  - KDChart::CartesianAxis, 456
  - KDChart::CartesianCoordinatePlane, 501
  - KDChart::DiagramObserver, 594
  - KDChart::Legend, 646
  - KDChart::PolarCoordinatePlane, 896
  - KDChart::TernaryAxis, 1082
  - KDChart::TernaryCoordinatePlane, 1107
  - KDChart::Widget, 1293
- diagramAttributesChanged
  - KDChart::DiagramObserver, 594
- diagramDataChanged
  - KDChart::DiagramObserver, 594
- diagramDataHidden
  - KDChart::DiagramObserver, 595
- diagramDestroyed
  - KDChart::DiagramObserver, 595
- DiagramList
  - KDChart, 32
- DiagramObserver

- KDChart::DiagramObserver, 594
- diagrams
  - KDChart::AbstractCoordinatePlane, 142, 143
  - KDChart::CartesianCoordinatePlane, 502
  - KDChart::Legend, 647
  - KDChart::PolarCoordinatePlane, 896, 897
  - KDChart::TernaryCoordinatePlane, 1107
- displayArea
  - KDChart::LineAttributes, 672
- DisplayRoles
  - KDChart, 33
- distance
  - KDChart::DataDimension, 568
- doesIsometricScaling
  - KDChart::CartesianCoordinatePlane, 503
- doItemsLayout
  - KDChart::AbstractCartesianDiagram, 105
  - KDChart::AbstractDiagram, 183
  - KDChart::AbstractPieDiagram, 230
  - KDChart::AbstractPolarDiagram, 278
  - KDChart::AbstractTernaryDiagram, 324
  - KDChart::BarDiagram, 412
  - KDChart::LineDiagram, 696
  - KDChart::PieDiagram, 795
  - KDChart::Plotter, 851
  - KDChart::PolarDiagram, 945
  - KDChart::RingDiagram, 1040
  - KDChart::TernaryLineDiagram, 1151
  - KDChart::TernaryPointDiagram, 1196
- doneSetZoomCenter
  - KDChart::CartesianCoordinatePlane, 503
- doneSetZoomFactorX
  - KDChart::CartesianCoordinatePlane, 503
- doneSetZoomFactorY
  - KDChart::CartesianCoordinatePlane, 503
- drawingArea
  - KDChart::CartesianCoordinatePlane, 504
- drawSolidExcessArrows
  - KDChart::BarAttributes, 384
- DUMP\_CACHE\_STATS
  - KDChartTextLabelCache.cpp, 1516
- East
  - KDChart::Position, 983
- emitSignal
  - KDChart::SignalCompressor, 1074
- end
  - KDChart::DataDimension, 569
- ExcludeCenter
  - KDChart::Position, 978
- expandingDirections
  - KDChart::AbstractCoordinatePlane, 143
  - KDChart::AutoSpacerLayoutItem, 372
  - KDChart::CartesianAxis, 457
  - KDChart::CartesianCoordinatePlane, 504
  - KDChart::HeaderFooter, 616
  - KDChart::HorizontalLineLayoutItem, 632
  - KDChart::LineLayoutItem, 734
  - KDChart::LineWithMarkerLayoutItem, 740
  - KDChart::MarkerLayoutItem, 752
  - KDChart::PolarCoordinatePlane, 897
  - KDChart::TernaryAxis, 1083
  - KDChart::TernaryCoordinatePlane, 1108
  - KDChart::TextArea, 1227
  - KDChart::TextLayoutItem, 1253
  - KDChart::VerticalLineLayoutItem, 1282
- explode
  - KDChart::PieAttributes, 774
- explodeFactor
  - KDChart::PieAttributes, 774
- filterAcceptsColumn
  - KDChart::DatasetProxyModel, 573
- filterAcceptsRow
  - KDChart::DatasetProxyModel, 573
- finallyEmit
  - KDChart::SignalCompressor, 1074
- findOrCreateLayoutByObjectName
  - KDChartChart.cpp, 1406
- firstHeaderFooter
  - KDChart::Widget, 1293
- fixedBarWidth
  - KDChart::BarAttributes, 384
- fixedDataValueGap
  - KDChart::BarAttributes, 385
- fixedValueBlockGap
  - KDChart::BarAttributes, 385
- flags
  - KDChart::DatasetProxyModel, 574
- Floating
  - KDChart::Position, 983
- floatingPosition
  - KDChart::Legend, 647
- font
  - KDChart::TextAttributes, 1243
  - PrerenderedLabel, 1328
- fontSize
  - KDChart::TextAttributes, 1244
- Footer
  - KDChart::HeaderFooter, 614
- forceRebuild
  - KDChart::AbstractAreaWidget, 58
  - KDChart::Legend, 647
- FrameAttributes
  - KDChart::FrameAttributes, 596
- frameAttributes
  - KDChart::AbstractArea, 39
  - KDChart::AbstractAreaBase, 50

- KDChart::AbstractAreaWidget, 58
- KDChart::AbstractAxis, 73
- KDChart::AbstractCoordinatePlane, 143
- KDChart::CartesianAxis, 457
- KDChart::CartesianCoordinatePlane, 504
- KDChart::Chart, 549
- KDChart::DataValueAttributes, 584
- KDChart::HeaderFooter, 616
- KDChart::Legend, 648
- KDChart::PolarCoordinatePlane, 897
- KDChart::TernaryAxis, 1083
- KDChart::TernaryCoordinatePlane, 1108
- KDChart::TextArea, 1227
- fromName
  - KDChart::Position, 979
- FullMarkerDistanceAC
  - TernaryConstants.cpp, 1541
  - TernaryConstants.h, 1542
- FullMarkerDistanceBA
  - TernaryConstants.cpp, 1541
  - TernaryConstants.h, 1543
- FullMarkerDistanceBC
  - TernaryConstants.cpp, 1541
  - TernaryConstants.h, 1543
- geometry
  - KDChart::AbstractAxis, 74
  - KDChart::AbstractCoordinatePlane, 143
  - KDChart::AutoSpacerLayoutItem, 372
  - KDChart::CartesianAxis, 457
  - KDChart::CartesianCoordinatePlane, 504
  - KDChart::HeaderFooter, 617
  - KDChart::HorizontalLineLayoutItem, 632
  - KDChart::LineLayoutItem, 734
  - KDChart::LineWithMarkerLayoutItem, 740
  - KDChart::MarkerLayoutItem, 752
  - KDChart::PolarCoordinatePlane, 897
  - KDChart::TernaryAxis, 1083
  - KDChart::TernaryCoordinatePlane, 1108
  - KDChart::TextArea, 1227
  - KDChart::TextLayoutItem, 1253
  - KDChart::VerticalLineLayoutItem, 1282
- getBrush
  - KDChart::Palette, 770
- getCellValues
  - KDChart::LineDiagram, 696
- getDataDimensionsList
  - KDChart::AbstractCoordinatePlane, 144
  - KDChart::CartesianCoordinatePlane, 505
  - KDChart::PolarCoordinatePlane, 898
  - KDChart::TernaryCoordinatePlane, 1109
- getFrameLeadings
  - KDChart::AbstractArea, 39
  - KDChart::AbstractAreaBase, 50
  - KDChart::AbstractAreaWidget, 58
  - KDChart::AbstractAxis, 74
  - KDChart::AbstractCoordinatePlane, 144
  - KDChart::CartesianAxis, 458
  - KDChart::CartesianCoordinatePlane, 506
  - KDChart::HeaderFooter, 617
  - KDChart::Legend, 648
  - KDChart::PolarCoordinatePlane, 898
  - KDChart::TernaryAxis, 1083
  - KDChart::TernaryCoordinatePlane, 1109
  - KDChart::TextArea, 1228
- getRawDataBoundingRectFromDiagrams
  - KDChart::CartesianCoordinatePlane, 506
- globalGridAttributes
  - KDChart::AbstractCoordinatePlane, 144
  - KDChart::CartesianCoordinatePlane, 507
  - KDChart::PolarCoordinatePlane, 898
  - KDChart::TernaryCoordinatePlane, 1109
- globalLeadingBottom
  - KDChart::Chart, 550, 564
  - KDChart::Widget, 1294
- globalLeadingLeft
  - KDChart::Chart, 550, 564
  - KDChart::Widget, 1294
- globalLeadingRight
  - KDChart::Chart, 550, 564
  - KDChart::Widget, 1294
- globalLeadingTop
  - KDChart::Chart, 550, 564
  - KDChart::Widget, 1294
- GlobalMeasureScaling
  - KDChart::GlobalMeasureScaling, 601
- granularity
  - KDChart::AbstractPieDiagram, 231
  - KDChart::PieDiagram, 796
  - KDChart::RingDiagram, 1040
- GranularitySequence
  - KDChartEnums, 1310
- GranularitySequence\_10\_20
  - KDChartEnums, 1311
- GranularitySequence\_10\_50
  - KDChartEnums, 1311
- GranularitySequence\_125\_25
  - KDChartEnums, 1311
- GranularitySequence\_25\_50
  - KDChartEnums, 1311
- GranularitySequenceIrregular
  - KDChartEnums, 1311
- granularitySequenceToString
  - KDChartEnums, 1314
- GridAttributes
  - KDChart::GridAttributes, 604
- gridAttributes
  - KDChart::CartesianCoordinatePlane, 507



- KDChart::PolarCoordinatePlane, 899
- gridDimensionsList
  - KDChart::AbstractCoordinatePlane, 145
  - KDChart::CartesianCoordinatePlane, 508
  - KDChart::PolarCoordinatePlane, 899
  - KDChart::TernaryCoordinatePlane, 1110
- gridGranularitySequence
  - KDChart::GridAttributes, 604
- gridPen
  - KDChart::GridAttributes, 605
- gridStepWidth
  - KDChart::GridAttributes, 605
- gridSubStepWidth
  - KDChart::GridAttributes, 605
- groupGapFactor
  - KDChart::BarAttributes, 385
- handleFixedDataCoordinateSpaceRelation
  - KDChart::CartesianCoordinatePlane, 508
- hasAbsoluteFontSize
  - KDChart::TextAttributes, 1244
- hasDefaultTitleTextAttributes
  - KDChart::CartesianAxis, 458
  - KDChart::TernaryAxis, 1084
- hasFixedDataCoordinateSpaceRelation
  - KDChart::CartesianCoordinatePlane, 509
- hasOwnGridAttributes
  - KDChart::CartesianCoordinatePlane, 509
  - KDChart::PolarCoordinatePlane, 900
- Header
  - KDChart::HeaderFooter, 614
- headerData
  - KDChart::AttributesModel, 362
  - KDChart::DatasetProxyModel, 574
  - KDChart::PrivateAttributesModel, 998
- HeaderFooter
  - KDChart::HeaderFooter, 614
- headerFooter
  - KDChart::Chart, 551
- HeaderFooterList
  - KDChart, 33
- headerFooters
  - KDChart::Chart, 551
- HeaderFooterType
  - KDChart::HeaderFooter, 614
- HitCount
  - KDChartTextLabelCache.cpp, 1517
- horizontalHeaderDataMap
  - KDChart::AttributesModel, 363
  - KDChart::PrivateAttributesModel, 999
- HorizontalLineLayoutItem
  - KDChart::HorizontalLineLayoutItem, 632
- horizontalOffset
  - KDChart::AbstractCartesianDiagram, 105
  - KDChart::AbstractDiagram, 183
  - KDChart::AbstractPieDiagram, 231
  - KDChart::AbstractPolarDiagram, 278
  - KDChart::AbstractTernaryDiagram, 324
  - KDChart::BarDiagram, 413
  - KDChart::LineDiagram, 696
  - KDChart::PieDiagram, 796
  - KDChart::Plotter, 851
  - KDChart::PolarDiagram, 946
  - KDChart::RingDiagram, 1041
  - KDChart::TernaryLineDiagram, 1151
  - KDChart::TernaryPointDiagram, 1196
- horizontalPadding
  - KDChart::RelativePosition, 1010
- horizontalRange
  - KDChart::CartesianCoordinatePlane, 510
- INC\_HIT\_COUNT
  - KDChartTextLabelCache.cpp, 1517
- INC\_MISS\_COUNT
  - KDChartTextLabelCache.cpp, 1517
- IncludeCenter
  - KDChart::Position, 978
- index
  - KDChart::AbstractProxyModel, 306
  - KDChart::AttributesModel, 363
  - KDChart::DatasetProxyModel, 575
  - KDChart::PrivateAttributesModel, 999
- indexAt
  - KDChart::AbstractCartesianDiagram, 105
  - KDChart::AbstractDiagram, 184
  - KDChart::AbstractPieDiagram, 231
  - KDChart::AbstractPolarDiagram, 278
  - KDChart::AbstractTernaryDiagram, 325
  - KDChart::BarDiagram, 413
  - KDChart::LineDiagram, 697
  - KDChart::PieDiagram, 796
  - KDChart::Plotter, 852
  - KDChart::PolarDiagram, 946
  - KDChart::RingDiagram, 1041
  - KDChart::TernaryLineDiagram, 1151
  - KDChart::TernaryPointDiagram, 1196
- indexesAt
  - KDChart::AbstractCartesianDiagram, 105
  - KDChart::AbstractDiagram, 184
  - KDChart::AbstractPieDiagram, 231
  - KDChart::AbstractPolarDiagram, 279
  - KDChart::AbstractTernaryDiagram, 325
  - KDChart::BarDiagram, 413
  - KDChart::LineDiagram, 697
  - KDChart::PieDiagram, 796
  - KDChart::Plotter, 852
  - KDChart::PolarDiagram, 946
  - KDChart::ReverseMapper, 1019

- KDChart::RingDiagram, 1041
- KDChart::TernaryLineDiagram, 1151
- KDChart::TernaryPointDiagram, 1196
- indexesIn
  - KDChart::ReverseMapper, 1020
- initFrom
  - KDChart::AttributesModel, 363
  - KDChart::PrivateAttributesModel, 1000
- innerRect
  - KDChart::AbstractArea, 40
  - KDChart::AbstractAreaBase, 51
  - KDChart::AbstractAreaWidget, 59
  - KDChart::AbstractAxis, 74
  - KDChart::AbstractCoordinatePlane, 145
  - KDChart::CartesianAxis, 458
  - KDChart::CartesianCoordinatePlane, 510
  - KDChart::HeaderFooter, 617
  - KDChart::Legend, 648
  - KDChart::PolarCoordinatePlane, 901
  - KDChart::TernaryAxis, 1084
  - KDChart::TernaryCoordinatePlane, 1110
  - KDChart::TextArea, 1228
- instance
  - KDChart::GlobalMeasureScaling, 601
- intersects
  - KDChart::HeaderFooter, 618
  - KDChart::TextArea, 1228, 1229
  - KDChart::TextLayoutItem, 1253, 1254
- invalidate
  - PrerenderedElement, 1323
  - PrerenderedLabel, 1328
- isAbscissa
  - KDChart::CartesianAxis, 459
- isCalculated
  - KDChart::DataDimension, 569
- isCartesian
  - KDChartWidget.cpp, 1532
- isCorner
  - KDChart::Position, 979
- isEastSide
  - KDChart::Position, 979
- isEmpty
  - KDChart::AbstractCoordinatePlane, 146
  - KDChart::AutoSpacerLayoutItem, 372
  - KDChart::CartesianAxis, 459
  - KDChart::CartesianCoordinatePlane, 510
  - KDChart::HeaderFooter, 619
  - KDChart::HorizontalLineLayoutItem, 632
  - KDChart::LineLayoutItem, 735
  - KDChart::LineWithMarkerLayoutItem, 741
  - KDChart::MarkerLayoutItem, 753
  - KDChart::PolarCoordinatePlane, 901
  - KDChart::TernaryAxis, 1084
  - KDChart::TernaryCoordinatePlane, 1111
  - KDChart::TextArea, 1229
  - KDChart::TextLayoutItem, 1254
  - KDChart::VerticalLineLayoutItem, 1282
- isEnabled
  - KDChart::AbstractThreeDAttributes, 351
  - KDChart::ThreeDBarAttributes, 1262
  - KDChart::ThreeDLineAttributes, 1267
  - KDChart::ThreeDPieAttributes, 1272
  - KDChart::ValueTrackerAttributes, 1277
- isEqualTo
  - KDChart::BackgroundAttributes, 379
- isFloating
  - KDChart::Position, 979
- isGridVisible
  - KDChart::GridAttributes, 606
- isHidden
  - KDChart::AbstractCartesianDiagram, 105, 106
  - KDChart::AbstractDiagram, 184, 185
  - KDChart::AbstractPieDiagram, 232, 233
  - KDChart::AbstractPolarDiagram, 279, 280
  - KDChart::AbstractTernaryDiagram, 325, 326
  - KDChart::BarDiagram, 413, 414
  - KDChart::LineDiagram, 697, 698
  - KDChart::PieDiagram, 797
  - KDChart::Plotter, 852, 853
  - KDChart::PolarDiagram, 946, 947
  - KDChart::RingDiagram, 1041, 1042
  - KDChart::TernaryLineDiagram, 1152, 1153
  - KDChart::TernaryPointDiagram, 1197, 1198
- isHorizontalRangeReversed
  - KDChart::CartesianCoordinatePlane, 511
- isIndexHidden
  - KDChart::AbstractCartesianDiagram, 107
  - KDChart::AbstractDiagram, 185
  - KDChart::AbstractPieDiagram, 233
  - KDChart::AbstractPolarDiagram, 280
  - KDChart::AbstractTernaryDiagram, 326
  - KDChart::BarDiagram, 415
  - KDChart::LineDiagram, 698
  - KDChart::PieDiagram, 798
  - KDChart::Plotter, 853
  - KDChart::PolarDiagram, 948
  - KDChart::RingDiagram, 1043
  - KDChart::TernaryLineDiagram, 1153
  - KDChart::TernaryPointDiagram, 1198
- isKnownAttributesRole
  - KDChart::AttributesModel, 363
  - KDChart::PrivateAttributesModel, 1000
- isNorthSide
  - KDChart::Position, 980
- isNull
  - KDChart::PositionPoints, 987
- isOrdinate

- KDChart::CartesianAxis, 459
- isPolar
  - KDChartWidget.cpp, 1532
- isPole
  - KDChart::Position, 980
- isRubberBandZoomingEnabled
  - KDChart::AbstractCoordinatePlane, 146
  - KDChart::CartesianCoordinatePlane, 511
  - KDChart::PolarCoordinatePlane, 901
  - KDChart::TernaryCoordinatePlane, 1111
- isSouthSide
  - KDChart::Position, 980
- isSubGridVisible
  - KDChart::GridAttributes, 606
- isUnknown
  - KDChart::Position, 980
- isValid
  - KDChart::Palette, 771
  - TernaryPoint, 1343
- isVerticalRangeReversed
  - KDChart::CartesianCoordinatePlane, 511
- isVisible
  - KDChart::BackgroundAttributes, 380
  - KDChart::DataValueAttributes, 585
  - KDChart::FrameAttributes, 597
  - KDChart::MarkerAttributes, 747
  - KDChart::TextAttributes, 1244
- isVisiblePoint
  - KDChart::AbstractCoordinatePlane, 146
  - KDChart::CartesianCoordinatePlane, 511
  - KDChart::PolarCoordinatePlane, 901
  - KDChart::TernaryCoordinatePlane, 1111
- isWestSide
  - KDChart::Position, 980
- itemRowLabels
  - KDChart::AbstractCartesianDiagram, 107
  - KDChart::AbstractDiagram, 185
  - KDChart::AbstractPieDiagram, 233
  - KDChart::AbstractPolarDiagram, 280
  - KDChart::AbstractTernaryDiagram, 327
  - KDChart::BarDiagram, 415
  - KDChart::LineDiagram, 699
  - KDChart::PieDiagram, 798
  - KDChart::Plotter, 854
  - KDChart::PolarDiagram, 948
  - KDChart::RingDiagram, 1043
  - KDChart::TernaryLineDiagram, 1153
  - KDChart::TernaryPointDiagram, 1198
- KDAB\_SET\_OBJECT\_NAME
  - KDChartGlobal.h, 1428
- KDChart, 27
  - AbstractDiagramList, 32
  - BarAttributesRole, 33
  - CartesianAxisList, 32
  - ConstAbstractDiagramList, 32
  - ConstDiagramList, 32
  - CoordinatePlaneList, 32
  - DataDimensionsList, 32
  - DataHiddenRole, 33
  - DatasetBrushRole, 33
  - DatasetDescriptionVector, 32
  - DatasetPenRole, 33
  - DataValueLabelAttributesRole, 33
  - DiagramList, 32
  - DisplayRoles, 33
  - HeaderFooterList, 33
  - LegendList, 33
  - LineAttributesRole, 33
  - operator<<, 34
  - PieAttributesRole, 33
  - TernaryAxisList, 33
  - ThreeDAttributesRole, 33
  - ThreeDBarAttributesRole, 33
  - ThreeDLineAttributesRole, 33
  - ThreeDPieAttributesRole, 33
  - ValueTrackerAttributesRole, 33
- KDChart::AbstractArea, 35
- KDChart::AbstractArea
  - ~AbstractArea, 37
  - AbstractArea, 37
  - alignToReferencePoint, 38
  - areaGeometry, 38
  - backgroundAttributes, 38
  - bottomOverlap, 38
  - compare, 39
  - frameAttributes, 39
  - getFrameLeadings, 39
  - innerRect, 40
  - leftOverlap, 40
  - mParent, 47
  - mParentLayout, 47
  - paint, 40
  - paintAll, 41
  - paintBackground, 41
  - paintBackgroundAttributes, 42
  - paintCtx, 43
  - paintFrame, 43
  - paintFrameAttributes, 43
  - paintIntoRect, 44
  - parentLayout, 44
  - positionChanged, 44
  - positionHasChanged, 44
  - removeFromParentLayout, 44
  - rightOverlap, 45
  - setBackgroundAttributes, 45
  - setFrameAttributes, 45
  - setParentLayout, 46



- setParentWidget, 46
- sizeHintChanged, 46
- topOverlap, 47
- KDChart::AbstractAreaBase, 48
- KDChart::AbstractAreaBase
  - ~AbstractAreaBase, 49
  - AbstractAreaBase, 49
  - alignToReferencePoint, 49
  - areaGeometry, 49
  - backgroundAttributes, 49
  - compare, 50
  - frameAttributes, 50
  - getFrameLeadings, 50
  - innerRect, 51
  - paintBackground, 51
  - paintBackgroundAttributes, 51
  - paintFrame, 52
  - paintFrameAttributes, 52
  - positionHasChanged, 53
  - setBackgroundAttributes, 53
  - setFrameAttributes, 53
- KDChart::AbstractAreaWidget, 55
- KDChart::AbstractAreaWidget
  - ~AbstractAreaWidget, 56
  - AbstractAreaWidget, 56
  - alignToReferencePoint, 57
  - areaGeometry, 57
  - backgroundAttributes, 57
  - compare, 57
  - forceRebuild, 58
  - frameAttributes, 58
  - getFrameLeadings, 58
  - innerRect, 59
  - needSizeHint, 59
  - paint, 59
  - paintAll, 59
  - paintBackground, 60
  - paintBackgroundAttributes, 61
  - paintEvent, 62
  - paintFrame, 62
  - paintFrameAttributes, 62
  - paintIntoRect, 63
  - positionChanged, 63
  - positionHasChanged, 63
  - resizeLayout, 64
  - setBackgroundAttributes, 64
  - setFrameAttributes, 64
- KDChart::AbstractAxis, 66
- KDChart::AbstractAxis
  - ~AbstractAxis, 69
  - AbstractAxis, 69
  - alignToReferencePoint, 69
  - areaGeometry, 69
  - backgroundAttributes, 70
  - bottomOverlap, 70
  - compare, 70, 71
  - connectSignals, 71
  - coordinatePlane, 72
  - createObserver, 72
  - customizedLabel, 72
  - delayedInit, 73
  - deleteObserver, 73
  - diagram, 73
  - frameAttributes, 73
  - geometry, 74
  - getFrameLeadings, 74
  - innerRect, 74
  - labels, 75
  - leftOverlap, 75
  - mParent, 85
  - mParentLayout, 85
  - observedBy, 75
  - paint, 76
  - paintAll, 76
  - paintBackground, 76
  - paintBackgroundAttributes, 77
  - paintCtx, 78
  - paintFrame, 78
  - paintFrameAttributes, 78
  - paintIntoRect, 79
  - parentLayout, 79
  - positionChanged, 79
  - positionHasChanged, 79
  - removeFromParentLayout, 80
  - rightOverlap, 80
  - setBackgroundAttributes, 80
  - setFrameAttributes, 81
  - setGeometry, 81
  - setLabels, 81
  - setParentLayout, 81
  - setParentWidget, 82
  - setShortLabels, 82
  - setTextAttributes, 82
  - shortLabels, 83
  - sizeHintChanged, 83
  - textAttributes, 84
  - topOverlap, 84
  - update, 84
- KDChart::AbstractCartesianDiagram, 86
- KDChart::AbstractCartesianDiagram
  - ~AbstractCartesianDiagram, 92
  - AbstractCartesianDiagram, 92
  - addAxis, 93
  - allowOverlappingDataValueTexts, 93
  - antiAliasing, 93
  - attributesModel, 93
  - attributesModelRootIndex, 94
  - axes, 94

- brush, 95, 96
- calculateDataBoundaries, 96
- checkInvariants, 96
- columnToIndex, 97
- compare, 97, 99
- coordinatePlane, 99
- dataBoundaries, 100
- dataChanged, 100
- dataHidden, 100
- datasetBrushes, 100
- datasetDimension, 101
- datasetLabels, 102
- datasetMarkers, 102
- datasetPens, 103
- dataValueAttributes, 103, 104
- doItemsLayout, 105
- horizontalOffset, 105
- indexAt, 105
- indexesAt, 105
- isHidden, 105, 106
- isIndexHidden, 107
- itemRowLabels, 107
- layoutChanged, 107
- layoutPlanes, 108
- modelsChanged, 108
- moveCursor, 108
- numberOfAbscissaSegments, 108
- numberOfOrdinateSegments, 108
- paint, 109
- paintDataValueText, 109
- paintDataValueTexts, 110
- paintMarker, 111
- paintMarkers, 113
- pen, 113, 114
- percentMode, 115
- propertiesChanged, 115
- referenceDiagram, 115
- referenceDiagramOffset, 116
- resize, 116
- scrollTo, 116
- setAllowOverlappingDataValueTexts, 116
- setAntiAliasing, 117
- setAttributesModel, 117
- setAttributesModelRootIndex, 118
- setBrush, 118, 119
- setCoordinatePlane, 119
- setDataBoundariesDirty, 120
- setDatasetDimension, 120
- setDataValueAttributes, 120, 121
- setHidden, 122, 123
- setModel, 123
- setPen, 123, 124
- setPercentMode, 125
- setReferenceDiagram, 125
- setRootIndex, 125
- setSelection, 126
- setUnitPrefix, 126
- setUnitSuffix, 127
- takeAxis, 127
- threeDItemDepth, 128
- unitPrefix, 128, 129
- unitSuffix, 129
- update, 130
- useDefaultColors, 130
- useRainbowColors, 130
- usesExternalAttributesModel, 131
- useSubduedColors, 131
- valueForCell, 131
- verticalOffset, 132
- visualRect, 132
- visualRegionForSelection, 132
- KDChart::AbstractCoordinatePlane, 134
  - Linear, 139
  - Logarithmic, 139
- KDChart::AbstractCoordinatePlane
  - ~AbstractCoordinatePlane, 139
  - AbstractCoordinatePlane, 139
  - addDiagram, 140
  - alignToReferencePoint, 140
  - areaGeometry, 140
  - AxesCalcMode, 139
  - backgroundAttributes, 141
  - bottomOverlap, 141
  - compare, 141
  - destroyedCoordinatePlane, 142
  - diagram, 142
  - diagrams, 142, 143
  - expandingDirections, 143
  - frameAttributes, 143
  - geometry, 143
  - getDataDimensionsList, 144
  - getFrameLeadings, 144
  - globalGridAttributes, 144
  - gridDimensionsList, 145
  - innerRect, 145
  - isEmpty, 146
  - isRubberBandZoomingEnabled, 146
  - isVisiblePoint, 146
  - layoutDiagrams, 146
  - layoutPlanes, 147
  - leftOverlap, 147
  - maximumSize, 147
  - minimumSize, 148
  - minimumSizeHint, 148
  - mouseDoubleClickEvent, 148
  - mouseMoveEvent, 148
  - mousePressEvent, 149
  - mouseReleaseEvent, 150

- mParent, 165
- mParentLayout, 165
- needLayoutPlanes, 150
- needRelayout, 151
- needUpdate, 151
- paint, 151
- paintAll, 151
- paintBackground, 152
- paintBackgroundAttributes, 152
- paintCtx, 153
- paintFrame, 153
- paintFrameAttributes, 154
- paintIntoRect, 154
- parent, 154, 155
- parentLayout, 155
- positionChanged, 155
- positionHasChanged, 155
- propertiesChanged, 155
- referenceCoordinatePlane, 156
- relayout, 156
- removeFromParentLayout, 156
- replaceDiagram, 157
- rightOverlap, 157
- setBackgroundAttributes, 158
- setFrameAttributes, 158
- setGeometry, 158
- setGlobalGridAttributes, 159
- setGridNeedsRecalculate, 159
- setParent, 159
- setParentLayout, 160
- setParentWidget, 160
- setReferenceCoordinatePlane, 160
- setRubberBandZoomingEnabled, 161
- setZoomCenter, 161
- setZoomFactorX, 161
- setZoomFactorY, 161
- sharedAxisMasterPlane, 162
- sizeHint, 162
- sizeHintChanged, 162
- sizePolicy, 163
- takeDiagram, 163
- topOverlap, 163
- translate, 164
- update, 164
- zoomCenter, 164
- zoomFactorX, 165
- zoomFactorY, 165
- KDChart::AbstractDiagram, 167
- KDChart::AbstractDiagram
  - ~AbstractDiagram, 173
  - AbstractDiagram, 172
  - allowOverlappingDataValueTexts, 173
  - antiAliasing, 173
  - attributesModel, 173
  - attributesModelRootIndex, 174
  - brush, 174, 175
  - calculateDataBoundaries, 176
  - checkInvariants, 176
  - columnToIndex, 176
  - compare, 176
  - coordinatePlane, 178
  - dataBoundaries, 178
  - dataChanged, 179
  - dataHidden, 179
  - datasetBrushes, 179
  - datasetDimension, 180
  - datasetLabels, 180
  - datasetMarkers, 181
  - datasetPens, 181
  - dataValueAttributes, 182, 183
  - doItemsLayout, 183
  - horizontalOffset, 183
  - indexAt, 184
  - indexesAt, 184
  - isHidden, 184, 185
  - isIndexHidden, 185
  - itemRowLabels, 185
  - layoutChanged, 186
  - modelsChanged, 186
  - moveCursor, 186
  - paint, 186
  - paintDataValueText, 187
  - paintDataValueTexts, 188
  - paintMarker, 189
  - paintMarkers, 191
  - pen, 191, 192
  - percentMode, 193
  - propertiesChanged, 193
  - resize, 193
  - scrollTo, 193
  - setAllowOverlappingDataValueTexts, 193
  - setAntiAliasing, 194
  - setAttributesModel, 194
  - setAttributesModelRootIndex, 195
  - setBrush, 195, 196
  - setCoordinatePlane, 196
  - setDataBoundariesDirty, 197
  - setDatasetDimension, 197
  - setDataValueAttributes, 197, 198
  - setHidden, 198, 199
  - setModel, 200
  - setPen, 200, 201
  - setPercentMode, 201
  - setRootIndex, 202
  - setSelection, 202
  - setUnitPrefix, 202, 203
  - setUnitSuffix, 203
  - unitPrefix, 204

- unitSuffix, 205
- update, 205
- useDefaultColors, 206
- useRainbowColors, 206
- usesExternalAttributesModel, 206
- useSubduedColors, 207
- valueForCell, 207
- verticalOffset, 207
- visualRect, 207
- visualRegionForSelection, 208
- KDChart::AbstractLayoutItem, 209
- KDChart::AbstractLayoutItem
  - AbstractLayoutItem, 210
  - mParent, 212
  - mParentLayout, 212
  - paint, 210
  - paintAll, 210
  - paintCtx, 210
  - parentLayout, 211
  - removeFromParentLayout, 211
  - setParentLayout, 211
  - setParentWidget, 211
  - sizeHintChanged, 212
- KDChart::AbstractPieDiagram, 213
- KDChart::AbstractPieDiagram
  - ~AbstractPieDiagram, 219
  - AbstractPieDiagram, 219
  - allowOverlappingDataValueTexts, 219
  - antiAliasing, 220
  - attributesModel, 220
  - attributesModelRootIndex, 220
  - brush, 221, 222
  - calculateDataBoundaries, 222
  - checkInvariants, 222
  - columnCount, 223
  - columnToIndex, 223
  - compare, 223
  - coordinatePlane, 225
  - dataBoundaries, 225
  - dataChanged, 226
  - dataHidden, 226
  - datasetBrushes, 226
  - datasetDimension, 227
  - datasetLabels, 228
  - datasetMarkers, 228
  - datasetPens, 229
  - dataValueAttributes, 229, 230
  - doItemsLayout, 230
  - granularity, 231
  - horizontalOffset, 231
  - indexAt, 231
  - indexesAt, 231
  - isHidden, 232, 233
  - isIndexHidden, 233
  - itemRowLabels, 233
  - layoutChanged, 234
  - modelsChanged, 234
  - moveCursor, 234
  - numberOfGridRings, 234
  - numberOfValuesPerDataset, 234
  - paint, 234
  - paintDataValueText, 235
  - paintDataValueTexts, 236
  - paintMarker, 237
  - paintMarkers, 239
  - pen, 239, 240
  - percentMode, 241
  - pieAttributes, 241
  - polarCoordinatePlane, 242
  - propertiesChanged, 242
  - resize, 242
  - scrollTo, 242
  - setAllowOverlappingDataValueTexts, 243
  - setAntiAliasing, 243
  - setAttributesModel, 243
  - setAttributesModelRootIndex, 244
  - setBrush, 244, 245
  - setCoordinatePlane, 246
  - setDataBoundariesDirty, 246
  - setDatasetDimension, 246
  - setDataValueAttributes, 247
  - setGranularity, 248
  - setHidden, 248, 249
  - setModel, 250
  - setPen, 250, 251
  - setPercentMode, 251
  - setPieAttributes, 252
  - setRootIndex, 252
  - setSelection, 252
  - setStartPosition, 253
  - setThreeDPieAttributes, 253
  - setUnitPrefix, 254
  - setUnitSuffix, 254, 255
  - startPosition, 255
  - threeDPieAttributes, 255, 256
  - unitPrefix, 256
  - unitSuffix, 257
  - update, 258
  - useDefaultColors, 258
  - useRainbowColors, 258
  - usesExternalAttributesModel, 258
  - useSubduedColors, 259
  - valueForCell, 259
  - valueTotals, 260
  - verticalOffset, 260
  - visualRect, 260
  - visualRegionForSelection, 260
- KDChart::AbstractPolarDiagram, 261

- KDChart::AbstractPolarDiagram
  - ~AbstractPolarDiagram, 267
  - AbstractPolarDiagram, 267
  - allowOverlappingDataValueTexts, 267
  - antiAliasing, 267
  - attributesModel, 267
  - attributesModelRootIndex, 268
  - brush, 268, 269
  - calculateDataBoundaries, 270
  - checkInvariants, 270
  - columnCount, 270
  - columnToIndex, 271
  - compare, 271
  - coordinatePlane, 273
  - dataBoundaries, 273
  - dataChanged, 274
  - dataHidden, 274
  - datasetBrushes, 274
  - datasetDimension, 275
  - datasetLabels, 275
  - datasetMarkers, 276
  - datasetPens, 276
  - dataValueAttributes, 277, 278
  - doItemsLayout, 278
  - horizontalOffset, 278
  - indexAt, 278
  - indexesAt, 279
  - isHidden, 279, 280
  - isIndexHidden, 280
  - itemRowLabels, 280
  - layoutChanged, 281
  - modelsChanged, 281
  - moveCursor, 281
  - numberOfGridRings, 281
  - numberOfValuesPerDataset, 282
  - paint, 282
  - paintDataValueText, 282
  - paintDataValueTexts, 283
  - paintMarker, 284
  - paintMarkers, 286
  - pen, 287
  - percentMode, 288
  - polarCoordinatePlane, 288
  - propertiesChanged, 288
  - resize, 289
  - scrollTo, 289
  - setAllowOverlappingDataValueTexts, 289
  - setAntiAliasing, 289
  - setAttributesModel, 290
  - setAttributesModelRootIndex, 291
  - setBrush, 291, 292
  - setCoordinatePlane, 292
  - setDataBoundariesDirty, 292
  - setDatasetDimension, 293
  - setDataValueAttributes, 293, 294
  - setHidden, 294, 295
  - setModel, 296
  - setPen, 296, 297
  - setPercentMode, 298
  - setRootIndex, 298
  - setSelection, 298
  - setUnitPrefix, 298, 299
  - setUnitSuffix, 299
  - unitPrefix, 300
  - unitSuffix, 301
  - update, 301
  - useDefaultColors, 302
  - useRainbowColors, 302
  - usesExternalAttributesModel, 302
  - useSubduedColors, 303
  - valueForCell, 303
  - valueTotals, 303
  - verticalOffset, 304
  - visualRect, 304
  - visualRegionForSelection, 304
- KDChart::AbstractProxyModel
  - AbstractProxyModel, 305
  - index, 306
  - mapFromSource, 306
  - mapToSource, 306
  - parent, 307
- KDChart::AbstractTernaryDiagram, 308
- KDChart::AbstractTernaryDiagram
  - allowOverlappingDataValueTexts, 313
  - antiAliasing, 313
  - attributesModel, 314
  - attributesModelRootIndex, 314
  - brush, 315, 316
  - calculateDataBoundaries, 316
  - checkInvariants, 316
  - columnToIndex, 317
  - compare, 317
  - coordinatePlane, 319
  - dataBoundaries, 319
  - dataChanged, 320
  - dataHidden, 320
  - datasetBrushes, 320
  - datasetDimension, 321
  - datasetLabels, 321
  - datasetMarkers, 322
  - datasetPens, 322
  - dataValueAttributes, 323, 324
  - doItemsLayout, 324
  - horizontalOffset, 324
  - indexAt, 325
  - indexesAt, 325
  - isHidden, 325, 326

- isIndexHidden, 326
- itemRowLabels, 327
- layoutChanged, 327
- modelsChanged, 327
- moveCursor, 327
- paintDataValueText, 328
- paintDataValueTexts, 329
- paintMarker, 330
- paintMarkers, 332
- pen, 332, 333
- percentMode, 334
- propertiesChanged, 334
- scrollTo, 334
- setAllowOverlappingDataValueTexts, 334
- setAntiAliasing, 335
- setAttributesModel, 335
- setAttributesModelRootIndex, 336
- setBrush, 336, 337
- setCoordinatePlane, 337
- setDataBoundariesDirty, 338
- setDatasetDimension, 338
- setDataValueAttributes, 338, 339
- setHidden, 340, 341
- setModel, 341
- setPen, 342
- setPercentMode, 343
- setRootIndex, 343
- setSelection, 343
- setUnitPrefix, 344
- setUnitSuffix, 344, 345
- unitPrefix, 345
- unitSuffix, 346
- update, 347
- useDefaultColors, 347
- useRainbowColors, 347
- usesExternalAttributesModel, 348
- useSubduedColors, 348
- valueForCell, 348
- verticalOffset, 349
- visualRect, 349
- visualRegionForSelection, 349
- KDChart::AbstractThreeDAttributes, 350
- KDChart::AbstractThreeDAttributes
  - ~AbstractThreeDAttributes, 350
  - AbstractThreeDAttributes, 350
  - depth, 351
  - isEnabled, 351
  - operator!=, 351
  - operator=, 351
  - operator==, 351
  - setDepth, 352
  - setEnabled, 352
  - validDepth, 352
- KDChart::AttributesModel, 353
  - PaletteTypeDefault, 355
  - PaletteTypeRainbow, 355
  - PaletteTypeSubdued, 355
- KDChart::AttributesModel
  - ~AttributesModel, 356
  - attributesChanged, 356
  - AttributesModel, 356
  - columnCount, 356
  - compare, 356
  - compareAttributes, 359
  - data, 360, 361
  - dataMap, 361
  - headerData, 362
  - horizontalHeaderDataMap, 363
  - index, 363
  - initFrom, 363
  - isKnownAttributesRole, 363
  - mapFromSource, 364
  - mapToSource, 364
  - modelData, 365
  - modelDataMap, 365
  - PaletteType, 355
  - paletteType, 365
  - parent, 366
  - resetData, 366
  - resetHeaderData, 366
  - rowCount, 366
  - setData, 367
  - setDataMap, 367
  - setDefaultForRole, 367
  - setHeaderData, 368
  - setHorizontalHeaderDataMap, 368
  - setModelData, 368
  - setModelDataMap, 369
  - setPaletteType, 369
  - setSourceModel, 369
  - setVerticalHeaderDataMap, 370
  - verticalHeaderDataMap, 370
- KDChart::AutoSpacerLayoutItem, 371
- KDChart::AutoSpacerLayoutItem
  - AutoSpacerLayoutItem, 372
  - expandingDirections, 372
  - geometry, 372
  - isEmpty, 372
  - maximumSize, 373
  - minimumSize, 373
  - mParent, 377
  - mParentLayout, 377
  - paint, 373
  - paintAll, 374
  - paintCtx, 374
  - parentLayout, 374
  - removeFromParentLayout, 375
  - setGeometry, 375

- setParentLayout, 375
- setParentWidget, 375
- sizeHint, 376
- sizeHintChanged, 376
- KDChart::BackgroundAttributes, 378
  - BackgroundPixmapModeCentered, 378
  - BackgroundPixmapModeNone, 378
  - BackgroundPixmapModeScaled, 378
  - BackgroundPixmapModeStretched, 378
- KDChart::BackgroundAttributes
  - ~BackgroundAttributes, 379
  - BackgroundAttributes, 379
  - BackgroundPixmapMode, 378
  - brush, 379
  - isEqualTo, 379
  - isVisible, 380
  - operator!=, 380
  - operator=, 380
  - operator==, 380
  - pixmap, 381
  - pixmapMode, 381
  - setBrush, 381
  - setPixmap, 381
  - setPixmapMode, 381
  - setVisible, 382
- KDChart::BarAttributes, 383
- KDChart::BarAttributes
  - ~BarAttributes, 384
  - BarAttributes, 384
  - barGapFactor, 384
  - drawSolidExcessArrows, 384
  - fixedBarWidth, 384
  - fixedDataValueGap, 385
  - fixedValueBlockGap, 385
  - groupGapFactor, 385
  - operator!=, 385
  - operator=, 385
  - operator==, 386
  - setBarGapFactor, 386
  - setDrawSolidExcessArrows, 386
  - setFixedBarWidth, 386
  - setFixedDataValueGap, 387
  - setFixedValueBlockGap, 387
  - setGroupGapFactor, 387
  - setUseFixedBarWidth, 387
  - setUseFixedDataValueGap, 387
  - setUseFixedValueBlockGap, 388
  - useFixedBarWidth, 388
  - useFixedDataValueGap, 388
  - useFixedValueBlockGap, 388
- KDChart::BarDiagram, 390
  - Normal, 398
  - Percent, 398
  - Rows, 398
  - Stacked, 398
- KDChart::BarDiagram
  - ~BarDiagram, 398
  - addAxis, 399
  - allowOverlappingDataValueTexts, 399
  - antiAliasing, 399
  - attributesModel, 399
  - attributesModelRootIndex, 400
  - axes, 400
  - barAttributes, 401
  - BarDiagram, 398
  - BarType, 398
  - brush, 402, 403
  - calculateDataBoundaries, 403
  - checkInvariants, 403
  - clone, 404
  - columnToIndex, 404
  - compare, 404, 406, 407
  - coordinatePlane, 407
  - dataBoundaries, 407
  - dataChanged, 408
  - dataHidden, 408
  - datasetBrushes, 408
  - datasetDimension, 409
  - datasetLabels, 409
  - datasetMarkers, 410
  - datasetPens, 410
  - dataValueAttributes, 411, 412
  - doItemsLayout, 412
  - horizontalOffset, 413
  - indexAt, 413
  - indexesAt, 413
  - isHidden, 413, 414
  - isIndexHidden, 415
  - itemRowLabels, 415
  - layoutChanged, 415
  - layoutPlanes, 415
  - modelsChanged, 416
  - moveCursor, 416
  - numberOfAbscissaSegments, 416
  - numberOfOrdinateSegments, 416
  - paint, 417
  - paintDataValueText, 417
  - paintDataValueTexts, 419
  - paintEvent, 419
  - paintMarker, 419, 420
  - paintMarkers, 422
  - pen, 422, 423
  - percentMode, 423
  - propertiesChanged, 424
  - referenceDiagram, 424
  - referenceDiagramOffset, 424
  - resize, 425
  - resizeEvent, 425



- scrollTo, 425
- setAllowOverlappingDataValueTexts, 425
- setAntiAliasing, 426
- setAttributesModel, 426
- setAttributesModelRootIndex, 427
- setBarAttributes, 427, 428
- setBrush, 428, 429
- setCoordinatePlane, 429
- setDataBoundariesDirty, 430
- setDatasetDimension, 430
- setDataValueAttributes, 430, 431
- setHidden, 431, 432
- setModel, 433
- setPen, 433, 434
- setPercentMode, 435
- setReferenceDiagram, 435
- setRootIndex, 435
- setSelection, 435
- setThreeDBarAttributes, 436
- setType, 437
- setUnitPrefix, 437, 438
- setUnitSuffix, 438
- takeAxis, 439
- threeDBarAttributes, 439, 440
- threeDItemDepth, 440, 441
- type, 441
- unitPrefix, 441, 442
- unitSuffix, 442
- update, 443
- useDefaultColors, 443
- useRainbowColors, 443
- usesExternalAttributesModel, 444
- useSubduedColors, 444
- valueForCell, 444
- verticalOffset, 445
- visualRect, 445
- visualRegionForSelection, 445
- KDChart::CartesianAxis, 447
  - Bottom, 451
  - Left, 451
  - Right, 451
  - Top, 451
- KDChart::CartesianAxis
  - ~CartesianAxis, 452
  - alignToReferencePoint, 452
  - areaGeometry, 452
  - backgroundAttributes, 452
  - bottomOverlap, 453
  - CartesianAxis, 451
  - compare, 453, 454
  - connectSignals, 454
  - coordinatePlane, 455
  - createObserver, 455
  - customizedLabel, 455
  - delayedInit, 456
  - deleteObserver, 456
  - diagram, 456
  - expandingDirections, 457
  - frameAttributes, 457
  - geometry, 457
  - getFrameLeadings, 458
  - hasDefaultTitleTextAttributes, 458
  - innerRect, 458
  - isAbscissa, 459
  - isEmpty, 459
  - isOrdinate, 459
  - labels, 459
  - layoutPlanes, 460
  - leftOverlap, 460
  - maximumSize, 460
  - minimumSize, 463
  - mParent, 485
  - mParentLayout, 485
  - observedBy, 464
  - paint, 464
  - paintAll, 464
  - paintBackground, 465
  - paintBackgroundAttributes, 465
  - paintCtx, 466
  - paintFrame, 476
  - paintFrameAttributes, 476
  - paintIntoRect, 477
  - parentLayout, 477
  - Position, 451
  - position, 477
  - positionChanged, 477
  - positionHasChanged, 478
  - removeFromParentLayout, 478
  - resetTitleTextAttributes, 478
  - rightOverlap, 478
  - setBackgroundAttributes, 479
  - setFrameAttributes, 479
  - setGeometry, 479
  - setLabels, 480
  - setParentLayout, 480
  - setParentWidget, 480
  - setPosition, 481
  - setShortLabels, 481
  - setTextAttributes, 481
  - setTitleText, 482
  - setTitleTextAttributes, 482
  - shortLabels, 482
  - sizeHint, 483
  - sizeHintChanged, 483
  - textAttributes, 483
  - tickLength, 483
  - titleText, 484
  - titleTextAttributes, 484



- topOverlap, 485
- update, 485
- KDChart::CartesianCoordinatePlane, 487
  - Linear, 494
  - Logarithmic, 494
- KDChart::CartesianCoordinatePlane
  - ~CartesianCoordinatePlane, 495
  - addDiagram, 495
  - adjustedToMaxEmptyInnerPercentage, 495
  - adjustHorizontalRangeToData, 496
  - adjustRangesToData, 496
  - adjustVerticalRangeToData, 497
  - alignToReferencePoint, 497
  - areaGeometry, 497
  - autoAdjustGridToZoom, 498
  - autoAdjustHorizontalRangeToData, 498
  - autoAdjustVerticalRangeToData, 498
  - AxesCalcMode, 494
  - axesCalcModeX, 499
  - axesCalcModeY, 499
  - backgroundAttributes, 499
  - bottomOverlap, 500
  - calculateRawDataBoundingRect, 500
  - CartesianCoordinatePlane, 494
  - compare, 501
  - destroyedCoordinatePlane, 501
  - diagram, 501
  - diagrams, 502
  - doesIsometricScaling, 503
  - doneSetZoomCenter, 503
  - doneSetZoomFactorX, 503
  - doneSetZoomFactorY, 503
  - drawingArea, 504
  - expandingDirections, 504
  - frameAttributes, 504
  - geometry, 504
  - getDataDimensionsList, 505
  - getFrameLeadings, 506
  - getRawDataBoundingRectFromDiagrams, 506
  - globalGridAttributes, 507
  - gridAttributes, 507
  - gridDimensionsList, 508
  - handleFixedDataCoordinateSpaceRelation, 508
  - hasFixedDataCoordinateSpaceRelation, 509
  - hasOwnGridAttributes, 509
  - horizontalRange, 510
  - innerRect, 510
  - isEmpty, 510
  - isHorizontalRangeReversed, 511
  - isRubberBandZoomingEnabled, 511
  - isVerticalRangeReversed, 511
  - isVisiblePoint, 511
  - layoutDiagrams, 512
  - layoutPlanes, 514
  - leftOverlap, 514
  - maximumSize, 514
  - minimumSize, 515
  - minimumSizeHint, 515
  - mouseDoubleClickEvent, 515
  - mouseMoveEvent, 516
  - mousePressEvent, 516
  - mouseReleaseEvent, 517
  - mParent, 542
  - mParentLayout, 542
  - needLayoutPlanes, 518
  - needRelayout, 518
  - needUpdate, 518
  - paint, 518
  - paintAll, 519
  - paintBackground, 520
  - paintBackgroundAttributes, 520
  - paintCtx, 521
  - paintEvent, 521
  - paintFrame, 521
  - paintFrameAttributes, 522
  - paintIntoRect, 522
  - parent, 522, 523
  - parentLayout, 523
  - positionChanged, 523
  - positionHasChanged, 523
  - propertiesChanged, 523
  - referenceCoordinatePlane, 524
  - relayout, 524
  - removeFromParentLayout, 524
  - replaceDiagram, 525
  - resetGridAttributes, 525
  - rightOverlap, 526
  - setAutoAdjustGridToZoom, 526
  - setAutoAdjustHorizontalRangeToData, 527
  - setAutoAdjustVerticalRangeToData, 527
  - setAxesCalcModes, 528
  - setAxesCalcModeX, 528
  - setAxesCalcModeY, 528
  - setBackgroundAttributes, 529
  - setFixedDataCoordinateSpaceRelation, 529
  - setFrameAttributes, 529
  - setGeometry, 529
  - setGlobalGridAttributes, 530
  - setGridAttributes, 530
  - setGridNeedsRecalculate, 531
  - setHorizontalRange, 531
  - setHorizontalRangeReversed, 532
  - setIsometricScaling, 532
  - setParent, 533
  - setParentLayout, 533
  - setParentWidget, 533
  - setReferenceCoordinatePlane, 533

- setRubberBandZoomingEnabled, 534
- setVerticalRange, 534
- setVerticalRangeReversed, 535
- setZoomCenter, 535
- setZoomFactorX, 536
- setZoomFactorY, 536
- sharedAxisMasterPlane, 536
- sizeHint, 537
- sizeHintChanged, 538
- sizePolicy, 538
- slotLayoutChanged, 538
- takeDiagram, 538
- topOverlap, 539
- translate, 539
- translateBack, 540
- update, 540
- verticalRange, 540
- visibleDataRange, 541
- zoomCenter, 541
- zoomFactorX, 541
- zoomFactorY, 542
- KDChart::Chart, 543
  - ~Chart, 546
  - addCoordinatePlane, 546
  - addHeaderFooter, 547
  - addLegend, 547
  - backgroundAttributes, 548
  - Chart, 546
  - coordinatePlane, 549
  - coordinatePlaneLayout, 549
  - coordinatePlanes, 549
  - frameAttributes, 549
  - globalLeadingBottom, 550, 564
  - globalLeadingLeft, 550, 564
  - globalLeadingRight, 550, 564
  - globalLeadingTop, 550, 564
  - headerFooter, 551
  - headerFooters, 551
  - legend, 551
  - legends, 552
  - mouseDoubleClickEvent, 552
  - mouseMoveEvent, 553
  - mousePressEvent, 553
  - mouseReleaseEvent, 554
  - paint, 554
  - paintEvent, 556
  - propertiesChanged, 556
  - reLayoutFloatingLegends, 556
  - replaceCoordinatePlane, 557
  - replaceHeaderFooter, 557
  - replaceLegend, 558
  - resizeEvent, 559
  - setBackgroundAttributes, 559
  - setCoordinatePlaneLayout, 560
  - setFrameAttributes, 560
  - setGlobalLeading, 560
  - setGlobalLeadingBottom, 561
  - setGlobalLeadingLeft, 561
  - setGlobalLeadingRight, 562
  - setGlobalLeadingTop, 562
  - takeCoordinatePlane, 562
  - takeHeaderFooter, 563
  - takeLegend, 563
- KDChart::ChartGraphicsItem, 565
  - Type, 565
- KDChart::ChartGraphicsItem
  - ChartGraphicsItem, 566
  - column, 566
  - row, 566
  - type, 566
- KDChart::DataDimension, 567
- KDChart::DataDimension
  - calcMode, 569
  - DataDimension, 568
  - distance, 568
  - end, 569
  - isCalculated, 569
  - operator!=, 568
  - operator==, 568
  - sequence, 569
  - start, 569
  - stepWidth, 569
  - subStepWidth, 570
- KDChart::DatasetProxyModel, 571
- KDChart::DatasetProxyModel
  - buddy, 573
  - data, 573
  - DatasetProxyModel, 572
  - filterAcceptsColumn, 573
  - filterAcceptsRow, 573
  - flags, 574
  - headerData, 574
  - index, 575
  - mapFromSource, 575
  - mapToSource, 575
  - parent, 576
  - resetDatasetDescriptions, 576
  - setData, 576
  - setDatasetColumnDescriptionVector, 576
  - setDatasetDescriptionVectors, 577
  - setDatasetRowDescriptionVector, 577
  - setSourceModel, 577
  - setSourceRootIndex, 578
- KDChart::DatasetSelectorWidget, 579
- KDChart::DatasetSelectorWidget
  - configureDatasetProxyModel, 580
  - DatasetSelectorWidget, 579
  - mappingDisabled, 580

- setSourceColumnCount, 580
- setSourceRowCount, 580
- KDChart::DataValueAttributes, 581
- KDChart::DataValueAttributes
  - ~DataValueAttributes, 583
  - backgroundAttributes, 583
  - dataLabel, 583
  - DataValueAttributes, 583
  - decimalDigits, 584
  - defaultAttributes, 584
  - defaultAttributesAsVariant, 584
  - frameAttributes, 584
  - isVisible, 585
  - markerAttributes, 585
  - negativePosition, 585
  - operator!=, 586
  - operator=, 586
  - operator==, 586
  - position, 587
  - positivePosition, 587
  - prefix, 587
  - setBackgroundAttributes, 588
  - setDataLabel, 588
  - setDecimalDigits, 588
  - setFrameAttributes, 589
  - setMarkerAttributes, 589
  - setNegativePosition, 589
  - setPositivePosition, 590
  - setPrefix, 590
  - setSuffix, 590
  - setTextAttributes, 591
  - setVisible, 591
  - suffix, 591
  - textAttributes, 592
- KDChart::DiagramObserver, 593
- KDChart::DiagramObserver
  - ~DiagramObserver, 594
  - diagram, 594
  - diagramAttributesChanged, 594
  - diagramDataChanged, 594
  - diagramDataHidden, 595
  - diagramDestroyed, 595
  - DiagramObserver, 594
- KDChart::FrameAttributes, 596
- KDChart::FrameAttributes
  - ~FrameAttributes, 596
  - FrameAttributes, 596
  - isVisible, 597
  - operator!=, 597
  - operator=, 597
  - operator==, 597
  - padding, 597
  - pen, 598
  - setPadding, 598
  - setPen, 598
  - setVisible, 598
- KDChart::GlobalMeasureScaling, 600
- KDChart::GlobalMeasureScaling
  - ~GlobalMeasureScaling, 601
  - currentFactors, 601
  - GlobalMeasureScaling, 601
  - instance, 601
  - paintDevice, 601
  - resetFactors, 602
  - setFactors, 602
  - setPaintDevice, 602
- KDChart::GridAttributes, 603
- KDChart::GridAttributes
  - ~GridAttributes, 604
  - adjustLowerBoundToGrid, 604
  - adjustUpperBoundToGrid, 604
  - GridAttributes, 604
  - gridGranularitySequence, 604
  - gridPen, 605
  - gridStepWidth, 605
  - gridSubStepWidth, 605
  - isGridVisible, 606
  - isSubGridVisible, 606
  - operator!=, 606
  - operator=, 606
  - operator==, 606
  - setAdjustBoundsToGrid, 607
  - setGridGranularitySequence, 607
  - setGridPen, 607
  - setGridStepWidth, 608
  - setGridSubStepWidth, 608
  - setGridVisible, 609
  - setSubGridPen, 609
  - setSubGridVisible, 609
  - setZeroLinePen, 609
  - subGridPen, 609
  - zeroLinePen, 610
- KDChart::HeaderFooter, 611
- KDChart::HeaderFooter
  - Footer, 614
  - Header, 614
- KDChart::HeaderFooter
  - ~HeaderFooter, 614
  - alignToReferencePoint, 614
  - areaGeometry, 614
  - autoReferenceArea, 615
  - backgroundAttributes, 615
  - clone, 615
  - compare, 615, 616
  - destroyedHeaderFooter, 616
  - expandingDirections, 616
  - frameAttributes, 616
  - geometry, 617
  - getFrameLeadings, 617

- HeaderFooter, 614
- HeaderFooterType, 614
- innerRect, 617
- intersects, 618
- isEmpty, 619
- maximumSize, 619
- minimumSize, 619
- mParent, 629
- mParentLayout, 629
- paint, 619
- paintAll, 620
- paintBackground, 621
- paintBackgroundAttributes, 621
- paintCtx, 622
- paintFrame, 622
- paintFrameAttributes, 622
- paintIntoRect, 623
- parentLayout, 623
- position, 623
- positionChanged, 624
- positionHasChanged, 624
- realFont, 624
- realFontSize, 624
- removeFromParentLayout, 624
- setAutoReferenceArea, 625
- setBackgroundAttributes, 625
- setFrameAttributes, 625
- setGeometry, 626
- setParent, 626
- setParentLayout, 626
- setParentWidget, 626
- setPosition, 627
- setText, 627
- setTextAttributes, 627
- setType, 628
- sizeHint, 628
- sizeHintChanged, 628
- text, 629
- textAttributes, 629
- type, 629
- KDChart::HorizontalLineLayoutItem, 631
- KDChart::HorizontalLineLayoutItem
  - expandingDirections, 632
  - geometry, 632
  - HorizontalLineLayoutItem, 632
  - isEmpty, 632
  - maximumSize, 632
  - minimumSize, 633
  - mParent, 635
  - mParentLayout, 635
  - paint, 633
  - paintAll, 633
  - paintCtx, 633
  - parentLayout, 634
  - removeFromParentLayout, 634
  - setGeometry, 634
  - setParentLayout, 634
  - setParentWidget, 634
  - sizeHint, 635
  - sizeHintChanged, 635
- KDChart::Legend, 637
  - ~Legend, 642
  - addDiagram, 642
  - alignment, 643
  - alignToReferencePoint, 643
  - areaGeometry, 643
  - backgroundAttributes, 643
  - brush, 643
  - brushes, 644
  - clone, 644
  - compare, 644, 645
  - constDiagrams, 646
  - datasetCount, 646
  - destroyedLegend, 646
  - diagram, 646
  - diagrams, 647
  - floatingPosition, 647
  - forceRebuild, 647
  - frameAttributes, 648
  - getFrameLeadings, 648
  - innerRect, 648
  - Legend, 641
  - LegendStyle, 641
  - legendStyle, 649
  - LinesOnly, 641
  - markerAttributes, 649
  - MarkersAndLines, 641
  - MarkersOnly, 641
  - minimumSizeHint, 649
  - needSizeHint, 650
  - orientation, 650
  - paint, 650
  - paintAll, 651
  - paintBackground, 652
  - paintBackgroundAttributes, 652
  - paintEvent, 653
  - paintFrame, 653
  - paintFrameAttributes, 654
  - paintIntoRect, 654
  - pen, 655
  - pens, 655
  - position, 655
  - positionChanged, 655
  - positionHasChanged, 656
  - propertiesChanged, 656
  - referenceArea, 656
  - removeDiagram, 656
  - removeDiagrams, 657

- replaceDiagram, 657
- resetTexts, 658
- resizeEvent, 658
- resizeLayout, 658
- setAlignment, 659
- setBackgroundAttributes, 659
- setBrush, 659
- setBrushesFromDiagram, 659
- setColor, 660
- setDefaultColors, 660
- setDiagram, 660
- setFloatingPosition, 661
- setFrameAttributes, 662
- setLegendStyle, 662
- setMarkerAttributes, 662
- setOrientation, 663
- setPen, 663
- setPosition, 663
- setRainbowColors, 663
- setReferenceArea, 664
- setShowLines, 664
- setSpacing, 664
- setSubduedColors, 665
- setText, 666
- setTextAttributes, 666
- setTitleText, 666
- setTitleTextAttributes, 666
- setUseAutomaticMarkerSize, 666
- setVisible, 667
- showLines, 667
- sizeHint, 667
- spacing, 668
- text, 668
- textAttributes, 668
- texts, 668
- titleText, 668
- titleTextAttributes, 669
- useAutomaticMarkerSize, 669
- KDChart::LineAttributes, 670
  - MissingValuesAreBridged, 671
  - MissingValuesHideSegments, 671
  - MissingValuesPolicyIgnored, 671
  - MissingValuesShownAsZero, 671
- KDChart::LineAttributes
  - ~LineAttributes, 671
  - displayArea, 672
  - LineAttributes, 671
  - MissingValuesPolicy, 670
  - missingValuesPolicy, 672
  - operator!=, 672
  - operator=, 672
  - operator==, 672
  - setDisplayArea, 673
  - setMissingValuesPolicy, 673
  - setTransparency, 673
  - transparency, 673
- KDChart::LineDiagram, 674
  - Normal, 682
  - Percent, 682
  - Stacked, 682
- KDChart::LineDiagram
  - ~LineDiagram, 682
  - addAxis, 683
  - allowOverlappingDataValueTexts, 683
  - antiAliasing, 683
  - attributesModel, 684
  - attributesModelRootIndex, 684
  - axes, 685
  - brush, 685, 686
  - calculateDataBoundaries, 686
  - checkInvariants, 687
  - clone, 687
  - columnToIndex, 687
  - compare, 688–690
  - coordinatePlane, 690
  - dataBoundaries, 691
  - dataChanged, 691
  - dataHidden, 691
  - datasetBrushes, 692
  - datasetDimension, 692
  - datasetLabels, 693
  - datasetMarkers, 693
  - datasetPens, 694
  - dataValueAttributes, 694, 695
  - doItemsLayout, 696
  - getCellValues, 696
  - horizontalOffset, 696
  - indexAt, 697
  - indexesAt, 697
  - isHidden, 697, 698
  - isIndexHidden, 698
  - itemRowLabels, 699
  - layoutChanged, 699
  - layoutPlanes, 699
  - lineAttributes, 700
  - LineDiagram, 682
  - LineType, 682
  - modelsChanged, 701
  - moveCursor, 701
  - numberOfAbscissaSegments, 701
  - numberOfOrdinateSegments, 701
  - paint, 701
  - paintDataValueText, 702
  - paintDataValueTexts, 704
  - paintEvent, 704
  - paintMarker, 704, 705
  - paintMarkers, 707
  - pen, 707, 708

- percentMode, 708
- propertiesChanged, 709
- referenceDiagram, 709
- referenceDiagramOffset, 709
- resetLineAttributes, 710
- resize, 710
- resizeEvent, 710
- scrollTo, 711
- setAllowOverlappingDataValueTexts, 711
- setAntiAliasing, 711
- setAttributesModel, 711
- setAttributesModelRootIndex, 712
- setBrush, 712, 713
- setCoordinatePlane, 714
- setDataBoundariesDirty, 714
- setDatasetDimension, 714
- setDataValueAttributes, 715, 716
- setHidden, 716, 717
- setLineAttributes, 717, 718
- setModel, 718
- setPen, 719
- setPercentMode, 720
- setReferenceDiagram, 720
- setRootIndex, 720
- setSelection, 721
- setThreeDLineAttributes, 721, 722
- setType, 722
- setUnitPrefix, 723
- setUnitSuffix, 724
- setValueTrackerAttributes, 724
- takeAxis, 724
- threeDItemDepth, 725
- threeDLineAttributes, 726
- type, 726
- unitPrefix, 727
- unitSuffix, 728
- update, 728
- useDefaultColors, 729
- useRainbowColors, 729
- usesExternalAttributesModel, 729
- useSubduedColors, 730
- valueForCell, 730
- valueForCellTesting, 730
- valueTrackerAttributes, 731
- verticalOffset, 731
- visualRect, 731
- visualRegionForSelection, 731
- KDChart::LineLayoutItem, 733
- KDChart::LineLayoutItem
  - expandingDirections, 734
  - geometry, 734
  - isEmpty, 735
  - LineLayoutItem, 734
  - maximumSize, 735
  - minimumSize, 735
  - mParent, 738
  - mParentLayout, 738
  - paint, 735
  - paintAll, 735
  - paintCtx, 736
  - paintIntoRect, 736
  - parentLayout, 736
  - removeFromParentLayout, 736
  - setGeometry, 737
  - setParentLayout, 737
  - setParentWidget, 737
  - sizeHint, 737
  - sizeHintChanged, 738
- KDChart::LineWithMarkerLayoutItem, 739
- KDChart::LineWithMarkerLayoutItem
  - expandingDirections, 740
  - geometry, 740
  - isEmpty, 741
  - LineWithMarkerLayoutItem, 740
  - maximumSize, 741
  - minimumSize, 741
  - mParent, 744
  - mParentLayout, 744
  - paint, 741
  - paintAll, 741
  - paintCtx, 742
  - parentLayout, 742
  - removeFromParentLayout, 742
  - setGeometry, 742
  - setParentLayout, 743
  - setParentWidget, 743
  - sizeHint, 743
  - sizeHintChanged, 743
- KDChart::MarkerAttributes, 745
  - Marker1Pixel, 746
  - Marker4Pixels, 746
  - MarkerCircle, 746
  - MarkerCross, 746
  - MarkerDiamond, 746
  - MarkerFastCross, 746
  - MarkerRing, 746
  - MarkerSquare, 746
- KDChart::MarkerAttributes
  - ~MarkerAttributes, 746
  - isVisible, 747
  - MarkerAttributes, 746
  - markerColor, 747
  - markerSize, 747
  - MarkerStyle, 746
  - markerStyle, 747
  - MarkerStylesMap, 746
  - markerStylesMap, 748
  - operator!=, 748

- operator=, 748
- operator==, 748
- pen, 749
- setMarkerColor, 749
- setMarkerSize, 749
- setMarkerStyle, 749
- setMarkerStylesMap, 749
- setPen, 750
- setVisible, 750
- KDChart::MarkerLayoutItem, 751
- KDChart::MarkerLayoutItem
  - expandingDirections, 752
  - geometry, 752
  - isEmpty, 753
  - MarkerLayoutItem, 752
  - maximumSize, 753
  - minimumSize, 753
  - mParent, 756
  - mParentLayout, 757
  - paint, 753
  - paintAll, 753
  - paintCtx, 754
  - paintIntoRect, 754
  - parentLayout, 755
  - removeFromParentLayout, 755
  - setGeometry, 755
  - setParentLayout, 755
  - setParentWidget, 755
  - sizeHint, 756
  - sizeHintChanged, 756
- KDChart::Measure, 758
  - calculatedValue, 759, 760
  - calculationMode, 761
  - Measure, 759
  - operator!=, 761
  - operator=, 761
  - operator==, 761
  - referenceArea, 762
  - referenceOrientation, 762
  - setAbsoluteValue, 762
  - setCalculationMode, 762
  - setReferenceArea, 763
  - setReferenceOrientation, 763
  - setRelativeMode, 763
  - setValue, 763
  - sizeOfArea, 763
  - value, 764
- KDChart::PaintContext, 765
- KDChart::PaintContext
  - ~PaintContext, 765
  - coordinatePlane, 766
  - PaintContext, 765
  - painter, 766
  - rectangle, 766
  - setCoordinatePlane, 766
  - setPainter, 766
  - setRectangle, 767
- KDChart::Palette, 768
  - ~Palette, 769
  - addBrush, 770
  - changed, 770
  - defaultPalette, 770
  - getBrush, 770
  - isValid, 771
  - operator=, 771
  - Palette, 769
  - rainbowPalette, 771
  - removeBrush, 771
  - size, 772
  - subduedPalette, 772
- KDChart::PieAttributes, 773
- KDChart::PieAttributes
  - ~PieAttributes, 773
  - explode, 774
  - explodeFactor, 774
  - operator!=, 774
  - operator=, 774
  - operator==, 775
  - PieAttributes, 773
  - setExplode, 775
  - setExplodeFactor, 775
- KDChart::PieDiagram, 777
- KDChart::PieDiagram
  - ~PieDiagram, 783
  - allowOverlappingDataValueTexts, 784
  - antiAliasing, 784
  - attributesModel, 784
  - attributesModelRootIndex, 785
  - brush, 785, 786
  - calculateDataBoundaries, 787
  - checkInvariants, 787
  - clone, 788
  - columnCount, 788
  - columnToIndex, 788
  - compare, 788
  - coordinatePlane, 790
  - dataBoundaries, 790
  - dataChanged, 791
  - dataHidden, 791
  - datasetBrushes, 791
  - datasetDimension, 792
  - datasetLabels, 792
  - datasetMarkers, 793
  - datasetPens, 793
  - dataValueAttributes, 794, 795
  - doItemsLayout, 795
  - granularity, 796
  - horizontalOffset, 796



- indexAt, 796
- indexesAt, 796
- isHidden, 797
- isIndexHidden, 798
- itemRowLabels, 798
- layoutChanged, 798
- modelsChanged, 799
- moveCursor, 799
- numberOfGridRings, 799
- numberOfValuesPerDataset, 799
- paint, 799
- paintDataValueText, 802
- paintDataValueTexts, 804
- paintEvent, 804
- paintMarker, 804, 805
- paintMarkers, 807
- pen, 807, 808
- percentMode, 808
- pieAttributes, 809
- PieDiagram, 783
- polarCoordinatePlane, 809
- propertiesChanged, 810
- resize, 810
- resizeEvent, 810
- scrollTo, 810
- setAllowOverlappingDataValueTexts, 810
- setAntiAliasing, 811
- setAttributesModel, 811
- setAttributesModelRootIndex, 812
- setBrush, 812, 813
- setCoordinatePlane, 813
- setDataBoundariesDirty, 814
- setDatasetDimension, 814
- setDataValueAttributes, 815
- setGranularity, 816
- setHidden, 816, 817
- setModel, 818
- setPen, 818, 819
- setPercentMode, 819
- setPieAttributes, 820
- setRootIndex, 820
- setSelection, 820
- setStartPosition, 821
- setThreeDPieAttributes, 821
- setUnitPrefix, 822
- setUnitSuffix, 822, 823
- startPosition, 823
- threeDPieAttributes, 823, 824
- unitPrefix, 824
- unitSuffix, 825
- update, 826
- useDefaultColors, 826
- useRainbowColors, 826
- usesExternalAttributesModel, 827
- useSubduedColors, 827
- valueForCell, 827
- valueTotals, 828
- verticalOffset, 828
- visualRect, 828
- visualRegionForSelection, 828
- KDChart::Plotter, 830
  - ~Plotter, 838
  - addAxis, 838
  - allowOverlappingDataValueTexts, 839
  - antiAliasing, 839
  - attributesModel, 839
  - attributesModelRootIndex, 840
  - axes, 840
  - brush, 841, 842
  - calculateDataBoundaries, 842
  - checkInvariants, 842
  - clone, 843
  - columnToIndex, 843
  - compare, 843, 845, 846
  - coordinatePlane, 846
  - dataBoundaries, 846
  - dataChanged, 847
  - dataHidden, 847
  - datasetBrushes, 847
  - datasetDimension, 848
  - datasetLabels, 848
  - datasetMarkers, 849
  - datasetPens, 849
  - dataValueAttributes, 850, 851
  - doItemsLayout, 851
  - horizontalOffset, 851
  - indexAt, 852
  - indexesAt, 852
  - isHidden, 852, 853
  - isIndexHidden, 853
  - itemRowLabels, 854
  - layoutChanged, 854
  - layoutPlanes, 854
  - lineAttributes, 855
  - modelsChanged, 856
  - moveCursor, 856
  - Normal, 838
  - numberOfAbscissaSegments, 856
  - numberOfOrdinateSegments, 856
  - paint, 856
  - paintDataValueText, 857
  - paintDataValueTexts, 859
  - paintEvent, 859
  - paintMarker, 859, 860
  - paintMarkers, 862
  - pen, 862, 863
  - percentMode, 863
  - Plotter, 838



- PlotType, 838
- propertiesChanged, 864
- referenceDiagram, 864
- referenceDiagramOffset, 864
- resetLineAttributes, 865
- resize, 865
- resizeEvent, 865
- scrollTo, 866
- setAllowOverlappingDataValueTexts, 866
- setAntiAliasing, 866
- setAttributesModel, 866
- setAttributesModelRootIndex, 867
- setBrush, 867, 868
- setCoordinatePlane, 869
- setDataBoundariesDirty, 869
- setDatasetDimension, 869
- setDataValueAttributes, 870, 871
- setHidden, 871, 872
- setLineAttributes, 872, 873
- setModel, 873
- setPen, 874
- setPercentMode, 875
- setReferenceDiagram, 875
- setRootIndex, 875
- setSelection, 876
- setThreeDLineAttributes, 876, 877
- setType, 877
- setUnitPrefix, 878
- setUnitSuffix, 878, 879
- setValueTrackerAttributes, 879
- takeAxis, 879
- threeDItemDepth, 880
- threeDLineAttributes, 881
- type, 881
- unitPrefix, 882
- unitSuffix, 883
- update, 883
- useDefaultColors, 884
- useRainbowColors, 884
- usesExternalAttributesModel, 884
- useSubduedColors, 885
- valueForCell, 885
- valueTrackerAttributes, 885
- verticalOffset, 886
- visualRect, 886
- visualRegionForSelection, 886
- KDChart::PolarCoordinatePlane, 887
  - Linear, 893
  - Logarithmic, 893
- KDChart::PolarCoordinatePlane
  - ~PolarCoordinatePlane, 893
  - addDiagram, 893
  - alignToReferencePoint, 894
  - angleUnit, 894
  - areaGeometry, 894
  - AxesCalcMode, 893
  - backgroundAttributes, 894
  - bottomOverlap, 895
  - compare, 895
  - CoordinateTransformationList, 893
  - destroyedCoordinatePlane, 896
  - diagram, 896
  - diagrams, 896, 897
  - expandingDirections, 897
  - frameAttributes, 897
  - geometry, 897
  - getDataDimensionsList, 898
  - getFrameLeadings, 898
  - globalGridAttributes, 898
  - gridAttributes, 899
  - gridDimensionsList, 899
  - hasOwnGridAttributes, 900
  - innerRect, 901
  - isEmpty, 901
  - isRubberBandZoomingEnabled, 901
  - isVisiblePoint, 901
  - layoutDiagrams, 902
  - layoutPlanes, 903
  - leftOverlap, 903
  - maximumSize, 903
  - minimumSize, 904
  - minimumSizeHint, 904
  - mouseDoubleClickEvent, 904
  - mouseMoveEvent, 905
  - mousePressEvent, 905
  - mouseReleaseEvent, 906
  - mParent, 925
  - mParentLayout, 925
  - needLayoutPlanes, 907
  - needRelayout, 907
  - needUpdate, 907
  - paint, 907
  - paintAll, 908
  - paintBackground, 908
  - paintBackgroundAttributes, 909
  - paintCtx, 910
  - paintEvent, 910
  - paintFrame, 910
  - paintFrameAttributes, 910
  - paintIntoRect, 911
  - parent, 911
  - parentLayout, 912
  - PolarCoordinatePlane, 893
  - positionChanged, 912
  - positionHasChanged, 912
  - propertiesChanged, 912
  - radiusUnit, 912
  - referenceCoordinatePlane, 913

- relayout, 913
- removeFromParentLayout, 913
- replaceDiagram, 914
- resetGridAttributes, 914
- resizeEvent, 915
- rightOverlap, 915
- setBackgroundAttributes, 915
- setFrameAttributes, 916
- setGeometry, 916
- setGlobalGridAttributes, 916
- setGridAttributes, 917
- setGridNeedsRecalculate, 918
- setParent, 918
- setParentLayout, 918
- setParentWidget, 918
- setReferenceCoordinatePlane, 919
- setRubberBandZoomingEnabled, 919
- setStartPosition, 919
- setZoomCenter, 920
- setZoomFactorX, 920
- setZoomFactorY, 920
- sharedAxisMasterPlane, 921
- sizeHint, 921
- sizeHintChanged, 921
- sizePolicy, 922
- slotLayoutChanged, 922
- startPosition, 922
- takeDiagram, 922
- topOverlap, 923
- translate, 923
- translatePolar, 924
- update, 924
- zoomCenter, 924
- zoomFactorX, 925
- zoomFactorY, 925
- KDChart::PolarDiagram, 927
- KDChart::PolarDiagram
  - ~PolarDiagram, 933
  - allowOverlappingDataValueTexts, 933
  - antiAliasing, 934
  - attributesModel, 934
  - attributesModelRootIndex, 934
  - brush, 935, 936
  - calculateDataBoundaries, 936
  - checkInvariants, 937
  - clone, 937
  - closeDatasets, 938
  - columnCount, 938
  - columnToIndex, 938
  - compare, 938
  - coordinatePlane, 940
  - dataBoundaries, 940
  - dataChanged, 941
  - dataHidden, 941
  - datasetBrushes, 941
  - datasetDimension, 942
  - datasetLabels, 942
  - datasetMarkers, 943
  - datasetPens, 943
  - dataValueAttributes, 944, 945
  - doItemsLayout, 945
  - horizontalOffset, 946
  - indexAt, 946
  - indexesAt, 946
  - isHidden, 946, 947
  - isIndexHidden, 948
  - itemRowLabels, 948
  - layoutChanged, 948
  - modelsChanged, 948
  - moveCursor, 949
  - numberOfGridRings, 949
  - numberOfValuesPerDataset, 949
  - paint, 949
  - paintDataValueText, 950
  - paintDataValueTexts, 952
  - paintEvent, 952
  - paintMarker, 952, 953
  - paintMarkers, 955
  - paintPolarMarkers, 955
  - pen, 955, 956
  - percentMode, 957
  - polarCoordinatePlane, 957
  - PolarDiagram, 933
  - propertiesChanged, 957
  - resize, 957
  - resizeEvent, 958
  - rotateCircularLabels, 958
  - scrollTo, 958
  - setAllowOverlappingDataValueTexts, 958
  - setAntiAliasing, 958
  - setAttributesModel, 959
  - setAttributesModelRootIndex, 960
  - setBrush, 960, 961
  - setCloseDatasets, 961
  - setCoordinatePlane, 961
  - setDataBoundariesDirty, 962
  - setDatasetDimension, 962
  - setDataValueAttributes, 963
  - setHidden, 964, 965
  - setModel, 965
  - setPen, 966
  - setPercentMode, 967
  - setRootIndex, 967
  - setRotateCircularLabels, 967
  - setSelection, 968
  - setShowDelimitersAtPosition, 968
  - setShowLabelsAtPosition, 968
  - setUnitPrefix, 968, 969

- setUnitSuffix, 969
- setZeroDegreePosition, 970
- showDelimitersAtPosition, 970
- showLabelsAtPosition, 970
- unitPrefix, 970, 971
- unitSuffix, 971, 972
- update, 972
- useDefaultColors, 972
- useRainbowColors, 973
- usesExternalAttributesModel, 973
- useSubduedColors, 973
- valueForCell, 974
- valueTotals, 974
- verticalOffset, 974
- visualRect, 975
- visualRegionForSelection, 975
- zeroDegreePosition, 975
- KDChart::Position, 976
  - Center, 983
  - East, 983
  - ExcludeCenter, 978
  - Floating, 983
  - fromName, 979
  - IncludeCenter, 978
  - isCorner, 979
  - isEastSide, 979
  - isFloating, 979
  - isNorthSide, 980
  - isPole, 980
  - isSouthSide, 980
  - isUnknown, 980
  - isWestSide, 980
  - name, 981
  - names, 981
  - North, 983
  - NorthEast, 983
  - NorthWest, 983
  - operator!=, 981
  - operator==, 982
  - Option, 978
  - Position, 978
  - printableName, 982
  - printableNames, 982
  - South, 984
  - SouthEast, 984
  - SouthWest, 984
  - Unknown, 984
  - value, 983
  - West, 984
- KDChart::PositionPoints, 985
- KDChart::PositionPoints
  - isNull, 987
  - mPositionCenter, 988
  - mPositionEast, 988
  - mPositionNorth, 988
  - mPositionNorthEast, 988
  - mPositionNorthWest, 988
  - mPositionSouth, 988
  - mPositionSouthEast, 988
  - mPositionSouthWest, 988
  - mPositionUnknown, 988
  - mPositionWest, 988
  - point, 987
  - PositionPoints, 986
- KDChart::PrivateAttributesModel, 990
  - PaletteTypeDefault, 992
  - PaletteTypeRainbow, 992
  - PaletteTypeSubdued, 992
- KDChart::PrivateAttributesModel
  - attributesChanged, 993
  - columnCount, 993
  - compare, 993
  - compareAttributes, 995
  - data, 996–998
  - dataMap, 998
  - headerData, 998
  - horizontalHeaderDataMap, 999
  - index, 999
  - initFrom, 1000
  - isKnownAttributesRole, 1000
  - mapFromSource, 1001
  - mapToSource, 1001
  - modelData, 1002
  - modelDataMap, 1002
  - PaletteType, 992
  - paletteType, 1002
  - parent, 1002
  - PrivateAttributesModel, 992
  - resetData, 1003
  - resetHeaderData, 1003
  - rowCount, 1003
  - setData, 1003
  - setDataMap, 1004
  - setDefaultForRole, 1004
  - setHeaderData, 1004
  - setHorizontalHeaderDataMap, 1005
  - setModelData, 1005
  - setModelDataMap, 1006
  - setPaletteType, 1006
  - setSourceModel, 1006
  - setVerticalHeaderDataMap, 1007
  - verticalHeaderDataMap, 1007
- KDChart::RelativePosition, 1008
- KDChart::RelativePosition
  - ~RelativePosition, 1009
  - alignment, 1010
  - calculatedPoint, 1010
  - horizontalPadding, 1010

- operator!=, 1010
- operator=, 1011
- operator==, 1011
- referenceArea, 1011
- referencePoint, 1011
- referencePoints, 1012
- referencePosition, 1012
- RelativePosition, 1009
- resetReferencePosition, 1012
- rotation, 1013
- setAlignment, 1013
- setHorizontalPadding, 1013
- setReferenceArea, 1014
- setReferencePoints, 1014
- setReferencePosition, 1015
- setRotation, 1015
- setVerticalPadding, 1015
- verticalPadding, 1016
- KDChart::ReverseMapper, 1017
- KDChart::ReverseMapper
  - ~ReverseMapper, 1018
  - addCircle, 1018
  - addItem, 1018
  - addLine, 1018
  - addPolygon, 1019
  - addRect, 1019
  - clear, 1019
  - indexesAt, 1019
  - indexesIn, 1020
  - ReverseMapper, 1017
  - setDiagram, 1020
- KDChart::RingDiagram, 1022
- KDChart::RingDiagram
  - ~RingDiagram, 1028
  - allowOverlappingDataValueTexts, 1029
  - antiAliasing, 1029
  - attributesModel, 1029
  - attributesModelRootIndex, 1030
  - brush, 1030, 1031
  - calculateDataBoundaries, 1032
  - checkInvariants, 1032
  - clone, 1032
  - columnCount, 1033
  - columnToIndex, 1033
  - compare, 1033
  - coordinatePlane, 1035
  - dataBoundaries, 1035
  - dataChanged, 1036
  - dataHidden, 1036
  - datasetBrushes, 1036
  - datasetDimension, 1037
  - datasetLabels, 1037
  - datasetMarkers, 1038
  - datasetPens, 1038
  - dataValueAttributes, 1039, 1040
  - doItemsLayout, 1040
  - granularity, 1040
  - horizontalOffset, 1041
  - indexAt, 1041
  - indexesAt, 1041
  - isHidden, 1041, 1042
  - isIndexHidden, 1043
  - itemRowLabels, 1043
  - layoutChanged, 1043
  - modelsChanged, 1044
  - moveCursor, 1044
  - numberOfGridRings, 1044
  - numberOfValuesPerDataset, 1044
  - paint, 1044
  - paintDataValueText, 1045
  - paintDataValueTexts, 1047
  - paintEvent, 1047
  - paintMarker, 1047, 1048
  - paintMarkers, 1050
  - pen, 1050, 1051
  - percentMode, 1051
  - pieAttributes, 1052
  - polarCoordinatePlane, 1052
  - propertiesChanged, 1053
  - relativeThickness, 1053
  - resize, 1053
  - resizeEvent, 1053
  - RingDiagram, 1028
  - scrollTo, 1053
  - setAllowOverlappingDataValueTexts, 1054
  - setAntiAliasing, 1054
  - setAttributesModel, 1054
  - setAttributesModelRootIndex, 1055
  - setBrush, 1055, 1056
  - setCoordinatePlane, 1056
  - setDataBoundariesDirty, 1057
  - setDatasetDimension, 1057
  - setDataValueAttributes, 1058
  - setGranularity, 1059
  - setHidden, 1059, 1060
  - setModel, 1061
  - setPen, 1061, 1062
  - setPercentMode, 1062
  - setPieAttributes, 1063
  - setRelativeThickness, 1063
  - setRootIndex, 1063
  - setSelection, 1064
  - setStartPosition, 1064
  - setThreeDPieAttributes, 1064, 1065
  - setUnitPrefix, 1065
  - setUnitSuffix, 1065, 1066
  - startPosition, 1066
  - threeDPieAttributes, 1066, 1067

- unitPrefix, [1067](#), [1068](#)
- unitSuffix, [1068](#)
- update, [1069](#)
- useDefaultColors, [1069](#)
- useRainbowColors, [1069](#)
- usesExternalAttributesModel, [1070](#)
- useSubduedColors, [1070](#)
- valueForCell, [1070](#)
- valueTotals, [1071](#)
- verticalOffset, [1071](#)
- visualRect, [1071](#)
- visualRegionForSelection, [1072](#)
- KDChart::SignalCompressor, [1073](#)
- KDChart::SignalCompressor
  - emitSignal, [1074](#)
  - finallyEmit, [1074](#)
  - SignalCompressor, [1074](#)
- KDChart::TernaryAxis, [1075](#)
- KDChart::TernaryAxis
  - ~TernaryAxis, [1078](#)
  - alignToReferencePoint, [1079](#)
  - areaGeometry, [1079](#)
  - backgroundAttributes, [1079](#)
  - bottomOverlap, [1079](#)
  - compare, [1080](#)
  - connectSignals, [1081](#)
  - coordinatePlane, [1081](#)
  - createObserver, [1081](#)
  - customizedLabel, [1082](#)
  - delayedInit, [1082](#)
  - deleteObserver, [1082](#)
  - diagram, [1082](#)
  - expandingDirections, [1083](#)
  - frameAttributes, [1083](#)
  - geometry, [1083](#)
  - getFrameLeadings, [1083](#)
  - hasDefaultTitleTextAttributes, [1084](#)
  - innerRect, [1084](#)
  - isEmpty, [1084](#)
  - labels, [1084](#)
  - leftOverlap, [1085](#)
  - maximumSize, [1085](#)
  - minimumSize, [1085](#)
  - mParent, [1098](#)
  - mParentLayout, [1098](#)
  - observedBy, [1086](#)
  - paint, [1086](#)
  - paintAll, [1086](#)
  - paintBackground, [1086](#)
  - paintBackgroundAttributes, [1087](#)
  - paintCtx, [1088](#)
  - paintFrame, [1088](#)
  - paintFrameAttributes, [1088](#)
  - paintIntoRect, [1089](#)
  - parentLayout, [1089](#)
  - position, [1089](#)
  - positionChanged, [1090](#)
  - positionHasChanged, [1090](#)
  - removeFromParentLayout, [1090](#)
  - requiredMargins, [1090](#)
  - resetTitleTextAttributes, [1091](#)
  - rightOverlap, [1091](#)
  - setBackgroundAttributes, [1091](#)
  - setFrameAttributes, [1092](#)
  - setGeometry, [1092](#)
  - setLabels, [1092](#)
  - setParentLayout, [1093](#)
  - setParentWidget, [1093](#)
  - setPosition, [1093](#)
  - setShortLabels, [1094](#)
  - setTextAttributes, [1094](#)
  - setTitleText, [1095](#)
  - setTitleTextAttributes, [1095](#)
  - shortLabels, [1095](#)
  - sizeHint, [1096](#)
  - sizeHintChanged, [1096](#)
  - TernaryAxis, [1078](#)
  - textAttributes, [1096](#)
  - titleText, [1096](#)
  - titleTextAttributes, [1097](#)
  - topOverlap, [1097](#)
  - update, [1097](#)
- KDChart::TernaryCoordinatePlane, [1099](#)
  - Linear, [1104](#)
  - Logarithmic, [1104](#)
- KDChart::TernaryCoordinatePlane
  - ~TernaryCoordinatePlane, [1104](#)
  - addDiagram, [1104](#)
  - alignToReferencePoint, [1105](#)
  - areaGeometry, [1105](#)
  - AxesCalcMode, [1104](#)
  - backgroundAttributes, [1105](#)
  - bottomOverlap, [1106](#)
  - compare, [1106](#)
  - destroyedCoordinatePlane, [1106](#)
  - diagram, [1107](#)
  - diagrams, [1107](#)
  - expandingDirections, [1108](#)
  - frameAttributes, [1108](#)
  - geometry, [1108](#)
  - getDataDimensionsList, [1109](#)
  - getFrameLeadings, [1109](#)
  - globalGridAttributes, [1109](#)
  - gridDimensionsList, [1110](#)
  - innerRect, [1110](#)
  - isEmpty, [1111](#)
  - isRubberBandZoomingEnabled, [1111](#)
  - isVisiblePoint, [1111](#)

- layoutDiagrams, 1111
- layoutPlanes, 1113
- leftOverlap, 1113
- maximumSize, 1113
- minimumSize, 1114
- minimumSizeHint, 1114
- mouseDoubleClickEvent, 1114
- mouseMoveEvent, 1115
- mousePressEvent, 1115
- mouseReleaseEvent, 1116
- mParent, 1133
- mParentLayout, 1133
- needLayoutPlanes, 1117
- needRelayout, 1117
- needUpdate, 1117
- paint, 1117
- paintAll, 1118
- paintBackground, 1119
- paintBackgroundAttributes, 1119
- paintCtx, 1120
- paintFrame, 1120
- paintFrameAttributes, 1120
- paintIntoRect, 1121
- parent, 1121
- parentLayout, 1122
- positionChanged, 1122
- positionHasChanged, 1122
- propertiesChanged, 1122
- referenceCoordinatePlane, 1122
- relayout, 1123
- removeFromParentLayout, 1123
- replaceDiagram, 1123
- rightOverlap, 1124
- setBackgroundAttributes, 1125
- setFrameAttributes, 1125
- setGeometry, 1125
- setGlobalGridAttributes, 1126
- setGridNeedsRecalculate, 1126
- setParent, 1126
- setParentLayout, 1127
- setParentWidget, 1127
- setReferenceCoordinatePlane, 1127
- setRubberBandZoomingEnabled, 1128
- setZoomCenter, 1128
- setZoomFactorX, 1128
- setZoomFactorY, 1129
- sharedAxisMasterPlane, 1129
- sizeHint, 1129
- sizeHintChanged, 1129
- sizePolicy, 1130
- takeDiagram, 1130
- TernaryCoordinatePlane, 1104
- topOverlap, 1131
- translate, 1131
- update, 1131
- zoomCenter, 1132
- zoomFactorX, 1132
- zoomFactorY, 1132
- KDChart::TernaryLineDiagram, 1134
- KDChart::TernaryLineDiagram
  - ~TernaryLineDiagram, 1140
  - allowOverlappingDataValueTexts, 1140
  - antiAliasing, 1140
  - attributesModel, 1140
  - attributesModelRootIndex, 1141
  - brush, 1141, 1142
  - calculateDataBoundaries, 1143
  - checkInvariants, 1143
  - columnToIndex, 1144
  - compare, 1144
  - coordinatePlane, 1146
  - dataBoundaries, 1146
  - dataChanged, 1146
  - dataHidden, 1147
  - datasetBrushes, 1147
  - datasetDimension, 1147
  - datasetLabels, 1148
  - datasetMarkers, 1148
  - datasetPens, 1149
  - dataValueAttributes, 1150
  - doItemsLayout, 1151
  - horizontalOffset, 1151
  - indexAt, 1151
  - indexesAt, 1151
  - isHidden, 1152, 1153
  - isIndexHidden, 1153
  - itemRowLabels, 1153
  - layoutChanged, 1154
  - modelsChanged, 1154
  - moveCursor, 1154
  - paint, 1154
  - paintDataValueText, 1156
  - paintDataValueTexts, 1157
  - paintMarker, 1157, 1158
  - paintMarkers, 1160
  - pen, 1160, 1161
  - percentMode, 1161
  - propertiesChanged, 1162
  - resize, 1162
  - scrollTo, 1162
  - setAllowOverlappingDataValueTexts, 1162
  - setAntiAliasing, 1163
  - setAttributesModel, 1163
  - setAttributesModelRootIndex, 1164
  - setBrush, 1164, 1165
  - setCoordinatePlane, 1165
  - setDataBoundariesDirty, 1166
  - setDatasetDimension, 1166

- setDataValueAttributes, 1167
- setHidden, 1168, 1169
- setModel, 1169
- setPen, 1170, 1171
- setPercentMode, 1171
- setRootIndex, 1171
- setSelection, 1172
- setUnitPrefix, 1172
- setUnitSuffix, 1173
- TernaryLineDiagram, 1139
- unitPrefix, 1173, 1174
- unitSuffix, 1174, 1175
- update, 1175
- useDefaultColors, 1175
- useRainbowColors, 1176
- usesExternalAttributesModel, 1176
- useSubduedColors, 1176
- valueForCell, 1177
- verticalOffset, 1177
- visualRect, 1177
- visualRegionForSelection, 1178
- KDChart::TernaryPointDiagram, 1179
- KDChart::TernaryPointDiagram
  - ~TernaryPointDiagram, 1185
  - allowOverlappingDataValueTexts, 1185
  - antiAliasing, 1185
  - attributesModel, 1185
  - attributesModelRootIndex, 1186
  - brush, 1186, 1187
  - calculateDataBoundaries, 1188
  - checkInvariants, 1188
  - columnToIndex, 1188
  - compare, 1189
  - coordinatePlane, 1190
  - dataBoundaries, 1191
  - dataChanged, 1191
  - dataHidden, 1192
  - datasetBrushes, 1192
  - datasetDimension, 1192
  - datasetLabels, 1193
  - datasetMarkers, 1193
  - datasetPens, 1194
  - dataValueAttributes, 1194, 1195
  - doItemsLayout, 1196
  - horizontalOffset, 1196
  - indexAt, 1196
  - indexesAt, 1196
  - isHidden, 1197, 1198
  - isIndexHidden, 1198
  - itemRowLabels, 1198
  - layoutChanged, 1199
  - modelsChanged, 1199
  - moveCursor, 1199
  - paint, 1199
  - paintDataValueText, 1200
  - paintDataValueTexts, 1202
  - paintMarker, 1202, 1203
  - paintMarkers, 1205
  - pen, 1205, 1206
  - percentMode, 1206
  - propertiesChanged, 1207
  - resize, 1207
  - scrollTo, 1207
  - setAllowOverlappingDataValueTexts, 1207
  - setAntiAliasing, 1208
  - setAttributesModel, 1208
  - setAttributesModelRootIndex, 1209
  - setBrush, 1209, 1210
  - setCoordinatePlane, 1210
  - setDataBoundariesDirty, 1211
  - setDatasetDimension, 1211
  - setDataValueAttributes, 1211, 1212
  - setHidden, 1213, 1214
  - setModel, 1214
  - setPen, 1215
  - setPercentMode, 1216
  - setRootIndex, 1216
  - setSelection, 1216
  - setUnitPrefix, 1217
  - setUnitSuffix, 1217, 1218
  - TernaryPointDiagram, 1184
  - unitPrefix, 1218
  - unitSuffix, 1219
  - update, 1220
  - useDefaultColors, 1220
  - useRainbowColors, 1220
  - usesExternalAttributesModel, 1221
  - useSubduedColors, 1221
  - valueForCell, 1221
  - verticalOffset, 1222
  - visualRect, 1222
  - visualRegionForSelection, 1222
- KDChart::TextArea, 1223
- KDChart::TextArea
  - ~TextArea, 1225
  - alignToReferencePoint, 1226
  - areaGeometry, 1226
  - autoReferenceArea, 1226
  - backgroundAttributes, 1226
  - compare, 1227
  - expandingDirections, 1227
  - frameAttributes, 1227
  - geometry, 1227
  - getFrameLeadings, 1228
  - innerRect, 1228
  - intersects, 1228, 1229
  - isEmpty, 1229
  - maximumSize, 1230



- minimumSize, 1230
- mParent, 1239
- mParentLayout, 1239
- paint, 1230
- paintAll, 1231
- paintBackground, 1231
- paintBackgroundAttributes, 1232
- paintCtx, 1233
- paintFrame, 1233
- paintFrameAttributes, 1233
- paintIntoRect, 1234
- parentLayout, 1234
- positionChanged, 1234
- positionHasChanged, 1234
- realFont, 1235
- realFontSize, 1235
- removeFromParentLayout, 1235
- setAutoReferenceArea, 1235
- setBackgroundAttributes, 1236
- setFrameAttributes, 1236
- setGeometry, 1236
- setParentLayout, 1236
- setParentWidget, 1237
- setText, 1237
- setTextAttributes, 1237
- sizeHint, 1238
- sizeHintChanged, 1238
- text, 1238
- TextArea, 1225
- textAttributes, 1239
- KDChart::TextAttributes, 1240
- KDChart::TextAttributes
  - ~TextAttributes, 1242
  - autoRotate, 1242
  - autoShrink, 1242
  - calculatedFont, 1243
  - calculatedFontSize, 1243
  - font, 1243
  - fontSize, 1244
  - hasAbsoluteFontSize, 1244
  - isVisible, 1244
  - minimalFontSize, 1244
  - operator!=, 1245
  - operator=, 1245
  - operator==, 1245
  - pen, 1246
  - rotation, 1246
  - setAutoRotate, 1246
  - setAutoShrink, 1247
  - setFont, 1247
  - setFontSize, 1247
  - setMinimalFontSize, 1248
  - setPen, 1248
  - setRotation, 1248
  - setVisible, 1249
  - TextAttributes, 1241, 1242
- KDChart::TextLayoutItem, 1250
- KDChart::TextLayoutItem
  - autoReferenceArea, 1252
  - expandingDirections, 1253
  - geometry, 1253
  - intersects, 1253, 1254
  - isEmpty, 1254
  - maximumSize, 1254
  - minimumSize, 1254
  - mParent, 1260
  - mParentLayout, 1260
  - paint, 1255
  - paintAll, 1255
  - paintCtx, 1256
  - parentLayout, 1256
  - realFont, 1256
  - realFontSize, 1256
  - removeFromParentLayout, 1256
  - setAutoReferenceArea, 1257
  - setGeometry, 1257
  - setParentLayout, 1257
  - setParentWidget, 1257
  - setText, 1258
  - setTextAttributes, 1258
  - sizeHint, 1258
  - sizeHintChanged, 1259
  - text, 1259
  - textAttributes, 1259
  - TextLayoutItem, 1252
- KDChart::ThreeDBarAttributes, 1261
- KDChart::ThreeDBarAttributes
  - ~ThreeDBarAttributes, 1262
  - angle, 1262
  - depth, 1262
  - isEnabled, 1262
  - operator!=, 1263
  - operator=, 1263
  - operator==, 1263, 1264
  - setAngle, 1264
  - setDepth, 1264
  - setEnabled, 1264
  - setUseShadowColors, 1264
  - ThreeDBarAttributes, 1262
  - useShadowColors, 1265
  - validDepth, 1265
- KDChart::ThreeDLineAttributes, 1266
- KDChart::ThreeDLineAttributes
  - ~ThreeDLineAttributes, 1267
  - depth, 1267
  - isEnabled, 1267
  - lineXRotation, 1267
  - lineYRotation, 1268



- operator!=, 1268
- operator=, 1268
- operator==, 1268, 1269
- setDepth, 1269
- setEnabled, 1269
- setLineXRotation, 1269
- setLineYRotation, 1270
- ThreeDLineAttributes, 1267
- validDepth, 1270
- KDChart::ThreeDPieAttributes, 1271
- KDChart::ThreeDPieAttributes
  - ~ThreeDPieAttributes, 1272
  - depth, 1272
  - isEnabled, 1272
  - operator!=, 1272
  - operator=, 1273
  - operator==, 1273
  - setDepth, 1273
  - setEnabled, 1274
  - setUseShadowColors, 1274
  - ThreeDPieAttributes, 1271, 1272
  - useShadowColors, 1274
  - validDepth, 1274
- KDChart::ValueTrackerAttributes, 1276
- KDChart::ValueTrackerAttributes
  - ~ValueTrackerAttributes, 1277
  - areaBrush, 1277
  - isEnabled, 1277
  - markerSize, 1278
  - operator!=, 1278
  - operator=, 1278
  - operator==, 1278
  - pen, 1278
  - setAreaBrush, 1279
  - setEnabled, 1279
  - setMarkerSize, 1279
  - setPen, 1280
  - ValueTrackerAttributes, 1277
- KDChart::VerticalLineLayoutItem, 1281
- KDChart::VerticalLineLayoutItem
  - expandingDirections, 1282
  - geometry, 1282
  - isEmpty, 1282
  - maximumSize, 1282
  - minimumSize, 1283
  - mParent, 1285
  - mParentLayout, 1285
  - paint, 1283
  - paintAll, 1283
  - paintCtx, 1283
  - parentLayout, 1284
  - removeFromParentLayout, 1284
  - setGeometry, 1284
  - setParentLayout, 1284
  - setParentWidget, 1284
  - sizeHint, 1285
  - sizeHintChanged, 1285
  - VerticalLineLayoutItem, 1282
- KDChart::Widget, 1287
  - ~Widget, 1291
  - addHeaderFooter, 1291
  - addLegend, 1292
  - allHeadersFooters, 1292
  - allLegends, 1292
  - Bar, 1290
  - barDiagram, 1293
  - ChartType, 1290
  - coordinatePlane, 1293
  - diagram, 1293
  - firstHeaderFooter, 1293
  - globalLeadingBottom, 1294
  - globalLeadingLeft, 1294
  - globalLeadingRight, 1294
  - globalLeadingTop, 1294
  - legend, 1295
  - Line, 1290
  - lineDiagram, 1295
  - Normal, 1290
  - NoType, 1290
  - Percent, 1290
  - Pie, 1290
  - pieDiagram, 1295
  - Polar, 1290
  - polarDiagram, 1295
  - replaceHeaderFooter, 1296
  - replaceLegend, 1296
  - resetData, 1296
  - Ring, 1290
  - ringDiagram, 1297
  - Rows, 1290
  - setDataCell, 1297
  - setDataset, 1298
  - setGlobalLeading, 1298
  - setGlobalLeadingBottom, 1299
  - setGlobalLeadingLeft, 1299
  - setGlobalLeadingRight, 1299
  - setGlobalLeadingTop, 1299
  - setSubType, 1300
  - setType, 1300
  - Stacked, 1290
  - SubType, 1290
  - subType, 1302
  - takeHeaderFooter, 1303
  - takeLegend, 1303
  - type, 1303
  - Widget, 1291
- KDChart::ZoomParameters, 1305
- KDChart::ZoomParameters

- center, [1306](#)
- setCenter, [1306](#)
- xCenter, [1306](#)
- xFactor, [1306](#)
- yCenter, [1306](#)
- yFactor, [1306](#)
- ZoomParameters, [1305](#)
- KDCHART\_DATA\_VALUE\_AUTO\_DIGITS
  - KDChartDataValueAttributes.cpp, [1414](#)
- KDCHART\_DECLARE\_DERIVED\_DIAGRAM
  - KDChartGlobal.h, [1428](#)
- KDCHART\_DECLARE\_PRIVATE\_BASE\_-POLYMORPHIC
  - KDChartGlobal.h, [1428](#)
- KDCHART\_DECLARE\_PRIVATE\_BASE\_-POLYMORPHIC\_QWIDGET
  - KDChartGlobal.h, [1428](#)
- KDCHART\_DECLARE\_PRIVATE\_BASE\_-VALUE
  - KDChartGlobal.h, [1428](#)
- KDCHART\_DECLARE\_PRIVATE\_DERIVED
  - KDChartGlobal.h, [1429](#)
- KDCHART\_DECLARE\_PRIVATE\_DERIVED\_-PARENT
  - KDChartGlobal.h, [1429](#)
- KDCHART\_DECLARE\_PRIVATE\_DERIVED\_-QWIDGET
  - KDChartGlobal.h, [1429](#)
- KDCHART\_DECLARE\_SWAP\_BASE
  - KDChartGlobal.h, [1429](#)
- KDCHART\_DECLARE\_SWAP\_DERIVED
  - KDChartGlobal.h, [1430](#)
- KDCHART\_DECLARE\_SWAP\_-SPECIALISATION
  - KDChartGlobal.h, [1430](#)
- KDCHART\_DECLARE\_SWAP\_-SPECIALISATION\_DERIVED
  - KDChartGlobal.h, [1430](#)
- KDCHART\_DERIVED\_PRIVATE\_FOOTER
  - KDChartGlobal.h, [1430](#)
- KDCHART\_IMPL\_DERIVED\_DIAGRAM
  - KDChartGlobal.h, [1430](#)
- KDCHART\_IMPL\_DERIVED\_PLANE
  - KDChartGlobal.h, [1431](#)
- KDChartAbstractArea.cpp, [1347](#)
  - d, [1347](#)
- KDChartAbstractArea.h, [1353](#)
- KDChartAbstractAreaBase.cpp, [1354](#)
  - d, [1354](#)
- KDChartAbstractAreaBase.h, [1355](#)
- KDChartAbstractAreaWidget.cpp, [1357](#)
- KDChartAbstractAreaWidget.cpp
  - d, [1357](#)
- KDChartAbstractAreaWidget.h, [1358](#)
- KDChartAbstractAxis.cpp, [1359](#)
  - d, [1359](#)
- KDChartAbstractAxis.h, [1360](#)
- KDChartAbstractCartesianDiagram.cpp, [1361](#)
  - d, [1361](#)
- KDChartAbstractCartesianDiagram.h, [1362](#)
- KDChartAbstractCoordinatePlane.cpp, [1364](#)
  - d, [1364](#)
- KDChartAbstractCoordinatePlane.h, [1365](#)
- KDChartAbstractDiagram.cpp, [1367](#)
  - d, [1368](#)
- KDChartAbstractDiagram.h, [1369](#)
- KDChartAbstractPieDiagram.cpp, [1371](#)
  - d, [1371](#)
- KDChartAbstractPieDiagram.h, [1372](#)
- KDChartAbstractPolarDiagram.cpp, [1373](#)
  - d, [1373](#)
- KDChartAbstractPolarDiagram.h, [1374](#)
- KDChartAbstractProxyModel.cpp, [1375](#)
- KDChartAbstractProxyModel.h, [1376](#)
- KDChartAbstractTernaryDiagram.cpp, [1377](#)
  - d, [1377](#)
- KDChartAbstractTernaryDiagram.h, [1378](#)
- KDChartAbstractThreeDAttributes.cpp, [1379](#)
  - d, [1379](#)
  - operator<<, [1379](#)
- KDChartAbstractThreeDAttributes.h, [1380](#)
- KDChartAbstractThreeDAttributes.h
  - operator<<, [1381](#)
- KDChartAttributesModel.cpp, [1382](#)
- KDChartAttributesModel.h, [1384](#)
- KDChartBackgroundAttributes.cpp, [1386](#)
  - d, [1386](#)
  - operator<<, [1386](#)
- KDChartBackgroundAttributes.h, [1387](#)
  - operator<<, [1388](#)
  - Q\_DECLARE\_TYPEINFO, [1388](#)
- KDChartBarAttributes.cpp, [1389](#)
  - d, [1389](#)
- KDChartBarAttributes.h, [1390](#)
- KDChartBarDiagram.cpp, [1391](#)

- KDChartBarDiagram.cpp
  - d, [1392](#)
- KDChartBarDiagram.h, [1393](#)
- KDChartBarDiagram\_p.cpp, [1394](#)
- KDChartCartesianAxis.cpp, [1395](#)
- KDChartCartesianAxis.cpp
  - calculateNextLabel, [1397](#)
  - calculateOverlap, [1397](#)
  - d, [1396](#)
  - referenceDiagramIsBarDiagram, [1397](#)
- KDChartCartesianAxis.h, [1399](#)
- KDChartCartesianCoordinatePlane.cpp, [1400](#)
- KDChartCartesianCoordinatePlane.cpp
  - d, [1401](#)
- KDChartCartesianCoordinatePlane.h, [1402](#)
- KDChartCartesianDiagramDataCompressor\_p.cpp, [1403](#)
- KDChartChart.cpp, [1404](#)
- KDChartChart.cpp
  - ADD\_AUTO\_SPACER\_IF\_NEEDED, [1406](#)
  - ADD\_VBOX\_WITH\_LEGENDS, [1406](#)
  - d, [1406](#)
  - findOrCreateLayoutByObjectName, [1406](#)
  - SET\_ALL\_MARGINS\_TO\_ZERO, [1406](#)
- KDChartChart.h, [1408](#)
- KDChartDatasetProxyModel.cpp, [1410](#)
- KDChartDatasetProxyModel.h, [1411](#)
- KDChartDatasetSelector.cpp, [1412](#)
- KDChartDatasetSelector.h, [1413](#)
- KDChartDataValueAttributes.cpp, [1414](#)
- KDChartDataValueAttributes.cpp
  - d, [1414](#)
  - KDCHART\_DATA\_VALUE\_AUTO\_-DIGITS, [1414](#)
  - operator<<, [1415](#)
- KDChartDataValueAttributes.h, [1416](#)
- KDChartDataValueAttributes.h
  - operator<<, [1417](#)
  - Q\_DECLARE\_TYPEINFO, [1417](#)
- KDChartDiagramObserver.cpp, [1418](#)
- KDChartDiagramObserver.h, [1419](#)
- KDChartEnums, [1308](#)
  - GranularitySequence\_10\_20, [1311](#)
  - GranularitySequence\_10\_50, [1311](#)
  - GranularitySequence\_125\_25, [1311](#)
  - GranularitySequence\_25\_50, [1311](#)
  - GranularitySequenceIrregular, [1311](#)
  - LayoutJustOverwrite, [1314](#)
  - LayoutPolicyRotate, [1314](#)
  - LayoutPolicyShiftHorizontally, [1314](#)
  - LayoutPolicyShiftVertically, [1314](#)
  - LayoutPolicyShrinkFontSize, [1314](#)
  - MeasureCalculationModeAbsolute, [1312](#)
  - MeasureCalculationModeAuto, [1312](#)
  - MeasureCalculationModeAutoArea, [1312](#)
  - MeasureCalculationModeAutoOrientation, [1312](#)
  - MeasureCalculationModeRelative, [1312](#)
  - MeasureOrientationAuto, [1313](#)
  - MeasureOrientationHorizontal, [1313](#)
  - MeasureOrientationMaximum, [1313](#)
  - MeasureOrientationMinimum, [1313](#)
  - MeasureOrientationVertical, [1313](#)
  - PositionCenter, [1313](#)
  - PositionEast, [1313](#)
  - PositionFloating, [1313](#)
  - PositionNorth, [1313](#)
  - PositionNorthEast, [1313](#)
  - PositionNorthWest, [1313](#)
  - PositionSouth, [1313](#)
  - PositionSouthEast, [1313](#)
  - PositionSouthWest, [1313](#)
  - PositionUnknown, [1313](#)
  - PositionWest, [1313](#)
- KDChartEnums
  - GranularitySequence, [1310](#)
  - granularitySequenceToString, [1314](#)
  - layoutPolicyToString, [1315](#)
  - MeasureCalculationMode, [1311](#)
  - measureCalculationModeToString, [1315](#)
  - MeasureOrientation, [1312](#)
  - measureOrientationToString, [1316](#)
  - PositionValue, [1313](#)
  - stringToGranularitySequence, [1316](#)
  - stringToLayoutPolicy, [1317](#)
  - stringToMeasureCalculationMode, [1317](#)
  - stringToMeasureOrientation, [1317](#)
  - TextLayoutPolicy, [1313](#)
- KDChartEnums.h, [1420](#)
- KDChartFrameAttributes.cpp, [1422](#)
- KDChartFrameAttributes.cpp
  - d, [1422](#)
  - operator<<, [1422](#)
- KDChartFrameAttributes.h, [1423](#)
- KDChartFrameAttributes.h
  - operator<<, [1424](#)
  - Q\_DECLARE\_TYPEINFO, [1424](#)
- KDChartGlobal.h, [1425](#)
- KDChartGlobal.h
  - \_\_kdab\_\_dereference\_for\_methodcall, [1431](#)
  - KDAB\_SET\_OBJECT\_NAME, [1428](#)
  - KDCHART\_DECLARE\_DERIVED\_-DIAGRAM, [1428](#)
  - KDCHART\_DECLARE\_PRIVATE\_BASE\_-POLYMORPHIC, [1428](#)
  - KDCHART\_DECLARE\_PRIVATE\_BASE\_-POLYMORPHIC\_QWIDGET, [1428](#)

- KDCHART\_DECLARE\_PRIVATE\_BASE\_-  
VALUE, 1428
- KDCHART\_DECLARE\_PRIVATE\_-  
DERIVED, 1429
- KDCHART\_DECLARE\_PRIVATE\_-  
DERIVED\_PARENT, 1429
- KDCHART\_DECLARE\_PRIVATE\_-  
DERIVED\_QWIDGET, 1429
- KDCHART\_DECLARE\_SWAP\_BASE, 1429
- KDCHART\_DECLARE\_SWAP\_DERIVED,  
1430
- KDCHART\_DECLARE\_SWAP\_-  
SPECIALISATION, 1430
- KDCHART\_DECLARE\_SWAP\_-  
SPECIALISATION\_DERIVED, 1430
- KDCHART\_DERIVED\_PRIVATE\_-  
FOOTER, 1430
- KDCHART\_IMPL\_DERIVED\_DIAGRAM,  
1430
- KDCHART\_IMPL\_DERIVED\_PLANE, 1431
- KDChartGridAttributes.cpp, 1432
- KDChartGridAttributes.cpp  
d, 1432
- operator<<, 1432
- KDChartGridAttributes.h, 1433
- KDChartGridAttributes.h  
operator<<, 1433
- Q\_DECLARE\_TYPEINFO, 1434
- KDChartHeaderFooter.cpp, 1435
- KDChartHeaderFooter.cpp  
d, 1436
- KDChartHeaderFooter.h, 1437
- KDChartLayoutItems.cpp, 1438
- KDChartLayoutItems.cpp  
PI, 1440
- rotatedPoint, 1440
- rotatedRect, 1440
- updateCommonBrush, 1440
- KDChartLayoutItems.h, 1442
- KDChartLegend.cpp, 1444
- KDChartLegend.cpp  
d, 1445
- KDChartLegend.h, 1446
- KDChartLineAttributes.cpp, 1447
- KDChartLineAttributes.cpp  
d, 1447
- operator<<, 1447
- KDChartLineAttributes.h, 1448
- KDChartLineAttributes.h  
operator<<, 1449
- Q\_DECLARE\_TYPEINFO, 1449
- KDChartLineDiagram.cpp, 1450
- KDChartLineDiagram.cpp  
d, 1451
- KDChartLineDiagram.h, 1452
- KDChartLineDiagram\_p.cpp, 1453
- KDChartMarkerAttributes.cpp, 1454
- KDChartMarkerAttributes.cpp  
d, 1454
- operator<<, 1455
- KDChartMarkerAttributes.h, 1456
- KDChartMarkerAttributes.h  
operator<<, 1457
- Q\_DECLARE\_TYPEINFO, 1457
- KDChartMeasure.cpp, 1458
- KDChartMeasure.cpp  
operator<<, 1458
- KDChartMeasure.h, 1460
- KDChartMeasure.h  
operator<<, 1461
- KDChartNormalBarDiagram\_p.cpp, 1462
- KDChartNormalLineDiagram\_p.cpp, 1463
- KDChartNormalPlotter\_p.cpp, 1464
- KDChartPaintContext.cpp, 1465
- KDChartPaintContext.cpp  
d, 1465
- KDChartPaintContext.h, 1466
- KDChartPalette.cpp, 1467
- KDChartPalette.cpp  
d, 1467
- makeDefaultPalette, 1467
- makeRainbowPalette, 1468
- makeSubduedPalette, 1468
- KDChartPalette.h, 1470
- KDChartPercentBarDiagram\_p.cpp, 1471
- KDChartPercentLineDiagram\_p.cpp, 1472
- KDChartPieAttributes.cpp, 1473
- KDChartPieAttributes.cpp  
d, 1473
- operator<<, 1473
- KDChartPieAttributes.h, 1474
- KDChartPieAttributes.h  
operator<<, 1474
- Q\_DECLARE\_TYPEINFO, 1475
- KDChartPieDiagram.cpp, 1476
- KDChartPieDiagram.cpp  
buildReferenceRect, 1477
- d, 1477
- KDChartPieDiagram.h, 1478
- KDChartPlotter.cpp, 1479
- KDChartPlotter.cpp  
d, 1479
- KDChartPlotter.h, 1480
- KDChartPlotter\_p.cpp, 1481
- KDChartPolarCoordinatePlane.cpp, 1482
- KDChartPolarCoordinatePlane.cpp  
d, 1483
- KDChartPolarCoordinatePlane.h, 1484

- KDChartPolarDiagram.cpp, 1485
- KDChartPolarDiagram.cpp
  - d, 1485
- KDChartPolarDiagram.h, 1486
- KDChartPosition.cpp, 1487
- KDChartPosition.cpp
  - maxPositionValue, 1488
  - operator<<, 1488
  - staticPositionCenter, 1488
  - staticPositionEast, 1488
  - staticPositionFloating, 1488
  - staticPositionNames, 1488
  - staticPositionNorth, 1489
  - staticPositionNorthEast, 1489
  - staticPositionNorthWest, 1489
  - staticPositionSouth, 1489
  - staticPositionSouthEast, 1489
  - staticPositionSouthWest, 1489
  - staticPositionUnknown, 1489
  - staticPositionWest, 1489
- KDChartPosition.h, 1490
- KDChartPosition.h
  - operator<<, 1491
  - Q\_DECLARE\_TYPEINFO, 1491
- KDChartRelativePosition.cpp, 1492
- KDChartRelativePosition.cpp
  - d, 1492
  - operator<<, 1492
- KDChartRelativePosition.h, 1494
- KDChartRelativePosition.h
  - operator<<, 1495
  - Q\_DECLARE\_TYPEINFO, 1495
- KDChartRingDiagram.cpp, 1496
- KDChartRingDiagram.cpp
  - d, 1496
- KDChartRingDiagram.h, 1497
- KDChartSignalCompressor.cpp, 1498
- KDChartSignalCompressor.h, 1499
- KDChartStackedBarDiagram\_p.cpp, 1500
- KDChartStackedLineDiagram\_p.cpp, 1501
- KDChartTernaryAxis.cpp, 1502
- KDChartTernaryAxis.h, 1503
- KDChartTernaryCoordinatePlane.cpp, 1504
- KDChartTernaryCoordinatePlane.cpp
  - d, 1504
- KDChartTernaryCoordinatePlane.h, 1505
- KDChartTernaryLineDiagram.cpp, 1506
- KDChartTernaryLineDiagram.cpp
  - d, 1506
- KDChartTernaryLineDiagram.h, 1507
- KDChartTernaryPointDiagram.cpp, 1508
- KDChartTernaryPointDiagram.cpp
  - d, 1508
- KDChartTernaryPointDiagram.h, 1509
- KDChartTextArea.cpp, 1510
- KDChartTextArea.h, 1511
- KDChartTextAttributes.cpp, 1512
- KDChartTextAttributes.cpp
  - d, 1512
  - operator<<, 1512
- KDChartTextAttributes.h, 1514
- KDChartTextAttributes.h
  - operator<<, 1515
  - Q\_DECLARE\_TYPEINFO, 1515
- KDChartTextLabelCache.cpp, 1516
- KDChartTextLabelCache.cpp
  - DUMP\_CACHE\_STATS, 1516
  - HitCount, 1517
  - INC\_HIT\_COUNT, 1517
  - INC\_MISS\_COUNT, 1517
  - MissCount, 1517
- KDChartTextLabelCache.h, 1518
- KDChartThreeDBarAttributes.cpp, 1519
- KDChartThreeDBarAttributes.cpp
  - d, 1519
  - operator<<, 1519
- KDChartThreeDBarAttributes.h, 1520
- KDChartThreeDBarAttributes.h
  - operator<<, 1520
  - Q\_DECLARE\_TYPEINFO, 1521
- KDChartThreeDLineAttributes.cpp, 1522
- KDChartThreeDLineAttributes.cpp
  - d, 1522
  - operator<<, 1522
- KDChartThreeDLineAttributes.h, 1523
- KDChartThreeDLineAttributes.h
  - operator<<, 1523
  - Q\_DECLARE\_TYPEINFO, 1524
- KDChartThreeDPieAttributes.cpp, 1525
- KDChartThreeDPieAttributes.cpp
  - d, 1525
  - operator<<, 1525
- KDChartThreeDPieAttributes.h, 1526
- KDChartThreeDPieAttributes.h
  - operator<<, 1526
  - Q\_DECLARE\_TYPEINFO, 1527
- KDChartValueTrackerAttributes.cpp, 1528
- KDChartValueTrackerAttributes.cpp
  - d, 1528
  - operator<<, 1528
- KDChartValueTrackerAttributes.h, 1529
- KDChartValueTrackerAttributes.h
  - operator<<, 1529
  - Q\_DECLARE\_TYPEINFO, 1530
- KDChartWidget.cpp, 1531
- KDChartWidget.cpp
  - d, 1532
  - isCartesian, 1532

- isPolar, [1532](#)
- SET\_SUB\_TYPE, [1532](#)
- TEST\_SUB\_TYPE, [1532](#)
- KDChartWidget.h, [1534](#)
- KDChartZoomParameters.h, [1535](#)
- KDTextDocument, [1319](#)
  - KDTextDocument, [1319](#)
- KDTextDocument
  - ~KDTextDocument, [1320](#)
  - KDTextDocument, [1319](#)
  - minimumSizeHint, [1320](#)
  - sizeHint, [1320](#)
- KDTextDocument.cpp, [1536](#)
- KDTextDocument.h, [1537](#)
- labels
  - KDChart::AbstractAxis, [75](#)
  - KDChart::CartesianAxis, [459](#)
  - KDChart::TernaryAxis, [1084](#)
- layoutChanged
  - KDChart::AbstractCartesianDiagram, [107](#)
  - KDChart::AbstractDiagram, [186](#)
  - KDChart::AbstractPieDiagram, [234](#)
  - KDChart::AbstractPolarDiagram, [281](#)
  - KDChart::AbstractTernaryDiagram, [327](#)
  - KDChart::BarDiagram, [415](#)
  - KDChart::LineDiagram, [699](#)
  - KDChart::PieDiagram, [798](#)
  - KDChart::Plotter, [854](#)
  - KDChart::PolarDiagram, [948](#)
  - KDChart::RingDiagram, [1043](#)
  - KDChart::TernaryLineDiagram, [1154](#)
  - KDChart::TernaryPointDiagram, [1199](#)
- layoutDiagrams
  - KDChart::AbstractCoordinatePlane, [146](#)
  - KDChart::CartesianCoordinatePlane, [512](#)
  - KDChart::PolarCoordinatePlane, [902](#)
  - KDChart::TernaryCoordinatePlane, [1111](#)
- LayoutJustOverwrite
  - KDChartEnums, [1314](#)
- layoutPlanes
  - KDChart::AbstractCartesianDiagram, [108](#)
  - KDChart::AbstractCoordinatePlane, [147](#)
  - KDChart::BarDiagram, [415](#)
  - KDChart::CartesianAxis, [460](#)
  - KDChart::CartesianCoordinatePlane, [514](#)
  - KDChart::LineDiagram, [699](#)
  - KDChart::Plotter, [854](#)
  - KDChart::PolarCoordinatePlane, [903](#)
  - KDChart::TernaryCoordinatePlane, [1113](#)
- LayoutPolicyRotate
  - KDChartEnums, [1314](#)
- LayoutPolicyShiftHorizontally
  - KDChartEnums, [1314](#)
- LayoutPolicyShiftVertically
  - KDChartEnums, [1314](#)
- LayoutPolicyShrinkFontSize
  - KDChartEnums, [1314](#)
- layoutPolicyToString
  - KDChartEnums, [1315](#)
- Left
  - KDChart::CartesianAxis, [451](#)
- leftOverlap
  - KDChart::AbstractArea, [40](#)
  - KDChart::AbstractAxis, [75](#)
  - KDChart::AbstractCoordinatePlane, [147](#)
  - KDChart::CartesianAxis, [460](#)
  - KDChart::CartesianCoordinatePlane, [514](#)
  - KDChart::PolarCoordinatePlane, [903](#)
  - KDChart::TernaryAxis, [1085](#)
  - KDChart::TernaryCoordinatePlane, [1113](#)
- Legend
  - KDChart::Legend, [641](#)
- legend
  - KDChart::Chart, [551](#)
  - KDChart::Widget, [1295](#)
- LegendList
  - KDChart, [33](#)
- legends
  - KDChart::Chart, [552](#)
- LegendStyle
  - KDChart::Legend, [641](#)
- legendStyle
  - KDChart::Legend, [649](#)
- Line
  - KDChart::Widget, [1290](#)
- Linear
  - KDChart::AbstractCoordinatePlane, [139](#)
  - KDChart::CartesianCoordinatePlane, [494](#)
  - KDChart::PolarCoordinatePlane, [893](#)
  - KDChart::TernaryCoordinatePlane, [1104](#)
- LineAttributes
  - KDChart::LineAttributes, [671](#)
- lineAttributes
  - KDChart::LineDiagram, [700](#)
  - KDChart::Plotter, [855](#)
- LineAttributesRole
  - KDChart, [33](#)
- LineDiagram
  - KDChart::LineDiagram, [682](#)
- lineDiagram
  - KDChart::Widget, [1295](#)
- LineLayoutItem
  - KDChart::LineLayoutItem, [734](#)
- LinesOnly
  - KDChart::Legend, [641](#)
- LineType
  - KDChart::LineDiagram, [682](#)



- LineWithMarkerLayoutItem
  - KDChart::LineWithMarkerLayoutItem, 740
- lineXRotation
  - KDChart::ThreeDLineAttributes, 1267
- lineYRotation
  - KDChart::ThreeDLineAttributes, 1268
- Logarithmic
  - KDChart::AbstractCoordinatePlane, 139
  - KDChart::CartesianCoordinatePlane, 494
  - KDChart::PolarCoordinatePlane, 893
  - KDChart::TernaryCoordinatePlane, 1104
- makeDefaultPalette
  - KDChartPalette.cpp, 1467
- makeRainbowPalette
  - KDChartPalette.cpp, 1468
- makeSubduedPalette
  - KDChartPalette.cpp, 1468
- mapFromSource
  - KDChart::AbstractProxyModel, 306
  - KDChart::AttributesModel, 364
  - KDChart::DatasetProxyModel, 575
  - KDChart::PrivateAttributesModel, 1001
- mappingDisabled
  - KDChart::DatasetSelectorWidget, 580
- mapToSource
  - KDChart::AbstractProxyModel, 306
  - KDChart::AttributesModel, 364
  - KDChart::DatasetProxyModel, 575
  - KDChart::PrivateAttributesModel, 1001
- Marker1Pixel
  - KDChart::MarkerAttributes, 746
- Marker4Pixels
  - KDChart::MarkerAttributes, 746
- MarkerAttributes
  - KDChart::MarkerAttributes, 746
- markerAttributes
  - KDChart::DataValueAttributes, 585
  - KDChart::Legend, 649
- MarkerCircle
  - KDChart::MarkerAttributes, 746
- markerColor
  - KDChart::MarkerAttributes, 747
- MarkerCross
  - KDChart::MarkerAttributes, 746
- MarkerDiamond
  - KDChart::MarkerAttributes, 746
- MarkerFastCross
  - KDChart::MarkerAttributes, 746
- MarkerLayoutItem
  - KDChart::MarkerLayoutItem, 752
- MarkerRing
  - KDChart::MarkerAttributes, 746
- MarkersAndLines
  - KDChart::Legend, 641
- markerSize
  - KDChart::MarkerAttributes, 747
  - KDChart::ValueTrackerAttributes, 1278
- MarkersOnly
  - KDChart::Legend, 641
- MarkerSquare
  - KDChart::MarkerAttributes, 746
- MarkerStyle
  - KDChart::MarkerAttributes, 746
- markerStyle
  - KDChart::MarkerAttributes, 747
- MarkerStylesMap
  - KDChart::MarkerAttributes, 746
- markerStylesMap
  - KDChart::MarkerAttributes, 748
- maximumSize
  - KDChart::AbstractCoordinatePlane, 147
  - KDChart::AutoSpacerLayoutItem, 373
  - KDChart::CartesianAxis, 460
  - KDChart::CartesianCoordinatePlane, 514
  - KDChart::HeaderFooter, 619
  - KDChart::HorizontalLineLayoutItem, 632
  - KDChart::LineLayoutItem, 735
  - KDChart::LineWithMarkerLayoutItem, 741
  - KDChart::MarkerLayoutItem, 753
  - KDChart::PolarCoordinatePlane, 903
  - KDChart::TernaryAxis, 1085
  - KDChart::TernaryCoordinatePlane, 1113
  - KDChart::TextArea, 1230
  - KDChart::TextLayoutItem, 1254
  - KDChart::VerticalLineLayoutItem, 1282
- maxPositionValue
  - KDChartPosition.cpp, 1488
- Measure
  - KDChart::Measure, 759
- MeasureCalculationMode
  - KDChartEnums, 1311
- MeasureCalculationModeAbsolute
  - KDChartEnums, 1312
- MeasureCalculationModeAuto
  - KDChartEnums, 1312
- MeasureCalculationModeAutoArea
  - KDChartEnums, 1312
- MeasureCalculationModeAutoOrientation
  - KDChartEnums, 1312
- MeasureCalculationModeRelative
  - KDChartEnums, 1312
- measureCalculationModeToString
  - KDChartEnums, 1315
- MeasureOrientation
  - KDChartEnums, 1312
- MeasureOrientationAuto
  - KDChartEnums, 1313

- MeasureOrientationHorizontal
  - KDChartEnums, [1313](#)
- MeasureOrientationMaximum
  - KDChartEnums, [1313](#)
- MeasureOrientationMinimum
  - KDChartEnums, [1313](#)
- measureOrientationToString
  - KDChartEnums, [1316](#)
- MeasureOrientationVertical
  - KDChartEnums, [1313](#)
- minimalFontSize
  - KDChart::TextAttributes, [1244](#)
- minimumSize
  - KDChart::AbstractCoordinatePlane, [148](#)
  - KDChart::AutoSpacerLayoutItem, [373](#)
  - KDChart::CartesianAxis, [463](#)
  - KDChart::CartesianCoordinatePlane, [515](#)
  - KDChart::HeaderFooter, [619](#)
  - KDChart::HorizontalLineLayoutItem, [633](#)
  - KDChart::LineLayoutItem, [735](#)
  - KDChart::LineWithMarkerLayoutItem, [741](#)
  - KDChart::MarkerLayoutItem, [753](#)
  - KDChart::PolarCoordinatePlane, [904](#)
  - KDChart::TernaryAxis, [1085](#)
  - KDChart::TernaryCoordinatePlane, [1114](#)
  - KDChart::TextArea, [1230](#)
  - KDChart::TextLayoutItem, [1254](#)
  - KDChart::VerticalLineLayoutItem, [1283](#)
- minimumSizeHint
  - KDChart::AbstractCoordinatePlane, [148](#)
  - KDChart::CartesianCoordinatePlane, [515](#)
  - KDChart::Legend, [649](#)
  - KDChart::PolarCoordinatePlane, [904](#)
  - KDChart::TernaryCoordinatePlane, [1114](#)
  - KDTextDocument, [1320](#)
- MissCount
  - KDChartTextLabelCache.cpp, [1517](#)
- MissingValuesAreBridged
  - KDChart::LineAttributes, [671](#)
- MissingValuesHideSegments
  - KDChart::LineAttributes, [671](#)
- MissingValuesPolicy
  - KDChart::LineAttributes, [670](#)
- missingValuesPolicy
  - KDChart::LineAttributes, [672](#)
- MissingValuesPolicyIgnored
  - KDChart::LineAttributes, [671](#)
- MissingValuesShownAsZero
  - KDChart::LineAttributes, [671](#)
- modelData
  - KDChart::AttributesModel, [365](#)
  - KDChart::PrivateAttributesModel, [1002](#)
- modelDataMap
  - KDChart::AttributesModel, [365](#)
- KDChart::PrivateAttributesModel, [1002](#)
- modelsChanged
  - KDChart::AbstractCartesianDiagram, [108](#)
  - KDChart::AbstractDiagram, [186](#)
  - KDChart::AbstractPieDiagram, [234](#)
  - KDChart::AbstractPolarDiagram, [281](#)
  - KDChart::AbstractTernaryDiagram, [327](#)
  - KDChart::BarDiagram, [416](#)
  - KDChart::LineDiagram, [701](#)
  - KDChart::PieDiagram, [799](#)
  - KDChart::Plotter, [856](#)
  - KDChart::PolarDiagram, [948](#)
  - KDChart::RingDiagram, [1044](#)
  - KDChart::TernaryLineDiagram, [1154](#)
  - KDChart::TernaryPointDiagram, [1199](#)
- mouseDoubleClickEvent
  - KDChart::AbstractCoordinatePlane, [148](#)
  - KDChart::CartesianCoordinatePlane, [515](#)
  - KDChart::Chart, [552](#)
  - KDChart::PolarCoordinatePlane, [904](#)
  - KDChart::TernaryCoordinatePlane, [1114](#)
- mouseMoveEvent
  - KDChart::AbstractCoordinatePlane, [148](#)
  - KDChart::CartesianCoordinatePlane, [516](#)
  - KDChart::Chart, [553](#)
  - KDChart::PolarCoordinatePlane, [905](#)
  - KDChart::TernaryCoordinatePlane, [1115](#)
- mousePressEvent
  - KDChart::AbstractCoordinatePlane, [149](#)
  - KDChart::CartesianCoordinatePlane, [516](#)
  - KDChart::Chart, [553](#)
  - KDChart::PolarCoordinatePlane, [905](#)
  - KDChart::TernaryCoordinatePlane, [1115](#)
- mouseReleaseEvent
  - KDChart::AbstractCoordinatePlane, [150](#)
  - KDChart::CartesianCoordinatePlane, [517](#)
  - KDChart::Chart, [554](#)
  - KDChart::PolarCoordinatePlane, [906](#)
  - KDChart::TernaryCoordinatePlane, [1116](#)
- moveCursor
  - KDChart::AbstractCartesianDiagram, [108](#)
  - KDChart::AbstractDiagram, [186](#)
  - KDChart::AbstractPieDiagram, [234](#)
  - KDChart::AbstractPolarDiagram, [281](#)
  - KDChart::AbstractTernaryDiagram, [327](#)
  - KDChart::BarDiagram, [416](#)
  - KDChart::LineDiagram, [701](#)
  - KDChart::PieDiagram, [799](#)
  - KDChart::Plotter, [856](#)
  - KDChart::PolarDiagram, [949](#)
  - KDChart::RingDiagram, [1044](#)
  - KDChart::TernaryLineDiagram, [1154](#)
  - KDChart::TernaryPointDiagram, [1199](#)
- mParent



- KDChart::AbstractArea, [47](#)
- KDChart::AbstractAxis, [85](#)
- KDChart::AbstractCoordinatePlane, [165](#)
- KDChart::AbstractLayoutItem, [212](#)
- KDChart::AutoSpacerLayoutItem, [377](#)
- KDChart::CartesianAxis, [485](#)
- KDChart::CartesianCoordinatePlane, [542](#)
- KDChart::HeaderFooter, [629](#)
- KDChart::HorizontalLineLayoutItem, [635](#)
- KDChart::LineLayoutItem, [738](#)
- KDChart::LineWithMarkerLayoutItem, [744](#)
- KDChart::MarkerLayoutItem, [756](#)
- KDChart::PolarCoordinatePlane, [925](#)
- KDChart::TernaryAxis, [1098](#)
- KDChart::TernaryCoordinatePlane, [1133](#)
- KDChart::TextArea, [1239](#)
- KDChart::TextLayoutItem, [1260](#)
- KDChart::VerticalLineLayoutItem, [1285](#)
- mParentLayout
  - KDChart::AbstractArea, [47](#)
  - KDChart::AbstractAxis, [85](#)
  - KDChart::AbstractCoordinatePlane, [165](#)
  - KDChart::AbstractLayoutItem, [212](#)
  - KDChart::AutoSpacerLayoutItem, [377](#)
  - KDChart::CartesianAxis, [485](#)
  - KDChart::CartesianCoordinatePlane, [542](#)
  - KDChart::HeaderFooter, [629](#)
  - KDChart::HorizontalLineLayoutItem, [635](#)
  - KDChart::LineLayoutItem, [738](#)
  - KDChart::LineWithMarkerLayoutItem, [744](#)
  - KDChart::MarkerLayoutItem, [757](#)
  - KDChart::PolarCoordinatePlane, [925](#)
  - KDChart::TernaryAxis, [1098](#)
  - KDChart::TernaryCoordinatePlane, [1133](#)
  - KDChart::TextArea, [1239](#)
  - KDChart::TextLayoutItem, [1260](#)
  - KDChart::VerticalLineLayoutItem, [1285](#)
- mPositionCenter
  - KDChart::PositionPoints, [988](#)
- mPositionEast
  - KDChart::PositionPoints, [988](#)
- mPositionNorth
  - KDChart::PositionPoints, [988](#)
- mPositionNorthEast
  - KDChart::PositionPoints, [988](#)
- mPositionNorthWest
  - KDChart::PositionPoints, [988](#)
- mPositionSouth
  - KDChart::PositionPoints, [988](#)
- mPositionSouthEast
  - KDChart::PositionPoints, [988](#)
- mPositionSouthWest
  - KDChart::PositionPoints, [988](#)
- mPositionUnknown
  - KDChart::PositionPoints, [988](#)
- mPositionWest
  - KDChart::PositionPoints, [988](#)
- name
  - KDChart::Position, [981](#)
- names
  - KDChart::Position, [981](#)
- needLayoutPlanes
  - KDChart::AbstractCoordinatePlane, [150](#)
  - KDChart::CartesianCoordinatePlane, [518](#)
  - KDChart::PolarCoordinatePlane, [907](#)
  - KDChart::TernaryCoordinatePlane, [1117](#)
- needRelayout
  - KDChart::AbstractCoordinatePlane, [151](#)
  - KDChart::CartesianCoordinatePlane, [518](#)
  - KDChart::PolarCoordinatePlane, [907](#)
  - KDChart::TernaryCoordinatePlane, [1117](#)
- needSizeHint
  - KDChart::AbstractAreaWidget, [59](#)
  - KDChart::Legend, [650](#)
- needUpdate
  - KDChart::AbstractCoordinatePlane, [151](#)
  - KDChart::CartesianCoordinatePlane, [518](#)
  - KDChart::PolarCoordinatePlane, [907](#)
  - KDChart::TernaryCoordinatePlane, [1117](#)
- negativePosition
  - KDChart::DataValueAttributes, [585](#)
- Norm\_B\_A
  - TernaryConstants.cpp, [1541](#)
  - TernaryConstants.h, [1543](#)
- Norm\_B\_C
  - TernaryConstants.cpp, [1541](#)
  - TernaryConstants.h, [1543](#)
- Norm\_C\_A
  - TernaryConstants.cpp, [1541](#)
  - TernaryConstants.h, [1543](#)
- Normal
  - KDChart::BarDiagram, [398](#)
  - KDChart::LineDiagram, [682](#)
  - KDChart::Plotter, [838](#)
  - KDChart::Widget, [1290](#)
- North
  - KDChart::Position, [983](#)
- NorthEast
  - KDChart::Position, [983](#)
- NorthWest
  - KDChart::Position, [983](#)
- NoType
  - KDChart::Widget, [1290](#)
- numberOfAbscissaSegments
  - KDChart::AbstractCartesianDiagram, [108](#)
  - KDChart::BarDiagram, [416](#)
  - KDChart::LineDiagram, [701](#)

- KDChart::Plotter, [856](#)
- numberOfGridRings
  - KDChart::AbstractPieDiagram, [234](#)
  - KDChart::AbstractPolarDiagram, [281](#)
  - KDChart::PieDiagram, [799](#)
  - KDChart::PolarDiagram, [949](#)
  - KDChart::RingDiagram, [1044](#)
- numberOfOrdinateSegments
  - KDChart::AbstractCartesianDiagram, [108](#)
  - KDChart::BarDiagram, [416](#)
  - KDChart::LineDiagram, [701](#)
  - KDChart::Plotter, [856](#)
- numberOfValuesPerDataset
  - KDChart::AbstractPieDiagram, [234](#)
  - KDChart::AbstractPolarDiagram, [282](#)
  - KDChart::PieDiagram, [799](#)
  - KDChart::PolarDiagram, [949](#)
  - KDChart::RingDiagram, [1044](#)
- observedBy
  - KDChart::AbstractAxis, [75](#)
  - KDChart::CartesianAxis, [464](#)
  - KDChart::TernaryAxis, [1086](#)
- operator!=
  - KDChart::AbstractThreeDAttributes, [351](#)
  - KDChart::BackgroundAttributes, [380](#)
  - KDChart::BarAttributes, [385](#)
  - KDChart::DataDimension, [568](#)
  - KDChart::DataValueAttributes, [586](#)
  - KDChart::FrameAttributes, [597](#)
  - KDChart::GridAttributes, [606](#)
  - KDChart::LineAttributes, [672](#)
  - KDChart::MarkerAttributes, [748](#)
  - KDChart::Measure, [761](#)
  - KDChart::PieAttributes, [774](#)
  - KDChart::Position, [981](#)
  - KDChart::RelativePosition, [1010](#)
  - KDChart::TextAttributes, [1245](#)
  - KDChart::ThreeDBarAttributes, [1263](#)
  - KDChart::ThreeDLineAttributes, [1268](#)
  - KDChart::ThreeDPieAttributes, [1272](#)
  - KDChart::ValueTrackerAttributes, [1278](#)
- operator<<
  - KDChart, [34](#)
  - KDChartAbstractThreeDAttributes.cpp, [1379](#)
  - KDChartAbstractThreeDAttributes.h, [1381](#)
  - KDChartBackgroundAttributes.cpp, [1386](#)
  - KDChartBackgroundAttributes.h, [1388](#)
  - KDChartDataValueAttributes.cpp, [1415](#)
  - KDChartDataValueAttributes.h, [1417](#)
  - KDChartFrameAttributes.cpp, [1422](#)
  - KDChartFrameAttributes.h, [1424](#)
  - KDChartGridAttributes.cpp, [1432](#)
  - KDChartGridAttributes.h, [1433](#)
  - KDChartLineAttributes.cpp, [1447](#)
  - KDChartLineAttributes.h, [1449](#)
  - KDChartMarkerAttributes.cpp, [1455](#)
  - KDChartMarkerAttributes.h, [1457](#)
  - KDChartMeasure.cpp, [1458](#)
  - KDChartMeasure.h, [1461](#)
  - KDChartPieAttributes.cpp, [1473](#)
  - KDChartPieAttributes.h, [1474](#)
  - KDChartPosition.cpp, [1488](#)
  - KDChartPosition.h, [1491](#)
  - KDChartRelativePosition.cpp, [1492](#)
  - KDChartRelativePosition.h, [1495](#)
  - KDChartTextAttributes.cpp, [1512](#)
  - KDChartTextAttributes.h, [1515](#)
  - KDChartThreeDBarAttributes.cpp, [1519](#)
  - KDChartThreeDBarAttributes.h, [1520](#)
  - KDChartThreeDLineAttributes.cpp, [1522](#)
  - KDChartThreeDLineAttributes.h, [1523](#)
  - KDChartThreeDPieAttributes.cpp, [1525](#)
  - KDChartThreeDPieAttributes.h, [1526](#)
  - KDChartValueTrackerAttributes.cpp, [1528](#)
  - KDChartValueTrackerAttributes.h, [1529](#)
  - TernaryPoint.cpp, [1544](#)
  - TernaryPoint.h, [1546](#)
- operator=
  - KDChart::AbstractThreeDAttributes, [351](#)
  - KDChart::BackgroundAttributes, [380](#)
  - KDChart::BarAttributes, [385](#)
  - KDChart::DataValueAttributes, [586](#)
  - KDChart::FrameAttributes, [597](#)
  - KDChart::GridAttributes, [606](#)
  - KDChart::LineAttributes, [672](#)
  - KDChart::MarkerAttributes, [748](#)
  - KDChart::Measure, [761](#)
  - KDChart::Palette, [771](#)
  - KDChart::PieAttributes, [774](#)
  - KDChart::RelativePosition, [1011](#)
  - KDChart::TextAttributes, [1245](#)
  - KDChart::ThreeDBarAttributes, [1263](#)
  - KDChart::ThreeDLineAttributes, [1268](#)
  - KDChart::ThreeDPieAttributes, [1273](#)
  - KDChart::ValueTrackerAttributes, [1278](#)
- operator==
  - KDChart::AbstractThreeDAttributes, [351](#)
  - KDChart::BackgroundAttributes, [380](#)
  - KDChart::BarAttributes, [386](#)
  - KDChart::DataDimension, [568](#)
  - KDChart::DataValueAttributes, [586](#)
  - KDChart::FrameAttributes, [597](#)
  - KDChart::GridAttributes, [606](#)
  - KDChart::LineAttributes, [672](#)
  - KDChart::MarkerAttributes, [748](#)
  - KDChart::Measure, [761](#)
  - KDChart::PieAttributes, [775](#)

- KDChart::Position, [982](#)
- KDChart::RelativePosition, [1011](#)
- KDChart::TextAttributes, [1245](#)
- KDChart::ThreeDBarAttributes, [1263](#), [1264](#)
- KDChart::ThreeDLineAttributes, [1268](#), [1269](#)
- KDChart::ThreeDPieAttributes, [1273](#)
- KDChart::ValueTrackerAttributes, [1278](#)
- Option
  - KDChart::Position, [978](#)
- orientation
  - KDChart::Legend, [650](#)
- padding
  - KDChart::FrameAttributes, [597](#)
- paint
  - KDChart::AbstractArea, [40](#)
  - KDChart::AbstractAreaWidget, [59](#)
  - KDChart::AbstractAxis, [76](#)
  - KDChart::AbstractCartesianDiagram, [109](#)
  - KDChart::AbstractCoordinatePlane, [151](#)
  - KDChart::AbstractDiagram, [186](#)
  - KDChart::AbstractLayoutItem, [210](#)
  - KDChart::AbstractPieDiagram, [234](#)
  - KDChart::AbstractPolarDiagram, [282](#)
  - KDChart::AutoSpacerLayoutItem, [373](#)
  - KDChart::BarDiagram, [417](#)
  - KDChart::CartesianAxis, [464](#)
  - KDChart::CartesianCoordinatePlane, [518](#)
  - KDChart::Chart, [554](#)
  - KDChart::HeaderFooter, [619](#)
  - KDChart::HorizontalLineLayoutItem, [633](#)
  - KDChart::Legend, [650](#)
  - KDChart::LineDiagram, [701](#)
  - KDChart::LineLayoutItem, [735](#)
  - KDChart::LineWithMarkerLayoutItem, [741](#)
  - KDChart::MarkerLayoutItem, [753](#)
  - KDChart::PieDiagram, [799](#)
  - KDChart::Plotter, [856](#)
  - KDChart::PolarCoordinatePlane, [907](#)
  - KDChart::PolarDiagram, [949](#)
  - KDChart::RingDiagram, [1044](#)
  - KDChart::TernaryAxis, [1086](#)
  - KDChart::TernaryCoordinatePlane, [1117](#)
  - KDChart::TernaryLineDiagram, [1154](#)
  - KDChart::TernaryPointDiagram, [1199](#)
  - KDChart::TextArea, [1230](#)
  - KDChart::TextLayoutItem, [1255](#)
  - KDChart::VerticalLineLayoutItem, [1283](#)
- paintAll
  - KDChart::AbstractArea, [41](#)
  - KDChart::AbstractAreaWidget, [59](#)
  - KDChart::AbstractAxis, [76](#)
  - KDChart::AbstractCoordinatePlane, [151](#)
  - KDChart::AbstractLayoutItem, [210](#)
  - KDChart::AutoSpacerLayoutItem, [374](#)
  - KDChart::CartesianAxis, [464](#)
  - KDChart::CartesianCoordinatePlane, [519](#)
  - KDChart::HeaderFooter, [620](#)
  - KDChart::HorizontalLineLayoutItem, [633](#)
  - KDChart::Legend, [651](#)
  - KDChart::LineLayoutItem, [735](#)
  - KDChart::LineWithMarkerLayoutItem, [741](#)
  - KDChart::MarkerLayoutItem, [753](#)
  - KDChart::PolarCoordinatePlane, [908](#)
  - KDChart::TernaryAxis, [1086](#)
  - KDChart::TernaryCoordinatePlane, [1118](#)
  - KDChart::TextArea, [1231](#)
  - KDChart::TextLayoutItem, [1255](#)
  - KDChart::VerticalLineLayoutItem, [1283](#)
- paintBackground
  - KDChart::AbstractArea, [41](#)
  - KDChart::AbstractAreaBase, [51](#)
  - KDChart::AbstractAreaWidget, [60](#)
  - KDChart::AbstractAxis, [76](#)
  - KDChart::AbstractCoordinatePlane, [152](#)
  - KDChart::CartesianAxis, [465](#)
  - KDChart::CartesianCoordinatePlane, [520](#)
  - KDChart::HeaderFooter, [621](#)
  - KDChart::Legend, [652](#)
  - KDChart::PolarCoordinatePlane, [908](#)
  - KDChart::TernaryAxis, [1086](#)
  - KDChart::TernaryCoordinatePlane, [1119](#)
  - KDChart::TextArea, [1231](#)
- paintBackgroundAttributes
  - KDChart::AbstractArea, [42](#)
  - KDChart::AbstractAreaBase, [51](#)
  - KDChart::AbstractAreaWidget, [61](#)
  - KDChart::AbstractAxis, [77](#)
  - KDChart::AbstractCoordinatePlane, [152](#)
  - KDChart::CartesianAxis, [465](#)
  - KDChart::CartesianCoordinatePlane, [520](#)
  - KDChart::HeaderFooter, [621](#)
  - KDChart::Legend, [652](#)
  - KDChart::PolarCoordinatePlane, [909](#)
  - KDChart::TernaryAxis, [1087](#)
  - KDChart::TernaryCoordinatePlane, [1119](#)
  - KDChart::TextArea, [1232](#)
- PaintContext
  - KDChart::PaintContext, [765](#)
- paintCtx
  - KDChart::AbstractArea, [43](#)
  - KDChart::AbstractAxis, [78](#)
  - KDChart::AbstractCoordinatePlane, [153](#)
  - KDChart::AbstractLayoutItem, [210](#)
  - KDChart::AutoSpacerLayoutItem, [374](#)
  - KDChart::CartesianAxis, [466](#)
  - KDChart::CartesianCoordinatePlane, [521](#)
  - KDChart::HeaderFooter, [622](#)

- KDChart::HorizontalLineLayoutItem, 633
- KDChart::LineLayoutItem, 736
- KDChart::LineWithMarkerLayoutItem, 742
- KDChart::MarkerLayoutItem, 754
- KDChart::PolarCoordinatePlane, 910
- KDChart::TernaryAxis, 1088
- KDChart::TernaryCoordinatePlane, 1120
- KDChart::TextArea, 1233
- KDChart::TextLayoutItem, 1256
- KDChart::VerticalLineLayoutItem, 1283
- paintDataValueText
  - KDChart::AbstractCartesianDiagram, 109
  - KDChart::AbstractDiagram, 187
  - KDChart::AbstractPieDiagram, 235
  - KDChart::AbstractPolarDiagram, 282
  - KDChart::AbstractTernaryDiagram, 328
  - KDChart::BarDiagram, 417
  - KDChart::LineDiagram, 702
  - KDChart::PieDiagram, 802
  - KDChart::Plotter, 857
  - KDChart::PolarDiagram, 950
  - KDChart::RingDiagram, 1045
  - KDChart::TernaryLineDiagram, 1156
  - KDChart::TernaryPointDiagram, 1200
- paintDataValueTexts
  - KDChart::AbstractCartesianDiagram, 110
  - KDChart::AbstractDiagram, 188
  - KDChart::AbstractPieDiagram, 236
  - KDChart::AbstractPolarDiagram, 283
  - KDChart::AbstractTernaryDiagram, 329
  - KDChart::BarDiagram, 419
  - KDChart::LineDiagram, 704
  - KDChart::PieDiagram, 804
  - KDChart::Plotter, 859
  - KDChart::PolarDiagram, 952
  - KDChart::RingDiagram, 1047
  - KDChart::TernaryLineDiagram, 1157
  - KDChart::TernaryPointDiagram, 1202
- paintDevice
  - KDChart::GlobalMeasureScaling, 601
- painter
  - KDChart::PaintContext, 766
- paintEvent
  - KDChart::AbstractAreaWidget, 62
  - KDChart::BarDiagram, 419
  - KDChart::CartesianCoordinatePlane, 521
  - KDChart::Chart, 556
  - KDChart::Legend, 653
  - KDChart::LineDiagram, 704
  - KDChart::PieDiagram, 804
  - KDChart::Plotter, 859
  - KDChart::PolarCoordinatePlane, 910
  - KDChart::PolarDiagram, 952
  - KDChart::RingDiagram, 1047
- paintFrame
  - KDChart::AbstractArea, 43
  - KDChart::AbstractAreaBase, 52
  - KDChart::AbstractAreaWidget, 62
  - KDChart::AbstractAxis, 78
  - KDChart::AbstractCoordinatePlane, 153
  - KDChart::CartesianAxis, 476
  - KDChart::CartesianCoordinatePlane, 521
  - KDChart::HeaderFooter, 622
  - KDChart::Legend, 653
  - KDChart::PolarCoordinatePlane, 910
  - KDChart::TernaryAxis, 1088
  - KDChart::TernaryCoordinatePlane, 1120
  - KDChart::TextArea, 1233
- paintFrameAttributes
  - KDChart::AbstractArea, 43
  - KDChart::AbstractAreaBase, 52
  - KDChart::AbstractAreaWidget, 62
  - KDChart::AbstractAxis, 78
  - KDChart::AbstractCoordinatePlane, 154
  - KDChart::CartesianAxis, 476
  - KDChart::CartesianCoordinatePlane, 522
  - KDChart::HeaderFooter, 622
  - KDChart::Legend, 654
  - KDChart::PolarCoordinatePlane, 910
  - KDChart::TernaryAxis, 1088
  - KDChart::TernaryCoordinatePlane, 1120
  - KDChart::TextArea, 1233
- paintIntoRect
  - KDChart::AbstractArea, 44
  - KDChart::AbstractAreaWidget, 63
  - KDChart::AbstractAxis, 79
  - KDChart::AbstractCoordinatePlane, 154
  - KDChart::CartesianAxis, 477
  - KDChart::CartesianCoordinatePlane, 522
  - KDChart::HeaderFooter, 623
  - KDChart::Legend, 654
  - KDChart::LineLayoutItem, 736
  - KDChart::MarkerLayoutItem, 754
  - KDChart::PolarCoordinatePlane, 911
  - KDChart::TernaryAxis, 1089
  - KDChart::TernaryCoordinatePlane, 1121
  - KDChart::TextArea, 1234
- paintMarker
  - KDChart::AbstractCartesianDiagram, 111
  - KDChart::AbstractDiagram, 189
  - KDChart::AbstractPieDiagram, 237
  - KDChart::AbstractPolarDiagram, 284
  - KDChart::AbstractTernaryDiagram, 330
  - KDChart::BarDiagram, 419, 420
  - KDChart::LineDiagram, 704, 705
  - KDChart::PieDiagram, 804, 805
  - KDChart::Plotter, 859, 860
  - KDChart::PolarDiagram, 952, 953

- KDChart::RingDiagram, 1047, 1048
- KDChart::TernaryLineDiagram, 1157, 1158
- KDChart::TernaryPointDiagram, 1202, 1203
- paintMarkers
  - KDChart::AbstractCartesianDiagram, 113
  - KDChart::AbstractDiagram, 191
  - KDChart::AbstractPieDiagram, 239
  - KDChart::AbstractPolarDiagram, 286
  - KDChart::AbstractTernaryDiagram, 332
  - KDChart::BarDiagram, 422
  - KDChart::LineDiagram, 707
  - KDChart::PieDiagram, 807
  - KDChart::Plotter, 862
  - KDChart::PolarDiagram, 955
  - KDChart::RingDiagram, 1050
  - KDChart::TernaryLineDiagram, 1160
  - KDChart::TernaryPointDiagram, 1205
- paintPolarMarkers
  - KDChart::PolarDiagram, 955
- Palette
  - KDChart::Palette, 769
- PaletteType
  - KDChart::AttributesModel, 355
  - KDChart::PrivateAttributesModel, 992
- paletteType
  - KDChart::AttributesModel, 365
  - KDChart::PrivateAttributesModel, 1002
- PaletteTypeDefault
  - KDChart::AttributesModel, 355
  - KDChart::PrivateAttributesModel, 992
- PaletteTypeRainbow
  - KDChart::AttributesModel, 355
  - KDChart::PrivateAttributesModel, 992
- PaletteTypeSubdued
  - KDChart::AttributesModel, 355
  - KDChart::PrivateAttributesModel, 992
- parent
  - KDChart::AbstractCoordinatePlane, 154, 155
  - KDChart::AbstractProxyModel, 307
  - KDChart::AttributesModel, 366
  - KDChart::CartesianCoordinatePlane, 522, 523
  - KDChart::DatasetProxyModel, 576
  - KDChart::PolarCoordinatePlane, 911
  - KDChart::PrivateAttributesModel, 1002
  - KDChart::TernaryCoordinatePlane, 1121
- parentLayout
  - KDChart::AbstractArea, 44
  - KDChart::AbstractAxis, 79
  - KDChart::AbstractCoordinatePlane, 155
  - KDChart::AbstractLayoutItem, 211
  - KDChart::AutoSpacerLayoutItem, 374
  - KDChart::CartesianAxis, 477
  - KDChart::CartesianCoordinatePlane, 523
  - KDChart::HeaderFooter, 623
  - KDChart::HorizontalLineLayoutItem, 634
  - KDChart::LineLayoutItem, 736
  - KDChart::LineWithMarkerLayoutItem, 742
  - KDChart::MarkerLayoutItem, 755
  - KDChart::PolarCoordinatePlane, 912
  - KDChart::TernaryAxis, 1089
  - KDChart::TernaryCoordinatePlane, 1122
  - KDChart::TextArea, 1234
  - KDChart::TextLayoutItem, 1256
  - KDChart::VerticalLineLayoutItem, 1284
- pen
  - KDChart::AbstractCartesianDiagram, 113, 114
  - KDChart::AbstractDiagram, 191, 192
  - KDChart::AbstractPieDiagram, 239, 240
  - KDChart::AbstractPolarDiagram, 287
  - KDChart::AbstractTernaryDiagram, 332, 333
  - KDChart::BarDiagram, 422, 423
  - KDChart::FrameAttributes, 598
  - KDChart::Legend, 655
  - KDChart::LineDiagram, 707, 708
  - KDChart::MarkerAttributes, 749
  - KDChart::PieDiagram, 807, 808
  - KDChart::Plotter, 862, 863
  - KDChart::PolarDiagram, 955, 956
  - KDChart::RingDiagram, 1050, 1051
  - KDChart::TernaryLineDiagram, 1160, 1161
  - KDChart::TernaryPointDiagram, 1205, 1206
  - KDChart::TextAttributes, 1246
  - KDChart::ValueTrackerAttributes, 1278
  - PrerenderedLabel, 1328
- pens
  - KDChart::Legend, 655
- Percent
  - KDChart::BarDiagram, 398
  - KDChart::LineDiagram, 682
  - KDChart::Widget, 1290
- percentMode
  - KDChart::AbstractCartesianDiagram, 115
  - KDChart::AbstractDiagram, 193
  - KDChart::AbstractPieDiagram, 241
  - KDChart::AbstractPolarDiagram, 288
  - KDChart::AbstractTernaryDiagram, 334
  - KDChart::BarDiagram, 423
  - KDChart::LineDiagram, 708
  - KDChart::PieDiagram, 808
  - KDChart::Plotter, 863
  - KDChart::PolarDiagram, 957
  - KDChart::RingDiagram, 1051
  - KDChart::TernaryLineDiagram, 1161
  - KDChart::TernaryPointDiagram, 1206
- PI
  - KDChartLayoutItems.cpp, 1440
- Pie



- KDChart::Widget, 1290
- PieAttributes
  - KDChart::PieAttributes, 773
- pieAttributes
  - KDChart::AbstractPieDiagram, 241
  - KDChart::PieDiagram, 809
  - KDChart::RingDiagram, 1052
- PieAttributesRole
  - KDChart, 33
- PieDiagram
  - KDChart::PieDiagram, 783
- pieDiagram
  - KDChart::Widget, 1295
- pixmap
  - KDChart::BackgroundAttributes, 381
  - PrerenderedElement, 1323
  - PrerenderedLabel, 1328
- pixmapMode
  - KDChart::BackgroundAttributes, 381
- Plotter
  - KDChart::Plotter, 838
- PlotType
  - KDChart::Plotter, 838
- point
  - KDChart::PositionPoints, 987
- Polar
  - KDChart::Widget, 1290
- PolarCoordinatePlane
  - KDChart::PolarCoordinatePlane, 893
- polarCoordinatePlane
  - KDChart::AbstractPieDiagram, 242
  - KDChart::AbstractPolarDiagram, 288
  - KDChart::PieDiagram, 809
  - KDChart::PolarDiagram, 957
  - KDChart::RingDiagram, 1052
- PolarDiagram
  - KDChart::PolarDiagram, 933
- polarDiagram
  - KDChart::Widget, 1295
- Position
  - KDChart::CartesianAxis, 451
  - KDChart::Position, 978
- position
  - KDChart::CartesianAxis, 477
  - KDChart::DataValueAttributes, 587
  - KDChart::HeaderFooter, 623
  - KDChart::Legend, 655
  - KDChart::TernaryAxis, 1089
  - PrerenderedElement, 1323
  - PrerenderedLabel, 1328
- PositionCenter
  - KDChartEnums, 1313
- positionChanged
  - KDChart::AbstractArea, 44
- KDChart::AbstractAreaWidget, 63
- KDChart::AbstractAxis, 79
- KDChart::AbstractCoordinatePlane, 155
- KDChart::CartesianAxis, 477
- KDChart::CartesianCoordinatePlane, 523
- KDChart::HeaderFooter, 624
- KDChart::Legend, 655
- KDChart::PolarCoordinatePlane, 912
- KDChart::TernaryAxis, 1090
- KDChart::TernaryCoordinatePlane, 1122
- KDChart::TextArea, 1234
- PositionEast
  - KDChartEnums, 1313
- PositionFloating
  - KDChartEnums, 1313
- positionHasChanged
  - KDChart::AbstractArea, 44
  - KDChart::AbstractAreaBase, 53
  - KDChart::AbstractAreaWidget, 63
  - KDChart::AbstractAxis, 79
  - KDChart::AbstractCoordinatePlane, 155
  - KDChart::CartesianAxis, 478
  - KDChart::CartesianCoordinatePlane, 523
  - KDChart::HeaderFooter, 624
  - KDChart::Legend, 656
  - KDChart::PolarCoordinatePlane, 912
  - KDChart::TernaryAxis, 1090
  - KDChart::TernaryCoordinatePlane, 1122
  - KDChart::TextArea, 1234
- PositionNorth
  - KDChartEnums, 1313
- PositionNorthEast
  - KDChartEnums, 1313
- PositionNorthWest
  - KDChartEnums, 1313
- PositionPoints
  - KDChart::PositionPoints, 986
- PositionSouth
  - KDChartEnums, 1313
- PositionSouthEast
  - KDChartEnums, 1313
- PositionSouthWest
  - KDChartEnums, 1313
- PositionUnknown
  - KDChartEnums, 1313
- PositionValue
  - KDChartEnums, 1313
- PositionWest
  - KDChartEnums, 1313
- positivePosition
  - KDChart::DataValueAttributes, 587
- prefix
  - KDChart::DataValueAttributes, 587
- PrerenderedElement, 1322

- PrerenderedElement, 1323
- PrerenderedElement
  - ~PrerenderedElement, 1323
  - invalidate, 1323
  - pixmap, 1323
  - position, 1323
  - PrerenderedElement, 1323
  - referencePoint, 1323
  - referencePointLocation, 1324
  - setPosition, 1324
  - setReferencePoint, 1324
- PrerenderedLabel, 1325
  - PrerenderedLabel, 1327
- PrerenderedLabel
  - ~PrerenderedLabel, 1327
  - angle, 1327
  - brush, 1327
  - font, 1328
  - invalidate, 1328
  - pen, 1328
  - pixmap, 1328
  - position, 1328
  - PrerenderedLabel, 1327
  - referencePoint, 1329
  - referencePointLocation, 1329
  - setAngle, 1330
  - setBrush, 1330
  - setFont, 1330
  - setPen, 1331
  - setPosition, 1331
  - setReferencePoint, 1331
  - setText, 1331
  - text, 1331
- printableName
  - KDChart::Position, 982
- printableNames
  - KDChart::Position, 982
- PrivateAttributesModel
  - KDChart::PrivateAttributesModel, 992
- propertiesChanged
  - KDChart::AbstractCartesianDiagram, 115
  - KDChart::AbstractCoordinatePlane, 155
  - KDChart::AbstractDiagram, 193
  - KDChart::AbstractPieDiagram, 242
  - KDChart::AbstractPolarDiagram, 288
  - KDChart::AbstractTernaryDiagram, 334
  - KDChart::BarDiagram, 424
  - KDChart::CartesianCoordinatePlane, 523
  - KDChart::Chart, 556
  - KDChart::Legend, 656
  - KDChart::LineDiagram, 709
  - KDChart::PieDiagram, 810
  - KDChart::Plotter, 864
  - KDChart::PolarCoordinatePlane, 912
  - KDChart::PolarDiagram, 957
  - KDChart::RingDiagram, 1053
  - KDChart::TernaryCoordinatePlane, 1122
  - KDChart::TernaryLineDiagram, 1162
  - KDChart::TernaryPointDiagram, 1207
- Q\_DECLARE\_TYPEINFO
  - KDChartBackgroundAttributes.h, 1388
  - KDChartDataValueAttributes.h, 1417
  - KDChartFrameAttributes.h, 1424
  - KDChartGridAttributes.h, 1434
  - KDChartLineAttributes.h, 1449
  - KDChartMarkerAttributes.h, 1457
  - KDChartPieAttributes.h, 1475
  - KDChartPosition.h, 1491
  - KDChartRelativePosition.h, 1495
  - KDChartTextAttributes.h, 1515
  - KDChartThreeDBarAttributes.h, 1521
  - KDChartThreeDLineAttributes.h, 1524
  - KDChartThreeDPieAttributes.h, 1527
  - KDChartValueTrackerAttributes.h, 1530
- QAbstractItemView, 1333
- QAbstractProxyModel, 1334
- QFrame, 1335
- QGraphicsPolygonItem, 1336
- QLayoutItem, 1337
- QObject, 1338
- QSortFilterProxyModel, 1339
- QTextDocument, 1340
- QWidget, 1341
- radiusUnit
  - KDChart::PolarCoordinatePlane, 912
- rainbowPalette
  - KDChart::Palette, 771
- realFont
  - KDChart::HeaderFooter, 624
  - KDChart::TextArea, 1235
  - KDChart::TextLayoutItem, 1256
- realFontSize
  - KDChart::HeaderFooter, 624
  - KDChart::TextArea, 1235
  - KDChart::TextLayoutItem, 1256
- rectangle
  - KDChart::PaintContext, 766
- referenceArea
  - KDChart::Legend, 656
  - KDChart::Measure, 762
  - KDChart::RelativePosition, 1011
- referenceCoordinatePlane
  - KDChart::AbstractCoordinatePlane, 156
  - KDChart::CartesianCoordinatePlane, 524
  - KDChart::PolarCoordinatePlane, 913
  - KDChart::TernaryCoordinatePlane, 1122

- referenceDiagram
  - KDChart::AbstractCartesianDiagram, 115
  - KDChart::BarDiagram, 424
  - KDChart::LineDiagram, 709
  - KDChart::Plotter, 864
- referenceDiagramIsBarDiagram
  - KDChartCartesianAxis.cpp, 1397
- referenceDiagramOffset
  - KDChart::AbstractCartesianDiagram, 116
  - KDChart::BarDiagram, 424
  - KDChart::LineDiagram, 709
  - KDChart::Plotter, 864
- referenceOrientation
  - KDChart::Measure, 762
- referencePoint
  - KDChart::RelativePosition, 1011
  - PrerenderedElement, 1323
  - PrerenderedLabel, 1329
- referencePointLocation
  - PrerenderedElement, 1324
  - PrerenderedLabel, 1329
- referencePoints
  - KDChart::RelativePosition, 1012
- referencePosition
  - KDChart::RelativePosition, 1012
- RelativePosition
  - KDChart::RelativePosition, 1009
- relativeThickness
  - KDChart::RingDiagram, 1053
- relayout
  - KDChart::AbstractCoordinatePlane, 156
  - KDChart::CartesianCoordinatePlane, 524
  - KDChart::PolarCoordinatePlane, 913
  - KDChart::TernaryCoordinatePlane, 1123
- reLayoutFloatingLegends
  - KDChart::Chart, 556
- RelMarkerLength
  - TernaryConstants.cpp, 1541
  - TernaryConstants.h, 1543
- removeBrush
  - KDChart::Palette, 771
- removeDiagram
  - KDChart::Legend, 656
- removeDiagrams
  - KDChart::Legend, 657
- removeFromParentLayout
  - KDChart::AbstractArea, 44
  - KDChart::AbstractAxis, 80
  - KDChart::AbstractCoordinatePlane, 156
  - KDChart::AbstractLayoutItem, 211
  - KDChart::AutoSpacerLayoutItem, 375
  - KDChart::CartesianAxis, 478
  - KDChart::CartesianCoordinatePlane, 524
  - KDChart::HeaderFooter, 624
  - KDChart::HorizontalLineLayoutItem, 634
  - KDChart::LineLayoutItem, 736
  - KDChart::LineWithMarkerLayoutItem, 742
  - KDChart::MarkerLayoutItem, 755
  - KDChart::PolarCoordinatePlane, 913
  - KDChart::TernaryAxis, 1090
  - KDChart::TernaryCoordinatePlane, 1123
  - KDChart::TextArea, 1235
  - KDChart::TextLayoutItem, 1256
  - KDChart::VerticalLineLayoutItem, 1284
- replaceCoordinatePlane
  - KDChart::Chart, 557
- replaceDiagram
  - KDChart::AbstractCoordinatePlane, 157
  - KDChart::CartesianCoordinatePlane, 525
  - KDChart::Legend, 657
  - KDChart::PolarCoordinatePlane, 914
  - KDChart::TernaryCoordinatePlane, 1123
- replaceHeaderFooter
  - KDChart::Chart, 557
  - KDChart::Widget, 1296
- replaceLegend
  - KDChart::Chart, 558
  - KDChart::Widget, 1296
- requiredMargins
  - KDChart::TernaryAxis, 1090
- resetData
  - KDChart::AttributesModel, 366
  - KDChart::PrivateAttributesModel, 1003
  - KDChart::Widget, 1296
- resetDatasetDescriptions
  - KDChart::DatasetProxyModel, 576
- resetFactors
  - KDChart::GlobalMeasureScaling, 602
- resetGridAttributes
  - KDChart::CartesianCoordinatePlane, 525
  - KDChart::PolarCoordinatePlane, 914
- resetHeaderData
  - KDChart::AttributesModel, 366
  - KDChart::PrivateAttributesModel, 1003
- resetLineAttributes
  - KDChart::LineDiagram, 710
  - KDChart::Plotter, 865
- resetReferencePosition
  - KDChart::RelativePosition, 1012
- resetTexts
  - KDChart::Legend, 658
- resetTitleTextAttributes
  - KDChart::CartesianAxis, 478
  - KDChart::TernaryAxis, 1091
- resize
  - KDChart::AbstractCartesianDiagram, 116
  - KDChart::AbstractDiagram, 193
  - KDChart::AbstractPieDiagram, 242



- KDChart::AbstractPolarDiagram, 289
- KDChart::BarDiagram, 425
- KDChart::LineDiagram, 710
- KDChart::PieDiagram, 810
- KDChart::Plotter, 865
- KDChart::PolarDiagram, 957
- KDChart::RingDiagram, 1053
- KDChart::TernaryLineDiagram, 1162
- KDChart::TernaryPointDiagram, 1207
- resizeEvent
  - KDChart::BarDiagram, 425
  - KDChart::Chart, 559
  - KDChart::Legend, 658
  - KDChart::LineDiagram, 710
  - KDChart::PieDiagram, 810
  - KDChart::Plotter, 865
  - KDChart::PolarCoordinatePlane, 915
  - KDChart::PolarDiagram, 958
  - KDChart::RingDiagram, 1053
- resizeLayout
  - KDChart::AbstractAreaWidget, 64
  - KDChart::Legend, 658
- ReverseMapper
  - KDChart::ReverseMapper, 1017
- ReverseMapper.cpp, 1538
- ReverseMapper.h, 1539
- Right
  - KDChart::CartesianAxis, 451
- rightOverlap
  - KDChart::AbstractArea, 45
  - KDChart::AbstractAxis, 80
  - KDChart::AbstractCoordinatePlane, 157
  - KDChart::CartesianAxis, 478
  - KDChart::CartesianCoordinatePlane, 526
  - KDChart::PolarCoordinatePlane, 915
  - KDChart::TernaryAxis, 1091
  - KDChart::TernaryCoordinatePlane, 1124
- Ring
  - KDChart::Widget, 1290
- RingDiagram
  - KDChart::RingDiagram, 1028
- ringDiagram
  - KDChart::Widget, 1297
- rotateCircularLabels
  - KDChart::PolarDiagram, 958
- rotatedPoint
  - KDChartLayoutItems.cpp, 1440
- rotatedRect
  - KDChartLayoutItems.cpp, 1440
- rotation
  - KDChart::RelativePosition, 1013
  - KDChart::TextAttributes, 1246
- row
  - KDChart::ChartGraphicsItem, 566
- rowCount
  - KDChart::AttributesModel, 366
  - KDChart::PrivateAttributesModel, 1003
- Rows
  - KDChart::BarDiagram, 398
  - KDChart::Widget, 1290
- scrollTo
  - KDChart::AbstractCartesianDiagram, 116
  - KDChart::AbstractDiagram, 193
  - KDChart::AbstractPieDiagram, 242
  - KDChart::AbstractPolarDiagram, 289
  - KDChart::AbstractTernaryDiagram, 334
  - KDChart::BarDiagram, 425
  - KDChart::LineDiagram, 711
  - KDChart::PieDiagram, 810
  - KDChart::Plotter, 866
  - KDChart::PolarDiagram, 958
  - KDChart::RingDiagram, 1053
  - KDChart::TernaryLineDiagram, 1162
  - KDChart::TernaryPointDiagram, 1207
- sequence
  - KDChart::DataDimension, 569
- set
  - TernaryPoint, 1343
- SET\_ALL\_MARGINS\_TO\_ZERO
  - KDChartChart.cpp, 1406
- SET\_SUB\_TYPE
  - KDChartWidget.cpp, 1532
- setAbsoluteValue
  - KDChart::Measure, 762
- setAdjustBoundsToGrid
  - KDChart::GridAttributes, 607
- setAlignment
  - KDChart::Legend, 659
  - KDChart::RelativePosition, 1013
- setAllowOverlappingDataValueTexts
  - KDChart::AbstractCartesianDiagram, 116
  - KDChart::AbstractDiagram, 193
  - KDChart::AbstractPieDiagram, 243
  - KDChart::AbstractPolarDiagram, 289
  - KDChart::AbstractTernaryDiagram, 334
  - KDChart::BarDiagram, 425
  - KDChart::LineDiagram, 711
  - KDChart::PieDiagram, 810
  - KDChart::Plotter, 866
  - KDChart::PolarDiagram, 958
  - KDChart::RingDiagram, 1054
  - KDChart::TernaryLineDiagram, 1162
  - KDChart::TernaryPointDiagram, 1207
- setAngle
  - KDChart::ThreeDBarAttributes, 1264
  - PrerenderedLabel, 1330
- setAntiAliasing

- KDChart::AbstractCartesianDiagram, 117
- KDChart::AbstractDiagram, 194
- KDChart::AbstractPieDiagram, 243
- KDChart::AbstractPolarDiagram, 289
- KDChart::AbstractTernaryDiagram, 335
- KDChart::BarDiagram, 426
- KDChart::LineDiagram, 711
- KDChart::PieDiagram, 811
- KDChart::Plotter, 866
- KDChart::PolarDiagram, 958
- KDChart::RingDiagram, 1054
- KDChart::TernaryLineDiagram, 1163
- KDChart::TernaryPointDiagram, 1208
- setAreaBrush
  - KDChart::ValueTrackerAttributes, 1279
- setAttributesModel
  - KDChart::AbstractCartesianDiagram, 117
  - KDChart::AbstractDiagram, 194
  - KDChart::AbstractPieDiagram, 243
  - KDChart::AbstractPolarDiagram, 290
  - KDChart::AbstractTernaryDiagram, 335
  - KDChart::BarDiagram, 426
  - KDChart::LineDiagram, 711
  - KDChart::PieDiagram, 811
  - KDChart::Plotter, 866
  - KDChart::PolarDiagram, 959
  - KDChart::RingDiagram, 1054
  - KDChart::TernaryLineDiagram, 1163
  - KDChart::TernaryPointDiagram, 1208
- setAttributesModelRootIndex
  - KDChart::AbstractCartesianDiagram, 118
  - KDChart::AbstractDiagram, 195
  - KDChart::AbstractPieDiagram, 244
  - KDChart::AbstractPolarDiagram, 291
  - KDChart::AbstractTernaryDiagram, 336
  - KDChart::BarDiagram, 427
  - KDChart::LineDiagram, 712
  - KDChart::PieDiagram, 812
  - KDChart::Plotter, 867
  - KDChart::PolarDiagram, 960
  - KDChart::RingDiagram, 1055
  - KDChart::TernaryLineDiagram, 1164
  - KDChart::TernaryPointDiagram, 1209
- setAutoAdjustGridToZoom
  - KDChart::CartesianCoordinatePlane, 526
- setAutoAdjustHorizontalRangeToData
  - KDChart::CartesianCoordinatePlane, 527
- setAutoAdjustVerticalRangeToData
  - KDChart::CartesianCoordinatePlane, 527
- setAutoReferenceArea
  - KDChart::HeaderFooter, 625
  - KDChart::TextArea, 1235
  - KDChart::TextLayoutItem, 1257
- setAutoRotate
  - KDChart::TextAttributes, 1246
- setAutoShrink
  - KDChart::TextAttributes, 1247
- setAxesCalcModes
  - KDChart::CartesianCoordinatePlane, 528
- setAxesCalcModeX
  - KDChart::CartesianCoordinatePlane, 528
- setAxesCalcModeY
  - KDChart::CartesianCoordinatePlane, 528
- setBackgroundAttributes
  - KDChart::AbstractArea, 45
  - KDChart::AbstractAreaBase, 53
  - KDChart::AbstractAreaWidget, 64
  - KDChart::AbstractAxis, 80
  - KDChart::AbstractCoordinatePlane, 158
  - KDChart::CartesianAxis, 479
  - KDChart::CartesianCoordinatePlane, 529
  - KDChart::Chart, 559
  - KDChart::DataValueAttributes, 588
  - KDChart::HeaderFooter, 625
  - KDChart::Legend, 659
  - KDChart::PolarCoordinatePlane, 915
  - KDChart::TernaryAxis, 1091
  - KDChart::TernaryCoordinatePlane, 1125
  - KDChart::TextArea, 1236
- setBarAttributes
  - KDChart::BarDiagram, 427, 428
- setBarGapFactor
  - KDChart::BarAttributes, 386
- setBrush
  - KDChart::AbstractCartesianDiagram, 118, 119
  - KDChart::AbstractDiagram, 195, 196
  - KDChart::AbstractPieDiagram, 244, 245
  - KDChart::AbstractPolarDiagram, 291, 292
  - KDChart::AbstractTernaryDiagram, 336, 337
  - KDChart::BackgroundAttributes, 381
  - KDChart::BarDiagram, 428, 429
  - KDChart::Legend, 659
  - KDChart::LineDiagram, 712, 713
  - KDChart::PieDiagram, 812, 813
  - KDChart::Plotter, 867, 868
  - KDChart::PolarDiagram, 960, 961
  - KDChart::RingDiagram, 1055, 1056
  - KDChart::TernaryLineDiagram, 1164, 1165
  - KDChart::TernaryPointDiagram, 1209, 1210
  - PrerenderedLabel, 1330
- setBrushesFromDiagram
  - KDChart::Legend, 659
- setCalculationMode
  - KDChart::Measure, 762
- setCenter
  - KDChart::ZoomParameters, 1306
- setCloseDatasets

- KDChart::PolarDiagram, 961
- setColor
  - KDChart::Legend, 660
- setCoordinatePlane
  - KDChart::AbstractCartesianDiagram, 119
  - KDChart::AbstractDiagram, 196
  - KDChart::AbstractPieDiagram, 246
  - KDChart::AbstractPolarDiagram, 292
  - KDChart::AbstractTernaryDiagram, 337
  - KDChart::BarDiagram, 429
  - KDChart::LineDiagram, 714
  - KDChart::PaintContext, 766
  - KDChart::PieDiagram, 813
  - KDChart::Plotter, 869
  - KDChart::PolarDiagram, 961
  - KDChart::RingDiagram, 1056
  - KDChart::TernaryLineDiagram, 1165
  - KDChart::TernaryPointDiagram, 1210
- setCoordinatePlaneLayout
  - KDChart::Chart, 560
- setData
  - KDChart::AttributesModel, 367
  - KDChart::DatasetProxyModel, 576
  - KDChart::PrivateAttributesModel, 1003
- setDataBoundariesDirty
  - KDChart::AbstractCartesianDiagram, 120
  - KDChart::AbstractDiagram, 197
  - KDChart::AbstractPieDiagram, 246
  - KDChart::AbstractPolarDiagram, 292
  - KDChart::AbstractTernaryDiagram, 338
  - KDChart::BarDiagram, 430
  - KDChart::LineDiagram, 714
  - KDChart::PieDiagram, 814
  - KDChart::Plotter, 869
  - KDChart::PolarDiagram, 962
  - KDChart::RingDiagram, 1057
  - KDChart::TernaryLineDiagram, 1166
  - KDChart::TernaryPointDiagram, 1211
- setDataCell
  - KDChart::Widget, 1297
- setDataLabel
  - KDChart::DataValueAttributes, 588
- setDataMap
  - KDChart::AttributesModel, 367
  - KDChart::PrivateAttributesModel, 1004
- setDataset
  - KDChart::Widget, 1298
- setDatasetColumnDescriptionVector
  - KDChart::DatasetProxyModel, 576
- setDatasetDescriptionVectors
  - KDChart::DatasetProxyModel, 577
- setDatasetDimension
  - KDChart::AbstractCartesianDiagram, 120
  - KDChart::AbstractDiagram, 197
  - KDChart::AbstractPieDiagram, 246
  - KDChart::AbstractPolarDiagram, 293
  - KDChart::AbstractTernaryDiagram, 338
  - KDChart::BarDiagram, 430
  - KDChart::LineDiagram, 714
  - KDChart::PieDiagram, 814
  - KDChart::Plotter, 869
  - KDChart::PolarDiagram, 962
  - KDChart::RingDiagram, 1057
  - KDChart::TernaryLineDiagram, 1166
  - KDChart::TernaryPointDiagram, 1211
- setDatasetRowDescriptionVector
  - KDChart::DatasetProxyModel, 577
- setDataValueAttributes
  - KDChart::AbstractCartesianDiagram, 120, 121
  - KDChart::AbstractDiagram, 197, 198
  - KDChart::AbstractPieDiagram, 247
  - KDChart::AbstractPolarDiagram, 293, 294
  - KDChart::AbstractTernaryDiagram, 338, 339
  - KDChart::BarDiagram, 430, 431
  - KDChart::LineDiagram, 715, 716
  - KDChart::PieDiagram, 815
  - KDChart::Plotter, 870, 871
  - KDChart::PolarDiagram, 963
  - KDChart::RingDiagram, 1058
  - KDChart::TernaryLineDiagram, 1167
  - KDChart::TernaryPointDiagram, 1211, 1212
- setDecimalDigits
  - KDChart::DataValueAttributes, 588
- setDefaultColors
  - KDChart::Legend, 660
- setDefaultForRole
  - KDChart::AttributesModel, 367
  - KDChart::PrivateAttributesModel, 1004
- setDepth
  - KDChart::AbstractThreeDAttributes, 352
  - KDChart::ThreeDBarAttributes, 1264
  - KDChart::ThreeDLineAttributes, 1269
  - KDChart::ThreeDPieAttributes, 1273
- setDiagram
  - KDChart::Legend, 660
  - KDChart::ReverseMapper, 1020
- setDisplayArea
  - KDChart::LineAttributes, 673
- setDrawSolidExcessArrows
  - KDChart::BarAttributes, 386
- setEnabled
  - KDChart::AbstractThreeDAttributes, 352
  - KDChart::ThreeDBarAttributes, 1264
  - KDChart::ThreeDLineAttributes, 1269
  - KDChart::ThreeDPieAttributes, 1274
  - KDChart::ValueTrackerAttributes, 1279
- setExplode

- KDChart::PieAttributes, 775
- setExplodeFactor
  - KDChart::PieAttributes, 775
- setFactors
  - KDChart::GlobalMeasureScaling, 602
- setFixedBarWidth
  - KDChart::BarAttributes, 386
- setFixedDataCoordinateSpaceRelation
  - KDChart::CartesianCoordinatePlane, 529
- setFixedDataValueGap
  - KDChart::BarAttributes, 387
- setFixedValueBlockGap
  - KDChart::BarAttributes, 387
- setFloatingPosition
  - KDChart::Legend, 661
- setFont
  - KDChart::TextAttributes, 1247
  - PrerenderedLabel, 1330
- setFontSize
  - KDChart::TextAttributes, 1247
- setFrameAttributes
  - KDChart::AbstractArea, 45
  - KDChart::AbstractAreaBase, 53
  - KDChart::AbstractAreaWidget, 64
  - KDChart::AbstractAxis, 81
  - KDChart::AbstractCoordinatePlane, 158
  - KDChart::CartesianAxis, 479
  - KDChart::CartesianCoordinatePlane, 529
  - KDChart::Chart, 560
  - KDChart::DataValueAttributes, 589
  - KDChart::HeaderFooter, 625
  - KDChart::Legend, 662
  - KDChart::PolarCoordinatePlane, 916
  - KDChart::TernaryAxis, 1092
  - KDChart::TernaryCoordinatePlane, 1125
  - KDChart::TextArea, 1236
- setGeometry
  - KDChart::AbstractAxis, 81
  - KDChart::AbstractCoordinatePlane, 158
  - KDChart::AutoSpacerLayoutItem, 375
  - KDChart::CartesianAxis, 479
  - KDChart::CartesianCoordinatePlane, 529
  - KDChart::HeaderFooter, 626
  - KDChart::HorizontalLineLayoutItem, 634
  - KDChart::LineLayoutItem, 737
  - KDChart::LineWithMarkerLayoutItem, 742
  - KDChart::MarkerLayoutItem, 755
  - KDChart::PolarCoordinatePlane, 916
  - KDChart::TernaryAxis, 1092
  - KDChart::TernaryCoordinatePlane, 1125
  - KDChart::TextArea, 1236
  - KDChart::TextLayoutItem, 1257
  - KDChart::VerticalLineLayoutItem, 1284
- setGlobalGridAttributes
  - KDChart::AbstractCoordinatePlane, 159
  - KDChart::CartesianCoordinatePlane, 530
  - KDChart::PolarCoordinatePlane, 916
  - KDChart::TernaryCoordinatePlane, 1126
- setGlobalLeading
  - KDChart::Chart, 560
  - KDChart::Widget, 1298
- setGlobalLeadingBottom
  - KDChart::Chart, 561
  - KDChart::Widget, 1299
- setGlobalLeadingLeft
  - KDChart::Chart, 561
  - KDChart::Widget, 1299
- setGlobalLeadingRight
  - KDChart::Chart, 562
  - KDChart::Widget, 1299
- setGlobalLeadingTop
  - KDChart::Chart, 562
  - KDChart::Widget, 1299
- setGranularity
  - KDChart::AbstractPieDiagram, 248
  - KDChart::PieDiagram, 816
  - KDChart::RingDiagram, 1059
- setGridAttributes
  - KDChart::CartesianCoordinatePlane, 530
  - KDChart::PolarCoordinatePlane, 917
- setGridGranularitySequence
  - KDChart::GridAttributes, 607
- setGridNeedsRecalculate
  - KDChart::AbstractCoordinatePlane, 159
  - KDChart::CartesianCoordinatePlane, 531
  - KDChart::PolarCoordinatePlane, 918
  - KDChart::TernaryCoordinatePlane, 1126
- setGridPen
  - KDChart::GridAttributes, 607
- setGridStepWidth
  - KDChart::GridAttributes, 608
- setGridSubStepWidth
  - KDChart::GridAttributes, 608
- setGridVisible
  - KDChart::GridAttributes, 609
- setGroupGapFactor
  - KDChart::BarAttributes, 387
- setHeaderData
  - KDChart::AttributesModel, 368
  - KDChart::PrivateAttributesModel, 1004
- setHidden
  - KDChart::AbstractCartesianDiagram, 122, 123
  - KDChart::AbstractDiagram, 198, 199
  - KDChart::AbstractPieDiagram, 248, 249
  - KDChart::AbstractPolarDiagram, 294, 295
  - KDChart::AbstractTernaryDiagram, 340, 341
  - KDChart::BarDiagram, 431, 432

- KDChart::LineDiagram, 716, 717
- KDChart::PieDiagram, 816, 817
- KDChart::Plotter, 871, 872
- KDChart::PolarDiagram, 964, 965
- KDChart::RingDiagram, 1059, 1060
- KDChart::TernaryLineDiagram, 1168, 1169
- KDChart::TernaryPointDiagram, 1213, 1214
- setHorizontalHeaderDataMap
  - KDChart::AttributesModel, 368
  - KDChart::PrivateAttributesModel, 1005
- setHorizontalPadding
  - KDChart::RelativePosition, 1013
- setHorizontalRange
  - KDChart::CartesianCoordinatePlane, 531
- setHorizontalRangeReversed
  - KDChart::CartesianCoordinatePlane, 532
- setIsometricScaling
  - KDChart::CartesianCoordinatePlane, 532
- setLabels
  - KDChart::AbstractAxis, 81
  - KDChart::CartesianAxis, 480
  - KDChart::TernaryAxis, 1092
- setLegendStyle
  - KDChart::Legend, 662
- setLineAttributes
  - KDChart::LineDiagram, 717, 718
  - KDChart::Plotter, 872, 873
- setLineXRotation
  - KDChart::ThreeDLineAttributes, 1269
- setLineYRotation
  - KDChart::ThreeDLineAttributes, 1270
- setMarkerAttributes
  - KDChart::DataValueAttributes, 589
  - KDChart::Legend, 662
- setMarkerColor
  - KDChart::MarkerAttributes, 749
- setMarkerSize
  - KDChart::MarkerAttributes, 749
  - KDChart::ValueTrackerAttributes, 1279
- setMarkerStyle
  - KDChart::MarkerAttributes, 749
- setMarkerStylesMap
  - KDChart::MarkerAttributes, 749
- setMinimalFontSize
  - KDChart::TextAttributes, 1248
- setMissingValuesPolicy
  - KDChart::LineAttributes, 673
- setModel
  - KDChart::AbstractCartesianDiagram, 123
  - KDChart::AbstractDiagram, 200
  - KDChart::AbstractPieDiagram, 250
  - KDChart::AbstractPolarDiagram, 296
  - KDChart::AbstractTernaryDiagram, 341
  - KDChart::BarDiagram, 433
  - KDChart::LineDiagram, 718
  - KDChart::PieDiagram, 818
  - KDChart::Plotter, 873
  - KDChart::PolarDiagram, 965
  - KDChart::RingDiagram, 1061
  - KDChart::TernaryLineDiagram, 1169
  - KDChart::TernaryPointDiagram, 1214
- setModelData
  - KDChart::AttributesModel, 368
  - KDChart::PrivateAttributesModel, 1005
- setModelDataMap
  - KDChart::AttributesModel, 369
  - KDChart::PrivateAttributesModel, 1006
- setNegativePosition
  - KDChart::DataValueAttributes, 589
- setOrientation
  - KDChart::Legend, 663
- setPadding
  - KDChart::FrameAttributes, 598
- setPaintDevice
  - KDChart::GlobalMeasureScaling, 602
- setPainter
  - KDChart::PaintContext, 766
- setPaletteType
  - KDChart::AttributesModel, 369
  - KDChart::PrivateAttributesModel, 1006
- setParent
  - KDChart::AbstractCoordinatePlane, 159
  - KDChart::CartesianCoordinatePlane, 533
  - KDChart::HeaderFooter, 626
  - KDChart::PolarCoordinatePlane, 918
  - KDChart::TernaryCoordinatePlane, 1126
- setParentLayout
  - KDChart::AbstractArea, 46
  - KDChart::AbstractAxis, 81
  - KDChart::AbstractCoordinatePlane, 160
  - KDChart::AbstractLayoutItem, 211
  - KDChart::AutoSpacerLayoutItem, 375
  - KDChart::CartesianAxis, 480
  - KDChart::CartesianCoordinatePlane, 533
  - KDChart::HeaderFooter, 626
  - KDChart::HorizontalLineLayoutItem, 634
  - KDChart::LineLayoutItem, 737
  - KDChart::LineWithMarkerLayoutItem, 743
  - KDChart::MarkerLayoutItem, 755
  - KDChart::PolarCoordinatePlane, 918
  - KDChart::TernaryAxis, 1093
  - KDChart::TernaryCoordinatePlane, 1127
  - KDChart::TextArea, 1236
  - KDChart::TextLayoutItem, 1257
  - KDChart::VerticalLineLayoutItem, 1284
- setParentWidget
  - KDChart::AbstractArea, 46
  - KDChart::AbstractAxis, 82



- KDChart::AbstractCoordinatePlane, 160
- KDChart::AbstractLayoutItem, 211
- KDChart::AutoSpacerLayoutItem, 375
- KDChart::CartesianAxis, 480
- KDChart::CartesianCoordinatePlane, 533
- KDChart::HeaderFooter, 626
- KDChart::HorizontalLineLayoutItem, 634
- KDChart::LineLayoutItem, 737
- KDChart::LineWithMarkerLayoutItem, 743
- KDChart::MarkerLayoutItem, 755
- KDChart::PolarCoordinatePlane, 918
- KDChart::TernaryAxis, 1093
- KDChart::TernaryCoordinatePlane, 1127
- KDChart::TextArea, 1237
- KDChart::TextLayoutItem, 1257
- KDChart::VerticalLineLayoutItem, 1284
- setPen
  - KDChart::AbstractCartesianDiagram, 123, 124
  - KDChart::AbstractDiagram, 200, 201
  - KDChart::AbstractPieDiagram, 250, 251
  - KDChart::AbstractPolarDiagram, 296, 297
  - KDChart::AbstractTernaryDiagram, 342
  - KDChart::BarDiagram, 433, 434
  - KDChart::FrameAttributes, 598
  - KDChart::Legend, 663
  - KDChart::LineDiagram, 719
  - KDChart::MarkerAttributes, 750
  - KDChart::PieDiagram, 818, 819
  - KDChart::Plotter, 874
  - KDChart::PolarDiagram, 966
  - KDChart::RingDiagram, 1061, 1062
  - KDChart::TernaryLineDiagram, 1170, 1171
  - KDChart::TernaryPointDiagram, 1215
  - KDChart::TextAttributes, 1248
  - KDChart::ValueTrackerAttributes, 1280
  - PrerenderedLabel, 1331
- setPercentMode
  - KDChart::AbstractCartesianDiagram, 125
  - KDChart::AbstractDiagram, 201
  - KDChart::AbstractPieDiagram, 251
  - KDChart::AbstractPolarDiagram, 298
  - KDChart::AbstractTernaryDiagram, 343
  - KDChart::BarDiagram, 435
  - KDChart::LineDiagram, 720
  - KDChart::PieDiagram, 819
  - KDChart::Plotter, 875
  - KDChart::PolarDiagram, 967
  - KDChart::RingDiagram, 1062
  - KDChart::TernaryLineDiagram, 1171
  - KDChart::TernaryPointDiagram, 1216
- setPieAttributes
  - KDChart::AbstractPieDiagram, 252
  - KDChart::PieDiagram, 820
  - KDChart::RingDiagram, 1063
- setPixmap
  - KDChart::BackgroundAttributes, 381
- setPixmapMode
  - KDChart::BackgroundAttributes, 381
- setPosition
  - KDChart::CartesianAxis, 481
  - KDChart::HeaderFooter, 627
  - KDChart::Legend, 663
  - KDChart::TernaryAxis, 1093
  - PrerenderedElement, 1324
  - PrerenderedLabel, 1331
- setPositivePosition
  - KDChart::DataValueAttributes, 590
- setPrefix
  - KDChart::DataValueAttributes, 590
- setRainbowColors
  - KDChart::Legend, 663
- setRectangle
  - KDChart::PaintContext, 767
- setReferenceArea
  - KDChart::Legend, 664
  - KDChart::Measure, 763
  - KDChart::RelativePosition, 1014
- setReferenceCoordinatePlane
  - KDChart::AbstractCoordinatePlane, 160
  - KDChart::CartesianCoordinatePlane, 533
  - KDChart::PolarCoordinatePlane, 919
  - KDChart::TernaryCoordinatePlane, 1127
- setReferenceDiagram
  - KDChart::AbstractCartesianDiagram, 125
  - KDChart::BarDiagram, 435
  - KDChart::LineDiagram, 720
  - KDChart::Plotter, 875
- setReferenceOrientation
  - KDChart::Measure, 763
- setReferencePoint
  - PrerenderedElement, 1324
  - PrerenderedLabel, 1331
- setReferencePoints
  - KDChart::RelativePosition, 1014
- setReferencePosition
  - KDChart::RelativePosition, 1015
- setRelativeMode
  - KDChart::Measure, 763
- setRelativeThickness
  - KDChart::RingDiagram, 1063
- setRootIndex
  - KDChart::AbstractCartesianDiagram, 125
  - KDChart::AbstractDiagram, 202
  - KDChart::AbstractPieDiagram, 252
  - KDChart::AbstractPolarDiagram, 298
  - KDChart::AbstractTernaryDiagram, 343
  - KDChart::BarDiagram, 435

- KDChart::LineDiagram, 720
- KDChart::PieDiagram, 820
- KDChart::Plotter, 875
- KDChart::PolarDiagram, 967
- KDChart::RingDiagram, 1063
- KDChart::TernaryLineDiagram, 1171
- KDChart::TernaryPointDiagram, 1216
- setRotateCircularLabels
  - KDChart::PolarDiagram, 967
- setRotation
  - KDChart::RelativePosition, 1015
  - KDChart::TextAttributes, 1248
- setRubberBandZoomingEnabled
  - KDChart::AbstractCoordinatePlane, 161
  - KDChart::CartesianCoordinatePlane, 534
  - KDChart::PolarCoordinatePlane, 919
  - KDChart::TernaryCoordinatePlane, 1128
- setSelection
  - KDChart::AbstractCartesianDiagram, 126
  - KDChart::AbstractDiagram, 202
  - KDChart::AbstractPieDiagram, 252
  - KDChart::AbstractPolarDiagram, 298
  - KDChart::AbstractTernaryDiagram, 343
  - KDChart::BarDiagram, 435
  - KDChart::LineDiagram, 721
  - KDChart::PieDiagram, 820
  - KDChart::Plotter, 876
  - KDChart::PolarDiagram, 968
  - KDChart::RingDiagram, 1064
  - KDChart::TernaryLineDiagram, 1172
  - KDChart::TernaryPointDiagram, 1216
- setShortLabels
  - KDChart::AbstractAxis, 82
  - KDChart::CartesianAxis, 481
  - KDChart::TernaryAxis, 1094
- setShowDelimitersAtPosition
  - KDChart::PolarDiagram, 968
- setShowLabelsAtPosition
  - KDChart::PolarDiagram, 968
- setShowLines
  - KDChart::Legend, 664
- setSourceColumnCount
  - KDChart::DatasetSelectorWidget, 580
- setSourceModel
  - KDChart::AttributesModel, 369
  - KDChart::DatasetProxyModel, 577
  - KDChart::PrivateAttributesModel, 1006
- setSourceRootIndex
  - KDChart::DatasetProxyModel, 578
- setSourceRowCount
  - KDChart::DatasetSelectorWidget, 580
- setSpacing
  - KDChart::Legend, 664
- setStartPosition
  - KDChart::AbstractPieDiagram, 253
  - KDChart::PieDiagram, 821
  - KDChart::PolarCoordinatePlane, 919
  - KDChart::RingDiagram, 1064
- setSubduedColors
  - KDChart::Legend, 665
- setSubGridPen
  - KDChart::GridAttributes, 609
- setSubGridVisible
  - KDChart::GridAttributes, 609
- setSubType
  - KDChart::Widget, 1300
- setSuffix
  - KDChart::DataValueAttributes, 590
- setText
  - KDChart::HeaderFooter, 627
  - KDChart::Legend, 666
  - KDChart::TextArea, 1237
  - KDChart::TextLayoutItem, 1258
  - PrerenderedLabel, 1331
- setTextAttributes
  - KDChart::AbstractAxis, 82
  - KDChart::CartesianAxis, 481
  - KDChart::DataValueAttributes, 591
  - KDChart::HeaderFooter, 627
  - KDChart::Legend, 666
  - KDChart::TernaryAxis, 1094
  - KDChart::TextArea, 1237
  - KDChart::TextLayoutItem, 1258
- setThreeDBarAttributes
  - KDChart::BarDiagram, 436
- setThreeDLineAttributes
  - KDChart::LineDiagram, 721, 722
  - KDChart::Plotter, 876, 877
- setThreeDPieAttributes
  - KDChart::AbstractPieDiagram, 253
  - KDChart::PieDiagram, 821
  - KDChart::RingDiagram, 1064, 1065
- setTitleText
  - KDChart::CartesianAxis, 482
  - KDChart::Legend, 666
  - KDChart::TernaryAxis, 1095
- setTitleTextAttributes
  - KDChart::CartesianAxis, 482
  - KDChart::Legend, 666
  - KDChart::TernaryAxis, 1095
- setTransparency
  - KDChart::LineAttributes, 673
- setType
  - KDChart::BarDiagram, 437
  - KDChart::HeaderFooter, 628
  - KDChart::LineDiagram, 722
  - KDChart::Plotter, 877
  - KDChart::Widget, 1300

- setUnitPrefix
  - KDChart::AbstractCartesianDiagram, 126
  - KDChart::AbstractDiagram, 202, 203
  - KDChart::AbstractPieDiagram, 254
  - KDChart::AbstractPolarDiagram, 298, 299
  - KDChart::AbstractTernaryDiagram, 344
  - KDChart::BarDiagram, 437, 438
  - KDChart::LineDiagram, 723
  - KDChart::PieDiagram, 822
  - KDChart::Plotter, 878
  - KDChart::PolarDiagram, 968, 969
  - KDChart::RingDiagram, 1065
  - KDChart::TernaryLineDiagram, 1172
  - KDChart::TernaryPointDiagram, 1217
- setUnitSuffix
  - KDChart::AbstractCartesianDiagram, 127
  - KDChart::AbstractDiagram, 203
  - KDChart::AbstractPieDiagram, 254, 255
  - KDChart::AbstractPolarDiagram, 299
  - KDChart::AbstractTernaryDiagram, 344, 345
  - KDChart::BarDiagram, 438
  - KDChart::LineDiagram, 724
  - KDChart::PieDiagram, 822, 823
  - KDChart::Plotter, 878, 879
  - KDChart::PolarDiagram, 969
  - KDChart::RingDiagram, 1065, 1066
  - KDChart::TernaryLineDiagram, 1173
  - KDChart::TernaryPointDiagram, 1217, 1218
- setUseAutomaticMarkerSize
  - KDChart::Legend, 666
- setUseFixedBarWidth
  - KDChart::BarAttributes, 387
- setUseFixedDataValueGap
  - KDChart::BarAttributes, 387
- setUseFixedValueBlockGap
  - KDChart::BarAttributes, 388
- setUseShadowColors
  - KDChart::ThreeDBarAttributes, 1264
  - KDChart::ThreeDPieAttributes, 1274
- setValue
  - KDChart::Measure, 763
- setValueTrackerAttributes
  - KDChart::LineDiagram, 724
  - KDChart::Plotter, 879
- setVerticalHeaderDataMap
  - KDChart::AttributesModel, 370
  - KDChart::PrivateAttributesModel, 1007
- setVerticalPadding
  - KDChart::RelativePosition, 1015
- setVerticalRange
  - KDChart::CartesianCoordinatePlane, 534
- setVerticalRangeReversed
  - KDChart::CartesianCoordinatePlane, 535
- setVisible
  - KDChart::BackgroundAttributes, 382
  - KDChart::DataValueAttributes, 591
  - KDChart::FrameAttributes, 598
  - KDChart::Legend, 667
  - KDChart::MarkerAttributes, 750
  - KDChart::TextAttributes, 1249
- setZeroDegreePosition
  - KDChart::PolarDiagram, 970
- setZeroLinePen
  - KDChart::GridAttributes, 609
- setZoomCenter
  - KDChart::AbstractCoordinatePlane, 161
  - KDChart::CartesianCoordinatePlane, 535
  - KDChart::PolarCoordinatePlane, 920
  - KDChart::TernaryCoordinatePlane, 1128
- setZoomFactorX
  - KDChart::AbstractCoordinatePlane, 161
  - KDChart::CartesianCoordinatePlane, 536
  - KDChart::PolarCoordinatePlane, 920
  - KDChart::TernaryCoordinatePlane, 1128
- setZoomFactorY
  - KDChart::AbstractCoordinatePlane, 161
  - KDChart::CartesianCoordinatePlane, 536
  - KDChart::PolarCoordinatePlane, 920
  - KDChart::TernaryCoordinatePlane, 1129
- sharedAxisMasterPlane
  - KDChart::AbstractCoordinatePlane, 162
  - KDChart::CartesianCoordinatePlane, 536
  - KDChart::PolarCoordinatePlane, 921
  - KDChart::TernaryCoordinatePlane, 1129
- shortLabels
  - KDChart::AbstractAxis, 83
  - KDChart::CartesianAxis, 482
  - KDChart::TernaryAxis, 1095
- showDelimitersAtPosition
  - KDChart::PolarDiagram, 970
- showLabelsAtPosition
  - KDChart::PolarDiagram, 970
- showLines
  - KDChart::Legend, 667
- SignalCompressor
  - KDChart::SignalCompressor, 1074
- size
  - KDChart::Palette, 772
- sizeHint
  - KDChart::AbstractCoordinatePlane, 162
  - KDChart::AutoSpacerLayoutItem, 376
  - KDChart::CartesianAxis, 483
  - KDChart::CartesianCoordinatePlane, 537
  - KDChart::HeaderFooter, 628
  - KDChart::HorizontalLineLayoutItem, 635
  - KDChart::Legend, 667
  - KDChart::LineLayoutItem, 737
  - KDChart::LineWithMarkerLayoutItem, 743



- KDChart::MarkerLayoutItem, 756
- KDChart::PolarCoordinatePlane, 921
- KDChart::TernaryAxis, 1096
- KDChart::TernaryCoordinatePlane, 1129
- KDChart::TextArea, 1238
- KDChart::TextLayoutItem, 1258
- KDChart::VerticalLineLayoutItem, 1285
- KDTextDocument, 1320
- sizeHintChanged
  - KDChart::AbstractArea, 46
  - KDChart::AbstractAxis, 83
  - KDChart::AbstractCoordinatePlane, 162
  - KDChart::AbstractLayoutItem, 212
  - KDChart::AutoSpacerLayoutItem, 376
  - KDChart::CartesianAxis, 483
  - KDChart::CartesianCoordinatePlane, 538
  - KDChart::HeaderFooter, 628
  - KDChart::HorizontalLineLayoutItem, 635
  - KDChart::LineLayoutItem, 738
  - KDChart::LineWithMarkerLayoutItem, 743
  - KDChart::MarkerLayoutItem, 756
  - KDChart::PolarCoordinatePlane, 921
  - KDChart::TernaryAxis, 1096
  - KDChart::TernaryCoordinatePlane, 1129
  - KDChart::TextArea, 1238
  - KDChart::TextLayoutItem, 1259
  - KDChart::VerticalLineLayoutItem, 1285
- sizeOfArea
  - KDChart::Measure, 763
- sizePolicy
  - KDChart::AbstractCoordinatePlane, 163
  - KDChart::CartesianCoordinatePlane, 538
  - KDChart::PolarCoordinatePlane, 922
  - KDChart::TernaryCoordinatePlane, 1130
- slotLayoutChanged
  - KDChart::CartesianCoordinatePlane, 538
  - KDChart::PolarCoordinatePlane, 922
- South
  - KDChart::Position, 984
- SouthEast
  - KDChart::Position, 984
- SouthWest
  - KDChart::Position, 984
- spacing
  - KDChart::Legend, 668
- Sqrt3
  - TernaryConstants.cpp, 1541
  - TernaryConstants.h, 1543
- src/ Directory Reference, 21
- src/PrerenderedElements/ Directory Reference, 19
- src/Scenery/ Directory Reference, 20
- src/Ternary/ Directory Reference, 25
- Stacked
  - KDChart::BarDiagram, 398
  - KDChart::LineDiagram, 682
  - KDChart::Widget, 1290
- start
  - KDChart::DataDimension, 569
- startPosition
  - KDChart::AbstractPieDiagram, 255
  - KDChart::PieDiagram, 823
  - KDChart::PolarCoordinatePlane, 922
  - KDChart::RingDiagram, 1066
- staticPositionCenter
  - KDChartPosition.cpp, 1488
- staticPositionEast
  - KDChartPosition.cpp, 1488
- staticPositionFloating
  - KDChartPosition.cpp, 1488
- staticPositionNames
  - KDChartPosition.cpp, 1488
- staticPositionNorth
  - KDChartPosition.cpp, 1489
- staticPositionNorthEast
  - KDChartPosition.cpp, 1489
- staticPositionNorthWest
  - KDChartPosition.cpp, 1489
- staticPositionSouth
  - KDChartPosition.cpp, 1489
- staticPositionSouthEast
  - KDChartPosition.cpp, 1489
- staticPositionSouthWest
  - KDChartPosition.cpp, 1489
- staticPositionUnknown
  - KDChartPosition.cpp, 1489
- staticPositionWest
  - KDChartPosition.cpp, 1489
- stepWidth
  - KDChart::DataDimension, 569
- stringToGranularitySequence
  - KDChartEnums, 1316
- stringToLayoutPolicy
  - KDChartEnums, 1317
- stringToMeasureCalculationMode
  - KDChartEnums, 1317
- stringToMeasureOrientation
  - KDChartEnums, 1317
- subduedPalette
  - KDChart::Palette, 772
- subGridPen
  - KDChart::GridAttributes, 609
- subStepWidth
  - KDChart::DataDimension, 570
- SubType
  - KDChart::Widget, 1290
- subType
  - KDChart::Widget, 1302
- suffix

- KDChart::DataValueAttributes, 591
- takeAxis
  - KDChart::AbstractCartesianDiagram, 127
  - KDChart::BarDiagram, 439
  - KDChart::LineDiagram, 724
  - KDChart::Plotter, 879
- takeCoordinatePlane
  - KDChart::Chart, 562
- takeDiagram
  - KDChart::AbstractCoordinatePlane, 163
  - KDChart::CartesianCoordinatePlane, 538
  - KDChart::PolarCoordinatePlane, 922
  - KDChart::TernaryCoordinatePlane, 1130
- takeHeaderFooter
  - KDChart::Chart, 563
  - KDChart::Widget, 1303
- takeLegend
  - KDChart::Chart, 563
  - KDChart::Widget, 1303
- TernaryAxis
  - KDChart::TernaryAxis, 1078
- TernaryAxisList
  - KDChart, 33
- TernaryConstants.cpp, 1540
- TernaryConstants.cpp
  - AxisVector\_B\_A, 1541
  - AxisVector\_B\_C, 1541
  - AxisVector\_C\_A, 1541
  - FullMarkerDistanceAC, 1541
  - FullMarkerDistanceBA, 1541
  - FullMarkerDistanceBC, 1541
  - Norm\_B\_A, 1541
  - Norm\_B\_C, 1541
  - Norm\_C\_A, 1541
  - RelMarkerLength, 1541
  - Sqrt3, 1541
  - TriangleBottomLeft, 1541
  - TriangleBottomRight, 1541
  - TriangleHeight, 1541
  - TriangleTop, 1541
  - TriangleWidth, 1541
- TernaryConstants.h, 1542
- TernaryConstants.h
  - AxisVector\_B\_A, 1542
  - AxisVector\_B\_C, 1542
  - AxisVector\_C\_A, 1542
  - FullMarkerDistanceAC, 1542
  - FullMarkerDistanceBA, 1543
  - FullMarkerDistanceBC, 1543
  - Norm\_B\_A, 1543
  - Norm\_B\_C, 1543
  - Norm\_C\_A, 1543
  - RelMarkerLength, 1543
  - Sqrt3, 1543
  - TriangleBottomLeft, 1543
  - TriangleBottomRight, 1543
  - TriangleHeight, 1543
  - TriangleTop, 1543
  - TriangleWidth, 1543
- TernaryCoordinatePlane
  - KDChart::TernaryCoordinatePlane, 1104
- TernaryLineDiagram
  - KDChart::TernaryLineDiagram, 1139
- TernaryPoint, 1342
  - TernaryPoint, 1342
- TernaryPoint
  - a, 1343
  - b, 1343
  - c, 1343
  - isValid, 1343
  - set, 1343
  - TernaryPoint, 1342
- TernaryPoint.cpp, 1544
- TernaryPoint.cpp
  - operator<<, 1544
  - translate, 1544
- TernaryPoint.h, 1546
- TernaryPoint.h
  - operator<<, 1546
  - translate, 1547
- TernaryPointDiagram
  - KDChart::TernaryPointDiagram, 1184
- TEST\_SUB\_TYPE
  - KDChartWidget.cpp, 1532
- text
  - KDChart::HeaderFooter, 629
  - KDChart::Legend, 668
  - KDChart::TextArea, 1238
  - KDChart::TextLayoutItem, 1259
  - PrerenderedLabel, 1331
- TextArea
  - KDChart::TextArea, 1225
- TextAttributes
  - KDChart::TextAttributes, 1241, 1242
- textAttributes
  - KDChart::AbstractAxis, 84
  - KDChart::CartesianAxis, 483
  - KDChart::DataValueAttributes, 592
  - KDChart::HeaderFooter, 629
  - KDChart::Legend, 668
  - KDChart::TernaryAxis, 1096
  - KDChart::TextArea, 1239
  - KDChart::TextLayoutItem, 1259
- TextLayoutItem
  - KDChart::TextLayoutItem, 1252
- TextLayoutPolicy
  - KDChartEnums, 1313

- texts
  - KDChart::Legend, 668
- ThreeDAttributesRole
  - KDChart, 33
- ThreeDBarAttributes
  - KDChart::ThreeDBarAttributes, 1262
- threeDBarAttributes
  - KDChart::BarDiagram, 439, 440
- ThreeDBarAttributesRole
  - KDChart, 33
- threeDItemDepth
  - KDChart::AbstractCartesianDiagram, 128
  - KDChart::BarDiagram, 440, 441
  - KDChart::LineDiagram, 725
  - KDChart::Plotter, 880
- ThreeDLineAttributes
  - KDChart::ThreeDLineAttributes, 1267
- threeDLineAttributes
  - KDChart::LineDiagram, 726
  - KDChart::Plotter, 881
- ThreeDLineAttributesRole
  - KDChart, 33
- ThreeDPieAttributes
  - KDChart::ThreeDPieAttributes, 1271, 1272
- threeDPieAttributes
  - KDChart::AbstractPieDiagram, 255, 256
  - KDChart::PieDiagram, 823, 824
  - KDChart::RingDiagram, 1066, 1067
- ThreeDPieAttributesRole
  - KDChart, 33
- tickLength
  - KDChart::CartesianAxis, 483
- titleText
  - KDChart::CartesianAxis, 484
  - KDChart::Legend, 668
  - KDChart::TernaryAxis, 1096
- titleTextAttributes
  - KDChart::CartesianAxis, 484
  - KDChart::Legend, 669
  - KDChart::TernaryAxis, 1097
- Top
  - KDChart::CartesianAxis, 451
- topOverlap
  - KDChart::AbstractArea, 47
  - KDChart::AbstractAxis, 84
  - KDChart::AbstractCoordinatePlane, 163
  - KDChart::CartesianAxis, 485
  - KDChart::CartesianCoordinatePlane, 539
  - KDChart::PolarCoordinatePlane, 923
  - KDChart::TernaryAxis, 1097
  - KDChart::TernaryCoordinatePlane, 1131
- translate
  - KDChart::AbstractCoordinatePlane, 164
  - KDChart::CartesianCoordinatePlane, 539
  - KDChart::PolarCoordinatePlane, 923
  - KDChart::TernaryCoordinatePlane, 1131
  - TernaryPoint.cpp, 1544
  - TernaryPoint.h, 1547
- translateBack
  - KDChart::CartesianCoordinatePlane, 540
- translatePolar
  - KDChart::PolarCoordinatePlane, 924
- transparency
  - KDChart::LineAttributes, 673
- TriangleBottomLeft
  - TernaryConstants.cpp, 1541
  - TernaryConstants.h, 1543
- TriangleBottomRight
  - TernaryConstants.cpp, 1541
  - TernaryConstants.h, 1543
- TriangleHeight
  - TernaryConstants.cpp, 1541
  - TernaryConstants.h, 1543
- TriangleTop
  - TernaryConstants.cpp, 1541
  - TernaryConstants.h, 1543
- TriangleWidth
  - TernaryConstants.cpp, 1541
  - TernaryConstants.h, 1543
- Type
  - KDChart::ChartGraphicsItem, 565
- type
  - KDChart::BarDiagram, 441
  - KDChart::ChartGraphicsItem, 566
  - KDChart::HeaderFooter, 629
  - KDChart::LineDiagram, 726
  - KDChart::Plotter, 881
  - KDChart::Widget, 1303
- unitPrefix
  - KDChart::AbstractCartesianDiagram, 128, 129
  - KDChart::AbstractDiagram, 204
  - KDChart::AbstractPieDiagram, 256
  - KDChart::AbstractPolarDiagram, 300
  - KDChart::AbstractTernaryDiagram, 345
  - KDChart::BarDiagram, 441, 442
  - KDChart::LineDiagram, 727
  - KDChart::PieDiagram, 824
  - KDChart::Plotter, 882
  - KDChart::PolarDiagram, 970, 971
  - KDChart::RingDiagram, 1067, 1068
  - KDChart::TernaryLineDiagram, 1173, 1174
  - KDChart::TernaryPointDiagram, 1218
- unitSuffix
  - KDChart::AbstractCartesianDiagram, 129
  - KDChart::AbstractDiagram, 205
  - KDChart::AbstractPieDiagram, 257

- KDChart::AbstractPolarDiagram, 301
- KDChart::AbstractTernaryDiagram, 346
- KDChart::BarDiagram, 442
- KDChart::LineDiagram, 728
- KDChart::PieDiagram, 825
- KDChart::Plotter, 883
- KDChart::PolarDiagram, 971, 972
- KDChart::RingDiagram, 1068
- KDChart::TernaryLineDiagram, 1174, 1175
- KDChart::TernaryPointDiagram, 1219
- Unknown
  - KDChart::Position, 984
- update
  - KDChart::AbstractAxis, 84
  - KDChart::AbstractCartesianDiagram, 130
  - KDChart::AbstractCoordinatePlane, 164
  - KDChart::AbstractDiagram, 205
  - KDChart::AbstractPieDiagram, 258
  - KDChart::AbstractPolarDiagram, 301
  - KDChart::AbstractTernaryDiagram, 347
  - KDChart::BarDiagram, 443
  - KDChart::CartesianAxis, 485
  - KDChart::CartesianCoordinatePlane, 540
  - KDChart::LineDiagram, 728
  - KDChart::PieDiagram, 826
  - KDChart::Plotter, 883
  - KDChart::PolarCoordinatePlane, 924
  - KDChart::PolarDiagram, 972
  - KDChart::RingDiagram, 1069
  - KDChart::TernaryAxis, 1097
  - KDChart::TernaryCoordinatePlane, 1131
  - KDChart::TernaryLineDiagram, 1175
  - KDChart::TernaryPointDiagram, 1220
- updateCommonBrush
  - KDChartLayoutItems.cpp, 1440
- useAutomaticMarkerSize
  - KDChart::Legend, 669
- useDefaultColors
  - KDChart::AbstractCartesianDiagram, 130
  - KDChart::AbstractDiagram, 206
  - KDChart::AbstractPieDiagram, 258
  - KDChart::AbstractPolarDiagram, 302
  - KDChart::AbstractTernaryDiagram, 347
  - KDChart::BarDiagram, 443
  - KDChart::LineDiagram, 729
  - KDChart::PieDiagram, 826
  - KDChart::Plotter, 884
  - KDChart::PolarDiagram, 972
  - KDChart::RingDiagram, 1069
  - KDChart::TernaryLineDiagram, 1175
  - KDChart::TernaryPointDiagram, 1220
- useFixedBarWidth
  - KDChart::BarAttributes, 388
- useFixedDataValueGap
  - KDChart::BarAttributes, 388
- useFixedValueBlockGap
  - KDChart::BarAttributes, 388
- useRainbowColors
  - KDChart::AbstractCartesianDiagram, 130
  - KDChart::AbstractDiagram, 206
  - KDChart::AbstractPieDiagram, 258
  - KDChart::AbstractPolarDiagram, 302
  - KDChart::AbstractTernaryDiagram, 347
  - KDChart::BarDiagram, 443
  - KDChart::LineDiagram, 729
  - KDChart::PieDiagram, 826
  - KDChart::Plotter, 884
  - KDChart::PolarDiagram, 973
  - KDChart::RingDiagram, 1069
  - KDChart::TernaryLineDiagram, 1176
  - KDChart::TernaryPointDiagram, 1220
- usesExternalAttributesModel
  - KDChart::AbstractCartesianDiagram, 131
  - KDChart::AbstractDiagram, 206
  - KDChart::AbstractPieDiagram, 258
  - KDChart::AbstractPolarDiagram, 302
  - KDChart::AbstractTernaryDiagram, 348
  - KDChart::BarDiagram, 444
  - KDChart::LineDiagram, 729
  - KDChart::PieDiagram, 827
  - KDChart::Plotter, 884
  - KDChart::PolarDiagram, 973
  - KDChart::RingDiagram, 1070
  - KDChart::TernaryLineDiagram, 1176
  - KDChart::TernaryPointDiagram, 1221
- useShadowColors
  - KDChart::ThreeDBarAttributes, 1265
  - KDChart::ThreeDPieAttributes, 1274
- useSubduedColors
  - KDChart::AbstractCartesianDiagram, 131
  - KDChart::AbstractDiagram, 207
  - KDChart::AbstractPieDiagram, 259
  - KDChart::AbstractPolarDiagram, 303
  - KDChart::AbstractTernaryDiagram, 348
  - KDChart::BarDiagram, 444
  - KDChart::LineDiagram, 730
  - KDChart::PieDiagram, 827
  - KDChart::Plotter, 885
  - KDChart::PolarDiagram, 973
  - KDChart::RingDiagram, 1070
  - KDChart::TernaryLineDiagram, 1176
  - KDChart::TernaryPointDiagram, 1221
- validDepth
  - KDChart::AbstractThreeDAttributes, 352
  - KDChart::ThreeDBarAttributes, 1265
  - KDChart::ThreeDLineAttributes, 1270
  - KDChart::ThreeDPieAttributes, 1274

- value
  - KDChart::Measure, 764
  - KDChart::Position, 983
- valueForCell
  - KDChart::AbstractCartesianDiagram, 131
  - KDChart::AbstractDiagram, 207
  - KDChart::AbstractPieDiagram, 259
  - KDChart::AbstractPolarDiagram, 303
  - KDChart::AbstractTernaryDiagram, 348
  - KDChart::BarDiagram, 444
  - KDChart::LineDiagram, 730
  - KDChart::PieDiagram, 827
  - KDChart::Plotter, 885
  - KDChart::PolarDiagram, 974
  - KDChart::RingDiagram, 1070
  - KDChart::TernaryLineDiagram, 1177
  - KDChart::TernaryPointDiagram, 1221
- valueForCellTesting
  - KDChart::LineDiagram, 730
- valueTotals
  - KDChart::AbstractPieDiagram, 260
  - KDChart::AbstractPolarDiagram, 303
  - KDChart::PieDiagram, 828
  - KDChart::PolarDiagram, 974
  - KDChart::RingDiagram, 1071
- ValueTrackerAttributes
  - KDChart::ValueTrackerAttributes, 1277
- valueTrackerAttributes
  - KDChart::LineDiagram, 731
  - KDChart::Plotter, 885
- ValueTrackerAttributesRole
  - KDChart, 33
- verticalHeaderDataMap
  - KDChart::AttributesModel, 370
  - KDChart::PrivateAttributesModel, 1007
- VerticalLineLayoutItem
  - KDChart::VerticalLineLayoutItem, 1282
- verticalOffset
  - KDChart::AbstractCartesianDiagram, 132
  - KDChart::AbstractDiagram, 207
  - KDChart::AbstractPieDiagram, 260
  - KDChart::AbstractPolarDiagram, 304
  - KDChart::AbstractTernaryDiagram, 349
  - KDChart::BarDiagram, 445
  - KDChart::LineDiagram, 731
  - KDChart::PieDiagram, 828
  - KDChart::Plotter, 886
  - KDChart::PolarDiagram, 974
  - KDChart::RingDiagram, 1071
  - KDChart::TernaryLineDiagram, 1177
  - KDChart::TernaryPointDiagram, 1222
- verticalPadding
  - KDChart::RelativePosition, 1016
- verticalRange
  - KDChart::CartesianCoordinatePlane, 540
- visibleDataRange
  - KDChart::CartesianCoordinatePlane, 541
- visualRect
  - KDChart::AbstractCartesianDiagram, 132
  - KDChart::AbstractDiagram, 207
  - KDChart::AbstractPieDiagram, 260
  - KDChart::AbstractPolarDiagram, 304
  - KDChart::AbstractTernaryDiagram, 349
  - KDChart::BarDiagram, 445
  - KDChart::LineDiagram, 731
  - KDChart::PieDiagram, 828
  - KDChart::Plotter, 886
  - KDChart::PolarDiagram, 975
  - KDChart::RingDiagram, 1071
  - KDChart::TernaryLineDiagram, 1177
  - KDChart::TernaryPointDiagram, 1222
- visualRegionForSelection
  - KDChart::AbstractCartesianDiagram, 132
  - KDChart::AbstractDiagram, 208
  - KDChart::AbstractPieDiagram, 260
  - KDChart::AbstractPolarDiagram, 304
  - KDChart::AbstractTernaryDiagram, 349
  - KDChart::BarDiagram, 445
  - KDChart::LineDiagram, 731
  - KDChart::PieDiagram, 828
  - KDChart::Plotter, 886
  - KDChart::PolarDiagram, 975
  - KDChart::RingDiagram, 1072
  - KDChart::TernaryLineDiagram, 1178
  - KDChart::TernaryPointDiagram, 1222
- West
  - KDChart::Position, 984
- Widget
  - KDChart::Widget, 1291
- xCenter
  - KDChart::ZoomParameters, 1306
- xFactor
  - KDChart::ZoomParameters, 1306
- yCenter
  - KDChart::ZoomParameters, 1306
- yFactor
  - KDChart::ZoomParameters, 1306
- zeroDegreePosition
  - KDChart::PolarDiagram, 975
- zeroLinePen
  - KDChart::GridAttributes, 610
- zoomCenter
  - KDChart::AbstractCoordinatePlane, 164
  - KDChart::CartesianCoordinatePlane, 541

- KDChart::PolarCoordinatePlane, [924](#)
  - KDChart::TernaryCoordinatePlane, [1132](#)
- zoomFactorX
  - KDChart::AbstractCoordinatePlane, [165](#)
  - KDChart::CartesianCoordinatePlane, [541](#)
  - KDChart::PolarCoordinatePlane, [925](#)
  - KDChart::TernaryCoordinatePlane, [1132](#)
- zoomFactorY
  - KDChart::AbstractCoordinatePlane, [165](#)
  - KDChart::CartesianCoordinatePlane, [542](#)
  - KDChart::PolarCoordinatePlane, [925](#)
  - KDChart::TernaryCoordinatePlane, [1132](#)
- ZoomParameters
  - KDChart::ZoomParameters, [1305](#)