

BTube

Generated by Doxygen 1.8.13

Contents

1	Todo List	1
2	Namespace Index	3
2.1	Namespace List	3
3	Hierarchical Index	5
3.1	Class Hierarchy	5
4	Class Index	9
4.1	Class List	9
5	Namespace Documentation	13
5.1	QCP Namespace Reference	13
5.1.1	Detailed Description	14
5.1.2	Enumeration Type Documentation	14
5.1.2.1	AntialiasedElement	14
5.1.2.2	ExportPen	14
5.1.2.3	Interaction	15
5.1.2.4	MarginSide	15
5.1.2.5	PlottingHint	16
5.1.2.6	ResolutionUnit	16
5.1.2.7	SelectionRectMode	17
5.1.2.8	SelectionType	17
5.1.2.9	SignDomain	18

6	Class Documentation	19
6.1	APITest Class Reference	19
6.1.1	Member Function Documentation	19
6.1.1.1	printTestResult()	19
6.2	QCPAxisPainterPrivate::CachedLabel Struct Reference	20
6.3	DatenEingabe Class Reference	20
6.4	Fluid Class Reference	20
6.4.1	Detailed Description	21
6.4.2	Member Function Documentation	21
6.4.2.1	get_cp_strom()	21
6.4.2.2	get_massenstrom()	22
6.5	MainWindow Class Reference	22
6.6	Plotter Class Reference	22
6.7	QCPAbstractItem Class Reference	23
6.7.1	Detailed Description	25
6.7.2	Clipping	25
6.7.3	Using items	25
6.7.4	Creating own items	26
6.7.4.1	Allowing the item to be positioned	26
6.7.4.2	The draw function	27
6.7.4.3	The selectTest function	27
6.7.4.4	Providing anchors	27
6.7.5	Constructor & Destructor Documentation	27
6.7.5.1	QCPAbstractItem()	27
6.7.6	Member Function Documentation	27
6.7.6.1	anchor()	28
6.7.6.2	anchors()	28
6.7.6.3	hasAnchor()	28
6.7.6.4	position()	29
6.7.6.5	positions()	29

6.7.6.6	selectionChanged	29
6.7.6.7	selectTest()	29
6.7.6.8	setClipAxisRect()	30
6.7.6.9	setClipToAxisRect()	30
6.7.6.10	setSelectable()	31
6.7.6.11	setSelected()	31
6.8	QCPAbstractLegendItem Class Reference	31
6.8.1	Detailed Description	33
6.8.2	Constructor & Destructor Documentation	33
6.8.2.1	QCPAbstractLegendItem()	33
6.8.3	Member Function Documentation	33
6.8.3.1	selectionChanged	33
6.8.3.2	selectTest()	34
6.8.3.3	setFont()	34
6.8.3.4	setSelectable()	34
6.8.3.5	setSelected()	35
6.8.3.6	setSelectedFont()	35
6.8.3.7	setSelectedTextColor()	35
6.8.3.8	setTextColor()	35
6.9	QCPAbstractPaintBuffer Class Reference	36
6.9.1	Detailed Description	36
6.9.2	Constructor & Destructor Documentation	37
6.9.2.1	QCPAbstractPaintBuffer()	37
6.9.3	Member Function Documentation	37
6.9.3.1	clear()	37
6.9.3.2	donePainting()	37
6.9.3.3	draw()	37
6.9.3.4	reallocateBuffer()	38
6.9.3.5	setDevicePixelRatio()	38
6.9.3.6	setInvalidated()	38

6.9.3.7	setSize()	39
6.9.3.8	startPainting()	39
6.10	QCPAbstractPlottable Class Reference	39
6.10.1	Detailed Description	42
6.10.2	Creating own plottables	42
6.10.3	Constructor & Destructor Documentation	43
6.10.3.1	QCPAbstractPlottable()	43
6.10.4	Member Function Documentation	43
6.10.4.1	addToLegend() [1/2]	44
6.10.4.2	addToLegend() [2/2]	44
6.10.4.3	coordsToPixels() [1/2]	44
6.10.4.4	coordsToPixels() [2/2]	45
6.10.4.5	getKeyRange()	45
6.10.4.6	getValueRange()	45
6.10.4.7	interface1D()	46
6.10.4.8	pixelsToCoords() [1/2]	46
6.10.4.9	pixelsToCoords() [2/2]	46
6.10.4.10	removeFromLegend() [1/2]	47
6.10.4.11	removeFromLegend() [2/2]	47
6.10.4.12	rescaleAxes()	47
6.10.4.13	rescaleKeyAxis()	48
6.10.4.14	rescaleValueAxis()	48
6.10.4.15	selectableChanged	48
6.10.4.16	selected()	48
6.10.4.17	selection()	48
6.10.4.18	selectionChanged [1/2]	49
6.10.4.19	selectionChanged [2/2]	49
6.10.4.20	selectionDecorator()	49
6.10.4.21	selectTest()	50
6.10.4.22	setAntialiasedFill()	50

6.10.4.23 setAntialiasedScatters()	51
6.10.4.24 setBrush()	51
6.10.4.25 setKeyAxis()	51
6.10.4.26 setName()	51
6.10.4.27 setPen()	52
6.10.4.28 setSelectable()	52
6.10.4.29 setSelection()	52
6.10.4.30 setSelectionDecorator()	53
6.10.4.31 setValueAxis()	53
6.11 QCPAbstractPlottable1D< DataType > Class Template Reference	53
6.11.1 Detailed Description	54
6.11.2 Constructor & Destructor Documentation	55
6.11.2.1 QCPAbstractPlottable1D()	55
6.11.3 Member Function Documentation	55
6.11.3.1 dataCount()	55
6.11.3.2 dataMainKey()	55
6.11.3.3 dataMainValue()	56
6.11.3.4 dataPixelPosition()	56
6.11.3.5 dataSortKey()	56
6.11.3.6 dataValueRange()	57
6.11.3.7 drawPolyline()	57
6.11.3.8 findBegin()	57
6.11.3.9 findEnd()	58
6.11.3.10 getDataSegments()	58
6.11.3.11 interface1D()	58
6.11.3.12 selectTest()	59
6.11.3.13 selectTestRect()	59
6.11.3.14 sortKeysMainKey()	59
6.12 QCPAxis Class Reference	60
6.12.1 Detailed Description	64

6.12.2	Member Enumeration Documentation	64
6.12.2.1	AxisType	64
6.12.2.2	LabelSide	64
6.12.2.3	ScaleType	65
6.12.2.4	SelectablePart	65
6.12.3	Constructor & Destructor Documentation	66
6.12.3.1	QCPAxis()	66
6.12.4	Member Function Documentation	66
6.12.4.1	coordToPixel()	66
6.12.4.2	getPartAt()	66
6.12.4.3	graphs()	67
6.12.4.4	grid()	67
6.12.4.5	items()	67
6.12.4.6	marginSideToAxisType()	67
6.12.4.7	moveRange()	67
6.12.4.8	opposite()	68
6.12.4.9	orientation() [1/2]	68
6.12.4.10	orientation() [2/2]	68
6.12.4.11	pixelOrientation()	68
6.12.4.12	pixelToCoord()	69
6.12.4.13	plottables()	69
6.12.4.14	rangeChanged [1/2]	69
6.12.4.15	rangeChanged [2/2]	69
6.12.4.16	rescale()	70
6.12.4.17	scaleRange() [1/2]	70
6.12.4.18	scaleRange() [2/2]	70
6.12.4.19	scaleTypeChanged	70
6.12.4.20	selectableChanged	71
6.12.4.21	selectionChanged	71
6.12.4.22	selectTest()	71

6.12.4.23 setBasePen()	72
6.12.4.24 setLabel()	72
6.12.4.25 setLabelColor()	72
6.12.4.26 setLabelFont()	72
6.12.4.27 setLabelPadding()	73
6.12.4.28 setLowerEnding()	73
6.12.4.29 setNumberFormat()	73
6.12.4.30 setNumberPrecision()	74
6.12.4.31 setOffset()	74
6.12.4.32 setPadding()	74
6.12.4.33 setRange() [1/3]	74
6.12.4.34 setRange() [2/3]	75
6.12.4.35 setRange() [3/3]	75
6.12.4.36 setRangeLower()	75
6.12.4.37 setRangeReversed()	75
6.12.4.38 setRangeUpper()	76
6.12.4.39 setScaleRatio()	76
6.12.4.40 setScaleType()	76
6.12.4.41 setSelectableParts()	76
6.12.4.42 setSelectedBasePen()	77
6.12.4.43 setSelectedLabelColor()	77
6.12.4.44 setSelectedLabelFont()	77
6.12.4.45 setSelectedParts()	77
6.12.4.46 setSelectedSubTickPen()	78
6.12.4.47 setSelectedTickLabelColor()	78
6.12.4.48 setSelectedTickLabelFont()	78
6.12.4.49 setSelectedTickPen()	78
6.12.4.50 setSubTickLength()	79
6.12.4.51 setSubTickLengthIn()	79
6.12.4.52 setSubTickLengthOut()	79

6.12.4.53 setSubTickPen()	79
6.12.4.54 setSubTicks()	80
6.12.4.55 setTicker()	80
6.12.4.56 setTickLabelColor()	80
6.12.4.57 setTickLabelFont()	81
6.12.4.58 setTickLabelPadding()	81
6.12.4.59 setTickLabelRotation()	81
6.12.4.60 setTickLabels()	81
6.12.4.61 setTickLabelSide()	81
6.12.4.62 setTickLength()	82
6.12.4.63 setTickLengthIn()	82
6.12.4.64 setTickLengthOut()	82
6.12.4.65 setTickPen()	82
6.12.4.66 setTicks()	83
6.12.4.67 setUpperEnding()	83
6.12.4.68 ticker()	83
6.13 QCPAxisPainterPrivate Class Reference	84
6.13.1 Constructor & Destructor Documentation	85
6.13.1.1 QCPAxisPainterPrivate()	85
6.14 QCPAxisRect Class Reference	85
6.14.1 Detailed Description	88
6.14.2 Constructor & Destructor Documentation	88
6.14.2.1 QCPAxisRect()	88
6.14.3 Member Function Documentation	88
6.14.3.1 addAxes()	89
6.14.3.2 addAxis()	89
6.14.3.3 axes() [1/2]	89
6.14.3.4 axes() [2/2]	90
6.14.3.5 axis()	90
6.14.3.6 axisCount()	90

6.14.3.7	bottom()	90
6.14.3.8	bottomLeft()	90
6.14.3.9	bottomRight()	91
6.14.3.10	center()	91
6.14.3.11	elements()	91
6.14.3.12	graphs()	91
6.14.3.13	height()	91
6.14.3.14	insetLayout()	92
6.14.3.15	items()	92
6.14.3.16	left()	92
6.14.3.17	mouseMoveEvent()	92
6.14.3.18	mousePressEvent()	93
6.14.3.19	mouseReleaseEvent()	93
6.14.3.20	plottables()	94
6.14.3.21	rangeDragAxes()	94
6.14.3.22	rangeDragAxis()	94
6.14.3.23	rangeZoomAxes()	94
6.14.3.24	rangeZoomAxis()	95
6.14.3.25	rangeZoomFactor()	95
6.14.3.26	removeAxis()	95
6.14.3.27	right()	95
6.14.3.28	setBackground() [1/3]	96
6.14.3.29	setBackground() [2/3]	96
6.14.3.30	setBackground() [3/3]	96
6.14.3.31	setBackgroundScaled()	97
6.14.3.32	setBackgroundScaledMode()	97
6.14.3.33	setRangeDrag()	97
6.14.3.34	setRangeDragAxes() [1/3]	98
6.14.3.35	setRangeDragAxes() [2/3]	98
6.14.3.36	setRangeDragAxes() [3/3]	98

6.14.3.37 setRangeZoom()	99
6.14.3.38 setRangeZoomAxes() [1/3]	99
6.14.3.39 setRangeZoomAxes() [2/3]	99
6.14.3.40 setRangeZoomAxes() [3/3]	100
6.14.3.41 setRangeZoomFactor() [1/2]	100
6.14.3.42 setRangeZoomFactor() [2/2]	100
6.14.3.43 setupFullAxesBox()	100
6.14.3.44 size()	101
6.14.3.45 top()	101
6.14.3.46 topLeft()	101
6.14.3.47 topRight()	101
6.14.3.48 update()	101
6.14.3.49 wheelEvent()	102
6.14.3.50 width()	102
6.14.3.51 zoom() [1/2]	102
6.14.3.52 zoom() [2/2]	103
6.15 QCPAxisTicker Class Reference	103
6.15.1 Detailed Description	104
6.15.2 Creating own axis tickers	104
6.15.3 Member Enumeration Documentation	105
6.15.3.1 TickStepStrategy	105
6.15.4 Constructor & Destructor Documentation	105
6.15.4.1 QCPAxisTicker()	105
6.15.5 Member Function Documentation	106
6.15.5.1 generate()	106
6.15.5.2 setTickCount()	106
6.15.5.3 setTickOrigin()	106
6.15.5.4 setTickStepStrategy()	107
6.16 QCPAxisTickerDateTime Class Reference	107
6.16.1 Detailed Description	108

6.16.2	Constructor & Destructor Documentation	109
6.16.2.1	QCPAxisTickerDateTime()	109
6.16.3	Member Function Documentation	109
6.16.3.1	dateTimeToKey() [1/2]	109
6.16.3.2	dateTimeToKey() [2/2]	109
6.16.3.3	keyToDateTime()	110
6.16.3.4	setDateTimeFormat()	110
6.16.3.5	setDateTimeSpec()	110
6.16.3.6	setTickOrigin() [1/2]	111
6.16.3.7	setTickOrigin() [2/2]	111
6.17	QCPAxisTickerFixed Class Reference	111
6.17.1	Detailed Description	112
6.17.2	Member Enumeration Documentation	112
6.17.2.1	ScaleStrategy	112
6.17.3	Constructor & Destructor Documentation	113
6.17.3.1	QCPAxisTickerFixed()	113
6.17.4	Member Function Documentation	113
6.17.4.1	setScaleStrategy()	113
6.17.4.2	setTickStep()	113
6.18	QCPAxisTickerLog Class Reference	114
6.18.1	Detailed Description	114
6.18.2	Constructor & Destructor Documentation	115
6.18.2.1	QCPAxisTickerLog()	115
6.18.3	Member Function Documentation	115
6.18.3.1	setLogBase()	115
6.18.3.2	setSubTickCount()	115
6.19	QCPAxisTickerPi Class Reference	115
6.19.1	Detailed Description	116
6.19.2	Member Enumeration Documentation	117
6.19.2.1	FractionStyle	117

6.19.3	Constructor & Destructor Documentation	117
6.19.3.1	QCPAxisTickerPi()	117
6.19.4	Member Function Documentation	117
6.19.4.1	setFractionStyle()	117
6.19.4.2	setPeriodicity()	118
6.19.4.3	setPiSymbol()	118
6.19.4.4	setPiValue()	118
6.20	QCPAxisTickerText Class Reference	118
6.20.1	Detailed Description	119
6.20.2	Constructor & Destructor Documentation	119
6.20.2.1	QCPAxisTickerText()	120
6.20.3	Member Function Documentation	120
6.20.3.1	addTick()	120
6.20.3.2	addTicks() [1/2]	120
6.20.3.3	addTicks() [2/2]	121
6.20.3.4	clear()	121
6.20.3.5	createTickVector()	121
6.20.3.6	getSubTickCount()	122
6.20.3.7	getTickLabel()	122
6.20.3.8	getTickStep()	122
6.20.3.9	setSubTickCount()	122
6.20.3.10	setTicks() [1/2]	123
6.20.3.11	setTicks() [2/2]	123
6.20.3.12	ticks()	123
6.21	QCPAxisTickerTime Class Reference	124
6.21.1	Detailed Description	125
6.21.2	Member Enumeration Documentation	125
6.21.2.1	TimeUnit	125
6.21.3	Constructor & Destructor Documentation	125
6.21.3.1	QCPAxisTickerTime()	126

6.21.4	Member Function Documentation	126
6.21.4.1	setFieldWidth()	126
6.21.4.2	setTimeFormat()	126
6.22	QCPBars Class Reference	127
6.22.1	Detailed Description	128
6.22.2	Changing the appearance	128
6.22.3	Usage	129
6.22.4	Member Enumeration Documentation	129
6.22.4.1	WidthType	129
6.22.5	Constructor & Destructor Documentation	129
6.22.5.1	QCPBars()	129
6.22.6	Member Function Documentation	130
6.22.6.1	addData() [1/2]	130
6.22.6.2	addData() [2/2]	130
6.22.6.3	barAbove()	130
6.22.6.4	barBelow()	131
6.22.6.5	data()	131
6.22.6.6	dataPixelPosition()	131
6.22.6.7	getKeyRange()	131
6.22.6.8	getValueRange()	132
6.22.6.9	moveAbove()	132
6.22.6.10	moveBelow()	133
6.22.6.11	selectTest()	133
6.22.6.12	selectTestRect()	133
6.22.6.13	setBarsGroup()	134
6.22.6.14	setBaseValue()	134
6.22.6.15	setData() [1/2]	134
6.22.6.16	setData() [2/2]	135
6.22.6.17	setStackingGap()	135
6.22.6.18	setWidth()	135

6.22.6.19	setWidthType()	135
6.23	QCPBarsData Class Reference	136
6.23.1	Detailed Description	136
6.23.2	Constructor & Destructor Documentation	136
6.23.2.1	QCPBarsData() [1/2]	137
6.23.2.2	QCPBarsData() [2/2]	137
6.23.3	Member Function Documentation	137
6.23.3.1	fromSortKey()	137
6.23.3.2	mainKey()	137
6.23.3.3	mainValue()	137
6.23.3.4	sortKey()	138
6.23.3.5	sortKeyIsMainKey()	138
6.23.3.6	valueRange()	138
6.24	QCPBarsGroup Class Reference	138
6.24.1	Detailed Description	139
6.24.2	Usage	140
6.24.3	Example	140
6.24.4	Member Enumeration Documentation	140
6.24.4.1	SpacingType	140
6.24.5	Constructor & Destructor Documentation	141
6.24.5.1	QCPBarsGroup()	141
6.24.6	Member Function Documentation	141
6.24.6.1	append()	141
6.24.6.2	bars() [1/2]	141
6.24.6.3	bars() [2/2]	141
6.24.6.4	clear()	142
6.24.6.5	contains()	142
6.24.6.6	insert()	142
6.24.6.7	isEmpty()	142
6.24.6.8	remove()	143

6.24.6.9	setSpacing()	143
6.24.6.10	setSpacingType()	143
6.24.6.11	size()	143
6.25	QCPColorGradient Class Reference	144
6.25.1	Detailed Description	145
6.25.2	Member Enumeration Documentation	145
6.25.2.1	ColorInterpolation	145
6.25.2.2	GradientPreset	145
6.25.3	Constructor & Destructor Documentation	146
6.25.3.1	QCPColorGradient() [1/2]	146
6.25.3.2	QCPColorGradient() [2/2]	146
6.25.4	Member Function Documentation	147
6.25.4.1	clearColorStops()	147
6.25.4.2	colorize() [1/2]	147
6.25.4.3	colorize() [2/2]	147
6.25.4.4	inverted()	148
6.25.4.5	loadPreset()	148
6.25.4.6	setColorInterpolation()	148
6.25.4.7	setColorStopAt()	148
6.25.4.8	setColorStops()	149
6.25.4.9	setLevelCount()	149
6.25.4.10	setPeriodic()	149
6.26	QCPColorMap Class Reference	150
6.26.1	Detailed Description	151
6.26.2	Changing the appearance	152
6.26.3	Transparency	152
6.26.4	Usage	152
6.26.5	Constructor & Destructor Documentation	152
6.26.5.1	QCPColorMap()	153
6.26.6	Member Function Documentation	153

6.26.6.1	data()	153
6.26.6.2	dataRangeChanged	153
6.26.6.3	dataScaleTypeChanged	153
6.26.6.4	getKeyRange()	154
6.26.6.5	getValueRange()	154
6.26.6.6	gradientChanged	155
6.26.6.7	rescaleDataRange()	155
6.26.6.8	selectTest()	155
6.26.6.9	setColorScale()	156
6.26.6.10	setData()	156
6.26.6.11	setDataRange()	157
6.26.6.12	setDataScaleType()	157
6.26.6.13	setGradient()	157
6.26.6.14	setInterpolate()	158
6.26.6.15	setTightBoundary()	158
6.26.6.16	updateLegendIcon()	158
6.27	QCPColorMapData Class Reference	159
6.27.1	Detailed Description	160
6.27.2	Constructor & Destructor Documentation	160
6.27.2.1	QCPColorMapData() [1/2]	160
6.27.2.2	QCPColorMapData() [2/2]	161
6.27.3	Member Function Documentation	161
6.27.3.1	alpha()	161
6.27.3.2	cellToCoord()	161
6.27.3.3	clear()	162
6.27.3.4	clearAlpha()	162
6.27.3.5	coordToCell()	162
6.27.3.6	fill()	162
6.27.3.7	fillAlpha()	163
6.27.3.8	isEmpty()	163

6.27.3.9	operator=()	163
6.27.3.10	recalculateDataBounds()	163
6.27.3.11	setAlpha()	164
6.27.3.12	setCell()	164
6.27.3.13	setData()	164
6.27.3.14	setKeyRange()	165
6.27.3.15	setKeySize()	165
6.27.3.16	setRange()	165
6.27.3.17	setSize()	166
6.27.3.18	setValueRange()	166
6.27.3.19	setValueSize()	166
6.28	QCPColorScale Class Reference	167
6.28.1	Detailed Description	168
6.28.2	Constructor & Destructor Documentation	169
6.28.2.1	QCPColorScale()	169
6.28.3	Member Function Documentation	169
6.28.3.1	axis()	169
6.28.3.2	colorMaps()	169
6.28.3.3	dataRangeChanged	169
6.28.3.4	dataScaleTypeChanged	170
6.28.3.5	gradientChanged	170
6.28.3.6	mouseMoveEvent()	170
6.28.3.7	mousePressEvent()	171
6.28.3.8	mouseReleaseEvent()	171
6.28.3.9	rescaleDataRange()	172
6.28.3.10	setBarWidth()	172
6.28.3.11	setDataRange()	172
6.28.3.12	setDataScaleType()	172
6.28.3.13	setGradient()	173
6.28.3.14	setLabel()	173

6.28.3.15 setRangeDrag()	173
6.28.3.16 setRangeZoom()	173
6.28.3.17 setType()	173
6.28.3.18 update()	174
6.28.3.19 wheelEvent()	174
6.29 QCPCColorScaleAxisRectPrivate Class Reference	175
6.29.1 Constructor & Destructor Documentation	175
6.29.1.1 QCPCColorScaleAxisRectPrivate()	176
6.30 QCPCurve Class Reference	176
6.30.1 Detailed Description	178
6.30.2 Changing the appearance	178
6.30.3 Usage	178
6.30.4 Member Enumeration Documentation	178
6.30.4.1 LineStyle	178
6.30.5 Constructor & Destructor Documentation	179
6.30.5.1 QCPCurve()	179
6.30.6 Member Function Documentation	179
6.30.6.1 addData() [1/4]	179
6.30.6.2 addData() [2/4]	180
6.30.6.3 addData() [3/4]	180
6.30.6.4 addData() [4/4]	180
6.30.6.5 data()	180
6.30.6.6 getKeyRange()	181
6.30.6.7 getValueRange()	181
6.30.6.8 selectTest()	182
6.30.6.9 setData() [1/3]	182
6.30.6.10 setData() [2/3]	182
6.30.6.11 setData() [3/3]	183
6.30.6.12 setLineStyle()	183
6.30.6.13 setScatterSkip()	183

6.30.6.14	setScatterStyle()	184
6.31	QCPCurveData Class Reference	184
6.31.1	Detailed Description	185
6.31.2	Constructor & Destructor Documentation	185
6.31.2.1	QCPCurveData() [1/2]	185
6.31.2.2	QCPCurveData() [2/2]	185
6.31.3	Member Function Documentation	185
6.31.3.1	fromSortKey()	186
6.31.3.2	mainKey()	186
6.31.3.3	mainValue()	186
6.31.3.4	sortKey()	186
6.31.3.5	sortKeysMainKey()	186
6.31.3.6	valueRange()	187
6.32	QCPCDataContainer< DataType > Class Template Reference	187
6.32.1	Detailed Description	188
6.32.2	Requirements for the DataType template parameter	189
6.32.3	Constructor & Destructor Documentation	189
6.32.3.1	QCPCDataContainer()	189
6.32.4	Member Function Documentation	189
6.32.4.1	add() [1/3]	190
6.32.4.2	add() [2/3]	190
6.32.4.3	add() [3/3]	190
6.32.4.4	at()	191
6.32.4.5	begin()	191
6.32.4.6	clear()	191
6.32.4.7	constBegin()	191
6.32.4.8	constEnd()	191
6.32.4.9	dataRange()	192
6.32.4.10	end()	192
6.32.4.11	findBegin()	192

6.32.4.12 findEnd()	192
6.32.4.13 isEmpty()	193
6.32.4.14 keyRange()	193
6.32.4.15 limitIteratorsToDataRange()	193
6.32.4.16 remove() [1/2]	193
6.32.4.17 remove() [2/2]	194
6.32.4.18 removeAfter()	194
6.32.4.19 removeBefore()	194
6.32.4.20 set() [1/2]	195
6.32.4.21 set() [2/2]	195
6.32.4.22 setAutoSqueeze()	195
6.32.4.23 size()	196
6.32.4.24 sort()	196
6.32.4.25 squeeze()	196
6.32.4.26 valueRange()	196
6.32.5 Friends And Related Function Documentation	197
6.32.5.1 qcpLessThanSortKey()	197
6.33 QCPDataRange Class Reference	197
6.33.1 Detailed Description	198
6.33.2 Constructor & Destructor Documentation	198
6.33.2.1 QCPDataRange() [1/2]	198
6.33.2.2 QCPDataRange() [2/2]	198
6.33.3 Member Function Documentation	199
6.33.3.1 adjusted()	199
6.33.3.2 bounded()	199
6.33.3.3 contains()	199
6.33.3.4 expanded()	199
6.33.3.5 intersection()	200
6.33.3.6 intersects()	200
6.33.3.7 isEmpty()	200

6.33.3.8	isValid()	200
6.33.3.9	length()	201
6.33.3.10	setBegin()	201
6.33.3.11	setEnd()	201
6.33.3.12	size()	201
6.33.4	Friends And Related Function Documentation	202
6.33.4.1	operator<<()	202
6.34	QCPDataSelection Class Reference	202
6.34.1	Detailed Description	203
6.34.2	Iterating over a data selection	203
6.34.3	Constructor & Destructor Documentation	203
6.34.3.1	QCPDataSelection() [1/2]	204
6.34.3.2	QCPDataSelection() [2/2]	204
6.34.4	Member Function Documentation	204
6.34.4.1	addDataRange()	204
6.34.4.2	clear()	204
6.34.4.3	contains()	204
6.34.4.4	dataPointCount()	205
6.34.4.5	dataRange()	205
6.34.4.6	dataRangeCount()	205
6.34.4.7	dataRanges()	205
6.34.4.8	enforceType()	206
6.34.4.9	intersection() [1/2]	206
6.34.4.10	intersection() [2/2]	206
6.34.4.11	inverse()	206
6.34.4.12	isEmpty()	207
6.34.4.13	operator+=() [1/2]	207
6.34.4.14	operator+=() [2/2]	207
6.34.4.15	operator-=() [1/2]	207
6.34.4.16	operator-=() [2/2]	207

6.34.4.17	operator==()	207
6.34.4.18	simplify()	208
6.34.4.19	span()	208
6.34.5	Friends And Related Function Documentation	208
6.34.5.1	operator+ [1/4]	208
6.34.5.2	operator+ [2/4]	208
6.34.5.3	operator+ [3/4]	208
6.34.5.4	operator+ [4/4]	209
6.34.5.5	operator- [1/4]	209
6.34.5.6	operator- [2/4]	209
6.34.5.7	operator- [3/4]	209
6.34.5.8	operator- [4/4]	209
6.34.5.9	operator<<()	209
6.35	QCPErrors Class Reference	210
6.35.1	Detailed Description	211
6.35.2	Changing the appearance	212
6.35.3	Member Enumeration Documentation	212
6.35.3.1	ErrorType	212
6.35.4	Constructor & Destructor Documentation	212
6.35.4.1	QCPErrors()	212
6.35.5	Member Function Documentation	213
6.35.5.1	addData() [1/4]	213
6.35.5.2	addData() [2/4]	213
6.35.5.3	addData() [3/4]	213
6.35.5.4	addData() [4/4]	214
6.35.5.5	data()	214
6.35.5.6	dataCount()	214
6.35.5.7	dataMainKey()	214
6.35.5.8	dataMainValue()	215
6.35.5.9	dataPixelPosition()	215

6.35.5.10 dataSortKey()	215
6.35.5.11 dataValueRange()	215
6.35.5.12 findBegin()	216
6.35.5.13 findEnd()	216
6.35.5.14 getKeyRange()	217
6.35.5.15 getValueRange()	217
6.35.5.16 interface1D()	218
6.35.5.17 selectTest()	218
6.35.5.18 selectTestRect()	219
6.35.5.19 setData() [1/3]	219
6.35.5.20 setData() [2/3]	220
6.35.5.21 setData() [3/3]	220
6.35.5.22 setDataPlottable()	220
6.35.5.23 setErrorType()	221
6.35.5.24 setSymbolGap()	221
6.35.5.25 setWhiskerWidth()	221
6.35.5.26 sortKeysMainKey()	221
6.36 QCPErrorsData Class Reference	221
6.36.1 Detailed Description	222
6.36.2 Constructor & Destructor Documentation	222
6.36.2.1 QCPErrorsData() [1/3]	222
6.36.2.2 QCPErrorsData() [2/3]	222
6.36.2.3 QCPErrorsData() [3/3]	222
6.37 QCPFinancial Class Reference	223
6.37.1 Detailed Description	224
6.37.2 Changing the appearance	225
6.37.3 Usage	225
6.37.4 Member Enumeration Documentation	225
6.37.4.1 ChartStyle	225
6.37.4.2 WidthType	226

6.37.5	Constructor & Destructor Documentation	226
6.37.5.1	QCPFinancial()	226
6.37.6	Member Function Documentation	226
6.37.6.1	addData() [1/2]	227
6.37.6.2	addData() [2/2]	227
6.37.6.3	data()	228
6.37.6.4	getKeyRange()	228
6.37.6.5	getValueRange()	228
6.37.6.6	selectTest()	229
6.37.6.7	selectTestRect()	229
6.37.6.8	setBrushNegative()	229
6.37.6.9	setBrushPositive()	230
6.37.6.10	setChartStyle()	230
6.37.6.11	setData() [1/2]	230
6.37.6.12	setData() [2/2]	231
6.37.6.13	setPenNegative()	231
6.37.6.14	setPenPositive()	231
6.37.6.15	setTwoColored()	232
6.37.6.16	setWidth()	232
6.37.6.17	setWidthType()	232
6.37.6.18	timeSeriesToOhlc()	233
6.38	QCPFinancialData Class Reference	233
6.38.1	Detailed Description	234
6.38.2	Constructor & Destructor Documentation	234
6.38.2.1	QCPFinancialData() [1/2]	234
6.38.2.2	QCPFinancialData() [2/2]	234
6.38.3	Member Function Documentation	234
6.38.3.1	fromSortKey()	235
6.38.3.2	mainKey()	235
6.38.3.3	mainValue()	235

6.38.3.4	sortKey()	235
6.38.3.5	sortKeysMainKey()	235
6.38.3.6	valueRange()	236
6.39	QCPGraph Class Reference	236
6.39.1	Detailed Description	238
6.39.2	Changing the appearance	238
6.39.2.1	Filling under or between graphs	238
6.39.3	Member Enumeration Documentation	239
6.39.3.1	LineStyle	239
6.39.4	Constructor & Destructor Documentation	239
6.39.4.1	QCPGraph()	239
6.39.5	Member Function Documentation	240
6.39.5.1	addData() [1/2]	240
6.39.5.2	addData() [2/2]	240
6.39.5.3	data()	240
6.39.5.4	getKeyRange()	241
6.39.5.5	getValueRange()	241
6.39.5.6	getVisibleDataBounds()	242
6.39.5.7	selectTest()	242
6.39.5.8	setAdaptiveSampling()	242
6.39.5.9	setChannelFillGraph()	243
6.39.5.10	setData() [1/2]	243
6.39.5.11	setData() [2/2]	244
6.39.5.12	setLineStyle()	244
6.39.5.13	setScatterSkip()	244
6.39.5.14	setScatterStyle()	245
6.40	QCPGraphData Class Reference	245
6.40.1	Detailed Description	246
6.40.2	Constructor & Destructor Documentation	246
6.40.2.1	QCPGraphData() [1/2]	246

6.40.2.2	QCPGraphData() [2/2]	246
6.40.3	Member Function Documentation	246
6.40.3.1	fromSortKey()	246
6.40.3.2	mainKey()	247
6.40.3.3	mainValue()	247
6.40.3.4	sortKey()	247
6.40.3.5	sortKeysIsMainKey()	247
6.40.3.6	valueRange()	247
6.41	QCPGrid Class Reference	248
6.41.1	Detailed Description	249
6.41.2	Constructor & Destructor Documentation	249
6.41.2.1	QCPGrid()	249
6.41.3	Member Function Documentation	249
6.41.3.1	setAntialiasedSubGrid()	249
6.41.3.2	setAntialiasedZeroLine()	249
6.41.3.3	setPen()	250
6.41.3.4	setSubGridPen()	250
6.41.3.5	setSubGridVisible()	250
6.41.3.6	setZeroLinePen()	250
6.42	QCPIItemAnchor Class Reference	251
6.42.1	Detailed Description	251
6.42.2	Constructor & Destructor Documentation	252
6.42.2.1	QCPIItemAnchor()	252
6.42.3	Member Function Documentation	252
6.42.3.1	pixelPosition()	252
6.42.3.2	toQCPIItemPosition()	252
6.43	QCPIItemBracket Class Reference	253
6.43.1	Detailed Description	254
6.43.2	Member Enumeration Documentation	254
6.43.2.1	BracketStyle	254

6.43.3	Constructor & Destructor Documentation	255
6.43.3.1	QCPIItemBracket()	255
6.43.4	Member Function Documentation	255
6.43.4.1	selectTest()	255
6.43.4.2	setLength()	256
6.43.4.3	setPen()	256
6.43.4.4	setSelectedPen()	256
6.43.4.5	setStyle()	257
6.44	QCPIItemCurve Class Reference	257
6.44.1	Detailed Description	258
6.44.2	Constructor & Destructor Documentation	258
6.44.2.1	QCPIItemCurve()	258
6.44.3	Member Function Documentation	259
6.44.3.1	selectTest()	259
6.44.3.2	setHead()	259
6.44.3.3	setPen()	260
6.44.3.4	setSelectedPen()	260
6.44.3.5	setTail()	260
6.45	QCPIItemEllipse Class Reference	261
6.45.1	Detailed Description	262
6.45.2	Constructor & Destructor Documentation	262
6.45.2.1	QCPIItemEllipse()	262
6.45.3	Member Function Documentation	262
6.45.3.1	selectTest()	263
6.45.3.2	setBrush()	263
6.45.3.3	setPen()	264
6.45.3.4	setSelectedBrush()	264
6.45.3.5	setSelectedPen()	264
6.46	QCPIItemLine Class Reference	265
6.46.1	Detailed Description	266

6.46.2	Constructor & Destructor Documentation	266
6.46.2.1	QCPItemLine()	266
6.46.3	Member Function Documentation	266
6.46.3.1	selectTest()	266
6.46.3.2	setHead()	267
6.46.3.3	setPen()	267
6.46.3.4	setSelectedPen()	267
6.46.3.5	setTail()	268
6.47	QCPItemPixmap Class Reference	268
6.47.1	Detailed Description	269
6.47.2	Constructor & Destructor Documentation	270
6.47.2.1	QCPItemPixmap()	270
6.47.3	Member Function Documentation	270
6.47.3.1	selectTest()	270
6.47.3.2	setPen()	271
6.47.3.3	setPixmap()	271
6.47.3.4	setScaled()	271
6.47.3.5	setSelectedPen()	271
6.48	QCPItemPosition Class Reference	272
6.48.1	Detailed Description	273
6.48.2	Member Enumeration Documentation	273
6.48.2.1	PositionType	273
6.48.3	Constructor & Destructor Documentation	274
6.48.3.1	QCPItemPosition()	274
6.48.4	Member Function Documentation	274
6.48.4.1	parentAnchor()	274
6.48.4.2	pixelPosition()	275
6.48.4.3	setAxes()	275
6.48.4.4	setAxisRect()	275
6.48.4.5	setCoords() [1/2]	275

6.48.4.6	setCoords() [2/2]	276
6.48.4.7	setParentAnchor()	276
6.48.4.8	setParentAnchorX()	276
6.48.4.9	setParentAnchorY()	277
6.48.4.10	setPixelPosition()	277
6.48.4.11	setType()	277
6.48.4.12	setTypeX()	278
6.48.4.13	setTypeY()	278
6.48.4.14	toQCPItemPosition()	278
6.48.4.15	type()	279
6.49	QCPItemRect Class Reference	279
6.49.1	Detailed Description	280
6.49.2	Constructor & Destructor Documentation	280
6.49.2.1	QCPItemRect()	281
6.49.3	Member Function Documentation	281
6.49.3.1	selectTest()	281
6.49.3.2	setBrush()	282
6.49.3.3	setPen()	282
6.49.3.4	setSelectedBrush()	282
6.49.3.5	setSelectedPen()	282
6.50	QCPItemStraightLine Class Reference	283
6.50.1	Detailed Description	283
6.50.2	Constructor & Destructor Documentation	284
6.50.2.1	QCPItemStraightLine()	284
6.50.3	Member Function Documentation	284
6.50.3.1	selectTest()	284
6.50.3.2	setPen()	285
6.50.3.3	setSelectedPen()	285
6.51	QCPItemText Class Reference	285
6.51.1	Detailed Description	287

6.51.2	Constructor & Destructor Documentation	287
6.51.2.1	QCPIItemText()	287
6.51.3	Member Function Documentation	288
6.51.3.1	selectTest()	288
6.51.3.2	setBrush()	288
6.51.3.3	setColor()	289
6.51.3.4	setFont()	289
6.51.3.5	setPadding()	289
6.51.3.6	setPen()	289
6.51.3.7	setPositionAlignment()	289
6.51.3.8	setRotation()	290
6.51.3.9	setSelectedBrush()	290
6.51.3.10	setSelectedColor()	290
6.51.3.11	setSelectedFont()	290
6.51.3.12	setSelectedPen()	290
6.51.3.13	setText()	291
6.51.3.14	setTextAlignment()	291
6.52	QCPIItemTracer Class Reference	291
6.52.1	Detailed Description	293
6.52.2	Member Enumeration Documentation	293
6.52.2.1	TracerStyle	293
6.52.3	Constructor & Destructor Documentation	294
6.52.3.1	QCPIItemTracer()	294
6.52.4	Member Function Documentation	294
6.52.4.1	selectTest()	294
6.52.4.2	setBrush()	295
6.52.4.3	setGraph()	295
6.52.4.4	setGraphKey()	295
6.52.4.5	setInterpolating()	296
6.52.4.6	setPen()	296

6.52.4.7	setSelectedBrush()	296
6.52.4.8	setSelectedPen()	296
6.52.4.9	setSize()	297
6.52.4.10	setStyle()	297
6.52.4.11	updatePosition()	297
6.53	QCPLayer Class Reference	297
6.53.1	Detailed Description	298
6.53.2	Default layers	299
6.53.3	Controlling the rendering order via layers	299
6.53.4	Replotting only a specific layer	299
6.53.5	Member Enumeration Documentation	299
6.53.5.1	LayerMode	299
6.53.6	Constructor & Destructor Documentation	300
6.53.6.1	QCPLayer()	300
6.53.7	Member Function Documentation	300
6.53.7.1	children()	300
6.53.7.2	index()	300
6.53.7.3	replot()	301
6.53.7.4	setMode()	301
6.53.7.5	setVisible()	301
6.54	QCPLayerable Class Reference	302
6.54.1	Detailed Description	303
6.54.2	Constructor & Destructor Documentation	303
6.54.2.1	QCPLayerable()	303
6.54.3	Member Function Documentation	304
6.54.3.1	layerChanged	304
6.54.3.2	mouseDoubleClickEvent()	304
6.54.3.3	mouseMoveEvent()	305
6.54.3.4	mousePressEvent()	305
6.54.3.5	mouseReleaseEvent()	306

6.54.3.6	parentLayerable()	306
6.54.3.7	realVisibility()	306
6.54.3.8	selectTest()	307
6.54.3.9	setAntialiased()	307
6.54.3.10	setLayer() [1/2]	308
6.54.3.11	setLayer() [2/2]	308
6.54.3.12	setVisible()	308
6.54.3.13	wheelEvent()	308
6.55	QCPLayout Class Reference	309
6.55.1	Detailed Description	310
6.55.2	Constructor & Destructor Documentation	310
6.55.2.1	QCPLayout()	310
6.55.3	Member Function Documentation	310
6.55.3.1	clear()	310
6.55.3.2	elementAt()	311
6.55.3.3	elementCount()	311
6.55.3.4	elements()	311
6.55.3.5	remove()	312
6.55.3.6	removeAt()	312
6.55.3.7	simplify()	312
6.55.3.8	sizeConstraintsChanged()	313
6.55.3.9	take()	313
6.55.3.10	takeAt()	313
6.55.3.11	update()	314
6.56	QCPLayoutElement Class Reference	314
6.56.1	Detailed Description	316
6.56.2	Member Enumeration Documentation	316
6.56.2.1	UpdatePhase	316
6.56.3	Constructor & Destructor Documentation	316
6.56.3.1	QCPLayoutElement()	317

6.56.4	Member Function Documentation	317
6.56.4.1	elements()	317
6.56.4.2	layout()	317
6.56.4.3	maximumSizeHint()	317
6.56.4.4	minimumSizeHint()	318
6.56.4.5	rect()	318
6.56.4.6	selectTest()	318
6.56.4.7	setAutoMargins()	319
6.56.4.8	setMarginGroup()	319
6.56.4.9	setMargins()	319
6.56.4.10	setMaximumSize() [1/2]	320
6.56.4.11	setMaximumSize() [2/2]	320
6.56.4.12	setMinimumMargins()	320
6.56.4.13	setMinimumSize() [1/2]	320
6.56.4.14	setMinimumSize() [2/2]	321
6.56.4.15	setOuterRect()	321
6.56.4.16	update()	321
6.57	QCPLayoutGrid Class Reference	322
6.57.1	Detailed Description	323
6.57.2	Member Enumeration Documentation	323
6.57.2.1	FillOrder	323
6.57.3	Constructor & Destructor Documentation	324
6.57.3.1	QCPLayoutGrid()	324
6.57.4	Member Function Documentation	324
6.57.4.1	addElement() [1/2]	324
6.57.4.2	addElement() [2/2]	325
6.57.4.3	columnCount()	325
6.57.4.4	element()	325
6.57.4.5	elementAt()	326
6.57.4.6	elementCount()	326

6.57.4.7	elements()	326
6.57.4.8	expandTo()	327
6.57.4.9	hasElement()	327
6.57.4.10	indexToRowCol()	327
6.57.4.11	insertColumn()	328
6.57.4.12	insertRow()	328
6.57.4.13	maximumSizeHint()	328
6.57.4.14	minimumSizeHint()	328
6.57.4.15	rowColToIndex()	329
6.57.4.16	rowCount()	329
6.57.4.17	setColumnSpacing()	329
6.57.4.18	setColumnStretchFactor()	330
6.57.4.19	setColumnStretchFactors()	330
6.57.4.20	setFillOrder()	330
6.57.4.21	setRowSpacing()	331
6.57.4.22	setRowStretchFactor()	331
6.57.4.23	setRowStretchFactors()	331
6.57.4.24	setWrap()	332
6.57.4.25	simplify()	332
6.57.4.26	take()	332
6.57.4.27	takeAt()	333
6.58	QCPLayoutInset Class Reference	333
6.58.1	Detailed Description	334
6.58.2	Member Enumeration Documentation	334
6.58.2.1	InsetPlacement	334
6.58.3	Constructor & Destructor Documentation	335
6.58.3.1	QCPLayoutInset()	335
6.58.4	Member Function Documentation	335
6.58.4.1	addElement() [1/2]	335
6.58.4.2	addElement() [2/2]	335

6.58.4.3	elementAt()	336
6.58.4.4	elementCount()	336
6.58.4.5	insetAlignment()	336
6.58.4.6	insetPlacement()	336
6.58.4.7	insetRect()	337
6.58.4.8	selectTest()	337
6.58.4.9	setInsetAlignment()	337
6.58.4.10	setInsetPlacement()	337
6.58.4.11	setInsetRect()	338
6.58.4.12	simplify()	338
6.58.4.13	take()	338
6.58.4.14	takeAt()	339
6.59	QCPLegend Class Reference	339
6.59.1	Detailed Description	341
6.59.2	Member Enumeration Documentation	342
6.59.2.1	SelectablePart	342
6.59.3	Constructor & Destructor Documentation	342
6.59.3.1	QCPLegend()	342
6.59.4	Member Function Documentation	342
6.59.4.1	addItem()	342
6.59.4.2	clearItems()	343
6.59.4.3	hasItem()	343
6.59.4.4	hasItemWithPlottable()	343
6.59.4.5	item()	343
6.59.4.6	itemCount()	344
6.59.4.7	itemWithPlottable()	344
6.59.4.8	removeItem() [1/2]	344
6.59.4.9	removeItem() [2/2]	345
6.59.4.10	selectedItems()	345
6.59.4.11	selectionChanged	345

6.59.4.12 selectTest()	346
6.59.4.13 setBorderPen()	346
6.59.4.14 setBrush()	346
6.59.4.15 setFont()	346
6.59.4.16 setIconBorderPen()	347
6.59.4.17 setIconSize() [1/2]	347
6.59.4.18 setIconSize() [2/2]	347
6.59.4.19 setIconTextPadding()	347
6.59.4.20 setSelectableParts()	347
6.59.4.21 setSelectedBorderPen()	348
6.59.4.22 setSelectedBrush()	348
6.59.4.23 setSelectedFont()	348
6.59.4.24 setSelectedIconBorderPen()	348
6.59.4.25 setSelectedParts()	349
6.59.4.26 setSelectedTextColor()	349
6.59.4.27 setTextColor()	349
6.60 QCPLineEnding Class Reference	350
6.60.1 Detailed Description	350
6.60.2 Member Enumeration Documentation	351
6.60.2.1 EndingStyle	351
6.60.3 Constructor & Destructor Documentation	351
6.60.3.1 QCPLineEnding() [1/2]	351
6.60.3.2 QCPLineEnding() [2/2]	351
6.60.4 Member Function Documentation	352
6.60.4.1 realLength()	352
6.60.4.2 setInverted()	352
6.60.4.3 setLength()	352
6.60.4.4 setStyle()	352
6.60.4.5 setWidth()	353
6.61 QCPLMarginGroup Class Reference	353

6.61.1 Detailed Description	354
6.61.2 Example	354
6.61.3 Constructor & Destructor Documentation	354
6.61.3.1 QCPMarginGroup()	354
6.61.4 Member Function Documentation	355
6.61.4.1 clear()	355
6.61.4.2 elements()	355
6.61.4.3 isEmpty()	355
6.62 QCPPaintBufferPixmap Class Reference	355
6.62.1 Detailed Description	356
6.62.2 Constructor & Destructor Documentation	356
6.62.2.1 QCPPaintBufferPixmap()	356
6.62.3 Member Function Documentation	356
6.62.3.1 clear()	356
6.62.3.2 draw()	357
6.62.3.3 reallocateBuffer()	357
6.62.3.4 startPainting()	357
6.63 QCPPainter Class Reference	358
6.63.1 Detailed Description	358
6.63.2 Member Enumeration Documentation	359
6.63.2.1 PainterMode	359
6.63.3 Constructor & Destructor Documentation	359
6.63.3.1 QCPPainter() [1/2]	359
6.63.3.2 QCPPainter() [2/2]	359
6.63.4 Member Function Documentation	359
6.63.4.1 begin()	360
6.63.4.2 drawLine()	360
6.63.4.3 makeNonCosmetic()	360
6.63.4.4 restore()	360
6.63.4.5 save()	361

6.63.4.6	setAntialiasing()	361
6.63.4.7	setMode()	361
6.63.4.8	setModes()	361
6.63.4.9	setPen() [1/3]	362
6.63.4.10	setPen() [2/3]	362
6.63.4.11	setPen() [3/3]	362
6.64	QCPPlottableInterface1D Class Reference	363
6.64.1	Detailed Description	363
6.64.2	Member Function Documentation	363
6.64.2.1	dataCount()	364
6.64.2.2	dataMainKey()	364
6.64.2.3	dataMainValue()	364
6.64.2.4	dataPixelPosition()	364
6.64.2.5	dataSortKey()	365
6.64.2.6	dataValueRange()	365
6.64.2.7	findBegin()	365
6.64.2.8	findEnd()	366
6.64.2.9	selectTestRect()	366
6.64.2.10	sortKeysMainKey()	367
6.65	QCPPlottableLegendItem Class Reference	367
6.65.1	Detailed Description	368
6.65.2	Constructor & Destructor Documentation	368
6.65.2.1	QCPPlottableLegendItem()	368
6.65.3	Member Function Documentation	368
6.65.3.1	minimumSizeHint()	369
6.66	QCPRange Class Reference	369
6.66.1	Detailed Description	370
6.66.2	Constructor & Destructor Documentation	370
6.66.2.1	QCPRange() [1/2]	370
6.66.2.2	QCPRange() [2/2]	371

6.66.3	Member Function Documentation	371
6.66.3.1	bounded()	371
6.66.3.2	center()	371
6.66.3.3	contains()	371
6.66.3.4	expand() [1/2]	372
6.66.3.5	expand() [2/2]	372
6.66.3.6	expanded() [1/2]	372
6.66.3.7	expanded() [2/2]	373
6.66.3.8	normalize()	373
6.66.3.9	operator*=()	373
6.66.3.10	operator+=()	373
6.66.3.11	operator-=()	373
6.66.3.12	operator/=()	374
6.66.3.13	sanitizedForLinScale()	374
6.66.3.14	sanitizedForLogScale()	374
6.66.3.15	size()	374
6.66.3.16	validRange() [1/2]	374
6.66.3.17	validRange() [2/2]	375
6.66.4	Friends And Related Function Documentation	375
6.66.4.1	operator* [1/2]	375
6.66.4.2	operator* [2/2]	375
6.66.4.3	operator+ [1/2]	375
6.66.4.4	operator+ [2/2]	375
6.66.4.5	operator-	376
6.66.4.6	operator/	376
6.66.4.7	operator<<()	376
6.66.5	Member Data Documentation	376
6.66.5.1	maxRange	376
6.66.5.2	minRange	377
6.67	QCPScatterStyle Class Reference	377

6.67.1 Detailed Description	378
6.67.2 Specifying a scatter style	379
6.67.3 Leaving the color/pen up to the plottable	379
6.67.4 Custom shapes and pixmaps	379
6.67.5 Member Enumeration Documentation	379
6.67.5.1 ScatterProperty	379
6.67.5.2 ScatterShape	380
6.67.6 Constructor & Destructor Documentation	380
6.67.6.1 QCPScatterStyle() [1/7]	381
6.67.6.2 QCPScatterStyle() [2/7]	381
6.67.6.3 QCPScatterStyle() [3/7]	381
6.67.6.4 QCPScatterStyle() [4/7]	381
6.67.6.5 QCPScatterStyle() [5/7]	382
6.67.6.6 QCPScatterStyle() [6/7]	382
6.67.6.7 QCPScatterStyle() [7/7]	382
6.67.7 Member Function Documentation	382
6.67.7.1 applyTo()	383
6.67.7.2 drawShape() [1/2]	383
6.67.7.3 drawShape() [2/2]	383
6.67.7.4 isNone()	383
6.67.7.5 isPenDefined()	384
6.67.7.6 setBrush()	384
6.67.7.7 setCustomPath()	384
6.67.7.8 setFromOther()	384
6.67.7.9 setPen()	385
6.67.7.10 setPixmap()	385
6.67.7.11 setShape()	385
6.67.7.12 setSize()	385
6.67.7.13 undefinePen()	386
6.68 QCPSelectionDecorator Class Reference	386

6.68.1 Detailed Description	387
6.68.2 Constructor & Destructor Documentation	387
6.68.2.1 QCPSelectionDecorator()	387
6.68.3 Member Function Documentation	387
6.68.3.1 applyBrush()	388
6.68.3.2 applyPen()	388
6.68.3.3 copyFrom()	388
6.68.3.4 drawDecoration()	388
6.68.3.5 getFinalScatterStyle()	389
6.68.3.6 setBrush()	389
6.68.3.7 setPen()	389
6.68.3.8 setScatterStyle()	389
6.68.3.9 setUsedScatterProperties()	389
6.69 QCPSelectionDecoratorBracket Class Reference	390
6.69.1 Detailed Description	391
6.69.2 Member Enumeration Documentation	391
6.69.2.1 BracketStyle	391
6.69.3 Constructor & Destructor Documentation	391
6.69.3.1 QCPSelectionDecoratorBracket()	391
6.69.4 Member Function Documentation	391
6.69.4.1 drawBracket()	392
6.69.4.2 drawDecoration()	392
6.69.4.3 setBracketBrush()	392
6.69.4.4 setBracketHeight()	392
6.69.4.5 setBracketPen()	392
6.69.4.6 setBracketStyle()	393
6.69.4.7 setBracketWidth()	393
6.69.4.8 setTangentAverage()	393
6.69.4.9 setTangentToData()	393
6.70 QCPSelectionRect Class Reference	394

6.70.1 Detailed Description	395
6.70.2 Constructor & Destructor Documentation	395
6.70.2.1 QCPSelectionRect()	395
6.70.3 Member Function Documentation	395
6.70.3.1 accepted	395
6.70.3.2 cancel()	396
6.70.3.3 canceled	396
6.70.3.4 changed	396
6.70.3.5 isActive()	396
6.70.3.6 range()	397
6.70.3.7 setBrush()	397
6.70.3.8 setPen()	397
6.70.3.9 started	397
6.71 QCPStatisticalBox Class Reference	398
6.71.1 Detailed Description	399
6.71.2 Changing the appearance	399
6.71.3 Usage	400
6.71.4 Constructor & Destructor Documentation	400
6.71.4.1 QCPStatisticalBox()	400
6.71.5 Member Function Documentation	400
6.71.5.1 addData() [1/2]	400
6.71.5.2 addData() [2/2]	401
6.71.5.3 data()	401
6.71.5.4 drawStatisticalBox()	401
6.71.5.5 getKeyRange()	402
6.71.5.6 getValueRange()	402
6.71.5.7 selectTest()	403
6.71.5.8 selectTestRect()	403
6.71.5.9 setData() [1/2]	403
6.71.5.10 setData() [2/2]	404

6.71.5.11 setMedianPen()	404
6.71.5.12 setOutlierStyle()	404
6.71.5.13 setWhiskerAntialiased()	404
6.71.5.14 setWhiskerBarPen()	405
6.71.5.15 setWhiskerPen()	405
6.71.5.16 setWhiskerWidth()	405
6.71.5.17 setWidth()	406
6.72 QCPStatisticalBoxData Class Reference	406
6.72.1 Detailed Description	407
6.72.2 Constructor & Destructor Documentation	407
6.72.2.1 QCPStatisticalBoxData() [1/2]	407
6.72.2.2 QCPStatisticalBoxData() [2/2]	408
6.72.3 Member Function Documentation	408
6.72.3.1 fromSortKey()	408
6.72.3.2 mainKey()	408
6.72.3.3 mainValue()	408
6.72.3.4 sortKey()	409
6.72.3.5 sortKeysMainKey()	409
6.72.3.6 valueRange()	409
6.73 QCPTextElement Class Reference	409
6.73.1 Detailed Description	411
6.73.2 Constructor & Destructor Documentation	411
6.73.2.1 QCPTextElement() [1/5]	411
6.73.2.2 QCPTextElement() [2/5]	411
6.73.2.3 QCPTextElement() [3/5]	412
6.73.2.4 QCPTextElement() [4/5]	412
6.73.2.5 QCPTextElement() [5/5]	412
6.73.3 Member Function Documentation	412
6.73.3.1 clicked	413
6.73.3.2 doubleClicked	413

6.73.3.3	maximumSizeHint()	413
6.73.3.4	minimumSizeHint()	413
6.73.3.5	mouseDoubleClickEvent()	414
6.73.3.6	mousePressEvent()	414
6.73.3.7	mouseReleaseEvent()	414
6.73.3.8	selectionChanged	414
6.73.3.9	selectTest()	415
6.73.3.10	setFont()	415
6.73.3.11	setSelectable()	415
6.73.3.12	setSelected()	415
6.73.3.13	setSelectedFont()	416
6.73.3.14	setSelectedTextColor()	416
6.73.3.15	setText()	416
6.73.3.16	setTextColor()	416
6.73.3.17	setTextFlags()	417
6.74	QCPVector2D Class Reference	417
6.74.1	Detailed Description	418
6.74.2	Constructor & Destructor Documentation	419
6.74.2.1	QCPVector2D() [1/4]	419
6.74.2.2	QCPVector2D() [2/4]	419
6.74.2.3	QCPVector2D() [3/4]	419
6.74.2.4	QCPVector2D() [4/4]	419
6.74.3	Member Function Documentation	419
6.74.3.1	distanceSquaredToLine() [1/2]	419
6.74.3.2	distanceSquaredToLine() [2/2]	420
6.74.3.3	distanceToStraightLine()	420
6.74.3.4	dot()	420
6.74.3.5	isNull()	420
6.74.3.6	length()	421
6.74.3.7	lengthSquared()	421

6.74.3.8	normalize()	421
6.74.3.9	normalized()	421
6.74.3.10	operator*=()	422
6.74.3.11	operator+=()	422
6.74.3.12	operator-=()	422
6.74.3.13	operator/=()	422
6.74.3.14	perpendicular()	422
6.74.3.15	setX()	422
6.74.3.16	setY()	423
6.74.3.17	toPoint()	423
6.74.3.18	toPointF()	423
6.74.4	Friends And Related Function Documentation	423
6.74.4.1	operator<<()	423
6.75	QCustomPlot Class Reference	424
6.75.1	Detailed Description	428
6.75.2	Member Enumeration Documentation	428
6.75.2.1	LayerInsertMode	428
6.75.2.2	RefreshPriority	428
6.75.3	Constructor & Destructor Documentation	429
6.75.3.1	QCustomPlot()	429
6.75.4	Member Function Documentation	429
6.75.4.1	addGraph()	429
6.75.4.2	addLayer()	430
6.75.4.3	afterReplot	430
6.75.4.4	axisClick	430
6.75.4.5	axisDoubleClick	431
6.75.4.6	axisRect()	431
6.75.4.7	axisRectAt()	431
6.75.4.8	axisRectCount()	432
6.75.4.9	axisRects()	432

6.75.4.10 beforeReplot	432
6.75.4.11 clearGraphs()	432
6.75.4.12 clearItems()	433
6.75.4.13 clearPlottables()	433
6.75.4.14 currentLayer()	433
6.75.4.15 deselectAll()	433
6.75.4.16 graph() [1/2]	434
6.75.4.17 graph() [2/2]	434
6.75.4.18 graphCount()	434
6.75.4.19 hasInvalidatedPaintBuffers()	434
6.75.4.20 hasItem()	435
6.75.4.21 hasPlottable()	435
6.75.4.22 item() [1/2]	435
6.75.4.23 item() [2/2]	435
6.75.4.24 itemAt()	436
6.75.4.25 itemClick	436
6.75.4.26 itemCount()	436
6.75.4.27 itemDoubleClick	437
6.75.4.28 layer() [1/2]	437
6.75.4.29 layer() [2/2]	437
6.75.4.30 layerCount()	438
6.75.4.31 layoutElementAt()	438
6.75.4.32 legendClick	438
6.75.4.33 legendDoubleClick	439
6.75.4.34 mouseDoubleClick	439
6.75.4.35 mouseMove	439
6.75.4.36 mousePress	439
6.75.4.37 mouseRelease	440
6.75.4.38 mouseWheel	440
6.75.4.39 moveLayer()	440

6.75.4.40 plotLayout()	440
6.75.4.41 plottable() [1/2]	441
6.75.4.42 plottable() [2/2]	441
6.75.4.43 plottableAt()	441
6.75.4.44 plottableClick	442
6.75.4.45 plottableCount()	442
6.75.4.46 plottableDoubleClick	442
6.75.4.47 removeGraph() [1/2]	443
6.75.4.48 removeGraph() [2/2]	443
6.75.4.49 removeItem() [1/2]	443
6.75.4.50 removeItem() [2/2]	443
6.75.4.51 removeLayer()	444
6.75.4.52 removePlottable() [1/2]	444
6.75.4.53 removePlottable() [2/2]	444
6.75.4.54 replot()	445
6.75.4.55 rescaleAxes()	445
6.75.4.56 saveBmp()	445
6.75.4.57 saveJpg()	446
6.75.4.58 savePdf()	447
6.75.4.59 savePng()	448
6.75.4.60 saveRastered()	449
6.75.4.61 selectedAxes()	449
6.75.4.62 selectedGraphs()	450
6.75.4.63 selectedItems()	450
6.75.4.64 selectedLegends()	450
6.75.4.65 selectedPlottables()	450
6.75.4.66 selectionChangedByUser	451
6.75.4.67 selectionRect()	451
6.75.4.68 setAntialiasedElement()	451
6.75.4.69 setAntialiasedElements()	452

6.75.4.70	setAutoAddPlottableToLegend()	452
6.75.4.71	setBackground() [1/3]	452
6.75.4.72	setBackground() [2/3]	453
6.75.4.73	setBackground() [3/3]	453
6.75.4.74	setBackgroundScaled()	453
6.75.4.75	setBackgroundScaledMode()	454
6.75.4.76	setBufferDevicePixelRatio()	454
6.75.4.77	setCurrentLayer() [1/2]	454
6.75.4.78	setCurrentLayer() [2/2]	455
6.75.4.79	setInteraction()	455
6.75.4.80	setInteractions()	455
6.75.4.81	setMultiSelectModifier()	456
6.75.4.82	setNoAntialiasingOnDrag()	457
6.75.4.83	setNotAntialiasedElement()	457
6.75.4.84	setNotAntialiasedElements()	457
6.75.4.85	setOpenGL()	458
6.75.4.86	setPlottingHint()	458
6.75.4.87	setPlottingHints()	459
6.75.4.88	setSelectionRect()	459
6.75.4.89	setSelectionRectMode()	459
6.75.4.90	setSelectionTolerance()	460
6.75.4.91	setViewport()	460
6.75.4.92	toPainter()	460
6.75.4.93	toPixmap()	461
6.75.5	Member Data Documentation	461
6.75.5.1	legend	461
6.75.5.2	xAxis	461
6.75.5.3	xAxis2	462
6.75.5.4	yAxis	462
6.75.5.5	yAxis2	462
6.76	Rohr Class Reference	463
6.76.1	Detailed Description	463
6.76.2	Member Function Documentation	463
6.76.2.1	get_kA() [1/2]	464
6.76.2.2	get_kA() [2/2]	464
6.76.2.3	get_querschnitt()	464
6.76.2.4	get_radius()	465
6.77	Rohrstroemung Class Reference	465
6.77.1	Detailed Description	466
6.77.2	Member Function Documentation	466
6.77.2.1	get_lambda()	466
6.78	QCPAxisPainterPrivate::TickLabelData Struct Reference	466

Chapter 1

Todo List

Member `Fluid::get_massenstrom ()`

: prüfen ob gesetzt

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

QCP	13
-------------------------------	--------------------

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

APITest	19
QCPAxisPainterPrivate::CachedLabel	20
Fluid	20
QCPAbstractPaintBuffer	36
QCPPaintBufferPixmap	355
QCPAxisPainterPrivate	84
QCPAxisTicker	103
QCPAxisTickerDateTime	107
QCPAxisTickerFixed	111
QCPAxisTickerLog	114
QCPAxisTickerPi	115
QCPAxisTickerText	118
QCPAxisTickerTime	124
QCPBarsData	136
QCPColorGradient	144
QCPColorMapData	159
QCPCurveData	184
QCPDataContainer< DataType >	187
QCPDataRange	197
QCPDataSelection	202
QCPErrorBarsData	221
QCPFinancialData	233
QCPGraphData	245
QCPItemAnchor	251
QCPItemPosition	272
QCPLineEnding	350
QCPPlottableInterface1D	363
QCPAbstractPlottable1D< DataType >	53
QCPAbstractPlottable1D< QCPBarsData >	53
QCPBars	127
QCPAbstractPlottable1D< QCPCurveData >	53
QCPCurve	176
QCPAbstractPlottable1D< QCPFinancialData >	53
QCPFinancial	223

QCPAbstractPlottable1D< QCPGraphData >	53
QCPGraph	236
QCPAbstractPlottable1D< QCPStatisticalBoxData >	53
QCPStatisticalBox	398
QCPErrorsBars	210
QCPRange	369
QCPScatterStyle	377
QCPSelectionDecorator	386
QCPSelectionDecoratorBracket	390
QCPStatisticalBoxData	406
QCPVector2D	417
QDialog	
DatenEingabe	20
Plotter	22
QMainWindow	
MainWindow	22
QObject	
QCPBarsGroup	138
QCPLayer	297
QCPLayerable	302
QCPAbstractItem	23
QCPItemBracket	253
QCPItemCurve	257
QCPItemEllipse	261
QCPItemLine	265
QCPItemPixmap	268
QCPItemRect	279
QCPItemStraightLine	283
QCPItemText	285
QCPItemTracer	291
QCPAbstractPlottable	39
QCPAbstractPlottable1D< DataType >	53
QCPAbstractPlottable1D< QCPBarsData >	53
QCPAbstractPlottable1D< QCPCurveData >	53
QCPAbstractPlottable1D< QCPFinancialData >	53
QCPAbstractPlottable1D< QCPGraphData >	53
QCPAbstractPlottable1D< QCPStatisticalBoxData >	53
QCPColorMap	150
QCPErrorsBars	210
QCPAxis	60
QCPGrid	248
QCPLayoutElement	314
QCPAbstractLegendItem	31
QCPPlottableLegendItem	367
QCPAxisRect	85
QCPColorScaleAxisRectPrivate	175
QCPColorScale	167
QCPLayout	309
QCPLayoutGrid	322
QCPLegend	339
QCPLayoutInset	333
QCPTextElement	409
QCPSelectionRect	394
QCPMarginGroup	353
QPainter	
QCPPainter	358
QWidget	

QCustomPlot	424
Rohr	463
Rohrstroemung	465
QCPAxisPainterPrivate::TickLabelData	466

Chapter 4

Class Index

4.1 Class List

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

APITest	19
QCPAxisPainterPrivate::CachedLabel	20
DatenEingabe	20
Fluid	
Stellt ein Fluid zur Verfügung	20
MainWindow	22
Plotter	22
QCPAbstractItem	
The abstract base class for all items in a plot	23
QCPAbstractLegendItem	
The abstract base class for all entries in a QCPLegend	31
QCPAbstractPaintBuffer	
The abstract base class for paint buffers, which define the rendering backend	36
QCPAbstractPlottable	
The abstract base class for all data representing objects in a plot	39
QCPAbstractPlottable1D< DataType >	
A template base class for plottables with one-dimensional data	53
QCPAxis	
Manages a single axis inside a QCustomPlot	60
QCPAxisPainterPrivate	84
QCPAxisRect	
Holds multiple axes and arranges them in a rectangular shape	85
QCPAxisTicker	
The base class tick generator used by QCPAxis to create tick positions and tick labels	103
QCPAxisTickerDateTime	
Specialized axis ticker for calendar dates and times as axis ticks	107
QCPAxisTickerFixed	
Specialized axis ticker with a fixed tick step	111
QCPAxisTickerLog	
Specialized axis ticker suited for logarithmic axes	114
QCPAxisTickerPi	
Specialized axis ticker to display ticks in units of an arbitrary constant, for example pi	115
QCPAxisTickerText	
Specialized axis ticker which allows arbitrary labels at specified coordinates	118
QCPAxisTickerTime	
Specialized axis ticker for time spans in units of milliseconds to days	124

QCPBars	
A plottable representing a bar chart in a plot	127
QCPBarsData	
Holds the data of one single data point (one bar) for QCPBars	136
QCPBarsGroup	
Groups multiple QCPBars together so they appear side by side	138
QCPColorGradient	
Defines a color gradient for use with e.g. QCPColorMap	144
QCPColorMap	
A plottable representing a two-dimensional color map in a plot	150
QCPColorMapData	
Holds the two-dimensional data of a QCPColorMap plottable	159
QCPColorScale	
A color scale for use with color coding data such as QCPColorMap	167
QCPColorScaleAxisRectPrivate	175
QCPCurve	
A plottable representing a parametric curve in a plot	176
QCPCurveData	
Holds the data of one single data point for QCPCurve	184
QCPDataContainer< DataType >	
The generic data container for one-dimensional plottables	187
QCPDataRange	
Describes a data range given by begin and end index	197
QCPDataSelection	
Describes a data set by holding multiple QCPDataRange instances	202
QCPErrorBars	
A plottable that adds a set of error bars to other plottables	210
QCPErrorBarsData	
Holds the data of one single error bar for QCPErrorBars	221
QCPFinancial	
A plottable representing a financial stock chart	223
QCPFinancialData	
Holds the data of one single data point for QCPFinancial	233
QCPGraph	
A plottable representing a graph in a plot	236
QCPGraphData	
Holds the data of one single data point for QCPGraph	245
QCPGrid	
Responsible for drawing the grid of a QCPAxis	248
QCPItemAnchor	
An anchor of an item to which positions can be attached to	251
QCPItemBracket	
A bracket for referencing/highlighting certain parts in the plot	253
QCPItemCurve	
A curved line from one point to another	257
QCPItemEllipse	
An ellipse	261
QCPItemLine	
A line from one point to another	265
QCPItemPixmap	
An arbitrary pixmap	268
QCPItemPosition	
Manages the position of an item	272
QCPItemRect	
A rectangle	279
QCPItemStraightLine	
A straight line that spans infinitely in both directions	283

QCItemText	
A text label	285
QCItemTracer	
Item that sticks to QCPGraph data points	291
QCPLayer	
A layer that may contain objects, to control the rendering order	297
QCPLayerable	
Base class for all drawable objects	302
QCPLayout	
The abstract base class for layouts	309
QCPLayoutElement	
The abstract base class for all objects that form the layout system	314
QCPLayoutGrid	
A layout that arranges child elements in a grid	322
QCPLayoutInset	
A layout that places child elements aligned to the border or arbitrarily positioned	333
QCPLegend	
Manages a legend inside a QCustomPlot	339
QCPLineEnding	
Handles the different ending decorations for line-like items	350
QCPMarginGroup	
A margin group allows synchronization of margin sides if working with multiple layout elements	353
QCPPaintBufferPixmap	
A paint buffer based on QPixmap, using software raster rendering	355
QCPPainter	
QPainter subclass used internally	358
QCPPlottableInterface1D	
Defines an abstract interface for one-dimensional plottables	363
QCPPlottableLegendItem	
A legend item representing a plottable with an icon and the plottable name	367
QCPRange	
Represents the range an axis is encompassing	369
QCPScatterStyle	
Represents the visual appearance of scatter points	377
QCPSelectionDecorator	
Controls how a plottable's data selection is drawn	386
QCPSelectionDecoratorBracket	
A selection decorator which draws brackets around each selected data segment	390
QCPSelectionRect	
Provides rect/rubber-band data selection and range zoom interaction	394
QCPStatisticalBox	
A plottable representing a single statistical box in a plot	398
QCPStatisticalBoxData	
Holds the data of one single data point for QCPStatisticalBox	406
QCPTextElement	
A layout element displaying a text	409
QCPVector2D	
Represents two doubles as a mathematical 2D vector	417
QCustomPlot	
The central class of the library. This is the QWidget which displays the plot and interacts with the user	424
Rohr	
Stellt ein Rohrbauteil zur Verfügung	463
Rohrstroemung	
Beschreibt eine Rohrströmung	465
QCPAxisPainterPrivate::TickLabelData	466

Chapter 5

Namespace Documentation

5.1 QCP Namespace Reference

Enumerations

- enum [ResolutionUnit](#) { [ruDotsPerMeter](#), [ruDotsPerCentimeter](#), [ruDotsPerInch](#) }
- enum [ExportPen](#) { [epNoCosmetic](#), [epAllowCosmetic](#) }
- enum [SignDomain](#) { [sdNegative](#), [sdBoth](#), [sdPositive](#) }
- enum [MarginSide](#) {
 [msLeft](#) = 0x01, [msRight](#) = 0x02, [msTop](#) = 0x04, [msBottom](#) = 0x08,
 [msAll](#) = 0xFF, [msNone](#) = 0x00 }
- enum [AntialiasedElement](#) {
 [aeAxes](#) = 0x0001, [aeGrid](#) = 0x0002, [aeSubGrid](#) = 0x0004, [aeLegend](#) = 0x0008,
 [aeLegendItems](#) = 0x0010, [aePlottables](#) = 0x0020, [aeItems](#) = 0x0040, [aeScatters](#) = 0x0080,
 [aeFills](#) = 0x0100, [aeZeroLine](#) = 0x0200, [aeOther](#) = 0x8000, [aeAll](#) = 0xFFFF,
 [aeNone](#) = 0x0000 }
- enum [PlottingHint](#) { [phNone](#) = 0x000, [phFastPolylines](#) = 0x001, [phImmediateRefresh](#) = 0x002, [phCacheLabels](#) = 0x004 }
- enum [Interaction](#) {
 [iRangeDrag](#) = 0x001, [iRangeZoom](#) = 0x002, [iMultiSelect](#) = 0x004, [iSelectPlottables](#) = 0x008,
 [iSelectAxes](#) = 0x010, [iSelectLegend](#) = 0x020, [iSelectItems](#) = 0x040, [iSelectOther](#) = 0x080 }
- enum [SelectionRectMode](#) { [srmNone](#), [srmZoom](#), [srmSelect](#), [srmCustom](#) }
- enum [SelectionType](#) {
 [stNone](#), [stWhole](#), [stSingleData](#), [stDataRange](#),
 [stMultipleDataRanges](#) }

Functions

- bool **isInvalidData** (double value)
- bool **isInvalidData** (double value1, double value2)
- void **setMarginValue** (QMargins &margins, [QCP::MarginSide](#) side, int value)
- int **getMarginValue** (const QMargins &margins, [QCP::MarginSide](#) side)

Variables

- const QMetaObject **staticMetaObject**

5.1.1 Detailed Description

The [QCP](#) Namespace contains general enums, QFlags and functions used throughout the [QCustomPlot](#) library.

It provides QMetaObject-based reflection of its enums and flags via *QCP::staticMetaObject*.

5.1.2 Enumeration Type Documentation

5.1.2.1 AntialiasedElement

enum [QCP::AntialiasedElement](#)

Defines what objects of a plot can be forcibly drawn antialiased/not antialiased. If an object is neither forcibly drawn antialiased nor forcibly drawn not antialiased, it is up to the respective element how it is drawn. Typically it provides a *setAntialiased* function for this.

`AntialiasedElements` is a flag of or-combined elements of this enum type.

See also

[QCustomPlot::setAntialiasedElements](#), [QCustomPlot::setNotAntialiasedElements](#)

Enumerator

<code>aeAxes</code>	<code>0x0001</code> Axis base line and tick marks
<code>aeGrid</code>	<code>0x0002</code> Grid lines
<code>aeSubGrid</code>	<code>0x0004</code> Sub grid lines
<code>aeLegend</code>	<code>0x0008</code> Legend box
<code>aeLegendItems</code>	<code>0x0010</code> Legend items
<code>aePlottables</code>	<code>0x0020</code> Main lines of plottables
<code>aeItems</code>	<code>0x0040</code> Main lines of items
<code>aeScatters</code>	<code>0x0080</code> Scatter symbols of plottables (excluding scatter symbols of type <code>ssPixmap</code>)
<code>aeFills</code>	<code>0x0100</code> Borders of fills (e.g. under or between graphs)
<code>aeZeroLine</code>	<code>0x0200</code> Zero-lines, see QCPGrid::setZeroLinePen
<code>aeOther</code>	<code>0x8000</code> Other elements that don't fit into any of the existing categories
<code>aeAll</code>	<code>0xFFFF</code> All elements
<code>aeNone</code>	<code>0x0000</code> No elements

5.1.2.2 ExportPen

enum [QCP::ExportPen](#)

Defines how cosmetic pens (pens with numerical width 0) are handled during export.

See also

[QCustomPlot::savePdf](#)

Enumerator

epNoCosmetic	Cosmetic pens are converted to pens with pixel width 1 when exporting.
epAllowCosmetic	Cosmetic pens are exported normally (e.g. in PDF exports, cosmetic pens always appear as 1 pixel on screen, independent of viewer zoom level)

5.1.2.3 Interaction

enum [QCP::Interaction](#)

Defines the mouse interactions possible with [QCustomPlot](#).

`Interactions` is a flag of or-combined elements of this enum type.

See also

[QCustomPlot::setInteractions](#)

Enumerator

iRangeDrag	0x001 Axis ranges are draggable (see QCPAxisRect::setRangeDrag , QCPAxisRect::setRangeDragAxes)
iRangeZoom	0x002 Axis ranges are zoomable with the mouse wheel (see QCPAxisRect::setRangeZoom , QCPAxisRect::setRangeZoomAxes)
iMultiSelect	0x004 The user can select multiple objects by holding the modifier set by QCustomPlot::setMultiSelectModifier while clicking
iSelectPlottables	0x008 Plottables are selectable (e.g. graphs, curves, bars,... see QCPAbstractPlottable)
iSelectAxes	0x010 Axes are selectable (or parts of them, see QCPAxis::setSelectableParts)
iSelectLegend	0x020 Legends are selectable (or their child items, see QCPLegend::setSelectableParts)
iSelectItems	0x040 Items are selectable (Rectangles, Arrows, Textitems, etc. see QCPAbstractItem)
iSelectOther	0x080 All other objects are selectable (e.g. your own derived layerables, other layout elements,...)

5.1.2.4 MarginSide

enum [QCP::MarginSide](#)

Defines the sides of a rectangular entity to which margins can be applied.

See also

[QCPLayoutElement::setAutoMargins](#), [QCPAxisRect::setAutoMargins](#)

Enumerator

msLeft	0x01 left margin
msRight	0x02 right margin
msTop	0x04 top margin
msBottom	0x08 bottom margin
msAll	0xFF all margins
msNone	0x00 no margin

5.1.2.5 PlottingHint

enum [QCP::PlottingHint](#)

Defines plotting hints that control various aspects of the quality and speed of plotting.

See also

[QCustomPlot::setPlottingHints](#)

Enumerator

phNone	0x000 No hints are set
phFastPolylines	0x001 Graph/Curve lines are drawn with a faster method. This reduces the quality especially of the line segment joins, thus is most effective for pen sizes larger than 1. It is only used for solid line pens.
phImmediateRefresh	0x002 causes an immediate repaint() instead of a soft update() when QCustomPlot::replot() is called with parameter QCustomPlot::rpRefreshHint . This is set by default to prevent the plot from freezing on fast consecutive replots (e.g. user drags ranges with mouse).
phCacheLabels	0x004 axis (tick) labels will be cached as pixmaps, increasing replot performance.

5.1.2.6 ResolutionUnit

enum [QCP::ResolutionUnit](#)

Defines the different units in which the image resolution can be specified in the export functions.

See also

[QCustomPlot::savePng](#), [QCustomPlot::saveJpg](#), [QCustomPlot::saveBmp](#), [QCustomPlot::saveRastered](#)

Enumerator

ruDotsPerMeter	Resolution is given in dots per meter (dpm)
ruDotsPerCentimeter	Resolution is given in dots per centimeter (dpcm)
ruDotsPerInch	Resolution is given in dots per inch (DPI/PPI)

5.1.2.7 SelectionRectMode

enum [QCP::SelectionRectMode](#)

Defines the behaviour of the selection rect.

See also

[QCustomPlot::setSelectionRectMode](#), [QCustomPlot::selectionRect](#), [QCPSelectionRect](#)

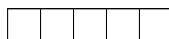
Enumerator

srnNone	The selection rect is disabled, and all mouse events are forwarded to the underlying objects, e.g. for axis range dragging.
srnZoom	When dragging the mouse, a selection rect becomes active. Upon releasing, the axes that are currently set as range zoom axes (QCPAxisRect::setRangeZoomAxes) will have their ranges zoomed accordingly.
srnSelect	When dragging the mouse, a selection rect becomes active. Upon releasing, plottable data points that were within the selection rect are selected, if the plottable's selectability setting permits. (See data selection mechanism for details.)
srnCustom	When dragging the mouse, a selection rect becomes active. It is the programmer's responsibility to connect according slots to the selection rect's signals (e.g. QCPSelectionRect::accepted) in order to process the user interaction.

5.1.2.8 SelectionType

enum [QCP::SelectionType](#)

Defines the different ways a plottable can be selected. These images show the effect of the different selection types, when the indicated selection rect was dragged:



See also

[QCPAbstractPlottable::setSelectable](#), [QCPDataSelection::enforceType](#)

Enumerator

stNone	The plottable is not selectable.
stWhole	Selection behaves like stMultipleDataRanges , but if there are any data points selected, the entire plottable is drawn as selected.

Enumerator

stSingleData	One individual data point can be selected at a time.
stDataRange	Multiple contiguous data points (a data range) can be selected.
stMultipleDataRanges	Any combination of data points/ranges can be selected.

5.1.2.9 SignDomain

enum [QCP::SignDomain](#)

Represents negative and positive sign domain, e.g. for passing to [QCPAbstractPlottable::getKeyRange](#) and [QCPAbstractPlottable::getValueRange](#).

This is primarily needed when working with logarithmic axis scales, since only one of the sign domains can be visible at a time.

Enumerator

sdNegative	The negative sign domain, i.e. numbers smaller than zero.
sdBoth	Both sign domains, including zero, i.e. all numbers.
sdPositive	The positive sign domain, i.e. numbers greater than zero.

Chapter 6

Class Documentation

6.1 APITest Class Reference

Static Public Member Functions

- static void [printTestResult](#) (const bool result, const char *testName, const char *autor, const char *testFall="", const char *dateiName="")
Setzt die Daten für den jeweiligen Test.
- static void [printTestStartHeader](#) ()
Nicht relevant: Gibt den Header am Anfang der Tests aus.
- static void [printTestEndFooter](#) ()
Gibt die Zusammenfassung am Ende der Tests aus.

6.1.1 Member Function Documentation

6.1.1.1 printTestResult()

```
static void APITest::printTestResult (  
    const bool result,  
    const char * testName,  
    const char * autor,  
    const char * testFall = "",  
    const char * dateiName = "" ) [inline], [static]
```

Setzt die Daten für den jeweiligen Test.

Parameters

<i>result</i>	Ergebnis des Tests true=Erfolgreich false=Fehlgeschlagen
<i>testName</i>	Name des Tests, damit dieser anschließend identifiziert werden kann
<i>autor</i>	Name des Autors, der den Test geschrieben hat
<i>testFall</i>	Was wird getestet. Ein Stichpunkt bzw. ein Satz genügt
<i>dateiName</i>	In welcher Datei befindet sich der Test

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

- [/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/tests/test.h](#)

6.2 QCPAxisPainterPrivate::CachedLabel Struct Reference

Public Attributes

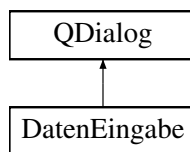
- **QPointF** **offset**
- **QPixmap** **pixmap**

The documentation for this struct was generated from the following file:

- [/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h](#)

6.3 DatenEingabe Class Reference

Inheritance diagram for DatenEingabe:



Public Member Functions

- **DatenEingabe** (QWidget *parent=0)

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

- [/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/dateneingabe.h](#)
- [/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/dateneingabe.cpp](#)

6.4 Fluid Class Reference

Stellt ein **Fluid** zur Verfügung.

```
#include <fluid.h>
```


Public Member Functions

- **Fluid** (double dichte, double nue, double cp)
- double [get_massenstrom](#) ()
Massenstrom wird zurückgegeben.
- double [get_dichte](#) ()
Dichte wird zurückgegeben.
- double [get_nue](#) ()
Viskosität wird zurückgegeben.
- double [get_cp](#) ()
Wärmekapazität wird zurückgegeben.
- double [get_cp_strom](#) ()
Wärmekapazitätsstrom wird zurückgegeben.
- double [get_t_ein](#) ()
Gibt Eintrittstemperatur des Fluids zurück.
- void [set_massenstrom](#) (double massenstrom)
Setzt den Massenstrom des Fluids.
- void [set_t_ein](#) (double t_ein)
Setzt die Eintrittstemperatur.

6.4.1 Detailed Description

Stellt ein [Fluid](#) zur Verfügung.

6.4.2 Member Function Documentation

6.4.2.1 [get_cp_strom\(\)](#)

```
double Fluid::get_cp_strom ( )
```

Wärmekapazitätsstrom wird zurückgegeben.

Berechnet den Wärmekapazitätsstrom.

Warning

Ist es wirklich nötig die Variablen der Funktion zu übergeben oder ist auch ein anderer Ansatz über entsprechende get-Funktionen möglich?

See also

[Rohr::set_kA](#)

6.4.2.2 get_massenstrom()

```
double Fluid::get_massenstrom ( )
```

Massenstrom wird zurückgegeben.

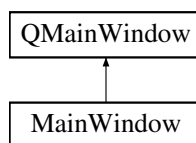
Todo : prüfen ob gesetzt

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

- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/fluid.↔
h
- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/fluid.↔
cpp

6.5 MainWindow Class Reference

Inheritance diagram for MainWindow:



Public Member Functions

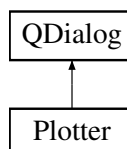
- **MainWindow** (QWidget *parent=0)

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

- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/mainwindow.↔
h
- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/mainwindow.↔
cpp

6.6 Plotter Class Reference

Inheritance diagram for Plotter:



Public Member Functions

- **Plotter** (QWidget *parent=0)

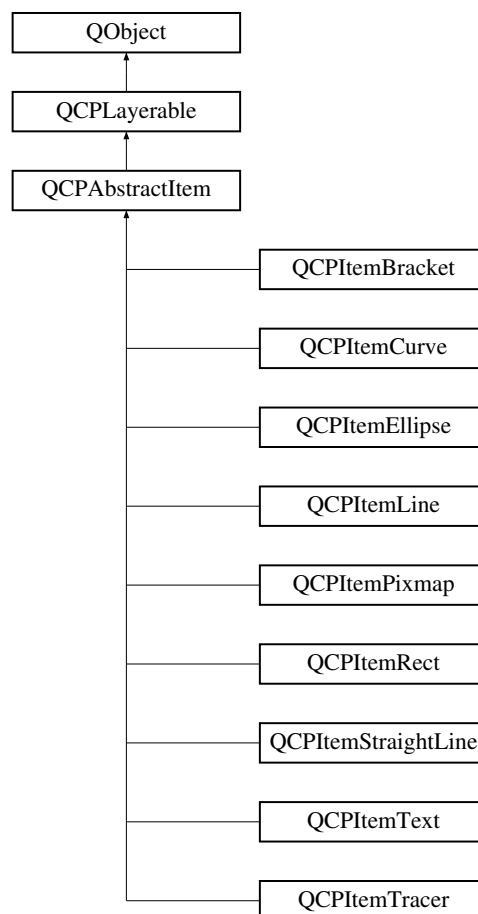
The documentation for this class was generated from the following files:

- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/plotter.↵
h
- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/plotter.↵
cpp

6.7 QCPAbstractItem Class Reference

The abstract base class for all items in a plot.

Inheritance diagram for QCPAbstractItem:



Signals

- void **selectionChanged** (bool selected)
- void **selectableChanged** (bool selectable)

Public Member Functions

- [QCPAbstractItem](#) ([QCustomPlot](#) *parentPlot)
- bool **clipToAxisRect** () const
- [QCPAxisRect](#) * **clipAxisRect** () const
- bool **selectable** () const
- bool **selected** () const
- void **setClipToAxisRect** (bool clip)
- void **setClipAxisRect** ([QCPAxisRect](#) *rect)
- Q_SLOT void **setSelectable** (bool selectable)
- Q_SLOT void **setSelected** (bool selected)
- virtual double **selectTest** (const QPointF &pos, bool onlySelectable, QVariant *details=0) const Q_DECL_OVERRIDE↔
- QList< [QCPItemPosition](#) * > **positions** () const
- QList< [QCPItemAnchor](#) * > **anchors** () const
- [QCPItemPosition](#) * **position** (const QString &name) const
- [QCPItemAnchor](#) * **anchor** (const QString &name) const
- bool **hasAnchor** (const QString &name) const

Protected Member Functions

- virtual [QCP::Interaction](#) **selectionCategory** () const Q_DECL_OVERRIDE
- virtual QRect **clipRect** () const Q_DECL_OVERRIDE
- virtual void **applyDefaultAntialiasingHint** ([QCPPainter](#) *painter) const Q_DECL_OVERRIDE
- virtual void **draw** ([QCPPainter](#) *painter) Q_DECL_OVERRIDE=0
- virtual void **selectEvent** (QMouseEvent *event, bool additive, const QVariant &details, bool *selection↔StateChanged) Q_DECL_OVERRIDE
- virtual void **deselectEvent** (bool *selectionStateChanged) Q_DECL_OVERRIDE
- virtual QPointF **anchorPixelPosition** (int anchorId) const
- double **rectDistance** (const QRectF &rect, const QPointF &pos, bool filledRect) const
- [QCPItemPosition](#) * **createPosition** (const QString &name)
- [QCPItemAnchor](#) * **createAnchor** (const QString &name, int anchorId)

Protected Attributes

- bool **mClipToAxisRect**
- QPointer< [QCPAxisRect](#) > **mClipAxisRect**
- QList< [QCPItemPosition](#) * > **mPositions**
- QList< [QCPItemAnchor](#) * > **mAnchors**
- bool **mSelectable**
- bool **mSelected**

Friends

- class **QCustomPlot**
- class **QCPItemAnchor**

6.7.1 Detailed Description

The abstract base class for all items in a plot.

In [QCustomPlot](#), items are supplemental graphical elements that are neither plottables ([QCPAbstractPlottable](#)) nor axes ([QCPAxis](#)). While plottables are always tied to two axes and thus plot coordinates, items can also be placed in absolute coordinates independent of any axes. Each specific item has at least one [QCPItemPosition](#) member which controls the positioning. Some items are defined by more than one coordinate and thus have two or more [QCPItemPosition](#) members (For example, [QCPItemRect](#) has *topLeft* and *bottomRight*).

This abstract base class defines a very basic interface like visibility and clipping. Since this class is abstract, it can't be instantiated. Use one of the subclasses or create a subclass yourself to create new items.

The built-in items are:

QCPItemLine	A line defined by a start and an end point. May have different ending styles on each side (e.g. arrows).
QCPItemStraightLine	A straight line defined by a start and a direction point. Unlike QCPItemLine , the straight line is infinitely long and has no endings.
QCPItemCurve	A curve defined by start, end and two intermediate control points. May have different ending styles on each side (e.g. arrows).
QCPItemRect	A rectangle
QCPItemEllipse	An ellipse
QCPItemPixmap	An arbitrary pixmap
QCPItemText	A text label
QCPItemBracket	A bracket which may be used to reference/highlight certain parts in the plot.
QCPItemTracer	An item that can be attached to a QCPGraph and sticks to its data points, given a key coordinate.

6.7.2 Clipping

Items are by default clipped to the main axis rect (they are only visible inside the axis rect). To make an item visible outside that axis rect, disable clipping via [setClipToAxisRect\(false\)](#).

On the other hand if you want the item to be clipped to a different axis rect, specify it via [setClipAxisRect](#). This `clipAxisRect` property of an item is only used for clipping behaviour, and in principle is independent of the coordinate axes the item might be tied to via its position members ([QCPItemPosition::setAxes](#)). However, it is common that the axis rect for clipping also contains the axes used for the item positions.

6.7.3 Using items

First you instantiate the item you want to use and add it to the plot:

by default, the positions of the item are bound to the x- and y-Axis of the plot. So we can just set the plot coordinates where the line should start/end:

If we don't want the line to be positioned in plot coordinates but a different coordinate system, e.g. absolute pixel positions on the [QCustomPlot](#) surface, we need to change the position type like this:

Then we can set the coordinates, this time in pixels:

and make the line visible on the entire [QCustomPlot](#), by disabling clipping to the axis rect:

For more advanced plots, it is even possible to set different types and parent anchors per X/Y coordinate of an item position, using for example [QCItemPosition::setTypeX](#) or [QCItemPosition::setParentAnchorX](#). For details, see the documentation of [QCItemPosition](#).

6.7.4 Creating own items

To create an own item, you implement a subclass of [QCPAbstractItem](#). These are the pure virtual functions, you must implement:

- [selectTest](#)
- [draw](#)

See the documentation of those functions for what they need to do.

6.7.4.1 Allowing the item to be positioned

As mentioned, item positions are represented by [QCItemPosition](#) members. Let's assume the new item shall have only one point as its position (as opposed to two like a rect or multiple like a polygon). You then add a public member of type [QCItemPosition](#) like so:

```
QCItemPosition * const myPosition;
```

the const makes sure the pointer itself can't be modified from the user of your new item (the [QCItemPosition](#) instance it points to, can be modified, of course). The initialization of this pointer is made easy with the `createPosition` function. Just assign the return value of this function to each [QCItemPosition](#) in the constructor of your item. `createPosition` takes a string which is the name of the position, typically this is identical to the variable name. For example, the constructor of `QCItemExample` could look like this:

```
QCItemExample::QCItemExample(QCustomPlot *parentPlot) :
    QCPAbstractItem(parentPlot),
    myPosition(createPosition("myPosition"))
{
    // other constructor code
}
```

6.7.4.2 The draw function

To give your item a visual representation, reimplement the draw function and use the passed [QCPPainter](#) to draw the item. You can retrieve the item position in pixel coordinates from the position member(s) via [QCPItemPosition](#)↔[::pixelPosition](#).

To optimize performance you should calculate a bounding rect first (don't forget to take the pen width into account), check whether it intersects the clipRect, and only draw the item at all if this is the case.

6.7.4.3 The selectTest function

Your implementation of the [selectTest](#) function may use the helpers [QCPVector2D::distanceSquaredToLine](#) and [rectDistance](#). With these, the implementation of the selection test becomes significantly simpler for most items. See the documentation of [selectTest](#) for what the function parameters mean and what the function should return.

6.7.4.4 Providing anchors

Providing anchors ([QCPItemAnchor](#)) starts off like adding a position. First you create a public member, e.g.

```
QCPItemAnchor * const bottom;
```

and create it in the constructor with the [createAnchor](#) function, assigning it a name and an anchor id (an integer enumerating all anchors on the item, you may create an own enum for this). Since anchors can be placed anywhere, relative to the item's position(s), your item needs to provide the position of every anchor with the reimplementation of the [anchorPixelPosition\(int anchorId\)](#) function.

In essence the [QCPItemAnchor](#) is merely an intermediary that itself asks your item for the pixel position when anything attached to the anchor needs to know the coordinates.

6.7.5 Constructor & Destructor Documentation

6.7.5.1 QCPAbstractItem()

```
QCPAbstractItem::QCPAbstractItem (
    QCustomPlot * parentPlot ) [explicit]
```

Base class constructor which initializes base class members.

6.7.6 Member Function Documentation

6.7.6.1 anchor()

```
QCItemAnchor * QCPAbstractItem::anchor (
    const QString & name ) const
```

Returns the [QCItemAnchor](#) with the specified *name*. If this item doesn't have an anchor by that name, returns 0.

This function provides an alternative way to access item anchors. Normally, you access anchors directly by their member pointers (which typically have the same variable name as *name*).

See also

[anchors](#), [position](#)

6.7.6.2 anchors()

```
QList< QCItemAnchor * > QCPAbstractItem::anchors ( ) const [inline]
```

Returns all anchors of the item in a list. Note that since a position ([QCItemPosition](#)) is always also an anchor, the list will also contain the positions of this item.

See also

[positions](#), [anchor](#)

6.7.6.3 hasAnchor()

```
bool QCPAbstractItem::hasAnchor (
    const QString & name ) const
```

Returns whether this item has an anchor with the specified *name*.

Note that you can check for positions with this function, too. This is because every position is also an anchor ([QCItemPosition](#) inherits from [QCItemAnchor](#)).

See also

[anchor](#), [position](#)

6.7.6.4 position()

```
QCPItemPosition * QCPAbstractItem::position (
    const QString & name ) const
```

Returns the [QCPItemPosition](#) with the specified *name*. If this item doesn't have a position by that name, returns 0.

This function provides an alternative way to access item positions. Normally, you access positions directly by their member pointers (which typically have the same variable name as *name*).

See also

[positions](#), [anchor](#)

6.7.6.5 positions()

```
QList< QCPItemPosition * > QCPAbstractItem::positions ( ) const [inline]
```

Returns all positions of the item in a list.

See also

[anchors](#), [position](#)

6.7.6.6 selectionChanged

```
void QCPAbstractItem::selectionChanged (
    bool selected ) [signal]
```

This signal is emitted when the selection state of this item has changed, either by user interaction or by a direct call to [setSelected](#).

6.7.6.7 selectTest()

```
virtual double QCPAbstractItem::selectTest (
    const QPointF & pos,
    bool onlySelectable,
    QVariant * details = 0 ) const [pure virtual]
```

This function is used to decide whether a click hits a layerable object or not.

pos is a point in pixel coordinates on the [QCustomPlot](#) surface. This function returns the shortest pixel distance of this point to the object. If the object is either invisible or the distance couldn't be determined, -1.0 is returned. Further, if *onlySelectable* is true and the object is not selectable, -1.0 is returned, too.

If the object is represented not by single lines but by an area like a [QCItemText](#) or the bars of a [QCPBars](#) plottable, a click inside the area should also be considered a hit. In these cases this function thus returns a constant value greater zero but still below the parent plot's selection tolerance. (typically the selectionTolerance multiplied by 0.99).

Providing a constant value for area objects allows selecting line objects even when they are obscured by such area objects, by clicking close to the lines (i.e. closer than $0.99 \cdot \text{selectionTolerance}$).

The actual setting of the selection state is not done by this function. This is handled by the parent [QCustomPlot](#) when the `mouseReleaseEvent` occurs, and the finally selected object is notified via the `selectEvent/deselectEvent` methods.

details is an optional output parameter. Every layerable subclass may place any information in *details*. This information will be passed to `selectEvent` when the parent [QCustomPlot](#) decides on the basis of this `selectTest` call, that the object was successfully selected. The subsequent call to `selectEvent` will carry the *details*. This is useful for multi-part objects (like [QCPAxis](#)). This way, a possibly complex calculation to decide which part was clicked is only done once in `selectTest`. The result (i.e. the actually clicked part) can then be placed in *details*. So in the subsequent `selectEvent`, the decision which part was selected doesn't have to be done a second time for a single selection operation.

You may pass 0 as *details* to indicate that you are not interested in those selection details.

See also

`selectEvent`, `deselectEvent`, [mousePressEvent](#), [wheelEvent](#), [QCustomPlot::setInteractions](#)

Reimplemented from [QCPLayerable](#).

Implemented in [QCItemBracket](#), [QCItemTracer](#), [QCItemPixmap](#), [QCItemEllipse](#), [QCItemText](#), [QCItemRect](#), [QCItemCurve](#), [QCItemLine](#), and [QCItemStraightLine](#).

6.7.6.8 `setClipAxisRect()`

```
void QCPAbstractItem::setClipAxisRect (
    QCPAxisRect * rect )
```

Sets the clip axis rect. It defines the rect that will be used to clip the item when [setClipToAxisRect](#) is set to true.

See also

[setClipToAxisRect](#)

6.7.6.9 `setClipToAxisRect()`

```
void QCPAbstractItem::setClipToAxisRect (
    bool clip )
```

Sets whether the item shall be clipped to an axis rect or whether it shall be visible on the entire [QCustomPlot](#). The axis rect can be set with [setClipAxisRect](#).

See also

[setClipAxisRect](#)

6.7.6.10 `setSelectable()`

```
void QCPAbstractItem::setSelectable (
    bool selectable )
```

Sets whether the user can (de-)select this item by clicking on the [QCustomPlot](#) surface. (When [QCustomPlot::setInteractions](#) contains [QCustomPlot::iSelectItems](#).)

However, even when *selectable* was set to false, it is possible to set the selection manually, by calling [setSelected](#).

See also

[QCustomPlot::setInteractions](#), [setSelected](#)

6.7.6.11 `setSelected()`

```
void QCPAbstractItem::setSelected (
    bool selected )
```

Sets whether this item is selected or not. When selected, it might use a different visual appearance (e.g. pen and brush), this depends on the specific item though.

The entire selection mechanism for items is handled automatically when [QCustomPlot::setInteractions](#) contains [QCustomPlot::iSelectItems](#). You only need to call this function when you wish to change the selection state manually.

This function can change the selection state even when [setSelectable](#) was set to false.

emits the [selectionChanged](#) signal when *selected* is different from the previous selection state.

See also

[setSelectable](#), [selectTest](#)

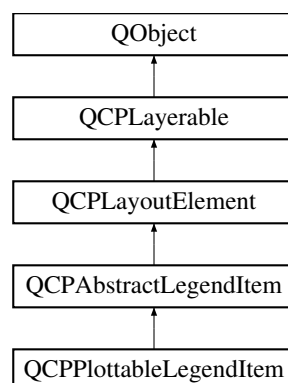
The documentation for this class was generated from the following files:

- `/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h`
- `/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp`

6.8 QCPAbstractLegendItem Class Reference

The abstract base class for all entries in a [QCPLegend](#).

Inheritance diagram for QCPAbstractLegendItem:



Signals

- void [selectionChanged](#) (bool selected)
- void **selectableChanged** (bool selectable)

Public Member Functions

- [QCPLegendItem](#) ([QCPLegend](#) *parent)
- [QCPLegend](#) * **parentLegend** () const
- QFont **font** () const
- QColor **textColor** () const
- QFont **selectedFont** () const
- QColor **selectedTextColor** () const
- bool **selectable** () const
- bool **selected** () const
- void [setFont](#) (const QFont &font)
- void [setTextColor](#) (const QColor &color)
- void [setSelectedFont](#) (const QFont &font)
- void [setSelectedTextColor](#) (const QColor &color)
- Q_SLOT void [setSelectable](#) (bool selectable)
- Q_SLOT void [setSelected](#) (bool selected)
- virtual double [selectTest](#) (const QPointF &pos, bool onlySelectable, QVariant *details=0) const Q_DECL_OVERRIDE

Protected Member Functions

- virtual [QCP::Interaction](#) **selectionCategory** () const Q_DECL_OVERRIDE
- virtual void **applyDefaultAntialiasingHint** ([QCPPainter](#) *painter) const Q_DECL_OVERRIDE
- virtual QRect **clipRect** () const Q_DECL_OVERRIDE
- virtual void **draw** ([QCPPainter](#) *painter) Q_DECL_OVERRIDE=0
- virtual void **selectEvent** (QMouseEvent *event, bool additive, const QVariant &details, bool *selectionStateChanged) Q_DECL_OVERRIDE
- virtual void **deselectEvent** (bool *selectionStateChanged) Q_DECL_OVERRIDE

Protected Attributes

- [QCPLegend](#) * **mParentLegend**
- QFont **mFont**
- QColor **mTextColor**
- QFont **mSelectedFont**
- QColor **mSelectedTextColor**
- bool **mSelectable**
- bool **mSelected**

Friends

- class **QCPLegend**

Additional Inherited Members

6.8.1 Detailed Description

The abstract base class for all entries in a [QCPLegend](#).

It defines a very basic interface for entries in a [QCPLegend](#). For representing plottables in the legend, the subclass [QCPPlottableLegendItem](#) is more suitable.

Only derive directly from this class when you need absolute freedom (e.g. a custom legend entry that's not even associated with a plottable).

You must implement the following pure virtual functions:

- draw (from [QCPLayerable](#))

You inherit the following members you may use:

QCPLegend * mParentLegend	A pointer to the parent QCPLegend .
QFont mFont	The generic font of the item. You should use this font for all or at least the most prominent text of the item.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 QCPAbstractLegendItem()

```
QCPAbstractLegendItem::QCPAbstractLegendItem (  
    QCPLegend * parent )    [explicit]
```

Constructs a [QCPAbstractLegendItem](#) and associates it with the [QCPLegend](#) *parent*. This does not cause the item to be added to *parent*, so [QCPLegend::addItem](#) must be called separately.

6.8.3 Member Function Documentation

6.8.3.1 selectionChanged

```
void QCPAbstractLegendItem::selectionChanged (  
    bool selected )    [signal]
```

This signal is emitted when the selection state of this legend item has changed, either by user interaction or by a direct call to [setSelected](#).

6.8.3.2 selectTest()

```
double QCPAbstractLegendItem::selectTest (
    const QPointF & pos,
    bool onlySelectable,
    QVariant * details = 0 ) const [virtual]
```

Layout elements are sensitive to events inside their outer rect. If *pos* is within the outer rect, this method returns a value corresponding to 0.99 times the parent plot's selection tolerance. However, layout elements are not selectable by default. So if *onlySelectable* is true, -1.0 is returned.

See [QCPLayerable::selectTest](#) for a general explanation of this virtual method.

[QCPLayoutElement](#) subclasses may reimplement this method to provide more specific selection test behaviour.

Reimplemented from [QCPLayoutElement](#).

6.8.3.3 setFont()

```
void QCPAbstractLegendItem::setFont (
    const QFont & font )
```

Sets the default font of this specific legend item to *font*.

See also

[setTextColor](#), [QCPLegend::setFont](#)

6.8.3.4 setSelectable()

```
void QCPAbstractLegendItem::setSelectable (
    bool selectable )
```

Sets whether this specific legend item is selectable.

See also

[setSelectedParts](#), [QCustomPlot::setInteractions](#)

6.8.3.5 setSelected()

```
void QCPAbstractLegendItem::setSelected (
    bool selected )
```

Sets whether this specific legend item is selected.

It is possible to set the selection state of this item by calling this function directly, even if `setSelectable` is set to false.

See also

`setSelectableParts`, [QCustomPlot::setInteractions](#)

6.8.3.6 setSelectedFont()

```
void QCPAbstractLegendItem::setSelectedFont (
    const QFont & font )
```

When this legend item is selected, *font* is used to draw generic text, instead of the normal font set with [setFont](#).

See also

[setFont](#), [QCPLegend::setSelectedFont](#)

6.8.3.7 setSelectedTextColor()

```
void QCPAbstractLegendItem::setSelectedTextColor (
    const QColor & color )
```

When this legend item is selected, *color* is used to draw generic text, instead of the normal color set with [setTextColor](#).

See also

[setTextColor](#), [QCPLegend::setSelectedTextColor](#)

6.8.3.8 setTextColor()

```
void QCPAbstractLegendItem::setTextColor (
    const QColor & color )
```

Sets the default text color of this specific legend item to *color*.

See also

[setFont](#), [QCPLegend::setTextColor](#)

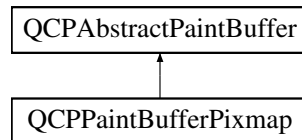
The documentation for this class was generated from the following files:

- `/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h`
- `/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp`

6.9 QCPAbstractPaintBuffer Class Reference

The abstract base class for paint buffers, which define the rendering backend.

Inheritance diagram for QCPAbstractPaintBuffer:



Public Member Functions

- [QCPAbstractPaintBuffer](#) (const QSize &size, double devicePixelRatio)
- QSize **size** () const
- bool **invalidated** () const
- double **devicePixelRatio** () const
- void [setSize](#) (const QSize &size)
- void [setInvalidated](#) (bool invalidated=true)
- void [setDevicePixelRatio](#) (double ratio)
- virtual [QCPPainter](#) * [startPainting](#) ()=0
- virtual void [donePainting](#) ()
- virtual void [draw](#) ([QCPPainter](#) *painter) const =0
- virtual void [clear](#) (const QColor &color)=0

Protected Member Functions

- virtual void [reallocateBuffer](#) ()=0

Protected Attributes

- QSize **mSize**
- double **mDevicePixelRatio**
- bool **mInvalidated**

6.9.1 Detailed Description

The abstract base class for paint buffers, which define the rendering backend.

This abstract base class defines the basic interface that a paint buffer needs to provide in order to be usable by [QCustomPlot](#).

A paint buffer manages both a surface to draw onto, and the matching paint device. The size of the surface can be changed via [setSize](#). External classes ([QCustomPlot](#) and [QCPLayer](#)) request a painter via [startPainting](#) and then perform the draw calls. Once the painting is complete, [donePainting](#) is called, so the paint buffer implementation can do clean up if necessary. Before rendering a frame, each paint buffer is usually filled with a color using [clear](#) (usually the color is `Qt::transparent`), to remove the contents of the previous frame.

The simplest paint buffer implementation is [QCPPaintBufferPixmap](#) which allows regular software rendering via the raster engine. Hardware accelerated rendering via pixel buffers and frame buffer objects is provided by [QCPaintBufferGLPbuffer](#) and [QCPPaintBufferGLFbo](#). They are used automatically if [QCustomPlot::setOpenGl](#) is enabled.

6.9.2 Constructor & Destructor Documentation

6.9.2.1 QCPAbstractPaintBuffer()

```
QCPAbstractPaintBuffer::QCPAbstractPaintBuffer (
    const QSize & size,
    double devicePixelRatio ) [explicit]
```

Creates a paint buffer and initializes it with the provided *size* and *devicePixelRatio*.

Subclasses must call their [reallocateBuffer](#) implementation in their respective constructors.

6.9.3 Member Function Documentation

6.9.3.1 clear()

```
void QCPAbstractPaintBuffer::clear (
    const QColor & color ) [pure virtual]
```

Fills the entire buffer with the provided *color*. To have an empty transparent buffer, use the named color `Qt::transparent`.

This method must not be called if there is currently a painter (acquired with [startPainting](#)) active.

Implemented in [QCPPaintBufferPixmap](#).

6.9.3.2 donePainting()

```
void QCPAbstractPaintBuffer::donePainting ( ) [inline], [virtual]
```

If you have acquired a [QCPPainter](#) to paint onto this paint buffer via [startPainting](#), call this method as soon as you are done with the painting operations and have deleted the painter.

paint buffer subclasses may use this method to perform any type of cleanup that is necessary. The default implementation does nothing.

6.9.3.3 draw()

```
void QCPAbstractPaintBuffer::draw (
    QCPPainter * painter ) const [pure virtual]
```

Draws the contents of this buffer with the provided *painter*. This is the method that is used to finally join all paint buffers and draw them onto the screen.

Implemented in [QCPPaintBufferPixmap](#).

6.9.3.4 `reallocateBuffer()`

```
void QCPAbstractPaintBuffer::reallocateBuffer ( ) [protected], [pure virtual]
```

Reallocates the internal buffer with the currently configured size ([setSize](#)) and device pixel ratio, if applicable ([setDevicePixelRatio](#)). It is called as soon as any of those properties are changed on this paint buffer.

Note

Subclasses of [QCPAbstractPaintBuffer](#) must call their reimplementation of this method in their constructor, to perform the first allocation (this can not be done by the base class because calling pure virtual methods in base class constructors is not possible).

Implemented in [QCPPaintBufferPixmap](#).

6.9.3.5 `setDevicePixelRatio()`

```
void QCPAbstractPaintBuffer::setDevicePixelRatio (
    double ratio )
```

Sets the the device pixel ratio to *ratio*. This is useful to render on high-DPI output devices. The ratio is automatically set to the device pixel ratio used by the parent [QCustomPlot](#) instance.

The buffer is reallocated (by calling [reallocateBuffer](#)), so any painters that were obtained by [startPainting](#) are invalidated and must not be used after calling this method.

Note

This method is only available for Qt versions 5.4 and higher.

6.9.3.6 `setInvalidated()`

```
void QCPAbstractPaintBuffer::setInvalidated (
    bool invalidated = true )
```

Sets the invalidated flag to *invalidated*.

This mechanism is used internally in conjunction with isolated replotting of [QCPLayer](#) instances (in [QCPLayer::ImBuffered](#) mode). If [QCPLayer::replot](#) is called on a buffered layer, i.e. an isolated repaint of only that layer (and its dedicated paint buffer) is requested, [QCustomPlot](#) will decide depending on the invalidated flags of other paint buffers whether it also replots them, instead of only the layer on which the replot was called.

The invalidated flag is set to true when [QCPLayer](#) association has changed, i.e. if layers were added or removed from this buffer, or if they were reordered. It is set to false as soon as all associated [QCPLayer](#) instances are drawn onto the buffer.

Under normal circumstances, it is not necessary to manually call this method.

6.9.3.7 setSize()

```
void QCPAbstractPaintBuffer::setSize (
    const QSize & size )
```

Sets the paint buffer size.

The buffer is reallocated (by calling [reallocateBuffer](#)), so any painters that were obtained by [startPainting](#) are invalidated and must not be used after calling this method.

If *size* is already the current buffer size, this method does nothing.

6.9.3.8 startPainting()

```
QCPPainter * QCPAbstractPaintBuffer::startPainting ( ) [pure virtual]
```

Returns a [QCPPainter](#) which is ready to draw to this buffer. The ownership and thus the responsibility to delete the painter after the painting operations are complete is given to the caller of this method.

Once you are done using the painter, delete the painter and call [donePainting](#).

While a painter generated with this method is active, you must not call [setSize](#), [setDevicePixelRatio](#) or [clear](#).

This method may return 0, if a painter couldn't be activated on the buffer. This usually indicates a problem with the respective painting backend.

Implemented in [QCPPaintBufferPixmap](#).

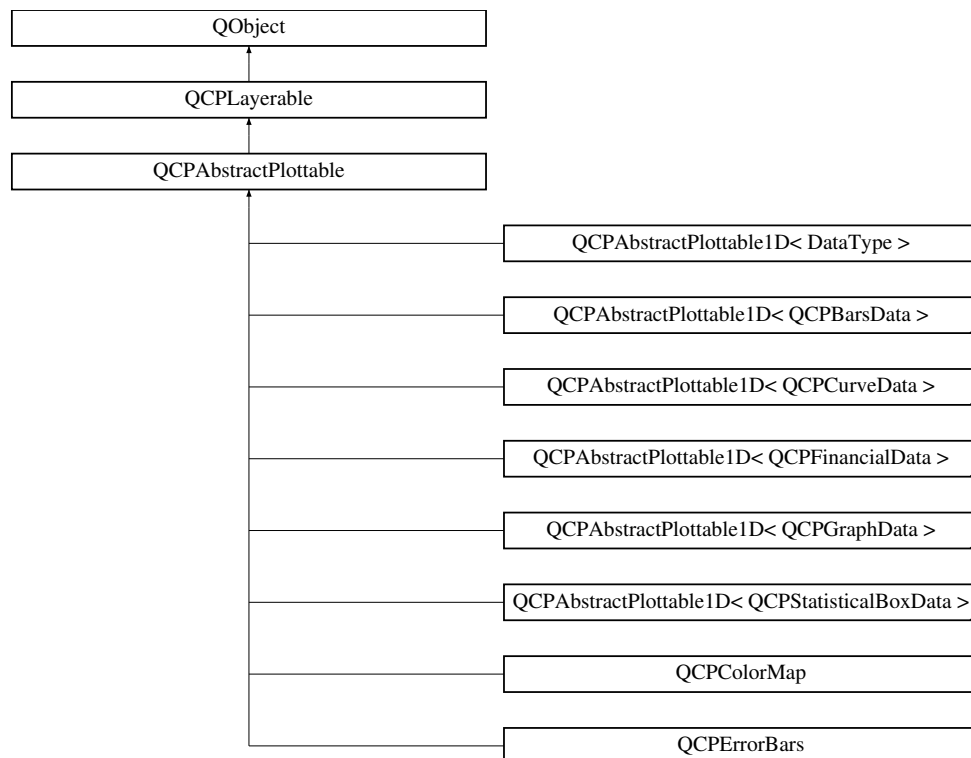
The documentation for this class was generated from the following files:

- [/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h](#)↔
- [/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp](#)↔

6.10 QCPAbstractPlottable Class Reference

The abstract base class for all data representing objects in a plot.

Inheritance diagram for QCPAbstractPlottable:



Signals

- void [selectionChanged](#) (bool [selected](#))
- void [selectionChanged](#) (const [QCPDataSelection](#) &[selection](#))
- void [selectableChanged](#) ([QCP::SelectionType](#) selectable)

Public Member Functions

- [QCPAbstractPlottable](#) ([QCPAxis](#) *keyAxis, [QCPAxis](#) *valueAxis)
- QString **name** () const
- bool **antialiasedFill** () const
- bool **antialiasedScatters** () const
- QPen **pen** () const
- QBrush **brush** () const
- [QCPAxis](#) * **keyAxis** () const
- [QCPAxis](#) * **valueAxis** () const
- [QCP::SelectionType](#) **selectable** () const
- bool [selected](#) () const
- [QCPDataSelection](#) [selection](#) () const
- [QCPSelectionDecorator](#) * [selectionDecorator](#) () const
- void [setName](#) (const QString &name)
- void [setAntialiasedFill](#) (bool enabled)
- void [setAntialiasedScatters](#) (bool enabled)
- void [setPen](#) (const QPen &pen)
- void [setBrush](#) (const QBrush &brush)
- void [setKeyAxis](#) ([QCPAxis](#) *axis)
- void [setValueAxis](#) ([QCPAxis](#) *axis)
- Q_SLOT void [setSelectable](#) ([QCP::SelectionType](#) selectable)

- Q_SLOT void [setSelection](#) (QCPDataSelection selection)
- void [setSelectionDecorator](#) (QCPSelectionDecorator *decorator)
- virtual double [selectTest](#) (const QPointF &pos, bool onlySelectable, QVariant *details=0) const =0
- virtual [QCPPlottableInterface1D](#) * [interface1D](#) ()
- virtual [QCPRange](#) [getKeyRange](#) (bool &foundRange, [QCP::SignDomain](#) inSignDomain=[QCP::sdBoth](#)) const =0
- virtual [QCPRange](#) [getValueRange](#) (bool &foundRange, [QCP::SignDomain](#) inSignDomain=[QCP::sdBoth](#), const [QCPRange](#) &inKeyRange=[QCPRange](#)()) const =0
- void [coordsToPixels](#) (double key, double value, double &x, double &y) const
- const QPointF [coordsToPixels](#) (double key, double value) const
- void [pixelsToCoords](#) (double x, double y, double &key, double &value) const
- void [pixelsToCoords](#) (const QPointF &pixelPos, double &key, double &value) const
- void [rescaleAxes](#) (bool onlyEnlarge=false) const
- void [rescaleKeyAxis](#) (bool onlyEnlarge=false) const
- void [rescaleValueAxis](#) (bool onlyEnlarge=false, bool inKeyRange=false) const
- bool [addToLegend](#) (QCPLegend *legend)
- bool [addToLegend](#) ()
- bool [removeFromLegend](#) (QCPLegend *legend) const
- bool [removeFromLegend](#) () const

Protected Member Functions

- virtual QRect [clipRect](#) () const Q_DECL_OVERRIDE
- virtual void [draw](#) (QCPPainter *painter) Q_DECL_OVERRIDE=0
- virtual [QCP::Interaction](#) [selectionCategory](#) () const Q_DECL_OVERRIDE
- void [applyDefaultAntialiasingHint](#) (QCPPainter *painter) const Q_DECL_OVERRIDE
- virtual void [selectEvent](#) (QMouseEvent *event, bool additive, const QVariant &details, bool *selection<→ StateChanged) Q_DECL_OVERRIDE
- virtual void [deselectEvent](#) (bool *selectionStateChanged) Q_DECL_OVERRIDE
- virtual void [drawLegendIcon](#) (QCPPainter *painter, const QRectF &rect) const =0
- void [applyFillAntialiasingHint](#) (QCPPainter *painter) const
- void [applyScattersAntialiasingHint](#) (QCPPainter *painter) const

Protected Attributes

- QString **mName**
- bool **mAntialiasedFill**
- bool **mAntialiasedScatters**
- QPen **mPen**
- QBrush **mBrush**
- QPointer< [QCPAxis](#) > **mKeyAxis**
- QPointer< [QCPAxis](#) > **mValueAxis**
- [QCP::SelectionType](#) **mSelectable**
- [QCPDataSelection](#) **mSelection**
- [QCPSelectionDecorator](#) * **mSelectionDecorator**

Friends

- class **QCustomPlot**
- class **QCPAxis**
- class **QCPPlottableLegendItem**

6.10.1 Detailed Description

The abstract base class for all data representing objects in a plot.

It defines a very basic interface like name, pen, brush, visibility etc. Since this class is abstract, it can't be instantiated. Use one of the subclasses or create a subclass yourself to create new ways of displaying data (see "Creating own plottables" below). Plottables that display one-dimensional data (i.e. data points have a single key dimension and one or multiple values at each key) are based off of the template subclass [QCPAbstractPlottable1D](#), see details there.

All further specifics are in the subclasses, for example:

- A normal graph with possibly a line and/or scatter points [QCPGraph](#) (typically created with [QCustomPlot::addGraph](#))
- A parametric curve: [QPCurve](#)
- A bar chart: [QCPBars](#)
- A statistical box plot: [QCPStatisticalBox](#)
- A color encoded two-dimensional map: [QCPColorMap](#)
- An OHLC/Candlestick chart: [QCPFinancial](#)

6.10.2 Creating own plottables

Subclassing directly from [QCPAbstractPlottable](#) is only recommended if you wish to display two-dimensional data like [QCPColorMap](#), i.e. two logical key dimensions and one (or more) data dimensions. If you want to display data with only one logical key dimension, you should rather derive from [QCPAbstractPlottable1D](#).

If subclassing [QCPAbstractPlottable](#) directly, these are the pure virtual functions you must implement:

- [selectTest](#)
- [draw](#)
- [drawLegendIcon](#)
- [getKeyRange](#)
- [getValueRange](#)

See the documentation of those functions for what they need to do.

For drawing your plot, you can use the [coordsToPixels](#) functions to translate a point in plot coordinates to pixel coordinates. This function is quite convenient, because it takes the orientation of the key and value axes into account for you (x and y are swapped when the key axis is vertical and the value axis horizontal). If you are worried about performance (i.e. you need to translate many points in a loop like [QCPGraph](#)), you can directly use [QCPAxis::coordToPixel](#). However, you must then take care about the orientation of the axis yourself.

Here are some important members you inherit from [QCPAbstractPlottable](#):

QCustomPlot * mParentPlot	A pointer to the parent QCustomPlot instance. The parent plot is inferred from the axes that are passed in the constructor.
QString mName	The name of the plottable.
QPen mPen	The generic pen of the plottable. You should use this pen for the most prominent data representing lines in the plottable (e.g. QCPGraph uses this pen for its graph lines and scatters)
QBrush mBrush	The generic brush of the plottable. You should use this brush for the most prominent fillable structures in the plottable (e.g. QCPGraph uses this brush to control filling under the graph)
QPointer< QCPAxis > mKeyAxis, mValueAxis	The key and value axes this plottable is attached to. Call their QCPAxis::coordToPixel functions to translate coordinates to pixels in either the key or value dimension. Make sure to check whether the pointer is null before using it. If one of the axes is null, don't draw the plottable.
QCPSelectionDecorator mSelectionDecorator	The currently set selection decorator which specifies how selected data of the plottable shall be drawn and decorated. When drawing your data, you must consult this decorator for the appropriate pen/brush before drawing unselected/selected data segments. Finally, you should call its QCPSelectionDecorator::drawDecoration method at the end of your draw implementation.
QCP::SelectionType mSelectable	In which composition, if at all, this plottable's data may be selected. Enforcing this setting on the data selection is done by QCPAbstractPlottable automatically.
QCPDataSelection mSelection	Holds the current selection state of the plottable's data, i.e. the selected data ranges (QCPDataRange).

6.10.3 Constructor & Destructor Documentation

6.10.3.1 QCPAbstractPlottable()

```
QCPAbstractPlottable::QCPAbstractPlottable (
    QCPAxis * keyAxis,
    QCPAxis * valueAxis )
```

Constructs an abstract plottable which uses *keyAxis* as its key axis ("x") and *valueAxis* as its value axis ("y"). *keyAxis* and *valueAxis* must reside in the same [QCustomPlot](#) instance and have perpendicular orientations. If either of these restrictions is violated, a corresponding message is printed to the debug output (qDebug), the construction is not aborted, though.

Since [QCPAbstractPlottable](#) is an abstract class that defines the basic interface to plottables, it can't be directly instantiated.

You probably want one of the subclasses like [QCPGraph](#) or [QCPCurve](#) instead.

6.10.4 Member Function Documentation

6.10.4.1 addToLegend() [1/2]

```
bool QCPAbstractPlottable::addToLegend (
    QCPLegend * legend )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Adds this plottable to the specified *legend*.

Creates a [QCPPlottableLegendItem](#) which is inserted into the legend. Returns true on success, i.e. when the legend exists and a legend item associated with this plottable isn't already in the legend.

If the plottable needs a more specialized representation in the legend, you can create a corresponding subclass of [QCPPlottableLegendItem](#) and add it to the legend manually instead of calling this method.

See also

[removeFromLegend](#), [QCPLegend::addItem](#)

6.10.4.2 addToLegend() [2/2]

```
bool QCPAbstractPlottable::addToLegend ( )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Adds this plottable to the legend of the parent [QCustomPlot](#) ([QCustomPlot::legend](#)).

See also

[removeFromLegend](#)

6.10.4.3 coordsToPixels() [1/2]

```
void QCPAbstractPlottable::coordsToPixels (
    double key,
    double value,
    double & x,
    double & y ) const
```

Convenience function for transforming a key/value pair to pixels on the [QCustomPlot](#) surface, taking the orientations of the axes associated with this plottable into account (e.g. whether key represents x or y).

key and *value* are transformed to the coordinates in pixels and are written to *x* and *y*.

See also

[pixelsToCoords](#), [QCPAxis::coordToPixel](#)

6.10.4.4 coordsToPixels() [2/2]

```
const QPointF QCPAbstractPlottable::coordsToPixels (
    double key,
    double value ) const
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Transforms the given *key* and *value* to pixel coordinates and returns them in a QPointF.

6.10.4.5 getKeyRange()

```
QCPRange QCPAbstractPlottable::getKeyRange (
    bool & foundRange,
    QCP::SignDomain inSignDomain = QCP::sdBoth ) const [pure virtual]
```

Returns the coordinate range that all data in this plottable span in the key axis dimension. For logarithmic plots, one can set *inSignDomain* to either [QCP::sdNegative](#) or [QCP::sdPositive](#) in order to restrict the returned range to that sign domain. E.g. when only negative range is wanted, set *inSignDomain* to [QCP::sdNegative](#) and all positive points will be ignored for range calculation. For no restriction, just set *inSignDomain* to [QCP::sdBoth](#) (default). *foundRange* is an output parameter that indicates whether a range could be found or not. If this is false, you shouldn't use the returned range (e.g. no points in data).

Note that *foundRange* is not the same as [QCPRange::validRange](#), since the range returned by this function may have size zero (e.g. when there is only one data point). In this case *foundRange* would return true, but the returned range is not a valid range in terms of [QCPRange::validRange](#).

See also

[rescaleAxes](#), [getValueRange](#)

Implemented in [QCPErrorBars](#), [QCPFinancial](#), [QCPColorMap](#), [QCPStatisticalBox](#), [QCPBars](#), [QCPCurve](#), and [QCPGraph](#).

6.10.4.6 getValueRange()

```
QCPRange QCPAbstractPlottable::getValueRange (
    bool & foundRange,
    QCP::SignDomain inSignDomain = QCP::sdBoth,
    const QCPRange & inKeyRange = QCPRange() ) const [pure virtual]
```

Returns the coordinate range that the data points in the specified key range (*inKeyRange*) span in the value axis dimension. For logarithmic plots, one can set *inSignDomain* to either [QCP::sdNegative](#) or [QCP::sdPositive](#) in order to restrict the returned range to that sign domain. E.g. when only negative range is wanted, set *inSignDomain* to [QCP::sdNegative](#) and all positive points will be ignored for range calculation. For no restriction, just set *inSignDomain* to [QCP::sdBoth](#) (default). *foundRange* is an output parameter that indicates whether a range could be found or not. If this is false, you shouldn't use the returned range (e.g. no points in data).

If *inKeyRange* has both lower and upper bound set to zero (is equal to [QCPRange\(\)](#)), all data points are considered, without any restriction on the keys.

Note that *foundRange* is not the same as [QCPRange::validRange](#), since the range returned by this function may have size zero (e.g. when there is only one data point). In this case *foundRange* would return true, but the returned range is not a valid range in terms of [QCPRange::validRange](#).

See also

[rescaleAxes](#), [getKeyRange](#)

Implemented in [QCPErrorBars](#), [QCPFinancial](#), [QCPColorMap](#), [QCPStatisticalBox](#), [QCPBars](#), [QCPCurve](#), and [QCPGraph](#).

6.10.4.7 interface1D()

```
QCPPlottableInterface1D * QCPAbstractPlottable::interface1D ( ) [inline], [virtual]
```

If this plottable is a one-dimensional plottable, i.e. it implements the [QCPPlottableInterface1D](#), returns the *this* pointer with that type. Otherwise (e.g. in the case of a [QCPColorMap](#)) returns zero.

You can use this method to gain read access to data coordinates while holding a pointer to the abstract base class only.

Reimplemented in [QCPErrorBars](#), [QCPAbstractPlottable1D< DataType >](#), [QCPAbstractPlottable1D< QCPFinancialData >](#), [QCPAbstractPlottable1D< QCPStatisticalBoxData >](#), [QCPAbstractPlottable1D< QCPGraphData >](#), [QCPAbstractPlottable1D< QCPBarsData >](#), and [QCPAbstractPlottable1D< QCPCurveData >](#).

6.10.4.8 pixelsToCoords() [1/2]

```
void QCPAbstractPlottable::pixelsToCoords (
    double x,
    double y,
    double & key,
    double & value ) const
```

Convenience function for transforming a x/y pixel pair on the [QCustomPlot](#) surface to plot coordinates, taking the orientations of the axes associated with this plottable into account (e.g. whether key represents x or y).

x and y are transformed to the plot coordinates and are written to *key* and *value*.

See also

[coordsToPixels](#), [QCPAxis::coordToPixel](#)

6.10.4.9 pixelsToCoords() [2/2]

```
void QCPAbstractPlottable::pixelsToCoords (
    const QPointF & pixelPos,
    double & key,
    double & value ) const
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Returns the pixel input *pixelPos* as plot coordinates *key* and *value*.

6.10.4.10 removeFromLegend() [1/2]

```
bool QCPAbstractPlottable::removeFromLegend (
    QCPLegend * legend ) const
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Removes the plottable from the specified *legend*. This means the [QCPPlottableLegendItem](#) that is associated with this plottable is removed.

Returns true on success, i.e. if the legend exists and a legend item associated with this plottable was found and removed.

See also

[addToLegend](#), [QCPLegend::removeItem](#)

6.10.4.11 removeFromLegend() [2/2]

```
bool QCPAbstractPlottable::removeFromLegend ( ) const
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Removes the plottable from the legend of the parent [QCustomPlot](#).

See also

[addToLegend](#)

6.10.4.12 rescaleAxes()

```
void QCPAbstractPlottable::rescaleAxes (
    bool onlyEnlarge = false ) const
```

Rescales the key and value axes associated with this plottable to contain all displayed data, so the whole plottable is visible. If the scaling of an axis is logarithmic, `rescaleAxes` will make sure not to rescale to an illegal range i.e. a range containing different signs and/or zero. Instead it will stay in the current sign domain and ignore all parts of the plottable that lie outside of that domain.

onlyEnlarge makes sure the ranges are only expanded, never reduced. So it's possible to show multiple plottables in their entirety by multiple calls to `rescaleAxes` where the first call has *onlyEnlarge* set to false (the default), and all subsequent set to true.

See also

[rescaleKeyAxis](#), [rescaleValueAxis](#), [QCustomPlot::rescaleAxes](#), [QCPAxis::rescale](#)

6.10.4.13 `rescaleKeyAxis()`

```
void QCPAbstractPlottable::rescaleKeyAxis (
    bool onlyEnlarge = false ) const
```

Rescales the key axis of the plottable so the whole plottable is visible.

See [rescaleAxes](#) for detailed behaviour.

6.10.4.14 `rescaleValueAxis()`

```
void QCPAbstractPlottable::rescaleValueAxis (
    bool onlyEnlarge = false,
    bool inKeyRange = false ) const
```

Rescales the value axis of the plottable so the whole plottable is visible. If *inKeyRange* is set to true, only the data points which are in the currently visible key axis range are considered.

Returns true if the axis was actually scaled. This might not be the case if this plottable has an invalid range, e.g. because it has no data points.

See [rescaleAxes](#) for detailed behaviour.

6.10.4.15 `selectableChanged`

```
void QCPAbstractPlottable::selectableChanged (
    QCP::SelectionType selectable ) [signal]
```

This signal is emitted when the selectability of this plottable has changed.

See also

[setSelectable](#)

6.10.4.16 `selected()`

```
bool QCPAbstractPlottable::selected( ) const [inline]
```

Returns true if there are any data points of the plottable currently selected. Use [selection](#) to retrieve the current [QCPDataSelection](#).

6.10.4.17 `selection()`

```
QCPDataSelection QCPAbstractPlottable::selection ( ) const [inline]
```

Returns a [QCPDataSelection](#) encompassing all the data points that are currently selected on this plottable.

See also

[selected](#), [setSelection](#), [setSelectable](#)

6.10.4.18 selectionChanged [1/2]

```
void QCPAbstractPlottable::selectionChanged (
    bool selected ) [signal]
```

This signal is emitted when the selection state of this plottable has changed, either by user interaction or by a direct call to [setSelection](#). The parameter *selected* indicates whether there are any points selected or not.

See also

[selectionChanged\(const QCPDataSelection &selection\)](#)

6.10.4.19 selectionChanged [2/2]

```
void QCPAbstractPlottable::selectionChanged (
    const QCPDataSelection & selection ) [signal]
```

This signal is emitted when the selection state of this plottable has changed, either by user interaction or by a direct call to [setSelection](#). The parameter *selection* holds the currently selected data ranges.

See also

[selectionChanged\(bool selected\)](#)

6.10.4.20 selectionDecorator()

```
QCPSelectionDecorator * QCPAbstractPlottable::selectionDecorator ( ) const [inline]
```

Provides access to the selection decorator of this plottable. The selection decorator controls how selected data ranges are drawn (e.g. their pen color and fill), see [QCPSelectionDecorator](#) for details.

If you wish to use an own [QCPSelectionDecorator](#) subclass, pass an instance of it to [setSelectionDecorator](#).

6.10.4.21 selectTest()

```
virtual double QCPAbstractPlottable::selectTest (
    const QPointF & pos,
    bool onlySelectable,
    QVariant * details = 0 ) const [pure virtual]
```

This function is used to decide whether a click hits a layerable object or not.

pos is a point in pixel coordinates on the [QCustomPlot](#) surface. This function returns the shortest pixel distance of this point to the object. If the object is either invisible or the distance couldn't be determined, -1.0 is returned. Further, if *onlySelectable* is true and the object is not selectable, -1.0 is returned, too.

If the object is represented not by single lines but by an area like a [QCPItemText](#) or the bars of a [QCPBars](#) plottable, a click inside the area should also be considered a hit. In these cases this function thus returns a constant value greater zero but still below the parent plot's selection tolerance. (typically the selectionTolerance multiplied by 0.99).

Providing a constant value for area objects allows selecting line objects even when they are obscured by such area objects, by clicking close to the lines (i.e. closer than $0.99 \cdot \text{selectionTolerance}$).

The actual setting of the selection state is not done by this function. This is handled by the parent [QCustomPlot](#) when the `mouseReleaseEvent` occurs, and the finally selected object is notified via the `selectEvent/deselectEvent` methods.

details is an optional output parameter. Every layerable subclass may place any information in *details*. This information will be passed to `selectEvent` when the parent [QCustomPlot](#) decides on the basis of this `selectTest` call, that the object was successfully selected. The subsequent call to `selectEvent` will carry the *details*. This is useful for multi-part objects (like [QCPAxis](#)). This way, a possibly complex calculation to decide which part was clicked is only done once in `selectTest`. The result (i.e. the actually clicked part) can then be placed in *details*. So in the subsequent `selectEvent`, the decision which part was selected doesn't have to be done a second time for a single selection operation.

You may pass 0 as *details* to indicate that you are not interested in those selection details.

See also

`selectEvent`, `deselectEvent`, [mousePressEvent](#), [wheelEvent](#), [QCustomPlot::setInteractions](#)

Reimplemented from [QCPLayerable](#).

Implemented in [QCPErrorBars](#), [QCPFinancial](#), [QCPColorMap](#), [QCPStatisticalBox](#), [QCPBars](#), [QCPCurve](#), [QCPGraph](#), [QCPAbstractPlottable1D< DataType >](#), [QCPAbstractPlottable1D< QCPFinancialData >](#), [QCPAbstractPlottable1D< QCPStatisticalBoxData >](#), [QCPAbstractPlottable1D< QCPGraphData >](#), [QCPAbstractPlottable1D< QCPBarsData >](#), and [QCPAbstractPlottable1D< QCPCurveData >](#).

6.10.4.22 setAntialiasedFill()

```
void QCPAbstractPlottable::setAntialiasedFill (
    bool enabled )
```

Sets whether fills of this plottable are drawn antialiased or not.

Note that this setting may be overridden by [QCustomPlot::setAntialiasedElements](#) and [QCustomPlot::setNotAntialiasedElements](#).

6.10.4.23 setAntialiasedScatters()

```
void QCPAbstractPlottable::setAntialiasedScatters (
    bool enabled )
```

Sets whether the scatter symbols of this plottable are drawn antialiased or not.

Note that this setting may be overridden by [QCustomPlot::setAntialiasedElements](#) and [QCustomPlot::setNotAntialiasedElements](#).

6.10.4.24 setBrush()

```
void QCPAbstractPlottable::setBrush (
    const QBrush & brush )
```

The brush is used to draw basic fills of the plottable representation in the plot. The Fill can be a color, gradient or texture, see the usage of QBrush.

For example, the [QCPGraph](#) subclass draws the fill under the graph with this brush, when it's not set to Qt::NoBrush.

See also

[setPen](#)

6.10.4.25 setKeyAxis()

```
void QCPAbstractPlottable::setKeyAxis (
    QCPAxis * axis )
```

The key axis of a plottable can be set to any axis of a [QCustomPlot](#), as long as it is orthogonal to the plottable's value axis. This function performs no checks to make sure this is the case. The typical mathematical choice is to use the x-axis ([QCustomPlot::xAxis](#)) as key axis and the y-axis ([QCustomPlot::yAxis](#)) as value axis.

Normally, the key and value axes are set in the constructor of the plottable (or [QCustomPlot::addGraph](#) when working with QCPGraphs through the dedicated graph interface).

See also

[setValueAxis](#)

6.10.4.26 setName()

```
void QCPAbstractPlottable::setName (
    const QString & name )
```

The name is the textual representation of this plottable as it is displayed in the legend ([QCPLegend](#)). It may contain any UTF-8 characters, including newlines.

6.10.4.27 `setPen()`

```
void QCPAbstractPlottable::setPen (
    const QPen & pen )
```

The pen is used to draw basic lines that make up the plottable representation in the plot.

For example, the [QCPGraph](#) subclass draws its graph lines with this pen.

See also

[setBrush](#)

6.10.4.28 `setSelectable()`

```
void QCPAbstractPlottable::setSelectable (
    QCP::SelectionType selectable )
```

Sets whether and to which granularity this plottable can be selected.

A selection can happen by clicking on the [QCustomPlot](#) surface (When [QCustomPlot::setInteractions](#) contains [QCP::iSelectPlottables](#)), by dragging a selection rect (When [QCustomPlot::setSelectionRectMode](#) is [QCP::srmSelect](#)), or programmatically by calling [setSelection](#).

See also

[setSelection](#), [QCP::SelectionType](#)

6.10.4.29 `setSelection()`

```
void QCPAbstractPlottable::setSelection (
    QCPDataSelection selection )
```

Sets which data ranges of this plottable are selected. Selected data ranges are drawn differently (e.g. color) in the plot. This can be controlled via the selection decorator (see [selectionDecorator](#)).

The entire selection mechanism for plottables is handled automatically when [QCustomPlot::setInteractions](#) contains [iSelectPlottables](#). You only need to call this function when you wish to change the selection state programmatically.

Using [setSelectable](#) you can further specify for each plottable whether and to which granularity it is selectable. If *selection* is not compatible with the current [QCP::SelectionType](#) set via [setSelectable](#), the resulting selection will be adjusted accordingly (see [QCPDataSelection::enforceType](#)).

emits the [selectionChanged](#) signal when *selected* is different from the previous selection state.

See also

[setSelectable](#), [selectTest](#)

6.10.4.30 setSelectionDecorator()

```
void QCPAbstractPlottable::setSelectionDecorator (
    QCPSelectionDecorator * decorator )
```

Use this method to set an own [QCPSelectionDecorator](#) (subclass) instance. This allows you to customize the visual representation of selected data ranges further than by using the default [QCPSelectionDecorator](#).

The plottable takes ownership of the *decorator*.

The currently set decorator can be accessed via [selectionDecorator](#).

6.10.4.31 setValueAxis()

```
void QCPAbstractPlottable::setValueAxis (
    QCPAxis * axis )
```

The value axis of a plottable can be set to any axis of a [QCustomPlot](#), as long as it is orthogonal to the plottable's key axis. This function performs no checks to make sure this is the case. The typical mathematical choice is to use the x-axis ([QCustomPlot::xAxis](#)) as key axis and the y-axis ([QCustomPlot::yAxis](#)) as value axis.

Normally, the key and value axes are set in the constructor of the plottable (or [QCustomPlot::addGraph](#) when working with QCPGraphs through the dedicated graph interface).

See also

[setKeyAxis](#)

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

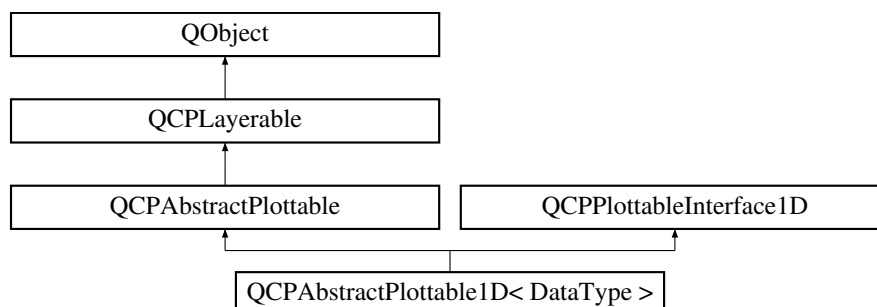
- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h
- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp

6.11 QCPAbstractPlottable1D< DataType > Class Template Reference

A template base class for plottables with one-dimensional data.

```
#include <qcustomplot.h>
```

Inheritance diagram for QCPAbstractPlottable1D< DataType >:



Public Member Functions

- [QCPAbstractPlottable1D](#) ([QCPAxis](#) *keyAxis, [QCPAxis](#) *valueAxis)
- virtual int [dataCount](#) () const
- virtual double [dataMainKey](#) (int index) const
- virtual double [dataSortKey](#) (int index) const
- virtual double [dataMainValue](#) (int index) const
- virtual [QCPRange](#) [dataValueRange](#) (int index) const
- virtual [QPointF](#) [dataPixelPosition](#) (int index) const
- virtual bool [sortKeysMainKey](#) () const
- virtual [QCPDataSelection](#) [selectTestRect](#) (const [QRectF](#) &rect, bool onlySelectable) const
- virtual int [findBegin](#) (double sortKey, bool expandedRange=true) const
- virtual int [findEnd](#) (double sortKey, bool expandedRange=true) const
- virtual double [selectTest](#) (const [QPointF](#) &pos, bool onlySelectable, [QVariant](#) *details=0) const
- virtual [QCPPlottableInterface1D](#) * [interface1D](#) ()

Protected Member Functions

- void [getDataSegments](#) ([QList](#)< [QCPDataRange](#) > &selectedSegments, [QList](#)< [QCPDataRange](#) > &unselectedSegments) const
- void [drawPolyline](#) ([QCPPainter](#) *painter, const [QVector](#)< [QPointF](#) > &lineData) const

Protected Attributes

- [QSharedPointer](#)< [QCPDataContainer](#)< *DataType* > > **mDataContainer**

Additional Inherited Members

6.11.1 Detailed Description

```
template<class DataType>
class QCPAbstractPlottable1D< DataType >
```

A template base class for plottables with one-dimensional data.

This template class derives from [QCPAbstractPlottable](#) and from the abstract interface [QCPPlottableInterface1D](#). It serves as a base class for all one-dimensional data (i.e. data with one key dimension), such as [QCPGraph](#) and [QCPCurve](#).

The template parameter *DataType* is the type of the data points of this plottable (e.g. [QCPGraphData](#) or [QCPCurveData](#)). The main purpose of this base class is to provide the member *mDataContainer* (a shared pointer to a [QCPDataContainer](#)<*DataType*>) and implement the according virtual methods of the [QCPPlottableInterface1D](#), such that most subclassed plottables don't need to worry about this anymore.

Further, it provides a convenience method for retrieving selected/unselected data segments via [getDataSegments](#). This is useful when subclasses implement their draw method and need to draw selected segments with a different pen/brush than unselected segments (also see [QCPSelectionDecorator](#)).

This class implements basic functionality of [QCPAbstractPlottable::selectTest](#) and [QCPPlottableInterface1D::selectTestRect](#), assuming point-like data points, based on the 1D data interface. In spite of that, most plottable subclasses will want to reimplement those methods again, to provide a more accurate hit test based on their specific data visualization geometry.

6.11.2 Constructor & Destructor Documentation

6.11.2.1 QCPAbstractPlottable1D()

```
template<class DataType >
QCPAbstractPlottable1D< DataType >::QCPAbstractPlottable1D (
    QCPAxis * keyAxis,
    QCPAxis * valueAxis )
```

Forwards *keyAxis* and *valueAxis* to the [QCPAbstractPlottable](#) constructor and allocates the *mDataContainer*.

6.11.3 Member Function Documentation

6.11.3.1 dataCount()

```
template<class DataType >
int QCPAbstractPlottable1D< DataType >::dataCount ( ) const [virtual]
```

Returns the number of data points of the plottable.

Implements [QCPPlottableInterface1D](#).

6.11.3.2 dataMainKey()

```
template<class DataType >
double QCPAbstractPlottable1D< DataType >::dataMainKey (
    int index ) const [virtual]
```

Returns the main key of the data point at the given *index*.

What the main key is, is defined by the plottable's data type. See the [QCPDataContainer DataType](#) documentation for details about this naming convention.

Implements [QCPPlottableInterface1D](#).

6.11.3.3 dataMainValue()

```
template<class DataType >
double QCPAbstractPlottable1D< DataType >::dataMainValue (
    int index ) const [virtual]
```

Returns the main value of the data point at the given *index*.

What the main value is, is defined by the plottable's data type. See the [QCPDataContainer DataType](#) documentation for details about this naming convention.

Implements [QCPPlottableInterface1D](#).

6.11.3.4 dataPixelPosition()

```
template<class DataType >
QPointF QCPAbstractPlottable1D< DataType >::dataPixelPosition (
    int index ) const [virtual]
```

Returns the pixel position on the widget surface at which the data point at the given *index* appears.

Usually this corresponds to the point of [dataMainKey/dataMainValue](#), in pixel coordinates. However, depending on the plottable, this might be a different apparent position than just a coord-to-pixel transform of those values. For example, [QCPBars](#) apparent data values can be shifted depending on their stacking, bar grouping or configured base value.

Implements [QCPPlottableInterface1D](#).

Reimplemented in [QCPBars](#).

6.11.3.5 dataSortKey()

```
template<class DataType >
double QCPAbstractPlottable1D< DataType >::dataSortKey (
    int index ) const [virtual]
```

Returns the sort key of the data point at the given *index*.

What the sort key is, is defined by the plottable's data type. See the [QCPDataContainer DataType](#) documentation for details about this naming convention.

Implements [QCPPlottableInterface1D](#).

6.11.3.6 dataValueRange()

```
template<class DataType >
QCPRange QCPAbstractPlottable1D< DataType >::dataValueRange (
    int index ) const [virtual]
```

Returns the value range of the data point at the given *index*.

What the value range is, is defined by the plottable's data type. See the [QCPDataContainer DataType](#) documentation for details about this naming convention.

Implements [QCPPlottableInterface1D](#).

6.11.3.7 drawPolyline()

```
template<class DataType >
void QCPAbstractPlottable1D< DataType >::drawPolyline (
    QCPPainter * painter,
    const QVector< QPointF > & lineData ) const [protected]
```

A helper method which draws a line with the passed *painter*, according to the pixel data in *lineData*. NaN points create gaps in the line, as expected from [QCustomPlot](#)'s plottables (this is the main difference to [QPainter](#)'s regular [drawPolyline](#), which handles NaNs by lagging or crashing).

Further it uses a faster line drawing technique based on [QCPPainter::drawLine](#) rather than [QPainter::drawPolyline](#) if the configured [QCustomPlot::setPlottingHints\(\)](#) and *painter* style allows.

6.11.3.8 findBegin()

```
template<class DataType >
int QCPAbstractPlottable1D< DataType >::findBegin (
    double sortKey,
    bool expandedRange = true ) const [virtual]
```

Returns the index of the data point with a (sort-)key that is equal to, just below, or just above *sortKey*. If *expandedRange* is true, the data point just below *sortKey* will be considered, otherwise the one just above.

This can be used in conjunction with [findEnd](#) to iterate over data points within a given key range, including or excluding the bounding data points that are just beyond the specified range.

If *expandedRange* is true but there are no data points below *sortKey*, 0 is returned.

If the container is empty, returns 0 (in that case, [findEnd](#) will also return 0, so a loop using these methods will not iterate over the index 0).

See also

[findEnd](#), [QCPDataContainer::findBegin](#)

Implements [QCPPlottableInterface1D](#).

6.11.3.9 findEnd()

```
template<class DataType >
int QCPAbstractPlottable1D< DataType >::findEnd (
    double sortKey,
    bool expandedRange = true ) const [virtual]
```

Returns the index one after the data point with a (sort-)key that is equal to, just above, or just below *sortKey*. If *expandedRange* is true, the data point just above *sortKey* will be considered, otherwise the one just below.

This can be used in conjunction with [findBegin](#) to iterate over data points within a given key range, including the bounding data points that are just below and above the specified range.

If *expandedRange* is true but there are no data points above *sortKey*, the index just above the highest data point is returned.

If the container is empty, returns 0.

See also

[findBegin](#), [QCPDataContainer::findEnd](#)

Implements [QCPPlottableInterface1D](#).

6.11.3.10 getDataSegments()

```
template<class DataType >
void QCPAbstractPlottable1D< DataType >::getDataSegments (
    QList< QCPDataRange > & selectedSegments,
    QList< QCPDataRange > & unselectedSegments ) const [protected]
```

Splits all data into selected and unselected segments and outputs them via *selectedSegments* and *unselectedSegments*, respectively.

This is useful when subclasses implement their draw method and need to draw selected segments with a different pen/brush than unselected segments (also see [QCPSelectionDecorator](#)).

See also

[setSelection](#)

6.11.3.11 interface1D()

```
template<class DataType>
QCPPlottableInterface1D * QCPAbstractPlottable1D< DataType >::interface1D ( ) [inline], [virtual]
```

Returns a [QCPPlottableInterface1D](#) pointer to this plottable, providing access to its 1D interface.

Reimplemented from [QCPAbstractPlottable](#).

6.11.3.12 selectTest()

```
template<class DataType >
double QCPAbstractPlottable1D< DataType >::selectTest (
    const QPointF & pos,
    bool onlySelectable,
    QVariant * details = 0 ) const [virtual]
```

Implements a point-selection algorithm assuming the data (accessed via the 1D data interface) is point-like. Most subclasses will want to reimplement this method again, to provide a more accurate hit test based on the true data visualization geometry.

Implements [QCPAbstractPlottable](#).

Reimplemented in [QCPFinancial](#), [QCPStatisticalBox](#), [QCPBars](#), [QCPCurve](#), and [QCPGraph](#).

6.11.3.13 selectTestRect()

```
template<class DataType >
QCPDataSelection QCPAbstractPlottable1D< DataType >::selectTestRect (
    const QRectF & rect,
    bool onlySelectable ) const [virtual]
```

Implements a rect-selection algorithm assuming the data (accessed via the 1D data interface) is point-like. Most subclasses will want to reimplement this method again, to provide a more accurate hit test based on the true data visualization geometry.

Implements [QCPPlottableInterface1D](#).

Reimplemented in [QCPFinancial](#), [QCPStatisticalBox](#), and [QCPBars](#).

6.11.3.14 sortKeysMainKey()

```
template<class DataType >
bool QCPAbstractPlottable1D< DataType >::sortKeyIsMainKey ( ) const [virtual]
```

Returns whether the sort key ([dataSortKey](#)) is identical to the main key ([dataMainKey](#)).

What the sort and main keys are, is defined by the plottable's data type. See the [QCPDataContainer DataType](#) documentation for details about this naming convention.

Implements [QCPPlottableInterface1D](#).

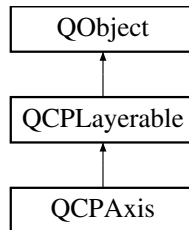
The documentation for this class was generated from the following file:

- `/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h`

6.12 QCPAxis Class Reference

Manages a single axis inside a [QCustomPlot](#).

Inheritance diagram for QCPAxis:



Public Types

- enum [AxisType](#) { [atLeft](#) = 0x01, [atRight](#) = 0x02, [atTop](#) = 0x04, [atBottom](#) = 0x08 }
- enum [LabelSide](#) { [lsInside](#), [lsOutside](#) }
- enum [ScaleType](#) { [stLinear](#), [stLogarithmic](#) }
- enum [SelectablePart](#) { [spNone](#) = 0, [spAxis](#) = 0x001, [spTickLabels](#) = 0x002, [spAxisLabel](#) = 0x004 }

Signals

- void [rangeChanged](#) (const [QCPRange](#) &newRange)
- void [rangeChanged](#) (const [QCPRange](#) &newRange, const [QCPRange](#) &oldRange)
- void [scaleTypeChanged](#) ([QCPAxis::ScaleType](#) scaleType)
- void [selectionChanged](#) (const [QCPAxis::SelectableParts](#) &parts)
- void [selectableChanged](#) (const [QCPAxis::SelectableParts](#) &parts)

Public Member Functions

- [QCPAxis](#) ([QCPAxisRect](#) *parent, [AxisType](#) type)
- [AxisType](#) **axisType** () const
- [QCPAxisRect](#) * **axisRect** () const
- [ScaleType](#) **scaleType** () const
- const [QCPRange](#) **range** () const
- bool **rangeReversed** () const
- QSharedPointer< [QCPAxisTicker](#) > **ticker** () const
- bool **ticks** () const
- bool **tickLabels** () const
- int **tickLabelPadding** () const
- QFont **tickLabelFont** () const
- QColor **tickLabelColor** () const
- double **tickLabelRotation** () const
- [LabelSide](#) **tickLabelSide** () const
- QString **numberFormat** () const
- int **numberPrecision** () const
- QVector< double > **tickVector** () const
- QVector< QString > **tickVectorLabels** () const
- int **tickLengthIn** () const

- int **tickLengthOut** () const
- bool **subTicks** () const
- int **subTickLengthIn** () const
- int **subTickLengthOut** () const
- QPen **basePen** () const
- QPen **tickPen** () const
- QPen **subTickPen** () const
- QFont **labelFont** () const
- QColor **labelColor** () const
- QString **label** () const
- int **labelPadding** () const
- int **padding** () const
- int **offset** () const
- SelectableParts **selectedParts** () const
- SelectableParts **selectableParts** () const
- QFont **selectedTickLabelFont** () const
- QFont **selectedLabelFont** () const
- QColor **selectedTickLabelColor** () const
- QColor **selectedLabelColor** () const
- QPen **selectedBasePen** () const
- QPen **selectedTickPen** () const
- QPen **selectedSubTickPen** () const
- QCPLineEnding **lowerEnding** () const
- QCPLineEnding **upperEnding** () const
- QCPGrid * **grid** () const
- Q_SLOT void **setScaleType** (QCPAxis::ScaleType type)
- Q_SLOT void **setRange** (const QCPRange &range)
- void **setRange** (double lower, double upper)
- void **setRange** (double position, double size, Qt::AlignmentFlag alignment)
- void **setRangeLower** (double lower)
- void **setRangeUpper** (double upper)
- void **setRangeReversed** (bool reversed)
- void **setTicker** (QSharedPointer< QCPAxisTicker > ticker)
- void **setTicks** (bool show)
- void **setTickLabels** (bool show)
- void **setTickLabelPadding** (int padding)
- void **setTickLabelFont** (const QFont &font)
- void **setTickLabelColor** (const QColor &color)
- void **setTickLabelRotation** (double degrees)
- void **setTickLabelSide** (LabelSide side)
- void **setNumberFormat** (const QString &formatCode)
- void **setNumberPrecision** (int precision)
- void **setTickLength** (int inside, int outside=0)
- void **setTickLengthIn** (int inside)
- void **setTickLengthOut** (int outside)
- void **setSubTicks** (bool show)
- void **setSubTickLength** (int inside, int outside=0)
- void **setSubTickLengthIn** (int inside)
- void **setSubTickLengthOut** (int outside)
- void **setBasePen** (const QPen &pen)
- void **setTickPen** (const QPen &pen)
- void **setSubTickPen** (const QPen &pen)
- void **setLabelFont** (const QFont &font)
- void **setLabelColor** (const QColor &color)
- void **setLabel** (const QString &str)

- void [setLabelPadding](#) (int padding)
- void [setPadding](#) (int padding)
- void [setOffset](#) (int offset)
- void [setSelectedTickLabelFont](#) (const QFont &font)
- void [setSelectedLabelFont](#) (const QFont &font)
- void [setSelectedTickLabelColor](#) (const QColor &color)
- void [setSelectedLabelColor](#) (const QColor &color)
- void [setSelectedBasePen](#) (const QPen &pen)
- void [setSelectedTickPen](#) (const QPen &pen)
- void [setSelectedSubTickPen](#) (const QPen &pen)
- Q_SLOT void [setSelectableParts](#) (const QCPAxis::SelectableParts &selectableParts)
- Q_SLOT void [setSelectedParts](#) (const QCPAxis::SelectableParts &selectedParts)
- void [setLowerEnding](#) (const QCPLLineEnding &ending)
- void [setUpperEnding](#) (const QCPLLineEnding &ending)
- virtual double [selectTest](#) (const QPointF &pos, bool onlySelectable, QVariant *details=0) const Q_DECL_OVERRIDE
- Qt::Orientation [orientation](#) () const
- int [pixelOrientation](#) () const
- void [moveRange](#) (double diff)
- void [scaleRange](#) (double factor)
- void [scaleRange](#) (double factor, double center)
- void [setScaleRatio](#) (const QCPAxis *otherAxis, double ratio=1.0)
- void [rescale](#) (bool onlyVisiblePlottables=false)
- double [pixelToCoord](#) (double value) const
- double [coordToPixel](#) (double value) const
- SelectablePart [getPartAt](#) (const QPointF &pos) const
- QList< QCPAbstractPlottable * > [plottables](#) () const
- QList< QCPGraph * > [graphs](#) () const
- QList< QCPAbstractItem * > [items](#) () const

Static Public Member Functions

- static AxisType [marginSideToAxisType](#) (QCP::MarginSide side)
- static Qt::Orientation [orientation](#) (AxisType type)
- static AxisType [opposite](#) (AxisType type)

Protected Member Functions

- virtual int [calculateMargin](#) ()
- virtual void [applyDefaultAntialiasingHint](#) (QCPPainter *painter) const Q_DECL_OVERRIDE
- virtual void [draw](#) (QCPPainter *painter) Q_DECL_OVERRIDE
- virtual QCP::Interaction [selectionCategory](#) () const Q_DECL_OVERRIDE
- virtual void [selectEvent](#) (QMouseEvent *event, bool additive, const QVariant &details, bool *selectionStateChanged) Q_DECL_OVERRIDE
- virtual void [deselectEvent](#) (bool *selectionStateChanged) Q_DECL_OVERRIDE
- void [setupTickVectors](#) ()
- QPen [getBasePen](#) () const
- QPen [getTickPen](#) () const
- QPen [getSubTickPen](#) () const
- QFont [getTickLabelFont](#) () const
- QFont [getLabelFont](#) () const
- QColor [getTickLabelColor](#) () const
- QColor [getLabelColor](#) () const

Protected Attributes

- [AxisType](#) **mAxisType**
- [QCPAxisRect](#) * **mAxisRect**
- int **mPadding**
- Qt::Orientation **mOrientation**
- SelectableParts **mSelectableParts**
- SelectableParts **mSelectedParts**
- QPen **mBasePen**
- QPen **mSelectedBasePen**
- QString **mLabel**
- QFont **mLabelFont**
- QFont **mSelectedLabelFont**
- QColor **mLabelColor**
- QColor **mSelectedLabelColor**
- bool **mTickLabels**
- QFont **mTickLabelFont**
- QFont **mSelectedTickLabelFont**
- QColor **mTickLabelColor**
- QColor **mSelectedTickLabelColor**
- int **mNumberPrecision**
- QLatin1Char **mNumberFormatChar**
- bool **mNumberBeautifulPowers**
- bool **mTicks**
- bool **mSubTicks**
- QPen **mTickPen**
- QPen **mSelectedTickPen**
- QPen **mSubTickPen**
- QPen **mSelectedSubTickPen**
- [QCPRange](#) **mRange**
- bool **mRangeReversed**
- [ScaleType](#) **mScaleType**
- [QCPGrid](#) * **mGrid**
- [QCPAxisPainterPrivate](#) * **mAxisPainter**
- QSharedPointer< [QCPAxisTicker](#) > **mTicker**
- QVector< double > **mTickVector**
- QVector< QString > **mTickVectorLabels**
- QVector< double > **mSubTickVector**
- bool **mCachedMarginValid**
- int **mCachedMargin**

Friends

- class **QCustomPlot**
- class **QCPGrid**
- class **QCPAxisRect**

6.12.1 Detailed Description

Manages a single axis inside a [QCustomPlot](#).

Usually doesn't need to be instantiated externally. Access [QCustomPlot](#)'s default four axes via [QCustomPlot::xAxis](#) (bottom), [QCustomPlot::yAxis](#) (left), [QCustomPlot::xAxis2](#) (top) and [QCustomPlot::yAxis2](#) (right).

Axes are always part of an axis rect, see [QCPAxisRect](#).

Naming convention of axis parts

Overview of the spacings and paddings that define the geometry of an axis. The dashed gray line on the left represents the [QCustomPlot](#) widget border.

Each axis holds an instance of [QCPAxisTicker](#) which is used to generate the tick coordinates and tick labels. You can access the currently installed [ticker](#) or set a new one (possibly one of the specialized subclasses, or your own subclass) via [setTicker](#). For details, see the documentation of [QCPAxisTicker](#).

6.12.2 Member Enumeration Documentation

6.12.2.1 AxisType

```
enum QCPAxis::AxisType
```

Defines at which side of the axis rect the axis will appear. This also affects how the tick marks are drawn, on which side the labels are placed etc.

Enumerator

atLeft	0x01 Axis is vertical and on the left side of the axis rect
atRight	0x02 Axis is vertical and on the right side of the axis rect
atTop	0x04 Axis is horizontal and on the top side of the axis rect
atBottom	0x08 Axis is horizontal and on the bottom side of the axis rect

6.12.2.2 LabelSide

```
enum QCPAxis::LabelSide
```

Defines on which side of the axis the tick labels (numbers) shall appear.

See also

[setTickLabelSide](#)

Enumerator

IsInside	Tick labels will be displayed inside the axis rect and clipped to the inner axis rect.
IsOutside	Tick labels will be displayed outside the axis rect.

6.12.2.3 ScaleType

enum [QCPAxis::ScaleType](#)

Defines the scale of an axis.

See also

[setScaleType](#)

Enumerator

stLinear	Linear scaling.
stLogarithmic	Logarithmic scaling with correspondingly transformed axis coordinates (possibly also setTicker to a QCPAxisTickerLog instance).

6.12.2.4 SelectablePart

enum [QCPAxis::SelectablePart](#)

Defines the selectable parts of an axis.

See also

[setSelectableParts](#), [setSelectedParts](#)

Enumerator

spNone	None of the selectable parts.
spAxis	The axis backbone and tick marks.
spTickLabels	Tick labels (numbers) of this axis (as a whole, not individually)
spAxisLabel	The axis label.

6.12.3 Constructor & Destructor Documentation

6.12.3.1 QCPAxis()

```
QCPAxis::QCPAxis (
    QCPAxisRect * parent,
    AxisType type ) [explicit]
```

Constructs an Axis instance of Type *type* for the axis rect *parent*.

Usually it isn't necessary to instantiate axes directly, because you can let [QCustomPlot](#) create them for you with [QCPAxisRect::addAxis](#). If you want to use own QCPAxis-subclasses however, create them manually and then inject them also via [QCPAxisRect::addAxis](#).

6.12.4 Member Function Documentation

6.12.4.1 coordToPixel()

```
double QCPAxis::coordToPixel (
    double value ) const
```

Transforms *value*, in coordinates of the axis, to pixel coordinates of the [QCustomPlot](#) widget.

6.12.4.2 getPartAt()

```
QCPAxis::SelectablePart QCPAxis::getPartAt (
    const QPointF & pos ) const
```

Returns the part of the axis that is hit by *pos* (in pixels). The return value of this function is independent of the user-selectable parts defined with [setSelectableParts](#). Further, this function does not change the current selection state of the axis.

If the axis is not visible ([setVisible](#)), this function always returns [spNone](#).

See also

[setSelectedParts](#), [setSelectableParts](#), [QCustomPlot::setInteractions](#)

6.12.4.3 graphs()

```
QList< QCPGraph * > QCPAxis::graphs ( ) const
```

Returns a list of all the graphs that have this axis as key or value axis.

See also

[plottables](#), [items](#)

6.12.4.4 grid()

```
QCPGrid * QCPAxis::grid ( ) const [inline]
```

Returns the [QCPGrid](#) instance belonging to this axis. Access it to set details about the way the grid is displayed.

6.12.4.5 items()

```
QList< QCPAbstractItem * > QCPAxis::items ( ) const
```

Returns a list of all the items that are associated with this axis. An item is considered associated with an axis if at least one of its positions uses the axis as key or value axis.

See also

[plottables](#), [graphs](#)

6.12.4.6 marginSideToAxisType()

```
QCPAxis::AxisType QCPAxis::marginSideToAxisType (
    QCP::MarginSide side ) [static]
```

Transforms a margin side to the logically corresponding axis type. ([QCP::msLeft](#) to [QCPAxis::atLeft](#), [QCP::msRight](#) to [QCPAxis::atRight](#), etc.)

6.12.4.7 moveRange()

```
void QCPAxis::moveRange (
    double diff )
```

If the scale type ([setScaleType](#)) is [stLinear](#), *diff* is added to the lower and upper bounds of the range. The range is simply moved by *diff*.

If the scale type is [stLogarithmic](#), the range bounds are multiplied by *diff*. This corresponds to an apparent "linear" move in logarithmic scaling by a distance of $\log(\text{diff})$.

6.12.4.8 opposite()

```
QCPAxis::AxisType QCPAxis::opposite (
    QCPAxis::AxisType type ) [static]
```

Returns the axis type that describes the opposite axis of an axis with the specified *type*.

6.12.4.9 orientation() [1/2]

```
Qt::Orientation QCPAxis::orientation ( ) const [inline]
```

Returns the orientation of this axis. The axis orientation (horizontal or vertical) is deduced from the axis type (left, top, right or bottom).

See also

[orientation\(AxisType type\)](#), [pixelOrientation](#)

6.12.4.10 orientation() [2/2]

```
static Qt::Orientation QCPAxis::orientation (
    AxisType type ) [inline], [static]
```

Returns the orientation of the specified axis type

See also

[orientation\(\)](#), [pixelOrientation](#)

6.12.4.11 pixelOrientation()

```
int QCPAxis::pixelOrientation ( ) const [inline]
```

Returns which direction points towards higher coordinate values/keys, in pixel space.

This method returns either 1 or -1. If it returns 1, then going in the positive direction along the orientation of the axis in pixels corresponds to going from lower to higher axis coordinates. On the other hand, if this method returns -1, going to smaller pixel values corresponds to going from lower to higher axis coordinates.

For example, this is useful to easily shift axis coordinates by a certain amount given in pixels, without having to care about reversed or vertically aligned axes:

```
double newKey = keyAxis->pixelToCoord(keyAxis->coordToPixel(oldKey)+10*keyAxis->pixelOrientation());
```

newKey will then contain a key that is ten pixels towards higher keys, starting from *oldKey*.

6.12.4.12 pixelToCoord()

```
double QCPAxis::pixelToCoord (
    double value ) const
```

Transforms *value*, in pixel coordinates of the [QCustomPlot](#) widget, to axis coordinates.

6.12.4.13 plottables()

```
QList< QCPAbstractPlottable * > QCPAxis::plottables ( ) const
```

Returns a list of all the plottables that have this axis as key or value axis.

If you are only interested in plottables of type [QCPGraph](#), see [graphs](#).

See also

[graphs](#), [items](#)

6.12.4.14 rangeChanged [1/2]

```
void QCPAxis::rangeChanged (
    const QCPRange & newRange ) [signal]
```

This signal is emitted when the range of this axis has changed. You can connect it to the [setRange](#) slot of another axis to communicate the new range to the other axis, in order for it to be synchronized.

You may also manipulate/correct the range with [setRange](#) in a slot connected to this signal. This is useful if for example a maximum range span shall not be exceeded, or if the lower/upper range shouldn't go beyond certain values (see [QCPRange::bounded](#)). For example, the following slot would limit the x axis to ranges between 0 and 10:

```
customPlot->xAxis->setRange(newRange.bounded(0, 10))
```

6.12.4.15 rangeChanged [2/2]

```
void QCPAxis::rangeChanged (
    const QCPRange & newRange,
    const QCPRange & oldRange ) [signal]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Additionally to the new range, this signal also provides the previous range held by the axis as *oldRange*.

6.12.4.16 `rescale()`

```
void QCPAxis::rescale (
    bool onlyVisiblePlottables = false )
```

Changes the axis range such that all plottables associated with this axis are fully visible in that dimension.

See also

[QCPAbstractPlottable::rescaleAxes](#), [QCustomPlot::rescaleAxes](#)

6.12.4.17 `scaleRange()` [1/2]

```
void QCPAxis::scaleRange (
    double factor )
```

Scales the range of this axis by *factor* around the center of the current axis range. For example, if *factor* is 2.0, then the axis range will double its size, and the point at the axis range center won't have changed its position in the [QCustomPlot](#) widget (i.e. coordinates around the center will have moved symmetrically closer).

If you wish to scale around a different coordinate than the current axis range center, use the overload [scaleRange\(double factor, double center\)](#).

6.12.4.18 `scaleRange()` [2/2]

```
void QCPAxis::scaleRange (
    double factor,
    double center )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Scales the range of this axis by *factor* around the coordinate *center*. For example, if *factor* is 2.0, *center* is 1.0, then the axis range will double its size, and the point at coordinate 1.0 won't have changed its position in the [QCustomPlot](#) widget (i.e. coordinates around 1.0 will have moved symmetrically closer to 1.0).

See also

[scaleRange\(double factor\)](#)

6.12.4.19 `scaleTypeChanged`

```
void QCPAxis::scaleTypeChanged (
    QCPAxis::ScaleType scaleType ) [signal]
```

This signal is emitted when the scale type changes, by calls to [setScaleType](#)

6.12.4.20 selectableChanged

```
void QCPAxis::selectableChanged (
    const QCPAxis::SelectableParts & parts ) [signal]
```

This signal is emitted when the selectability changes, by calls to [setSelectableParts](#)

6.12.4.21 selectionChanged

```
void QCPAxis::selectionChanged (
    const QCPAxis::SelectableParts & parts ) [signal]
```

This signal is emitted when the selection state of this axis has changed, either by user interaction or by a direct call to [setSelectedParts](#).

6.12.4.22 selectTest()

```
double QCPAxis::selectTest (
    const QPointF & pos,
    bool onlySelectable,
    QVariant * details = 0 ) const [virtual]
```

This function is used to decide whether a click hits a layerable object or not.

pos is a point in pixel coordinates on the [QCustomPlot](#) surface. This function returns the shortest pixel distance of this point to the object. If the object is either invisible or the distance couldn't be determined, -1.0 is returned. Further, if *onlySelectable* is true and the object is not selectable, -1.0 is returned, too.

If the object is represented not by single lines but by an area like a [QCPItemText](#) or the bars of a [QCPBars](#) plottable, a click inside the area should also be considered a hit. In these cases this function thus returns a constant value greater zero but still below the parent plot's selection tolerance. (typically the `selectionTolerance` multiplied by 0.99).

Providing a constant value for area objects allows selecting line objects even when they are obscured by such area objects, by clicking close to the lines (i.e. closer than $0.99 \cdot \text{selectionTolerance}$).

The actual setting of the selection state is not done by this function. This is handled by the parent [QCustomPlot](#) when the `mouseReleaseEvent` occurs, and the finally selected object is notified via the `selectEvent/deselectEvent` methods.

details is an optional output parameter. Every layerable subclass may place any information in *details*. This information will be passed to `selectEvent` when the parent [QCustomPlot](#) decides on the basis of this `selectTest` call, that the object was successfully selected. The subsequent call to `selectEvent` will carry the *details*. This is useful for multi-part objects (like [QCPAxis](#)). This way, a possibly complex calculation to decide which part was clicked is only done once in [selectTest](#). The result (i.e. the actually clicked part) can then be placed in *details*. So in the subsequent `selectEvent`, the decision which part was selected doesn't have to be done a second time for a single selection operation.

You may pass 0 as *details* to indicate that you are not interested in those selection details.

See also

`selectEvent`, `deselectEvent`, [mousePressEvent](#), [wheelEvent](#), [QCustomPlot::setInteractions](#)

Reimplemented from [QCPLayerable](#).

6.12.4.23 `setBasePen()`

```
void QCPAxis::setBasePen (
    const QPen & pen )
```

Sets the pen, the axis base line is drawn with.

See also

[setTickPen](#), [setSubTickPen](#)

6.12.4.24 `setLabel()`

```
void QCPAxis::setLabel (
    const QString & str )
```

Sets the text of the axis label that will be shown below/above or next to the axis, depending on its orientation. To disable axis labels, pass an empty string as *str*.

6.12.4.25 `setLabelColor()`

```
void QCPAxis::setLabelColor (
    const QColor & color )
```

Sets the color of the axis label.

See also

[setLabelFont](#)

6.12.4.26 `setLabelFont()`

```
void QCPAxis::setLabelFont (
    const QFont & font )
```

Sets the font of the axis label.

See also

[setLabelColor](#)

6.12.4.27 setLabelPadding()

```
void QCPAxis::setLabelPadding (
    int padding )
```

Sets the distance between the tick labels and the axis label.

See also

[setTickLabelPadding](#), [setPadding](#)

6.12.4.28 setLowerEnding()

```
void QCPAxis::setLowerEnding (
    const QCPLLineEnding & ending )
```

Sets the style for the lower axis ending. See the documentation of [QCPLLineEnding](#) for available styles.

For horizontal axes, this method refers to the left ending, for vertical axes the bottom ending. Note that this meaning does not change when the axis range is reversed with [setRangeReversed](#).

See also

[setUpperEnding](#)

6.12.4.29 setNumberFormat()

```
void QCPAxis::setNumberFormat (
    const QString & formatCode )
```

Sets the number format for the numbers in tick labels. This *formatCode* is an extended version of the format code used e.g. by `QString::number()` and `QLocale::toString()`. For reference about that, see the "Argument Formats" section in the detailed description of the `QString` class.

formatCode is a string of one, two or three characters. The first character is identical to the normal format code used by Qt. In short, this means: 'e'/'E' scientific format, 'f' fixed format, 'g'/'G' scientific or fixed, whichever is shorter.

The second and third characters are optional and specific to [QCustomPlot](#):

If the first char was 'e' or 'g', numbers are/might be displayed in the scientific format, e.g. "5.5e9", which is ugly in a plot. So when the second char of *formatCode* is set to 'b' (for "beautiful"), those exponential numbers are formatted in a more natural way, i.e. "5.5 [multiplication sign] 10 [superscript] 9". By default, the multiplication sign is a centered dot. If instead a cross should be shown (as is usual in the USA), the third char of *formatCode* can be set to 'c'. The inserted multiplication signs are the UTF-8 characters 215 (0xD7) for the cross and 183 (0xB7) for the dot.

Examples for *formatCode*:

- `g` normal format code behaviour. If number is small, fixed format is used, if number is large, normal scientific format is used
- `gb` If number is small, fixed format is used, if number is large, scientific format is used with beautifully typeset decimal powers and a dot as multiplication sign
- `ebc` All numbers are in scientific format with beautifully typeset decimal power and a cross as multiplication sign
- `fb` illegal format code, since fixed format doesn't support (or need) beautifully typeset decimal powers. Format code will be reduced to 'f'.
- `hello` illegal format code, since first char is not 'e', 'E', 'f', 'g' or 'G'. Current format code will not be changed.

6.12.4.30 `setNumberPrecision()`

```
void QCPAxis::setNumberPrecision (
    int precision )
```

Sets the precision of the tick label numbers. See `QLocale::toString(double i, char f, int prec)` for details. The effect of precisions are most notably for number Formats starting with 'e', see [setNumberFormat](#)

6.12.4.31 `setOffset()`

```
void QCPAxis::setOffset (
    int offset )
```

Sets the offset the axis has to its axis rect side.

If an axis rect side has multiple axes and automatic margin calculation is enabled for that side, only the offset of the inner most axis has meaning (even if it is set to be invisible). The offset of the other, outer axes is controlled automatically, to place them at appropriate positions.

6.12.4.32 `setPadding()`

```
void QCPAxis::setPadding (
    int padding )
```

Sets the padding of the axis.

When [QCPAxisRect::setAutoMargins](#) is enabled, the padding is the additional outer most space, that is left blank.

The axis padding has no meaning if [QCPAxisRect::setAutoMargins](#) is disabled.

See also

[setLabelPadding](#), [setTickLabelPadding](#)

6.12.4.33 `setRange()` [1/3]

```
void QCPAxis::setRange (
    const QCPRange & range )
```

Sets the range of the axis.

This slot may be connected with the [rangeChanged](#) signal of another axis so this axis is always synchronized with the other axis range, when it changes.

To invert the direction of an axis, use [setRangeReversed](#).

6.12.4.34 `setRange()` [2/3]

```
void QCPAxis::setRange (
    double lower,
    double upper )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Sets the lower and upper bound of the axis range.

To invert the direction of an axis, use [setRangeReversed](#).

There is also a slot to set a range, see [setRange\(const QCPRange &range\)](#).

6.12.4.35 `setRange()` [3/3]

```
void QCPAxis::setRange (
    double position,
    double size,
    Qt::AlignmentFlag alignment )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Sets the range of the axis.

The *position* coordinate indicates together with the *alignment* parameter, where the new range will be positioned. *size* defines the size of the new axis range. *alignment* may be `Qt::AlignLeft`, `Qt::AlignRight` or `Qt::AlignCenter`. This will cause the left border, right border, or center of the range to be aligned with *position*. Any other values of *alignment* will default to `Qt::AlignCenter`.

6.12.4.36 `setRangeLower()`

```
void QCPAxis::setRangeLower (
    double lower )
```

Sets the lower bound of the axis range. The upper bound is not changed.

See also

[setRange](#)

6.12.4.37 `setRangeReversed()`

```
void QCPAxis::setRangeReversed (
    bool reversed )
```

Sets whether the axis range (direction) is displayed reversed. Normally, the values on horizontal axes increase left to right, on vertical axes bottom to top. When *reversed* is set to true, the direction of increasing values is inverted.

Note that the range and data interface stays the same for reversed axes, e.g. the *lower* part of the [setRange](#) interface will still reference the mathematically smaller number than the *upper* part.

6.12.4.38 `setRangeUpper()`

```
void QCPAxis::setRangeUpper (
    double upper )
```

Sets the upper bound of the axis range. The lower bound is not changed.

See also

[setRange](#)

6.12.4.39 `setScaleRatio()`

```
void QCPAxis::setScaleRatio (
    const QCPAxis * otherAxis,
    double ratio = 1.0 )
```

Scales the range of this axis to have a certain scale *ratio* to *otherAxis*. The scaling will be done around the center of the current axis range.

For example, if *ratio* is 1, this axis is the *yAxis* and *otherAxis* is *xAxis*, graphs plotted with those axes will appear in a 1:1 aspect ratio, independent of the aspect ratio the axis rect has.

This is an operation that changes the range of this axis once, it doesn't fix the scale ratio indefinitely. Note that calling this function in the constructor of the [QCustomPlot](#)'s parent won't have the desired effect, since the widget dimensions aren't defined yet, and a `resizeEvent` will follow.

6.12.4.40 `setScaleType()`

```
void QCPAxis::setScaleType (
    QCPAxis::ScaleType type )
```

Sets whether the axis uses a linear scale or a logarithmic scale.

Note that this method controls the coordinate transformation. You will likely also want to use a logarithmic tick spacing and labeling, which can be achieved by setting an instance of [QCPAxisTickerLog](#) via [setTicker](#). See the documentation of [QCPAxisTickerLog](#) about the details of logarithmic axis tick creation.

[setNumberPrecision](#)

6.12.4.41 `setSelectableParts()`

```
void QCPAxis::setSelectableParts (
    const QCPAxis::SelectableParts & selectableParts )
```

Sets whether the user can (de-)select the parts in *selectable* by clicking on the [QCustomPlot](#) surface. (When [QCustomPlot::setInteractions](#) contains `iSelectAxes`.)

However, even when *selectable* is set to a value not allowing the selection of a specific part, it is still possible to set the selection of this part manually, by calling [setSelectedParts](#) directly.

See also

[SelectablePart](#), [setSelectedParts](#)

6.12.4.42 setSelectedBasePen()

```
void QCPAxis::setSelectedBasePen (
    const QPen & pen )
```

Sets the pen that is used to draw the axis base line when selected.

See also

[setBasePen](#), [setSelectableParts](#), [setSelectedParts](#), [QCustomPlot::setInteractions](#)

6.12.4.43 setSelectedLabelColor()

```
void QCPAxis::setSelectedLabelColor (
    const QColor & color )
```

Sets the color that is used for the axis label when it is selected.

See also

[setLabelColor](#), [setSelectableParts](#), [setSelectedParts](#), [QCustomPlot::setInteractions](#)

6.12.4.44 setSelectedLabelFont()

```
void QCPAxis::setSelectedLabelFont (
    const QFont & font )
```

Sets the font that is used for the axis label when it is selected.

See also

[setLabelFont](#), [setSelectableParts](#), [setSelectedParts](#), [QCustomPlot::setInteractions](#)

6.12.4.45 setSelectedParts()

```
void QCPAxis::setSelectedParts (
    const QCPAxis::SelectableParts & selectedParts )
```

Sets the selected state of the respective axis parts described by [SelectablePart](#). When a part is selected, it uses a different pen/font.

The entire selection mechanism for axes is handled automatically when [QCustomPlot::setInteractions](#) contains [iSelectAxes](#). You only need to call this function when you wish to change the selection state manually.

This function can change the selection state of a part, independent of the [setSelectableParts](#) setting.

emits the [selectionChanged](#) signal when *selected* is different from the previous selection state.

See also

[SelectablePart](#), [setSelectableParts](#), [selectTest](#), [setSelectedBasePen](#), [setSelectedTickPen](#), [setSelectedSub-
TickPen](#), [setSelectedTickLabelFont](#), [setSelectedLabelFont](#), [setSelectedTickLabelColor](#), [setSelectedLabel-
Color](#)

6.12.4.46 `setSelectedSubTickPen()`

```
void QCPAxis::setSelectedSubTickPen (
    const QPen & pen )
```

Sets the pen that is used to draw the subticks when selected.

See also

[setSubTickPen](#), [setSelectableParts](#), [setSelectedParts](#), [QCustomPlot::setInteractions](#)

6.12.4.47 `setSelectedTickLabelColor()`

```
void QCPAxis::setSelectedTickLabelColor (
    const QColor & color )
```

Sets the color that is used for tick labels when they are selected.

See also

[setTickLabelColor](#), [setSelectableParts](#), [setSelectedParts](#), [QCustomPlot::setInteractions](#)

6.12.4.48 `setSelectedTickLabelFont()`

```
void QCPAxis::setSelectedTickLabelFont (
    const QFont & font )
```

Sets the font that is used for tick labels when they are selected.

See also

[setTickLabelFont](#), [setSelectableParts](#), [setSelectedParts](#), [QCustomPlot::setInteractions](#)

6.12.4.49 `setSelectedTickPen()`

```
void QCPAxis::setSelectedTickPen (
    const QPen & pen )
```

Sets the pen that is used to draw the (major) ticks when selected.

See also

[setTickPen](#), [setSelectableParts](#), [setSelectedParts](#), [QCustomPlot::setInteractions](#)

6.12.4.50 setSubTickLength()

```
void QCPAxis::setSubTickLength (
    int inside,
    int outside = 0 )
```

Sets the length of the subticks in pixels. *inside* is the length the subticks will reach inside the plot and *outside* is the length they will reach outside the plot. If *outside* is greater than zero, the tick labels and axis label will increase their distance to the axis accordingly, so they won't collide with the ticks.

See also

[setTickLength](#), [setSubTickLengthIn](#), [setSubTickLengthOut](#)

6.12.4.51 setSubTickLengthIn()

```
void QCPAxis::setSubTickLengthIn (
    int inside )
```

Sets the length of the inward subticks in pixels. *inside* is the length the subticks will reach inside the plot.

See also

[setSubTickLengthOut](#), [setSubTickLength](#), [setTickLength](#)

6.12.4.52 setSubTickLengthOut()

```
void QCPAxis::setSubTickLengthOut (
    int outside )
```

Sets the length of the outward subticks in pixels. *outside* is the length the subticks will reach outside the plot. If *outside* is greater than zero, the tick labels will increase their distance to the axis accordingly, so they won't collide with the ticks.

See also

[setSubTickLengthIn](#), [setSubTickLength](#), [setTickLength](#)

6.12.4.53 setSubTickPen()

```
void QCPAxis::setSubTickPen (
    const QPen & pen )
```

Sets the pen, subtick marks will be drawn with.

See also

[setSubTickCount](#), [setSubTickLength](#), [setBasePen](#)

6.12.4.54 `setSubTicks()`

```
void QCPAxis::setSubTicks (
    bool show )
```

Sets whether sub tick marks are displayed.

Sub ticks are only potentially visible if (major) ticks are also visible (see [setTicks](#))

See also

[setTicks](#)

6.12.4.55 `setTicker()`

```
void QCPAxis::setTicker (
    QSharedPointer< QCPAxisTicker > ticker )
```

The axis ticker is responsible for generating the tick positions and tick labels. See the documentation of [QCPAxis<T> Ticker](#) for details on how to work with axis tickers.

You can change the tick positioning/labeling behaviour of this axis by setting a different [QCPAxisTicker](#) subclass using this method. If you only wish to modify the currently installed axis ticker, access it via [ticker](#).

Since the ticker is stored in the axis as a shared pointer, multiple axes may share the same axis ticker simply by passing the same shared pointer to multiple axes.

See also

[ticker](#)

6.12.4.56 `setTickLabelColor()`

```
void QCPAxis::setTickLabelColor (
    const QColor & color )
```

Sets the color of the tick labels.

See also

[setTickLabels](#), [setTickLabelFont](#)

6.12.4.57 setTickLabelFont()

```
void QCPAxis::setTickLabelFont (
    const QFont & font )
```

Sets the font of the tick labels.

See also

[setTickLabels](#), [setTickLabelColor](#)

6.12.4.58 setTickLabelPadding()

```
void QCPAxis::setTickLabelPadding (
    int padding )
```

Sets the distance between the axis base line (including any outward ticks) and the tick labels.

See also

[setLabelPadding](#), [setPadding](#)

6.12.4.59 setTickLabelRotation()

```
void QCPAxis::setTickLabelRotation (
    double degrees )
```

Sets the rotation of the tick labels. If *degrees* is zero, the labels are drawn normally. Else, the tick labels are drawn rotated by *degrees* clockwise. The specified angle is bound to values from -90 to 90 degrees.

If *degrees* is exactly -90, 0 or 90, the tick labels are centered on the tick coordinate. For other angles, the label is drawn with an offset such that it seems to point toward or away from the tick mark.

6.12.4.60 setTickLabels()

```
void QCPAxis::setTickLabels (
    bool show )
```

Sets whether tick labels are displayed. Tick labels are the numbers drawn next to tick marks.

6.12.4.61 setTickLabelSide()

```
void QCPAxis::setTickLabelSide (
    LabelSide side )
```

Sets whether the tick labels (numbers) shall appear inside or outside the axis rect.

The usual and default setting is [IsOutside](#). Very compact plots sometimes require tick labels to be inside the axis rect, to save space. If *side* is set to [IsInside](#), the tick labels appear on the inside are additionally clipped to the axis rect.

6.12.4.62 `setTickLength()`

```
void QCPAxis::setTickLength (
    int inside,
    int outside = 0 )
```

Sets the length of the ticks in pixels. *inside* is the length the ticks will reach inside the plot and *outside* is the length they will reach outside the plot. If *outside* is greater than zero, the tick labels and axis label will increase their distance to the axis accordingly, so they won't collide with the ticks.

See also

[setSubTickLength](#), [setTickLengthIn](#), [setTickLengthOut](#)

6.12.4.63 `setTickLengthIn()`

```
void QCPAxis::setTickLengthIn (
    int inside )
```

Sets the length of the inward ticks in pixels. *inside* is the length the ticks will reach inside the plot.

See also

[setTickLengthOut](#), [setTickLength](#), [setSubTickLength](#)

6.12.4.64 `setTickLengthOut()`

```
void QCPAxis::setTickLengthOut (
    int outside )
```

Sets the length of the outward ticks in pixels. *outside* is the length the ticks will reach outside the plot. If *outside* is greater than zero, the tick labels and axis label will increase their distance to the axis accordingly, so they won't collide with the ticks.

See also

[setTickLengthIn](#), [setTickLength](#), [setSubTickLength](#)

6.12.4.65 `setTickPen()`

```
void QCPAxis::setTickPen (
    const QPen & pen )
```

Sets the pen, tick marks will be drawn with.

See also

[setTickLength](#), [setBasePen](#)

6.12.4.66 setTicks()

```
void QCPAxis::setTicks (
    bool show )
```

Sets whether tick marks are displayed.

Note that setting *show* to false does not imply that tick labels are invisible, too. To achieve that, see [setTickLabels](#).

See also

[setSubTicks](#)

6.12.4.67 setUpperEnding()

```
void QCPAxis::setUpperEnding (
    const QCPLLineEnding & ending )
```

Sets the style for the upper axis ending. See the documentation of [QCPLLineEnding](#) for available styles.

For horizontal axes, this method refers to the right ending, for vertical axes the top ending. Note that this meaning does not change when the axis range is reversed with [setRangeReversed](#).

See also

[setLowerEnding](#)

6.12.4.68 ticker()

```
QSharedPointer< QCPAxisTicker > QCPAxis::ticker ( ) const [inline]
```

Returns a modifiable shared pointer to the currently installed axis ticker. The axis ticker is responsible for generating the tick positions and tick labels of this axis. You can access the [QCPAxisTicker](#) with this method and modify basic properties such as the approximate tick count ([QCPAxisTicker::setTickCount](#)).

You can gain more control over the axis ticks by setting a different [QCPAxisTicker](#) subclass, see the documentation there. A new axis ticker can be set with [setTicker](#).

Since the ticker is stored in the axis as a shared pointer, multiple axes may share the same axis ticker simply by passing the same shared pointer to multiple axes.

See also

[setTicker](#)

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

- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h↵
- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp↵

6.13 QCPAxisPainterPrivate Class Reference

Classes

- struct [CachedLabel](#)
- struct [TickLabelData](#)

Public Member Functions

- [QCPAxisPainterPrivate](#) ([QCustomPlot](#) *parentPlot)
- virtual void **draw** ([QCPPainter](#) *painter)
- virtual int **size** () const
- void **clearCache** ()
- [QRect](#) **axisSelectionBox** () const
- [QRect](#) **tickLabelsSelectionBox** () const
- [QRect](#) **labelSelectionBox** () const

Public Attributes

- [QCPAxis::AxisType](#) type
- [QPen](#) **basePen**
- [QCPLineEnding](#) **lowerEnding**
- [QCPLineEnding](#) **upperEnding**
- int **labelPadding**
- [QFont](#) **labelFont**
- [QColor](#) **labelColor**
- [QString](#) **label**
- int **tickLabelPadding**
- double **tickLabelRotation**
- [QCPAxis::LabelSide](#) **tickLabelSide**
- bool **substituteExponent**
- bool **numberMultiplyCross**
- int **tickLengthIn**
- int **tickLengthOut**
- int **subTickLengthIn**
- int **subTickLengthOut**
- [QPen](#) **tickPen**
- [QPen](#) **subTickPen**
- [QFont](#) **tickLabelFont**
- [QColor](#) **tickLabelColor**
- [QRect](#) **axisRect**
- [QRect](#) **viewportRect**
- double **offset**
- bool **abbreviateDecimalPowers**
- bool **reversedEndings**
- [QVector](#)< double > **subTickPositions**
- [QVector](#)< double > **tickPositions**
- [QVector](#)< [QString](#) > **tickLabels**

Protected Member Functions

- virtual QByteArray **generateLabelParameterHash** () const
- virtual void **placeTickLabel** (QCPPainter *painter, double position, int distanceToAxis, const QString &text, QSize *tickLabelsSize)
- virtual void **drawTickLabel** (QCPPainter *painter, double x, double y, const TickLabelData &labelData) const
- virtual TickLabelData **getTickLabelData** (const QFont &font, const QString &text) const
- virtual QPointF **getTickLabelDrawOffset** (const TickLabelData &labelData) const
- virtual void **getMaxTickLabelSize** (const QFont &font, const QString &text, QSize *tickLabelsSize) const

Protected Attributes

- **QCustomPlot * mParentPlot**
- **QByteArray mLabelParameterHash**
- **QCache< QString, [CachedLabel](#) > mLabelCache**
- **QRect mAxisSelectionBox**
- **QRect mTickLabelsSelectionBox**
- **QRect mLabelSelectionBox**

6.13.1 Constructor & Destructor Documentation

6.13.1.1 QCPAxisPainterPrivate()

```
QCPAxisPainterPrivate::QCPAxisPainterPrivate (
    QCustomPlot * parentPlot ) [explicit]
```

Constructs a `QCPAxisPainterPrivate` instance. Make sure to not create a new instance on every redraw, to utilize the caching mechanisms.

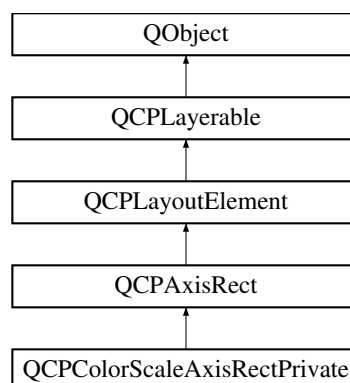
The documentation for this class was generated from the following files:

- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.←h
- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.←cpp

6.14 QCPAxisRect Class Reference

Holds multiple axes and arranges them in a rectangular shape.

Inheritance diagram for QCPAxisRect:



Public Member Functions

- [QCPAxisRect](#) ([QCustomPlot](#) *parentPlot, bool setupDefaultAxes=true)
- QPixmap **background** () const
- QBrush **backgroundBrush** () const
- bool **backgroundScaled** () const
- Qt::AspectRatioMode **backgroundScaledMode** () const
- Qt::Orientations **rangeDrag** () const
- Qt::Orientations **rangeZoom** () const
- [QCPAxis](#) * **rangeDragAxis** (Qt::Orientation orientation)
- [QCPAxis](#) * **rangeZoomAxis** (Qt::Orientation orientation)
- QList< [QCPAxis](#) * > **rangeDragAxes** (Qt::Orientation orientation)
- QList< [QCPAxis](#) * > **rangeZoomAxes** (Qt::Orientation orientation)
- double **rangeZoomFactor** (Qt::Orientation orientation)
- void **setBackground** (const QPixmap &pm)
- void **setBackground** (const QPixmap &pm, bool scaled, Qt::AspectRatioMode mode=Qt::KeepAspectRatio, Qt::ExpansionMode expansion=Qt::ByExpanding)
- void **setBackground** (const QBrush &brush)
- void **setBackgroundScaled** (bool scaled)
- void **setBackgroundScaledMode** (Qt::AspectRatioMode mode)
- void **setRangeDrag** (Qt::Orientations orientations)
- void **setRangeZoom** (Qt::Orientations orientations)
- void **setRangeDragAxes** ([QCPAxis](#) *horizontal, [QCPAxis](#) *vertical)
- void **setRangeDragAxes** (QList< [QCPAxis](#) * > axes)
- void **setRangeDragAxes** (QList< [QCPAxis](#) * > horizontal, QList< [QCPAxis](#) * > vertical)
- void **setRangeZoomAxes** ([QCPAxis](#) *horizontal, [QCPAxis](#) *vertical)
- void **setRangeZoomAxes** (QList< [QCPAxis](#) * > axes)
- void **setRangeZoomAxes** (QList< [QCPAxis](#) * > horizontal, QList< [QCPAxis](#) * > vertical)
- void **setRangeZoomFactor** (double horizontalFactor, double verticalFactor)
- void **setRangeZoomFactor** (double factor)
- int **axisCount** ([QCPAxis::AxisType](#) type) const
- [QCPAxis](#) * **axis** ([QCPAxis::AxisType](#) type, int index=0) const
- QList< [QCPAxis](#) * > **axes** ([QCPAxis::AxisTypes](#) types) const
- QList< [QCPAxis](#) * > **axes** () const
- [QCPAxis](#) * **addAxis** ([QCPAxis::AxisType](#) type, [QCPAxis](#) *axis=0)
- QList< [QCPAxis](#) * > **addAxes** ([QCPAxis::AxisTypes](#) types)
- bool **removeAxis** ([QCPAxis](#) *axis)
- [QCPLayoutInset](#) * **insetLayout** () const
- void **zoom** (const QRectF &pixelRect)
- void **zoom** (const QRectF &pixelRect, const QList< [QCPAxis](#) * > &affectedAxes)
- void **setupFullAxesBox** (bool connectRanges=false)
- QList< [QCPAbstractPlottable](#) * > **plottables** () const
- QList< [QCPGraph](#) * > **graphs** () const
- QList< [QCPAbstractItem](#) * > **items** () const
- int **left** () const
- int **right** () const
- int **top** () const
- int **bottom** () const
- int **width** () const
- int **height** () const
- QSize **size** () const
- QPoint **topLeft** () const
- QPoint **topRight** () const
- QPoint **bottomLeft** () const
- QPoint **bottomRight** () const
- QPoint **center** () const
- virtual void **update** ([UpdatePhase](#) phase) Q_DECL_OVERRIDE
- virtual QList< [QCPLayoutElement](#) * > **elements** (bool recursive) const Q_DECL_OVERRIDE

Protected Member Functions

- virtual void **applyDefaultAntialiasingHint** (QCPPainter *painter) const Q_DECL_OVERRIDE
- virtual void **draw** (QCPPainter *painter) Q_DECL_OVERRIDE
- virtual int **calculateAutoMargin** (QCP::MarginSide side) Q_DECL_OVERRIDE
- virtual void **layoutChanged** () Q_DECL_OVERRIDE
- virtual void **mousePressEvent** (QMouseEvent *event, const QVariant &details) Q_DECL_OVERRIDE
- virtual void **mouseMoveEvent** (QMouseEvent *event, const QPointF &startPos) Q_DECL_OVERRIDE
- virtual void **mouseReleaseEvent** (QMouseEvent *event, const QPointF &startPos) Q_DECL_OVERRIDE
- virtual void **wheelEvent** (QWheelEvent *event) Q_DECL_OVERRIDE
- void **drawBackground** (QCPPainter *painter)
- void **updateAxesOffset** (QCPAxis::AxisType type)

Protected Attributes

- QBrush **mBackgroundBrush**
- QPixmap **mBackgroundPixmap**
- QPixmap **mScaledBackgroundPixmap**
- bool **mBackgroundScaled**
- Qt::AspectRatioMode **mBackgroundScaledMode**
- QCPLayoutInset * **mInsetLayout**
- Qt::Orientations **mRangeDrag**
- Qt::Orientations **mRangeZoom**
- QList< QPointer< QCPAxis > > **mRangeDragHorzAxis**
- QList< QPointer< QCPAxis > > **mRangeDragVertAxis**
- QList< QPointer< QCPAxis > > **mRangeZoomHorzAxis**
- QList< QPointer< QCPAxis > > **mRangeZoomVertAxis**
- double **mRangeZoomFactorHorz**
- double **mRangeZoomFactorVert**
- QList< QCPRange > **mDragStartHorzRange**
- QList< QCPRange > **mDragStartVertRange**
- QCP::AntialiasedElements **mAADragBackup**
- QCP::AntialiasedElements **mNotAADragBackup**
- QPoint **mDragStart**
- bool **mDragging**
- QHash< QCPAxis::AxisType, QList< QCPAxis * > > **mAxes**

Friends

- class **QCustomPlot**

Additional Inherited Members

6.14.1 Detailed Description

Holds multiple axes and arranges them in a rectangular shape.

This class represents an axis rect, a rectangular area that is bounded on all sides with an arbitrary number of axes.

Initially `QCustomPlot` has one axis rect, accessible via `QCustomPlot::axisRect()`. However, the layout system allows to have multiple axis rects, e.g. arranged in a grid layout (`QCustomPlot::plotLayout`).

By default, `QCPAxisRect` comes with four axes, at bottom, top, left and right. They can be accessed via `axis` by providing the respective axis type (`QCPAxis::AxisType`) and index. If you need all axes in the axis rect, use `axes`. The top and right axes are set to be invisible initially (`QCPAxis::setVisible`). To add more axes to a side, use `addAxis` or `addAxes`. To remove an axis, use `removeAxis`.

The axis rect layerable itself only draws a background pixmap or color, if specified (`setBackground`). It is placed on the "background" layer initially (see `QCPLayer` for an explanation of the `QCustomPlot` layer system). The axes that are held by the axis rect can be placed on other layers, independently of the axis rect.

Every axis rect has a child layout of type `QCPLayoutInset`. It is accessible via `insetLayout` and can be used to have other layout elements (or even other layouts with multiple elements) hovering inside the axis rect.

If an axis rect is clicked and dragged, it processes this by moving certain axis ranges. The behaviour can be controlled with `setRangeDrag` and `setRangeDragAxes`. If the mouse wheel is scrolled while the cursor is on the axis rect, certain axes are scaled. This is controllable via `setRangeZoom`, `setRangeZoomAxes` and `setRangeZoomFactor`. These interactions are only enabled if `QCustomPlot::setInteractions` contains `QCP::iRangeDrag` and `QCP::iRangeZoom`.

Overview of the spacings and paddings that define the geometry of an axis. The dashed line on the far left indicates the viewport/widget border.

6.14.2 Constructor & Destructor Documentation

6.14.2.1 QCPAxisRect()

```
QCPAxisRect::QCPAxisRect (
    QCustomPlot * parentPlot,
    bool setupDefaultAxes = true ) [explicit]
```

Creates a `QCPAxisRect` instance and sets default values. An axis is added for each of the four sides, the top and right axes are set invisible initially.

6.14.3 Member Function Documentation

6.14.3.1 addAxes()

```
QList< QCPAxis * > QCPAxisRect::addAxes (
    QCPAxis::AxisTypes types )
```

Adds a new axis with [addAxis](#) to each axis rect side specified in *types*. This may be an `or`-combination of [QCPAxis::AxisType](#), so axes can be added to multiple sides at once.

Returns a list of the added axes.

See also

[addAxis](#), [setupFullAxesBox](#)

6.14.3.2 addAxis()

```
QCPAxis * QCPAxisRect::addAxis (
    QCPAxis::AxisType type,
    QCPAxis * axis = 0 )
```

Adds a new axis to the axis rect side specified with *type*, and returns it. If *axis* is 0, a new [QCPAxis](#) instance is created internally. [QCustomPlot](#) owns the returned axis, so if you want to remove an axis, use [removeAxis](#) instead of deleting it manually.

You may inject [QCPAxis](#) instances (or subclasses of [QCPAxis](#)) by setting *axis* to an axis that was previously created outside [QCustomPlot](#). It is important to note that [QCustomPlot](#) takes ownership of the axis, so you may not delete it afterwards. Further, the *axis* must have been created with this axis rect as parent and with the same axis type as specified in *type*. If this is not the case, a debug output is generated, the axis is not added, and the method returns 0.

This method can not be used to move *axis* between axis rects. The same *axis* instance must not be added multiple times to the same or different axis rects.

If an axis rect side already contains one or more axes, the lower and upper endings of the new axis ([QCPAxis::setLowerEnding](#), [QCPAxis::setUpperEnding](#)) are set to [QCPLineEnding::esHalfBar](#).

See also

[addAxes](#), [setupFullAxesBox](#)

6.14.3.3 axes() [1/2]

```
QList< QCPAxis * > QCPAxisRect::axes (
    QCPAxis::AxisTypes types ) const
```

Returns all axes on the axis rect sides specified with *types*.

types may be a single [QCPAxis::AxisType](#) or an `or`-combination, to get the axes of multiple sides.

See also

[axis](#)

6.14.3.4 axes() [2/2]

```
QList< QCPAxis * > QCPAxisRect::axes ( ) const
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Returns all axes of this axis rect.

6.14.3.5 axis()

```
QCPAxis * QCPAxisRect::axis (
    QCPAxis::AxisType type,
    int index = 0 ) const
```

Returns the axis with the given *index* on the axis rect side specified with *type*.

See also

[axisCount](#), [axes](#)

6.14.3.6 axisCount()

```
int QCPAxisRect::axisCount (
    QCPAxis::AxisType type ) const
```

Returns the number of axes on the axis rect side specified with *type*.

See also

[axis](#)

6.14.3.7 bottom()

```
int QCPAxisRect::bottom ( ) const [inline]
```

Returns the pixel position of the bottom border of this axis rect. Margins are not taken into account here, so the returned value is with respect to the inner [rect](#).

6.14.3.8 bottomLeft()

```
QPoint QCPAxisRect::bottomLeft ( ) const [inline]
```

Returns the bottom left corner of this axis rect in pixels. Margins are not taken into account here, so the returned value is with respect to the inner [rect](#).

6.14.3.9 bottomRight()

```
QPoint QCPAxisRect::bottomRight ( ) const [inline]
```

Returns the bottom right corner of this axis rect in pixels. Margins are not taken into account here, so the returned value is with respect to the inner [rect](#).

6.14.3.10 center()

```
QPoint QCPAxisRect::center ( ) const [inline]
```

Returns the center of this axis rect in pixels. Margins are not taken into account here, so the returned value is with respect to the inner [rect](#).

6.14.3.11 elements()

```
QList< QCPLayoutElement * > QCPAxisRect::elements (
    bool recursive ) const [virtual]
```

Returns a list of all child elements in this layout element. If *recursive* is true, all sub-child elements are included in the list, too.

Warning

There may be entries with value 0 in the returned list. (For example, [QCPLayoutGrid](#) may have empty cells which yield 0 at the respective index.)

Reimplemented from [QCPLayoutElement](#).

6.14.3.12 graphs()

```
QList< QCPGraph * > QCPAxisRect::graphs ( ) const
```

Returns a list of all the graphs that are associated with this axis rect.

A graph is considered associated with an axis rect if its key or value axis (or both) is in this axis rect.

See also

[plottables](#), [items](#)

6.14.3.13 height()

```
int QCPAxisRect::height ( ) const [inline]
```

Returns the pixel height of this axis rect. Margins are not taken into account here, so the returned value is with respect to the inner [rect](#).

6.14.3.14 insetLayout()

```
QCPLayoutInset * QCPAxisRect::insetLayout ( ) const [inline]
```

Returns the inset layout of this axis rect. It can be used to place other layout elements (or even layouts with multiple other elements) inside/on top of an axis rect.

See also

[QCPLayoutInset](#)

6.14.3.15 items()

```
QList< QCPAbstractItem * > QCPAxisRect::items ( ) const
```

Returns a list of all the items that are associated with this axis rect.

An item is considered associated with an axis rect if any of its positions has key or value axis set to an axis that is in this axis rect, or if any of its positions has [QCPLItemPosition::setAxisRect](#) set to the axis rect, or if the clip axis rect ([QCPAbstractItem::setClipAxisRect](#)) is set to this axis rect.

See also

[plottables](#), [graphs](#)

6.14.3.16 left()

```
int QCPAxisRect::left ( ) const [inline]
```

Returns the pixel position of the left border of this axis rect. Margins are not taken into account here, so the returned value is with respect to the inner [rect](#).

6.14.3.17 mouseMoveEvent()

```
void QCPAxisRect::mouseMoveEvent (
    QMouseEvent * event,
    const QPointF & startPos ) [protected], [virtual]
```

This event gets called when the user moves the mouse while holding a mouse button, after this layerable has become the mouse grabber by accepting the preceding [mousePressEvent](#).

The current pixel position of the cursor on the [QCustomPlot](#) widget is accessible via `event->pos()`. The parameter `startPos` indicates the position where the initial [mousePressEvent](#) occurred, that started the mouse interaction.

The default implementation does nothing.

See also

[mousePressEvent](#), [mouseReleaseEvent](#), [mouseDoubleClickEvent](#), [wheelEvent](#)

Reimplemented from [QCPLayerable](#).

6.14.3.18 mousePressEvent()

```
void QCPAxisRect::mousePressEvent (
    QMouseEvent * event,
    const QVariant & details ) [protected], [virtual]
```

This event gets called when the user presses a mouse button while the cursor is over the layerable. Whether a cursor is over the layerable is decided by a preceding call to [selectTest](#).

The current pixel position of the cursor on the [QCustomPlot](#) widget is accessible via `event->pos()`. The parameter *details* contains layerable-specific details about the hit, which were generated in the previous call to [selectTest](#). For example, One-dimensional plottables like [QCPGraph](#) or [QCPBars](#) convey the clicked data point in the *details* parameter, as [QCPDataSelection](#) packed as `QVariant`. Multi-part objects convey the specific `SelectablePart` that was hit (e.g. [QCPAxis::SelectablePart](#) in the case of axes).

[QCustomPlot](#) uses an event propagation system that works the same as Qt's system. If your layerable doesn't reimplement the [mousePressEvent](#) or explicitly calls `event->ignore()` in its reimplementation, the event will be propagated to the next layerable in the stacking order.

Once a layerable has accepted the [mousePressEvent](#), it is considered the mouse grabber and will receive all following calls to [mouseMoveEvent](#) or [mouseReleaseEvent](#) for this mouse interaction (a "mouse interaction" in this context ends with the release).

The default implementation does nothing except explicitly ignoring the event with `event->ignore()`.

See also

[mouseMoveEvent](#), [mouseReleaseEvent](#), [mouseDoubleClickEvent](#), [wheelEvent](#)

Reimplemented from [QCPLayerable](#).

6.14.3.19 mouseReleaseEvent()

```
void QCPAxisRect::mouseReleaseEvent (
    QMouseEvent * event,
    const QPointF & startPos ) [protected], [virtual]
```

This event gets called when the user releases the mouse button, after this layerable has become the mouse grabber by accepting the preceding [mousePressEvent](#).

The current pixel position of the cursor on the [QCustomPlot](#) widget is accessible via `event->pos()`. The parameter *startPos* indicates the position where the initial [mousePressEvent](#) occurred, that started the mouse interaction.

The default implementation does nothing.

See also

[mousePressEvent](#), [mouseMoveEvent](#), [mouseDoubleClickEvent](#), [wheelEvent](#)

Reimplemented from [QCPLayerable](#).

6.14.3.20 plottables()

```
QList< QCPAbstractPlottable * > QCPAxisRect::plottables ( ) const
```

Returns a list of all the plottables that are associated with this axis rect.

A plottable is considered associated with an axis rect if its key or value axis (or both) is in this axis rect.

See also

[graphs](#), [items](#)

6.14.3.21 rangeDragAxes()

```
QList< QCPAxis * > QCPAxisRect::rangeDragAxes (
    Qt::Orientation orientation )
```

Returns all range drag axes of the *orientation* provided.

See also

[rangeZoomAxis](#), [setRangeZoomAxes](#)

6.14.3.22 rangeDragAxis()

```
QCPAxis * QCPAxisRect::rangeDragAxis (
    Qt::Orientation orientation )
```

Returns the range drag axis of the *orientation* provided. If multiple axes were set, returns the first one (use [rangeDragAxes](#) to retrieve a list with all set axes).

See also

[setRangeDragAxes](#)

6.14.3.23 rangeZoomAxes()

```
QList< QCPAxis * > QCPAxisRect::rangeZoomAxes (
    Qt::Orientation orientation )
```

Returns all range zoom axes of the *orientation* provided.

See also

[rangeDragAxis](#), [setRangeDragAxes](#)

6.14.3.24 rangeZoomAxis()

```
QCPAxis * QCPAxisRect::rangeZoomAxis (
    Qt::Orientation orientation )
```

Returns the range zoom axis of the *orientation* provided. If multiple axes were set, returns the first one (use [rangeZoomAxes](#) to retrieve a list with all set axes).

See also

[setRangeZoomAxes](#)

6.14.3.25 rangeZoomFactor()

```
double QCPAxisRect::rangeZoomFactor (
    Qt::Orientation orientation )
```

Returns the range zoom factor of the *orientation* provided.

See also

[setRangeZoomFactor](#)

6.14.3.26 removeAxis()

```
bool QCPAxisRect::removeAxis (
    QCPAxis * axis )
```

Removes the specified *axis* from the axis rect and deletes it.

Returns true on success, i.e. if *axis* was a valid axis in this axis rect.

See also

[addAxis](#)

6.14.3.27 right()

```
int QCPAxisRect::right ( ) const [inline]
```

Returns the pixel position of the right border of this axis rect. Margins are not taken into account here, so the returned value is with respect to the inner [rect](#).

6.14.3.28 setBackground() [1/3]

```
void QCPAxisRect::setBackground (
    const QPixmap & pm )
```

Sets *pm* as the axis background pixmap. The axis background pixmap will be drawn inside the axis rect. Since axis rects place themselves on the "background" layer by default, the axis rect backgrounds are usually drawn below everything else.

For cases where the provided pixmap doesn't have the same size as the axis rect, scaling can be enabled with [setBackgroundScaled](#) and the scaling mode (i.e. whether and how the aspect ratio is preserved) can be set with [setBackgroundScaledMode](#). To set all these options in one call, consider using the overloaded version of this function.

Below the pixmap, the axis rect may be optionally filled with a brush, if specified with [setBackground\(const QBrush &brush\)](#).

See also

[setBackgroundScaled](#), [setBackgroundScaledMode](#), [setBackground\(const QBrush &brush\)](#)

6.14.3.29 setBackground() [2/3]

```
void QCPAxisRect::setBackground (
    const QPixmap & pm,
    bool scaled,
    Qt::AspectRatioMode mode = Qt::KeepAspectRatioByExpanding )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Allows setting the background pixmap of the axis rect, whether it shall be scaled and how it shall be scaled in one call.

See also

[setBackground\(const QPixmap &pm\)](#), [setBackgroundScaled](#), [setBackgroundScaledMode](#)

6.14.3.30 setBackground() [3/3]

```
void QCPAxisRect::setBackground (
    const QBrush & brush )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Sets *brush* as the background brush. The axis rect background will be filled with this brush. Since axis rects place themselves on the "background" layer by default, the axis rect backgrounds are usually drawn below everything else.

The brush will be drawn before (under) any background pixmap, which may be specified with [setBackground\(const QPixmap &pm\)](#).

To disable drawing of a background brush, set *brush* to `Qt::NoBrush`.

See also

[setBackground\(const QPixmap &pm\)](#)

6.14.3.31 setBackgroundScaled()

```
void QCPAxisRect::setBackgroundScaled (
    bool scaled )
```

Sets whether the axis background pixmap shall be scaled to fit the axis rect or not. If *scaled* is set to true, you may control whether and how the aspect ratio of the original pixmap is preserved with [setBackgroundScaledMode](#).

Note that the scaled version of the original pixmap is buffered, so there is no performance penalty on replots. (Except when the axis rect dimensions are changed continuously.)

See also

[setBackground](#), [setBackgroundScaledMode](#)

6.14.3.32 setBackgroundScaledMode()

```
void QCPAxisRect::setBackgroundScaledMode (
    Qt::AspectRatioMode mode )
```

If scaling of the axis background pixmap is enabled ([setBackgroundScaled](#)), use this function to define whether and how the aspect ratio of the original pixmap passed to [setBackground](#) is preserved.

See also

[setBackground](#), [setBackgroundScaled](#)

6.14.3.33 setRangeDrag()

```
void QCPAxisRect::setRangeDrag (
    Qt::Orientations orientations )
```

Sets which axis orientation may be range dragged by the user with mouse interaction. What orientation corresponds to which specific axis can be set with [setRangeDragAxes\(QCPAxis *horizontal, QCPAxis *vertical\)](#). By default, the horizontal axis is the bottom axis (xAxis) and the vertical axis is the left axis (yAxis).

To disable range dragging entirely, pass 0 as *orientations* or remove [QCP::iRangeDrag](#) from [QCustomPlot::setInteractions](#). To enable range dragging for both directions, pass `Qt::Horizontal | Qt::Vertical` as *orientations*.

In addition to setting *orientations* to a non-zero value, make sure [QCustomPlot::setInteractions](#) contains [QCP::iRangeDrag](#) to enable the range dragging interaction.

See also

[setRangeZoom](#), [setRangeDragAxes](#), [QCustomPlot::setNoAntialiasingOnDrag](#)

6.14.3.34 setRangeDragAxes() [1/3]

```
void QCPAxisRect::setRangeDragAxes (
    QCPAxis * horizontal,
    QCPAxis * vertical )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Sets the axes whose range will be dragged when [setRangeDrag](#) enables mouse range dragging on the [QCustomPlot](#) widget. Pass 0 if no axis shall be dragged in the respective orientation.

Use the overload taking a list of axes, if multiple axes (more than one per orientation) shall react to dragging interactions.

See also

[setRangeZoomAxes](#)

6.14.3.35 setRangeDragAxes() [2/3]

```
void QCPAxisRect::setRangeDragAxes (
    QList< QCPAxis *> axes )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This method allows to set up multiple axes to react to horizontal and vertical dragging. The drag orientation that the respective axis will react to is deduced from its orientation ([QCPAxis::orientation](#)).

In the unusual case that you wish to e.g. drag a vertically oriented axis with a horizontal drag motion, use the overload taking two separate lists for horizontal and vertical dragging.

6.14.3.36 setRangeDragAxes() [3/3]

```
void QCPAxisRect::setRangeDragAxes (
    QList< QCPAxis *> horizontal,
    QList< QCPAxis *> vertical )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This method allows to set multiple axes up to react to horizontal and vertical dragging, and define specifically which axis reacts to which drag orientation (irrespective of the axis orientation).

6.14.3.37 `setRangeZoom()`

```
void QCPAxisRect::setRangeZoom (
    Qt::Orientations orientations )
```

Sets which axis orientation may be zoomed by the user with the mouse wheel. What orientation corresponds to which specific axis can be set with [setRangeZoomAxes\(QCPAxis *horizontal, QCPAxis *vertical\)](#). By default, the horizontal axis is the bottom axis (xAxis) and the vertical axis is the left axis (yAxis).

To disable range zooming entirely, pass 0 as *orientations* or remove [QCP::iRangeZoom](#) from [QCustomPlot::setInteractions](#). To enable range zooming for both directions, pass `Qt::Horizontal | Qt::Vertical` as *orientations*.

In addition to setting *orientations* to a non-zero value, make sure [QCustomPlot::setInteractions](#) contains [QCP::iRangeZoom](#) to enable the range zooming interaction.

See also

[setRangeZoomFactor](#), [setRangeZoomAxes](#), [setRangeDrag](#)

6.14.3.38 `setRangeZoomAxes()` [1/3]

```
void QCPAxisRect::setRangeZoomAxes (
    QCPAxis * horizontal,
    QCPAxis * vertical )
```

Sets the axes whose range will be zoomed when [setRangeZoom](#) enables mouse wheel zooming on the [QCustomPlot](#) widget. Pass 0 if no axis shall be zoomed in the respective orientation.

The two axes can be zoomed with different strengths, when different factors are passed to [setRangeZoomFactor\(double horizontalFactor, double verticalFactor\)](#).

Use the overload taking a list of axes, if multiple axes (more than one per orientation) shall react to zooming interactions.

See also

[setRangeDragAxes](#)

6.14.3.39 `setRangeZoomAxes()` [2/3]

```
void QCPAxisRect::setRangeZoomAxes (
    QList< QCPAxis *> axes )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This method allows to set up multiple axes to react to horizontal and vertical range zooming. The zoom orientation that the respective axis will react to is deduced from its orientation ([QCPAxis::orientation](#)).

In the unusual case that you wish to e.g. zoom a vertically oriented axis with a horizontal zoom interaction, use the overload taking two separate lists for horizontal and vertical zooming.

6.14.3.40 `setRangeZoomAxes()` [3/3]

```
void QCPAxisRect::setRangeZoomAxes (
    QList< QCPAxis *> horizontal,
    QList< QCPAxis *> vertical )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This method allows to set multiple axes up to react to horizontal and vertical zooming, and define specifically which axis reacts to which zoom orientation (irrespective of the axis orientation).

6.14.3.41 `setRangeZoomFactor()` [1/2]

```
void QCPAxisRect::setRangeZoomFactor (
    double horizontalFactor,
    double verticalFactor )
```

Sets how strong one rotation step of the mouse wheel zooms, when range zoom was activated with [setRangeZoom](#). The two parameters *horizontalFactor* and *verticalFactor* provide a way to let the horizontal axis zoom at different rates than the vertical axis. Which axis is horizontal and which is vertical, can be set with [setRangeZoomAxes](#).

When the zoom factor is greater than one, scrolling the mouse wheel backwards (towards the user) will zoom in (make the currently visible range smaller). For zoom factors smaller than one, the same scrolling direction will zoom out.

6.14.3.42 `setRangeZoomFactor()` [2/2]

```
void QCPAxisRect::setRangeZoomFactor (
    double factor )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Sets both the horizontal and vertical zoom *factor*.

6.14.3.43 `setupFullAxesBox()`

```
void QCPAxisRect::setupFullAxesBox (
    bool connectRanges = false )
```

Convenience function to create an axis on each side that doesn't have any axes yet and set their visibility to true. Further, the top/right axes are assigned the following properties of the bottom/left axes:

- range ([QCPAxis::setRange](#))
- range reversed ([QCPAxis::setRangeReversed](#))
- scale type ([QCPAxis::setScaleType](#))
- tick visibility ([QCPAxis::setTicks](#))
- number format ([QCPAxis::setNumberFormat](#))
- number precision ([QCPAxis::setNumberPrecision](#))
- tick count of ticker ([QCPAxisTicker::setTickCount](#))
- tick origin of ticker ([QCPAxisTicker::setTickOrigin](#))

Tick label visibility ([QCPAxis::setTickLabels](#)) of the right and top axes are set to false.

If *connectRanges* is true, the [rangeChanged](#) signals of the bottom and left axes are connected to the [QCPAxis::setRange](#) slots of the top and right axes.

6.14.3.44 size()

```
QSize QCPAxisRect::size ( ) const [inline]
```

Returns the pixel size of this axis rect. Margins are not taken into account here, so the returned value is with respect to the inner [rect](#).

6.14.3.45 top()

```
int QCPAxisRect::top ( ) const [inline]
```

Returns the pixel position of the top border of this axis rect. Margins are not taken into account here, so the returned value is with respect to the inner [rect](#).

6.14.3.46 topLeft()

```
QPoint QCPAxisRect::topLeft ( ) const [inline]
```

Returns the top left corner of this axis rect in pixels. Margins are not taken into account here, so the returned value is with respect to the inner [rect](#).

6.14.3.47 topRight()

```
QPoint QCPAxisRect::topRight ( ) const [inline]
```

Returns the top right corner of this axis rect in pixels. Margins are not taken into account here, so the returned value is with respect to the inner [rect](#).

6.14.3.48 update()

```
void QCPAxisRect::update (
    UpdatePhase phase ) [virtual]
```

This method is called automatically upon replot and doesn't need to be called by users of [QCPAxisRect](#).

Calls the base class implementation to update the margins (see [QCPLayoutElement::update](#)), and finally passes the [rect](#) to the inset layout ([insetLayout](#)) and calls its `QCPInsetLayout::update` function.

Reimplemented from [QCPLayoutElement](#).

6.14.3.49 wheelEvent()

```
void QCPAxisRect::wheelEvent (
    QWheelEvent * event ) [protected], [virtual]
```

This event gets called when the user turns the mouse scroll wheel while the cursor is over the layerable. Whether a cursor is over the layerable is decided by a preceding call to [selectTest](#).

The current pixel position of the cursor on the [QCustomPlot](#) widget is accessible via `event->pos()`.

The `event->delta()` indicates how far the mouse wheel was turned, which is usually +/- 120 for single rotation steps. However, if the mouse wheel is turned rapidly, multiple steps may accumulate to one event, making `event->delta()` larger. On the other hand, if the wheel has very smooth steps or none at all, the delta may be smaller.

The default implementation does nothing.

See also

[mousePressEvent](#), [mouseMoveEvent](#), [mouseReleaseEvent](#), [mouseDoubleClickEvent](#)

Reimplemented from [QCPLayerable](#).

6.14.3.50 width()

```
int QCPAxisRect::width ( ) const [inline]
```

Returns the pixel width of this axis rect. Margins are not taken into account here, so the returned value is with respect to the inner [rect](#).

6.14.3.51 zoom() [1/2]

```
void QCPAxisRect::zoom (
    const QRectF & pixelRect )
```

Zooms in (or out) to the passed rectangular region *pixelRect*, given in pixel coordinates.

All axes of this axis rect will have their range zoomed accordingly. If you only wish to zoom specific axes, use the overloaded version of this method.

See also

[QCustomPlot::setSelectionRectMode](#)

6.14.3.52 zoom() [2/2]

```
void QCPAxisRect::zoom (
    const QRectF & pixelRect,
    const QList< QCPAxis *> & affectedAxes )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Zooms in (or out) to the passed rectangular region *pixelRect*, given in pixel coordinates.

Only the axes passed in *affectedAxes* will have their ranges zoomed accordingly.

See also

[QCustomPlot::setSelectionRectMode](#)

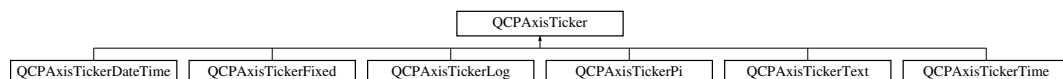
The documentation for this class was generated from the following files:

- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h↵
- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp↵

6.15 QCPAxisTicker Class Reference

The base class tick generator used by [QCPAxis](#) to create tick positions and tick labels.

Inheritance diagram for QCPAxisTicker:



Public Types

- enum [TickStepStrategy](#) { [tssReadability](#), [tssMeetTickCount](#) }

Public Member Functions

- [QCPAxisTicker](#) ()
- [TickStepStrategy](#) **tickStepStrategy** () const
- int **tickCount** () const
- double **tickOrigin** () const
- void **setTickStepStrategy** ([TickStepStrategy](#) strategy)
- void **setTickCount** (int count)
- void **setTickOrigin** (double origin)
- virtual void **generate** (const [QCPRange](#) &range, const QLocale &locale, QChar formatChar, int precision, QVector< double > &ticks, QVector< double > *subTicks, QVector< QString > *tickLabels)

Protected Member Functions

- virtual double **getTickStep** (const [QCPRange](#) &range)
- virtual int **getSubTickCount** (double tickStep)
- virtual QString **getTickLabel** (double tick, const QLocale &locale, QChar formatChar, int precision)
- virtual QVector< double > **createTickVector** (double tickStep, const [QCPRange](#) &range)
- virtual QVector< double > **createSubTickVector** (int subTickCount, const QVector< double > &ticks)
- virtual QVector< QString > **createLabelVector** (const QVector< double > &ticks, const QLocale &locale, QChar formatChar, int precision)
- void **trimTicks** (const [QCPRange](#) &range, QVector< double > &ticks, bool keepOneOutlier) const
- double **pickClosest** (double target, const QVector< double > &candidates) const
- double **getMantissa** (double input, double *magnitude=0) const
- double **cleanMantissa** (double input) const

Protected Attributes

- [TickStepStrategy](#) **mTickStepStrategy**
- int **mTickCount**
- double **mTickOrigin**

6.15.1 Detailed Description

The base class tick generator used by [QCPAxis](#) to create tick positions and tick labels.

Each [QCPAxis](#) has an internal [QCPAxisTicker](#) (or a subclass) in order to generate tick positions and tick labels for the current axis range. The ticker of an axis can be set via [QCPAxis::setTicker](#). Since that method takes a [QSharedPointer<QCPAxisTicker>](#), multiple axes can share the same ticker instance.

This base class generates normal tick coordinates and numeric labels for linear axes. It picks a reasonable tick step (the separation between ticks) which results in readable tick labels. The number of ticks that should be approximately generated can be set via [setTickCount](#). Depending on the current tick step strategy ([setTickStepStrategy](#)), the algorithm either sacrifices readability to better match the specified tick count ([QCPAxisTicker::tssMeetTickCount](#)) or relaxes the tick count in favor of better tick steps ([QCPAxisTicker::tssReadability](#)), which is the default.

The following more specialized axis ticker subclasses are available, see details in the respective class documentation:

QCPAxisTickerFixed	
QCPAxisTickerLog	
QCPAxisTickerPi	
QCPAxisTickerText	
QCPAxisTickerDateTime	
QCPAxisTickerTime	

6.15.2 Creating own axis tickers

Creating own axis tickers can be achieved very easily by subclassing [QCPAxisTicker](#) and reimplementing some or all of the available virtual methods.

In the simplest case you might wish to just generate different tick steps than the other tickers, so you only reimplement the method `getTickStep`. If you additionally want control over the string that will be shown as tick label, reimplement `getTickLabel`.

If you wish to have complete control, you can generate the tick vectors and tick label vectors yourself by reimplementing `createTickVector` and `createLabelVector`. The default implementations use the previously mentioned virtual methods `getTickStep` and `getTickLabel`, but your reimplementations don't necessarily need to do so. For example in the case of unequal tick steps, the method `getTickStep` loses its usefulness and can be ignored.

The sub tick count between major ticks can be controlled with `getSubTickCount`. Full sub tick placement control is obtained by reimplementing `createSubTickVector`.

See the documentation of all these virtual methods in [QCPAxisTicker](#) for detailed information about the parameters and expected return values.

6.15.3 Member Enumeration Documentation

6.15.3.1 TickStepStrategy

```
enum QCPAxisTicker::TickStepStrategy
```

Defines the strategies that the axis ticker may follow when choosing the size of the tick step.

See also

[setTickStepStrategy](#)

Enumerator

tssReadability	A nicely readable tick step is prioritized over matching the requested number of ticks (see setTickCount)
tssMeetTickCount	Less readable tick steps are allowed which in turn facilitates getting closer to the requested tick count.

6.15.4 Constructor & Destructor Documentation

6.15.4.1 QCPAxisTicker()

```
QCPAxisTicker::QCPAxisTicker ( )
```

Constructs the ticker and sets reasonable default values. Axis tickers are commonly created managed by a [QSharedPointer](#), which then can be passed to [QCPAxis::setTicker](#).

6.15.5 Member Function Documentation

6.15.5.1 generate()

```
void QCPAxisTicker::generate (
    const QCPRange & range,
    const QLocale & locale,
    QChar formatChar,
    int precision,
    QVector< double > & ticks,
    QVector< double > * subTicks,
    QVector< QString > * tickLabels ) [virtual]
```

This is the method called by [QCPAxis](#) in order to actually generate tick coordinates (*ticks*), tick label strings (*tickLabels*) and sub tick coordinates (*subTicks*).

The ticks are generated for the specified *range*. The generated labels typically follow the specified *locale*, *formatChar* and number *precision*, however this might be different (or even irrelevant) for certain [QCPAxisTicker](#) subclasses.

The output parameter *ticks* is filled with the generated tick positions in axis coordinates. The output parameters *subTicks* and *tickLabels* are optional (set them to 0 if not needed) and are respectively filled with sub tick coordinates, and tick label strings belonging to *ticks* by index.

6.15.5.2 setTickCount()

```
void QCPAxisTicker::setTickCount (
    int count )
```

Sets how many ticks this ticker shall aim to generate across the axis range. Note that *count* is not guaranteed to be matched exactly, as generating readable tick intervals may conflict with the requested number of ticks.

Whether the readability has priority over meeting the requested *count* can be specified with [setTickStepStrategy](#).

6.15.5.3 setTickOrigin()

```
void QCPAxisTicker::setTickOrigin (
    double origin )
```

Sets the mathematical coordinate (or "offset") of the zeroth tick. This tick coordinate is just a concept and doesn't need to be inside the currently visible axis range.

By default *origin* is zero, which for example yields ticks {-5, 0, 5, 10, 15,...} when the tick step is five. If *origin* is now set to 1 instead, the correspondingly generated ticks would be {-4, 1, 6, 11, 16,...}.

6.15.5.4 setTickStepStrategy()

```
void QCPAxisTicker::setTickStepStrategy (
    QCPAxisTicker::TickStepStrategy strategy )
```

Sets which strategy the axis ticker follows when choosing the size of the tick step. For the available strategies, see [TickStepStrategy](#).

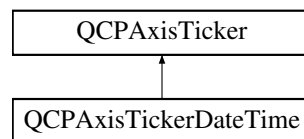
The documentation for this class was generated from the following files:

- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h↵
- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp↵

6.16 QCPAxisTickerDateTime Class Reference

Specialized axis ticker for calendar dates and times as axis ticks.

Inheritance diagram for QCPAxisTickerDateTime:



Public Member Functions

- [QCPAxisTickerDateTime](#) ()
- QString **dateTimeFormat** () const
- Qt::TimeSpec **dateTimeSpec** () const
- void [setDateTimeFormat](#) (const QString &format)
- void [setDateTimeSpec](#) (Qt::TimeSpec spec)
- void [setTickOrigin](#) (double origin)
- void [setTickOrigin](#) (const QDateTime &origin)

Static Public Member Functions

- static QDateTime [keyToDateTime](#) (double key)
- static double [dateTimeToKey](#) (const QDateTime dateTime)
- static double [dateTimeToKey](#) (const QDate date)

Protected Types

- enum **DateStrategy** { **dsNone**, **dsUniformTimeInDay**, **dsUniformDayInMonth** }

Protected Member Functions

- virtual double **getTickStep** (const [QCPRange](#) &range) Q_DECL_OVERRIDE
- virtual int **getSubTickCount** (double tickStep) Q_DECL_OVERRIDE
- virtual QString **getTickLabel** (double tick, const QLocale &locale, QChar formatChar, int precision) Q_DECL_OVERRIDE
- virtual QVector< double > **createTickVector** (double tickStep, const [QCPRange](#) &range) Q_DECL_OVERRIDE

Protected Attributes

- QString **mDateTimeFormat**
- Qt::TimeSpec **mDateTimeSpec**
- enum QCPAxisTickerDateTime::DateStrategy **mDateStrategy**

Additional Inherited Members

6.16.1 Detailed Description

Specialized axis ticker for calendar dates and times as axis ticks.

This [QCPAxisTicker](#) subclass generates ticks that correspond to real calendar dates and times. The plot axis coordinate is interpreted as Unix Time, so seconds since Epoch (January 1, 1970, 00:00 UTC). This is also used for example by `QDateTime` in the `toTime_t()` / `setTime_t()` methods with a precision of one second. Since Qt 4.7, millisecond accuracy can be obtained from `QDateTime` by using `QDateTime::fromMSecsSinceEpoch()` / `1000.0`. The static methods [dateTimeToKey](#) and [keyToDateTime](#) conveniently perform this conversion achieving a precision of one millisecond on all Qt versions.

The format of the date/time display in the tick labels is controlled with [setDateTimeFormat](#). If a different time spec (time zone) shall be used, see [setDateTimeSpec](#).

This ticker produces unequal tick spacing in order to provide intuitive date and time-of-day ticks. For example, if the axis range spans a few years such that there is one tick per year, ticks will be positioned on 1. January of every year. This is intuitive but, due to leap years, will result in slightly unequal tick intervals (visually unnoticeable). The same can be seen in the image above: even though the number of days varies month by month, this ticker generates ticks on the same day of each month.

If you would like to change the date/time that is used as a (mathematical) starting date for the ticks, use the [setTickOrigin\(const QDateTime &origin\)](#) method overload, which takes a `QDateTime`. If you pass 15. July, 9:45 to this method, the yearly ticks will end up on 15. July at 9:45 of every year.

The ticker can be created and assigned to an axis like this:

Note

If you rather wish to display relative times in terms of days, hours, minutes, seconds and milliseconds, and are not interested in the intricacies of real calendar dates with months and (leap) years, have a look at [QCPAxisTickerTime](#) instead.

6.16.2 Constructor & Destructor Documentation

6.16.2.1 QCPAxisTickerDateTime()

```
QCPAxisTickerDateTime::QCPAxisTickerDateTime ( )
```

Constructs the ticker and sets reasonable default values. Axis tickers are commonly created managed by a [QSharedPointer](#), which then can be passed to [QCPAxis::setTicker](#).

6.16.3 Member Function Documentation

6.16.3.1 dateTimeToKey() [1/2]

```
double QCPAxisTickerDateTime::dateTimeToKey (
    const QDateTime dateTime ) [static]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

A convenience method which turns a QDateTime object into a double value that corresponds to seconds since Epoch (1. Jan 1970, 00:00 UTC). This is the format used as axis coordinates by [QCPAxisTickerDateTime](#).

The accuracy achieved by this method is one millisecond, irrespective of the used Qt version (it works around the lack of a QDateTime::toMsecsSinceEpoch in Qt 4.6)

See also

[keyToDateTime](#)

6.16.3.2 dateTimeToKey() [2/2]

```
double QCPAxisTickerDateTime::dateTimeToKey (
    const QDate date ) [static]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

A convenience method which turns a QDate object into a double value that corresponds to seconds since Epoch (1. Jan 1970, 00:00 UTC). This is the format used as axis coordinates by [QCPAxisTickerDateTime](#).

See also

[keyToDateTime](#)

6.16.3.3 `keyToDateTime()`

```
QDateTime QCPAxisTickerDateTime::keyToDateTime (
    double key ) [static]
```

A convenience method which turns *key* (in seconds since Epoch 1. Jan 1970, 00:00 UTC) into a `QDateTime` object. This can be used to turn axis coordinates to actual `QDateTimes`.

The accuracy achieved by this method is one millisecond, irrespective of the used Qt version (it works around the lack of a `QDateTime::fromMsecsSinceEpoch` in Qt 4.6)

See also

[dateTimeToKey](#)

6.16.3.4 `setDateTimeFormat()`

```
void QCPAxisTickerDateTime::setDateTimeFormat (
    const QString & format )
```

Sets the format in which dates and times are displayed as tick labels. For details about the *format* string, see the documentation of `QDateTime::toString()`.

Newlines can be inserted with `"\n"`.

See also

[setDateTimeSpec](#)

6.16.3.5 `setDateTimeSpec()`

```
void QCPAxisTickerDateTime::setDateTimeSpec (
    Qt::TimeSpec spec )
```

Sets the time spec that is used for creating the tick labels from corresponding dates/times.

The default value of `QDateTime` objects (and also [QCPAxisTickerDateTime](#)) is `Qt::LocalTime`. However, if the date time values passed to [QCustomPlot](#) (e.g. in the form of axis ranges or keys of a plottable) are given in the UTC spec, set *spec* to `Qt::UTC` to get the correct axis labels.

See also

[setDateTimeFormat](#)

6.16.3.6 `setTickOrigin()` [1/2]

```
void QCPAxisTickerDateTime::setTickOrigin (
    double origin )
```

Sets the tick origin (see [QCPAxisTicker::setTickOrigin](#)) in seconds since Epoch (1. Jan 1970, 00:00 UTC). For the date time ticker it might be more intuitive to use the overload which directly takes a QDateTime, see [setTickOrigin\(const QDateTime &origin\)](#).

This is useful to define the month/day/time recurring at greater tick interval steps. For example, If you pass 15. July, 9:45 to this method and the tick interval happens to be one tick per year, the ticks will end up on 15. July at 9:45 of every year.

6.16.3.7 `setTickOrigin()` [2/2]

```
void QCPAxisTickerDateTime::setTickOrigin (
    const QDateTime & origin )
```

Sets the tick origin (see [QCPAxisTicker::setTickOrigin](#)) as a QDateTime *origin*.

This is useful to define the month/day/time recurring at greater tick interval steps. For example, If you pass 15. July, 9:45 to this method and the tick interval happens to be one tick per year, the ticks will end up on 15. July at 9:45 of every year.

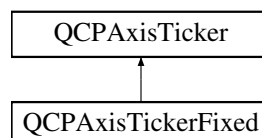
The documentation for this class was generated from the following files:

- [/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h](#)
- [/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp](#)

6.17 QCPAxisTickerFixed Class Reference

Specialized axis ticker with a fixed tick step.

Inheritance diagram for QCPAxisTickerFixed:



Public Types

- enum [ScaleStrategy](#) { [ssNone](#), [ssMultiples](#), [ssPowers](#) }

Public Member Functions

- [QCPAxisTickerFixed](#) ()
- double **tickStep** () const
- [ScaleStrategy](#) **scaleStrategy** () const
- void [setTickStep](#) (double step)
- void [setScaleStrategy](#) ([ScaleStrategy](#) strategy)

Protected Member Functions

- virtual double **getTickStep** (const [QCPRange](#) &range) Q_DECL_OVERRIDE

Protected Attributes

- double **mTickStep**
- [ScaleStrategy](#) **mScaleStrategy**

6.17.1 Detailed Description

Specialized axis ticker with a fixed tick step.

This [QCPAxisTicker](#) subclass generates ticks with a fixed tick step set with [setTickStep](#). It is also possible to allow integer multiples and integer powers of the specified tick step with [setScaleStrategy](#).

A typical application of this ticker is to make an axis only display integers, by setting the tick step of the ticker to 1.0 and the scale strategy to [ssMultiples](#).

Another case is when a certain number has a special meaning and axis ticks should only appear at multiples of that value. In this case you might also want to consider [QCPAxisTickerPi](#) because despite the name it is not limited to only pi symbols/values.

The ticker can be created and assigned to an axis like this:

6.17.2 Member Enumeration Documentation

6.17.2.1 ScaleStrategy

```
enum QCPAxisTickerFixed::ScaleStrategy
```

Defines how the axis ticker may modify the specified tick step ([setTickStep](#)) in order to control the number of ticks in the axis range.

See also

[setScaleStrategy](#)

Enumerator

ssNone	Modifications are not allowed, the specified tick step is absolutely fixed. This might cause a high tick density and overlapping labels if the axis range is zoomed out.
ssMultiples	An integer multiple of the specified tick step is allowed. The used factor follows the base class properties of setTickStepStrategy and setTickCount .
ssPowers	An integer power of the specified tick step is allowed.

6.17.3 Constructor & Destructor Documentation

6.17.3.1 QCPAxisTickerFixed()

```
QCPAxisTickerFixed::QCPAxisTickerFixed ( )
```

Constructs the ticker and sets reasonable default values. Axis tickers are commonly created managed by a [QSharedPointer](#), which then can be passed to [QCPAxis::setTicker](#).

6.17.4 Member Function Documentation

6.17.4.1 setScaleStrategy()

```
void QCPAxisTickerFixed::setScaleStrategy (
    QCPAxisTickerFixed::ScaleStrategy strategy )
```

Sets whether the specified tick step ([setTickStep](#)) is absolutely fixed or whether modifications may be applied to it before calculating the finally used tick step, such as permitting multiples or powers. See [ScaleStrategy](#) for details.

The default strategy is [ssNone](#), which means the tick step is absolutely fixed.

6.17.4.2 setTickStep()

```
void QCPAxisTickerFixed::setTickStep (
    double step )
```

Sets the fixed tick interval to *step*.

The axis ticker will only use this tick step when generating axis ticks. This might cause a very high tick density and overlapping labels if the axis range is zoomed out. Using [setScaleStrategy](#) it is possible to relax the fixed step and also allow multiples or powers of *step*. This will enable the ticker to reduce the number of ticks to a reasonable amount (see [setTickCount](#)).

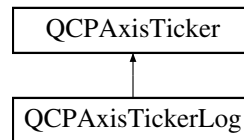
The documentation for this class was generated from the following files:

- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h
- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp

6.18 QCPAxisTickerLog Class Reference

Specialized axis ticker suited for logarithmic axes.

Inheritance diagram for QCPAxisTickerLog:



Public Member Functions

- [QCPAxisTickerLog](#) ()
- double **logBase** () const
- int **subTickCount** () const
- void [setLogBase](#) (double base)
- void [setSubTickCount](#) (int subTicks)

Protected Member Functions

- virtual double **getTickStep** (const [QCPRange](#) &range) Q_DECL_OVERRIDE
- virtual int **getSubTickCount** (double tickStep) Q_DECL_OVERRIDE
- virtual [QVector< double >](#) **createTickVector** (double tickStep, const [QCPRange](#) &range) Q_DECL_OVERRIDE

Protected Attributes

- double **mLogBase**
- int **mSubTickCount**
- double **mLogBaseLnInv**

Additional Inherited Members

6.18.1 Detailed Description

Specialized axis ticker suited for logarithmic axes.

This [QCPAxisTicker](#) subclass generates ticks with unequal tick intervals suited for logarithmic axis scales. The ticks are placed at powers of the specified log base ([setLogBase](#)).

Especially in the case of a log base equal to 10 (the default), it might be desirable to have tick labels in the form of powers of ten without mantissa display. To achieve this, set the number precision ([QCPAxis::setNumberPrecision](#)) to zero and the number format ([QCPAxis::setNumberFormat](#)) to scientific (exponential) display with beautifully typeset decimal powers, so a format string of "eb". This will result in the following axis tick labels:

The ticker can be created and assigned to an axis like this:

6.18.2 Constructor & Destructor Documentation

6.18.2.1 QCPAxisTickerLog()

```
QCPAxisTickerLog::QCPAxisTickerLog ( )
```

Constructs the ticker and sets reasonable default values. Axis tickers are commonly created managed by a [QSharedPointer](#), which then can be passed to [QCPAxis::setTicker](#).

6.18.3 Member Function Documentation

6.18.3.1 setLogBase()

```
void QCPAxisTickerLog::setLogBase (
    double base )
```

Sets the logarithm base used for tick coordinate generation. The ticks will be placed at integer powers of *base*.

6.18.3.2 setSubTickCount()

```
void QCPAxisTickerLog::setSubTickCount (
    int subTicks )
```

Sets the number of sub ticks in a tick interval. Within each interval, the sub ticks are spaced linearly to provide a better visual guide, so the sub tick density increases toward the higher tick.

Note that *subTicks* is the number of sub ticks (not sub intervals) in one tick interval. So in the case of logarithm base 10 an intuitive sub tick spacing would be achieved with eight sub ticks (the default). This means e.g. between the ticks 10 and 100 there will be eight ticks, namely at 20, 30, 40, 50, 60, 70, 80 and 90.

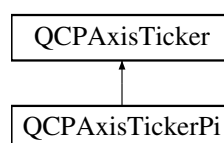
The documentation for this class was generated from the following files:

- [/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h](#)
- [/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp](#)

6.19 QCPAxisTickerPi Class Reference

Specialized axis ticker to display ticks in units of an arbitrary constant, for example pi.

Inheritance diagram for QCPAxisTickerPi:



Public Types

- enum [FractionStyle](#) { [fsFloatingPoint](#), [fsAsciiFractions](#), [fsUnicodeFractions](#) }

Public Member Functions

- [QCPAxisTickerPi](#) ()
- QString **piSymbol** () const
- double **piValue** () const
- bool **periodicity** () const
- [FractionStyle](#) **fractionStyle** () const
- void [setPiSymbol](#) (QString symbol)
- void [setPiValue](#) (double pi)
- void [setPeriodicity](#) (int multiplesOfPi)
- void [setFractionStyle](#) ([FractionStyle](#) style)

Protected Member Functions

- virtual double **getTickStep** (const [QCPRange](#) &range) Q_DECL_OVERRIDE
- virtual int **getSubTickCount** (double tickStep) Q_DECL_OVERRIDE
- virtual QString **getTickLabel** (double tick, const QLocale &locale, QChar formatChar, int precision) Q_DECL_OVERRIDE
- void **simplifyFraction** (int &numerator, int &denominator) const
- QString **fractionToString** (int numerator, int denominator) const
- QString **unicodeFraction** (int numerator, int denominator) const
- QString **unicodeSuperscript** (int number) const
- QString **unicodeSubscript** (int number) const

Protected Attributes

- QString **mPiSymbol**
- double **mPiValue**
- int **mPeriodicity**
- [FractionStyle](#) **mFractionStyle**
- double **mPiTickStep**

6.19.1 Detailed Description

Specialized axis ticker to display ticks in units of an arbitrary constant, for example pi.

This [QCPAxisTicker](#) subclass generates ticks that are expressed with respect to a given symbolic constant with a numerical value specified with [setPiValue](#) and an appearance in the tick labels specified with [setPiSymbol](#).

Ticks may be generated at fractions of the symbolic constant. How these fractions appear in the tick label can be configured with [setFractionStyle](#).

The ticker can be created and assigned to an axis like this:

6.19.2 Member Enumeration Documentation

6.19.2.1 FractionStyle

enum `QCPAxisTickerPi::FractionStyle`

Defines how fractions should be displayed in tick labels.

See also

[setFractionStyle](#)

Enumerator

<code>fsFloatingPoint</code>	Fractions are displayed as regular decimal floating point numbers, e.g. "0.25" or "0.125".
<code>fsAsciiFractions</code>	Fractions are written as rationals using ASCII characters only, e.g. "1/4" or "1/8".
<code>fsUnicodeFractions</code>	Fractions are written using sub- and superscript UTF-8 digits and the fraction symbol.

6.19.3 Constructor & Destructor Documentation

6.19.3.1 QCPAxisTickerPi()

`QCPAxisTickerPi::QCPAxisTickerPi ()`

Constructs the ticker and sets reasonable default values. Axis tickers are commonly created managed by a `QSharedPointer`, which then can be passed to [QCPAxis::setTicker](#).

6.19.4 Member Function Documentation

6.19.4.1 setFractionStyle()

```
void QCPAxisTickerPi::setFractionStyle (
    QCPAxisTickerPi::FractionStyle style )
```

Sets how the numerical/fractional part preceding the symbolic constant is displayed in tick labels. See [FractionStyle](#) for the various options.

6.19.4.2 setPeriodicity()

```
void QCPAxisTickerPi::setPeriodicity (
    int multiplesOfPi )
```

Sets whether the axis labels shall appear periodically and if so, at which multiplicity of the symbolic constant.

To disable periodicity, set *multiplesOfPi* to zero.

For example, an axis that identifies 0 with 2pi would set *multiplesOfPi* to two.

6.19.4.3 setPiSymbol()

```
void QCPAxisTickerPi::setPiSymbol (
    QString symbol )
```

Sets how the symbol part (which is always a suffix to the number) shall appear in the axis tick label.

If a space shall appear between the number and the symbol, make sure the space is contained in *symbol*.

6.19.4.4 setPiValue()

```
void QCPAxisTickerPi::setPiValue (
    double pi )
```

Sets the numerical value that the symbolic constant has.

This will be used to place the appropriate fractions of the symbol at the respective axis coordinates.

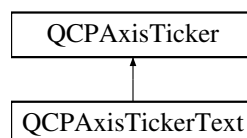
The documentation for this class was generated from the following files:

- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h↵
- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp↵

6.20 QCPAxisTickerText Class Reference

Specialized axis ticker which allows arbitrary labels at specified coordinates.

Inheritance diagram for QCPAxisTickerText:



Public Member Functions

- [QCPAxisTickerText](#) ()
- [QMap](#)< double, [QString](#) > & [ticks](#) ()
- int [subTickCount](#) () const
- void [setTicks](#) (const [QMap](#)< double, [QString](#) > &[ticks](#))
- void [setTicks](#) (const [QVector](#)< double > &positions, const [QVector](#)< [QString](#) > labels)
- void [setSubTickCount](#) (int subTicks)
- void [clear](#) ()
- void [addTick](#) (double position, [QString](#) label)
- void [addTicks](#) (const [QMap](#)< double, [QString](#) > &[ticks](#))
- void [addTicks](#) (const [QVector](#)< double > &positions, const [QVector](#)< [QString](#) > &labels)

Protected Member Functions

- virtual double [getTickStep](#) (const [QCPRange](#) &range) Q_DECL_OVERRIDE
- virtual int [getSubTickCount](#) (double tickStep) Q_DECL_OVERRIDE
- virtual [QString](#) [getTickLabel](#) (double tick, const [QLocale](#) &locale, [QChar](#) formatChar, int precision) Q_DECL_OVERRIDE
- virtual [QVector](#)< double > [createTickVector](#) (double tickStep, const [QCPRange](#) &range) Q_DECL_OVERRIDE

Protected Attributes

- [QMap](#)< double, [QString](#) > [mTicks](#)
- int [mSubTickCount](#)

Additional Inherited Members

6.20.1 Detailed Description

Specialized axis ticker which allows arbitrary labels at specified coordinates.

This [QCPAxisTicker](#) subclass generates ticks which can be directly specified by the user as coordinates and associated strings. They can be passed as a whole with [setTicks](#) or one at a time with [addTick](#). Alternatively you can directly access the internal storage via [ticks](#) and modify the tick/label data there.

This is useful for cases where the axis represents categories rather than numerical values.

If you are updating the ticks of this ticker regularly and in a dynamic fashion (e.g. dependent on the axis range), it is a sign that you should probably create an own ticker by subclassing [QCPAxisTicker](#), instead of using this one.

The ticker can be created and assigned to an axis like this:

6.20.2 Constructor & Destructor Documentation

6.20.2.1 QCPAxisTickerText()

```
QCPAxisTickerText::QCPAxisTickerText ( )
```

Constructs the ticker and sets reasonable default values. Axis tickers are commonly created managed by a [QSharedPointer](#), which then can be passed to [QCPAxis::setTicker](#).

6.20.3 Member Function Documentation

6.20.3.1 addTick()

```
void QCPAxisTickerText::addTick (
    double position,
    QString label )
```

Adds a single tick to the axis at the given axis coordinate *position*, with the provided tick *label*.

See also

[addTicks](#), [setTicks](#), [clear](#)

6.20.3.2 addTicks() [1/2]

```
void QCPAxisTickerText::addTicks (
    const QMap< double, QString > & ticks )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Adds the provided *ticks* to the ones already existing. The map key of *ticks* corresponds to the axis coordinate, and the map value is the string that will appear as tick label.

An alternative to manipulate ticks is to directly access the internal storage with the [ticks](#) getter.

See also

[addTick](#), [setTicks](#), [clear](#)

6.20.3.3 addTicks() [2/2]

```
void QCPAxisTickerText::addTicks (
    const QVector< double > & positions,
    const QVector< QString > & labels )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Adds the provided ticks to the ones already existing. The entries of *positions* correspond to the axis coordinates, and the entries of *labels* are the respective strings that will appear as tick labels.

An alternative to manipulate ticks is to directly access the internal storage with the [ticks](#) getter.

See also

[addTick](#), [setTicks](#), [clear](#)

6.20.3.4 clear()

```
void QCPAxisTickerText::clear ( )
```

Clears all ticks.

An alternative to manipulate ticks is to directly access the internal storage with the [ticks](#) getter.

See also

[setTicks](#), [addTicks](#), [addTick](#)

6.20.3.5 createTickVector()

```
QVector< double > QCPAxisTickerText::createTickVector (
    double tickStep,
    const QCPRange & range ) [protected], [virtual]
```

Returns the externally provided tick coordinates which are in the specified *range*. If available, one tick above and below the range is provided in addition, to allow possible sub tick calculation. The parameter *tickStep* is ignored.

Reimplemented from [QCPAxisTicker](#).

6.20.3.6 `getSubTickCount()`

```
int QCPAxisTickerText::getSubTickCount (
    double tickStep ) [protected], [virtual]
```

Returns the sub tick count that was configured with [setSubTickCount](#).

Reimplemented from [QCPAxisTicker](#).

6.20.3.7 `getTickLabel()`

```
QString QCPAxisTickerText::getTickLabel (
    double tick,
    const QLocale & locale,
    QChar formatChar,
    int precision ) [protected], [virtual]
```

Returns the tick label which corresponds to the key *tick* in the internal tick storage. Since the labels are provided externally, *locale*, *formatChar*, and *precision* are ignored.

Reimplemented from [QCPAxisTicker](#).

6.20.3.8 `getTickStep()`

```
double QCPAxisTickerText::getTickStep (
    const QCPRange & range ) [protected], [virtual]
```

Since the tick coordinates are provided externally, this method implementation does nothing.

Reimplemented from [QCPAxisTicker](#).

6.20.3.9 `setSubTickCount()`

```
void QCPAxisTickerText::setSubTickCount (
    int subTicks )
```

Sets the number of sub ticks that shall appear between ticks. For [QCPAxisTickerText](#), there is no automatic sub tick count calculation. So if sub ticks are needed, they must be configured with this method.

6.20.3.10 `setTicks()` [1/2]

```
void QCPAxisTickerText::setTicks (
    const QMap< double, QString > & ticks )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Sets the ticks that shall appear on the axis. The map key of *ticks* corresponds to the axis coordinate, and the map value is the string that will appear as tick label.

An alternative to manipulate ticks is to directly access the internal storage with the [ticks](#) getter.

See also

[addTicks](#), [addTick](#), [clear](#)

6.20.3.11 `setTicks()` [2/2]

```
void QCPAxisTickerText::setTicks (
    const QVector< double > & positions,
    const QVector< QString > labels )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Sets the ticks that shall appear on the axis. The entries of *positions* correspond to the axis coordinates, and the entries of *labels* are the respective strings that will appear as tick labels.

See also

[addTicks](#), [addTick](#), [clear](#)

6.20.3.12 `ticks()`

```
QMap< double, QString > & QCPAxisTickerText::ticks ( ) [inline]
```

Returns a non-const reference to the internal map which stores the tick coordinates and their labels.

You can access the map directly in order to add, remove or manipulate ticks, as an alternative to using the methods provided by [QCPAxisTickerText](#), such as [setTicks](#) and [addTick](#).

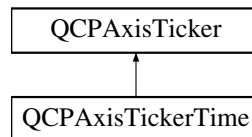
The documentation for this class was generated from the following files:

- [/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h](#)↔
- [/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp](#)↔

6.21 QCPAxisTickerTime Class Reference

Specialized axis ticker for time spans in units of milliseconds to days.

Inheritance diagram for QCPAxisTickerTime:



Public Types

- enum [TimeUnit](#) {
tuMilliseconds, **tuSeconds**, **tuMinutes**, **tuHours**,
tuDays }

Public Member Functions

- [QCPAxisTickerTime](#) ()
- QString **timeFormat** () const
- int **fieldWidth** ([TimeUnit](#) unit) const
- void **setTimeFormat** (const QString &format)
- void **setFieldWidth** ([TimeUnit](#) unit, int width)

Protected Member Functions

- virtual double **getTickStep** (const [QCPRange](#) &range) Q_DECL_OVERRIDE
- virtual int **getSubTickCount** (double tickStep) Q_DECL_OVERRIDE
- virtual QString **getTickLabel** (double tick, const QLocale &locale, QChar formatChar, int precision) Q_DECL_OVERRIDE
- void **replaceUnit** (QString &text, [TimeUnit](#) unit, int value) const

Protected Attributes

- QString **mTimeFormat**
- QHash< [TimeUnit](#), int > **mFieldWidth**
- [TimeUnit](#) **mSmallestUnit**
- [TimeUnit](#) **mBiggestUnit**
- QHash< [TimeUnit](#), QString > **mFormatPattern**

6.21.1 Detailed Description

Specialized axis ticker for time spans in units of milliseconds to days.

This [QCPAxisTicker](#) subclass generates ticks that corresponds to time intervals.

The format of the time display in the tick labels is controlled with [setTimeFormat](#) and [setFieldWidth](#). The time coordinate is in the unit of seconds with respect to the time coordinate zero. Unlike with [QCPAxisTickerDateTime](#), the ticks don't correspond to a specific calendar date and time.

The time can be displayed in milliseconds, seconds, minutes, hours and days. Depending on the largest available unit in the format specified with [setTimeFormat](#), any time spans above will be carried in that largest unit. So for example if the format string is "%m:%s" and a tick at coordinate value 7815 (being 2 hours, 10 minutes and 15 seconds) is created, the resulting tick label will show "130:15" (130 minutes, 15 seconds). If the format string is "%h:%m:%s", the hour unit will be used and the label will thus be "02:10:15". Negative times with respect to the axis zero will carry a leading minus sign.

The ticker can be created and assigned to an axis like this:

Here is an example of a time axis providing time information in days, hours and minutes. Due to the axis range spanning a few days and the wanted tick count ([setTickCount](#)), the ticker decided to use tick steps of 12 hours:

The format string for this example is

Note

If you rather wish to display calendar dates and times, have a look at [QCPAxisTickerDateTime](#) instead.

6.21.2 Member Enumeration Documentation

6.21.2.1 TimeUnit

```
enum QCPAxisTickerTime::TimeUnit
```

Defines the logical units in which fractions of time spans can be expressed.

See also

[setFieldWidth](#), [setTimeFormat](#)

6.21.3 Constructor & Destructor Documentation

6.21.3.1 QCPAxisTickerTime()

```
QCPAxisTickerTime::QCPAxisTickerTime ( )
```

Constructs the ticker and sets reasonable default values. Axis tickers are commonly created managed by a [QSharedPointer](#), which then can be passed to [QCPAxis::setTicker](#).

6.21.4 Member Function Documentation

6.21.4.1 setFieldWidth()

```
void QCPAxisTickerTime::setFieldWidth (
    QCPAxisTickerTime::TimeUnit unit,
    int width )
```

Sets the field width of the specified *unit* to be *width* digits, when displayed in the tick label. If the number for the specific unit is shorter than *width*, it will be padded with an according number of zeros to the left in order to reach the field width.

See also

[setTimeFormat](#)

6.21.4.2 setTimeFormat()

```
void QCPAxisTickerTime::setTimeFormat (
    const QString & format )
```

Sets the format that will be used to display time in the tick labels.

The available patterns are:

- %z for milliseconds
- %s for seconds
- %m for minutes
- %h for hours
- %d for days

The field width (zero padding) can be controlled for each unit with [setFieldWidth](#).

The largest unit that appears in *format* will carry all the remaining time of a certain tick coordinate, even if it overflows the natural limit of the unit. For example, if %m is the largest unit it might become larger than 59 in order to consume larger time values. If on the other hand %h is available, the minutes will wrap around to zero after 59 and the time will carry to the hour digit.

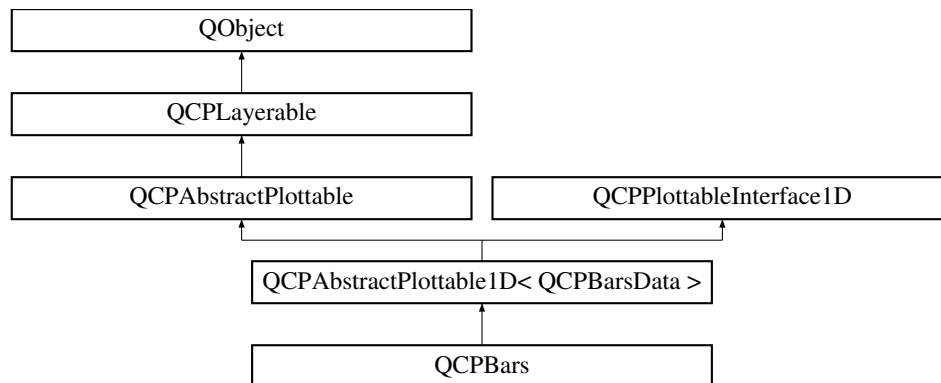
The documentation for this class was generated from the following files:

- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.[h](#)
- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.[cpp](#)

6.22 QCPBars Class Reference

A plottable representing a bar chart in a plot.

Inheritance diagram for QCPBars:



Public Types

- enum [WidthType](#) { [wtAbsolute](#), [wtAxisRectRatio](#), [wtPlotCoords](#) }

Public Member Functions

- [QCPBars](#) ([QCPAxis](#) *keyAxis, [QCPAxis](#) *valueAxis)
- double **width** () const
- [WidthType](#) **widthType** () const
- [QCPBarsGroup](#) * **barsGroup** () const
- double **baseValue** () const
- double **stackingGap** () const
- [QCPBars](#) * **barBelow** () const
- [QCPBars](#) * **barAbove** () const
- QSharedPointer< [QCPBarsDataContainer](#) > **data** () const
- void **setData** (QSharedPointer< [QCPBarsDataContainer](#) > data)
- void **setData** (const QVector< double > &keys, const QVector< double > &values, bool alreadySorted=false)
- void **setWidth** (double width)
- void **setWidthType** ([WidthType](#) widthType)
- void **setBarsGroup** ([QCPBarsGroup](#) *barsGroup)
- void **setBaseValue** (double baseValue)
- void **setStackingGap** (double pixels)
- void **addData** (const QVector< double > &keys, const QVector< double > &values, bool alreadySorted=false)
- void **addData** (double key, double value)
- void **moveBelow** ([QCPBars](#) *bars)
- void **moveAbove** ([QCPBars](#) *bars)
- virtual [QCPDataSelection](#) **selectTestRect** (const QRectF &rect, bool onlySelectable) const Q_DECL_OVERRIDE
- virtual double **selectTest** (const QPointF &pos, bool onlySelectable, QVariant *details=0) const Q_DECL_OVERRIDE
- virtual [QCPRange](#) **getKeyRange** (bool &foundRange, [QCP::SignDomain](#) inSignDomain=[QCP::sdBoth](#)) const Q_DECL_OVERRIDE
- virtual [QCPRange](#) **getValueRange** (bool &foundRange, [QCP::SignDomain](#) inSignDomain=[QCP::sdBoth](#), const [QCPRange](#) &inKeyRange=[QCPRange](#)()) const Q_DECL_OVERRIDE
- virtual QPointF **dataPixelPosition** (int index) const Q_DECL_OVERRIDE

Protected Member Functions

- virtual void **draw** ([QCPPainter](#) *painter) Q_DECL_OVERRIDE
- virtual void **drawLegendIcon** ([QCPPainter](#) *painter, const QRectF &rect) const Q_DECL_OVERRIDE
- void **getVisibleDataBounds** ([QCPBarsDataContainer::const_iterator](#) &begin, [QCPBarsDataContainer::const_iterator](#) &end) const
- QRectF **getBarRect** (double key, double value) const
- void **getPixelWidth** (double key, double &lower, double &upper) const
- double **getStackedBaseValue** (double key, bool positive) const

Static Protected Member Functions

- static void **connectBars** ([QCPBars](#) *lower, [QCPBars](#) *upper)

Protected Attributes

- double **mWidth**
- [WidthType](#) **mWidthType**
- [QCPBarsGroup](#) * **mBarsGroup**
- double **mBaseValue**
- double **mStackingGap**
- [QPointer](#)< [QCPBars](#) > **mBarBelow**
- [QPointer](#)< [QCPBars](#) > **mBarAbove**

Friends

- class **QCustomPlot**
- class **QCPLegend**
- class **QCPBarsGroup**

Additional Inherited Members

6.22.1 Detailed Description

A plottable representing a bar chart in a plot.

To plot data, assign it with the [setData](#) or [addData](#) functions.

6.22.2 Changing the appearance

The appearance of the bars is determined by the pen and the brush ([setPen](#), [setBrush](#)). The width of the individual bars can be controlled with [setWidthType](#) and [setWidth](#).

Bar charts are stackable. This means, two [QCPBars](#) plottables can be placed on top of each other (see [QCPBars::moveAbove](#)). So when two bars are at the same key position, they will appear stacked.

If you would like to group multiple [QCPBars](#) plottables together so they appear side by side as shown below, use [QCPBarsGroup](#).

6.22.3 Usage

Like all data representing objects in [QCustomPlot](#), the [QCPBars](#) is a plottable ([QCPAbstractPlottable](#)). So the plottable-interface of [QCustomPlot](#) applies ([QCustomPlot::plottable](#), [QCustomPlot::removePlottable](#), etc.)

Usually, you first create an instance:

which registers it with the [QCustomPlot](#) instance of the passed axes. Note that this [QCustomPlot](#) instance takes ownership of the plottable, so do not delete it manually but use [QCustomPlot::removePlottable\(\)](#) instead. The newly created plottable can be modified, e.g.:

6.22.4 Member Enumeration Documentation

6.22.4.1 WidthType

```
enum QCPBars::WidthType
```

Defines the ways the width of the bar can be specified. Thus it defines what the number passed to [setWidth](#) actually means.

See also

[setWidthType](#), [setWidth](#)

Enumerator

wtAbsolute	Bar width is in absolute pixels.
wtAxisRectRatio	Bar width is given by a fraction of the axis rect size.
wtPlotCoords	Bar width is in key coordinates and thus scales with the key axis range.

6.22.5 Constructor & Destructor Documentation

6.22.5.1 QCPBars()

```
QCPBars::QCPBars (
    QCPAxis * keyAxis,
    QCPAxis * valueAxis ) [explicit]
```

Constructs a bar chart which uses *keyAxis* as its key axis ("x") and *valueAxis* as its value axis ("y"). *keyAxis* and *valueAxis* must reside in the same [QCustomPlot](#) instance and not have the same orientation. If either of these restrictions is violated, a corresponding message is printed to the debug output (qDebug), the construction is not aborted, though.

The created [QCPBars](#) is automatically registered with the [QCustomPlot](#) instance inferred from *keyAxis*. This [QCustomPlot](#) instance takes ownership of the [QCPBars](#), so do not delete it manually but use [QCustomPlot::removePlottable\(\)](#) instead.

6.22.6 Member Function Documentation

6.22.6.1 `addData()` [1/2]

```
void QCPBars::addData (
    const QVector< double > & keys,
    const QVector< double > & values,
    bool alreadySorted = false )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Adds the provided points in *keys* and *values* to the current data. The provided vectors should have equal length. Else, the number of added points will be the size of the smallest vector.

If you can guarantee that the passed data points are sorted by *keys* in ascending order, you can set *alreadySorted* to true, to improve performance by saving a sorting run.

Alternatively, you can also access and modify the data directly via the [data](#) method, which returns a pointer to the internal data container.

6.22.6.2 `addData()` [2/2]

```
void QCPBars::addData (
    double key,
    double value )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts. Adds the provided data point as *key* and *value* to the current data.

Alternatively, you can also access and modify the data directly via the [data](#) method, which returns a pointer to the internal data container.

6.22.6.3 `barAbove()`

```
QCPBars * QCPBars::barAbove ( ) const [inline]
```

Returns the bars plottable that is directly above this bars plottable. If there is no such plottable, returns 0.

See also

[barBelow](#), [moveBelow](#), [moveAbove](#)

6.22.6.4 `barBelow()`

```
QCPBars * QCPBars::barBelow ( ) const [inline]
```

Returns the bars plottable that is directly below this bars plottable. If there is no such plottable, returns 0.

See also

[barAbove](#), [moveBelow](#), [moveAbove](#)

6.22.6.5 `data()`

```
QSharedPointer< QCPBarsDataContainer > QCPBars::data ( ) const [inline]
```

Returns a shared pointer to the internal data storage of type `QCPBarsDataContainer`. You may use it to directly manipulate the data, which may be more convenient and faster than using the regular [setData](#) or [addData](#) methods.

6.22.6.6 `dataPixelPosition()`

```
QPointF QCPBars::dataPixelPosition (
    int index ) const [virtual]
```

Returns the pixel position on the widget surface at which the data point at the given *index* appears.

Usually this corresponds to the point of [dataMainKey/dataMainValue](#), in pixel coordinates. However, depending on the plottable, this might be a different apparent position than just a coord-to-pixel transform of those values. For example, [QCPBars](#) apparent data values can be shifted depending on their stacking, bar grouping or configured base value.

Reimplemented from [QCPAbstractPlottable1D< QCPBarsData >](#).

6.22.6.7 `getKeyRange()`

```
QCPRange QCPBars::getKeyRange (
    bool & foundRange,
    QCP::SignDomain inSignDomain = QCP::sdBoth ) const [virtual]
```

Returns the coordinate range that all data in this plottable span in the key axis dimension. For logarithmic plots, one can set *inSignDomain* to either [QCP::sdNegative](#) or [QCP::sdPositive](#) in order to restrict the returned range to that sign domain. E.g. when only negative range is wanted, set *inSignDomain* to [QCP::sdNegative](#) and all positive points will be ignored for range calculation. For no restriction, just set *inSignDomain* to [QCP::sdBoth](#) (default). *foundRange* is an output parameter that indicates whether a range could be found or not. If this is false, you shouldn't use the returned range (e.g. no points in data).

Note that *foundRange* is not the same as [QCPRange::validRange](#), since the range returned by this function may have size zero (e.g. when there is only one data point). In this case *foundRange* would return true, but the returned range is not a valid range in terms of [QCPRange::validRange](#).

See also

[rescaleAxes](#), [getValueRange](#)

Implements [QCPAbstractPlottable](#).

6.22.6.8 `getValueRange()`

```
QCPRange QCPBars::getValueRange (
    bool & foundRange,
    QCP::SignDomain inSignDomain = QCP::sdBoth,
    const QCPRange & inKeyRange = QCPRange() ) const [virtual]
```

Returns the coordinate range that the data points in the specified key range (*inKeyRange*) span in the value axis dimension. For logarithmic plots, one can set *inSignDomain* to either [QCP::sdNegative](#) or [QCP::sdPositive](#) in order to restrict the returned range to that sign domain. E.g. when only negative range is wanted, set *inSignDomain* to [QCP::sdNegative](#) and all positive points will be ignored for range calculation. For no restriction, just set *inSignDomain* to [QCP::sdBoth](#) (default). *foundRange* is an output parameter that indicates whether a range could be found or not. If this is false, you shouldn't use the returned range (e.g. no points in data).

If *inKeyRange* has both lower and upper bound set to zero (is equal to [QCPRange\(\)](#)), all data points are considered, without any restriction on the keys.

Note that *foundRange* is not the same as [QCPRange::validRange](#), since the range returned by this function may have size zero (e.g. when there is only one data point). In this case *foundRange* would return true, but the returned range is not a valid range in terms of [QCPRange::validRange](#).

See also

[rescaleAxes](#), [getKeyRange](#)

Implements [QCPAbstractPlottable](#).

6.22.6.9 `moveAbove()`

```
void QCPBars::moveAbove (
    QCPBars * bars )
```

Moves this bars plottable above *bars*. In other words, the bars of this plottable will appear above the bars of *bars*. The move target *bars* must use the same key and value axis as this plottable.

Inserting into and removing from existing bar stacking is handled gracefully. If *bars* already has a bars object above itself, this bars object is inserted between the two. If this bars object is already between two other bars, the two other bars will be stacked on top of each other after the operation.

To remove this bars plottable from any stacking, set *bars* to 0.

See also

[moveBelow](#), [barBelow](#), [barAbove](#)

6.22.6.10 moveBelow()

```
void QCPBars::moveBelow (
    QCPBars * bars )
```

Moves this bars plottable below *bars*. In other words, the bars of this plottable will appear below the bars of *bars*. The move target *bars* must use the same key and value axis as this plottable.

Inserting into and removing from existing bar stacking is handled gracefully. If *bars* already has a bars object below itself, this bars object is inserted between the two. If this bars object is already between two other bars, the two other bars will be stacked on top of each other after the operation.

To remove this bars plottable from any stacking, set *bars* to 0.

See also

[moveBelow](#), [barAbove](#), [barBelow](#)

6.22.6.11 selectTest()

```
double QCPBars::selectTest (
    const QPointF & pos,
    bool onlySelectable,
    QVariant * details = 0 ) const [virtual]
```

Implements a point-selection algorithm assuming the data (accessed via the 1D data interface) is point-like. Most subclasses will want to reimplement this method again, to provide a more accurate hit test based on the true data visualization geometry.

Reimplemented from [QCPAbstractPlottable1D< QCPBarsData >](#).

6.22.6.12 selectTestRect()

```
QCPDataSelection QCPBars::selectTestRect (
    const QRectF & rect,
    bool onlySelectable ) const [virtual]
```

Returns a data selection containing all the data points of this plottable which are contained (or hit by) *rect*. This is used mainly in the selection rect interaction for data selection (data selection mechanism).

If *onlySelectable* is true, an empty [QCPDataSelection](#) is returned if this plottable is not selectable (i.e. if [QCPAbstractPlottable::setSelectable](#) is [QCP::stNone](#)).

Note

rect must be a normalized rect (positive or zero width and height). This is especially important when using the rect of [QCPSelectionRect::accepted](#), which is not necessarily normalized. Use `QRect::normalized()` when passing a rect which might not be normalized.

Reimplemented from [QCPAbstractPlottable1D< QCPBarsData >](#).

6.22.6.13 `setBarsGroup()`

```
void QCPBars::setBarsGroup (
    QCPBarsGroup * barsGroup )
```

Sets to which [QCPBarsGroup](#) this [QCPBars](#) instance belongs to. Alternatively, you can also use [QCPBarsGroup::append](#).

To remove this [QCPBars](#) from any group, set *barsGroup* to 0.

6.22.6.14 `setBaseValue()`

```
void QCPBars::setBaseValue (
    double baseValue )
```

Sets the base value of this bars plottable.

The base value defines where on the value coordinate the bars start. How far the bars extend from the base value is given by their individual value data. For example, if the base value is set to 1, a bar with data value 2 will have its lowest point at value coordinate 1 and highest point at 3.

For stacked bars, only the base value of the bottom-most [QCPBars](#) has meaning.

The default base value is 0.

6.22.6.15 `setData()` [1/2]

```
void QCPBars::setData (
    QSharedPointer< QCPBarsDataContainer > data )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Replaces the current data container with the provided *data* container.

Since a `QSharedPointer` is used, multiple [QCPBars](#) may share the same data container safely. Modifying the data in the container will then affect all bars that share the container. Sharing can be achieved by simply exchanging the data containers wrapped in shared pointers:

If you do not wish to share containers, but create a copy from an existing container, rather use the [QCPDataContainer<DataType>::set](#) method on the bar's data container directly:

See also

[addData](#)

6.22.6.16 setData() [2/2]

```
void QCPBars::setData (
    const QVector< double > & keys,
    const QVector< double > & values,
    bool alreadySorted = false )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Replaces the current data with the provided points in *keys* and *values*. The provided vectors should have equal length. Else, the number of added points will be the size of the smallest vector.

If you can guarantee that the passed data points are sorted by *keys* in ascending order, you can set *alreadySorted* to true, to improve performance by saving a sorting run.

See also

[addData](#)

6.22.6.17 setStackingGap()

```
void QCPBars::setStackingGap (
    double pixels )
```

If this bars plottable is stacked on top of another bars plottable ([moveAbove](#)), this method allows specifying a distance in *pixels*, by which the drawn bar rectangles will be separated by the bars below it.

6.22.6.18 setWidth()

```
void QCPBars::setWidth (
    double width )
```

Sets the width of the bars.

How the number passed as *width* is interpreted (e.g. screen pixels, plot coordinates,...), depends on the currently set width type, see [setWidthType](#) and [WidthType](#).

6.22.6.19 setWidthType()

```
void QCPBars::setWidthType (
    QCPBars::WidthType widthType )
```

Sets how the width of the bars is defined. See the documentation of [WidthType](#) for an explanation of the possible values for *widthType*.

The default value is [wtPlotCoords](#).

See also

[setWidth](#)

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

- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h↵
- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp↵

6.23 QCPBarsData Class Reference

Holds the data of one single data point (one bar) for [QCPBars](#).

Public Member Functions

- [QCPBarsData](#) ()
- [QCPBarsData](#) (double key, double value)
- double [sortKey](#) () const
- double [mainKey](#) () const
- double [mainValue](#) () const
- [QCPRange](#) [valueRange](#) () const

Static Public Member Functions

- static [QCPBarsData](#) [fromSortKey](#) (double [sortKey](#))
- static bool [sortKeysIsMainKey](#) ()

Public Attributes

- double **key**
- double **value**

6.23.1 Detailed Description

Holds the data of one single data point (one bar) for [QCPBars](#).

The stored data is:

- *key*: coordinate on the key axis of this bar (this is the *mainKey* and the *sortKey*)
- *value*: height coordinate on the value axis of this bar (this is the *mainValue*)

The container for storing multiple data points is [QCPBarsDataContainer](#). It is a typedef for [QCPDataContainer](#) with [QCPBarsData](#) as the `DataType` template parameter. See the documentation there for an explanation regarding the data type's generic methods.

See also

[QCPBarsDataContainer](#)

6.23.2 Constructor & Destructor Documentation

6.23.2.1 QCPBarsData() [1/2]

```
QCPBarsData::QCPBarsData ( )
```

Constructs a bar data point with key and value set to zero.

6.23.2.2 QCPBarsData() [2/2]

```
QCPBarsData::QCPBarsData (
    double key,
    double value )
```

Constructs a bar data point with the specified *key* and *value*.

6.23.3 Member Function Documentation

6.23.3.1 fromSortKey()

```
static QCPBarsData QCPBarsData::fromSortKey (
    double sortKey ) [inline], [static]
```

Returns a data point with the specified *sortKey*. All other members are set to zero.

For a general explanation of what this method is good for in the context of the data container, see the documentation of [QCPDataContainer](#).

6.23.3.2 mainKey()

```
double QCPBarsData::mainKey ( ) const [inline]
```

Returns the *key* member of this data point.

For a general explanation of what this method is good for in the context of the data container, see the documentation of [QCPDataContainer](#).

6.23.3.3 mainValue()

```
double QCPBarsData::mainValue ( ) const [inline]
```

Returns the *value* member of this data point.

For a general explanation of what this method is good for in the context of the data container, see the documentation of [QCPDataContainer](#).

6.23.3.4 `sortKey()`

```
double QCPBarsData::sortKey ( ) const [inline]
```

Returns the *key* member of this data point.

For a general explanation of what this method is good for in the context of the data container, see the documentation of [QCPDataContainer](#).

6.23.3.5 `sortKeyIsMainKey()`

```
static static bool QCPBarsData::sortKeyIsMainKey ( ) [inline], [static]
```

Since the member *key* is both the data point key coordinate and the data ordering parameter, this method returns true.

For a general explanation of what this method is good for in the context of the data container, see the documentation of [QCPDataContainer](#).

6.23.3.6 `valueRange()`

```
QCPRange QCPBarsData::valueRange ( ) const [inline]
```

Returns a [QCPRange](#) with both lower and upper boundary set to *value* of this data point.

For a general explanation of what this method is good for in the context of the data container, see the documentation of [QCPDataContainer](#).

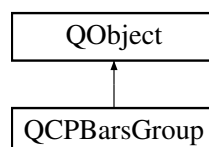
The documentation for this class was generated from the following files:

- `/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h`
- `/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp`

6.24 QCPBarsGroup Class Reference

Groups multiple [QCPBars](#) together so they appear side by side.

Inheritance diagram for QCPBarsGroup:



Public Types

- enum [SpacingType](#) { `stAbsolute`, `stAxisRectRatio`, `stPlotCoords` }

Public Member Functions

- [QCPBarsGroup](#) ([QCustomPlot](#) *parentPlot)
- [SpacingType](#) **spacingType** () const
- double **spacing** () const
- void **setSpacingType** ([SpacingType](#) spacingType)
- void **setSpacing** (double spacing)
- [QList](#)< [QCPBars](#) * > **bars** () const
- [QCPBars](#) * **bars** (int index) const
- int **size** () const
- bool **isEmpty** () const
- void **clear** ()
- bool **contains** ([QCPBars](#) *bars) const
- void **append** ([QCPBars](#) *bars)
- void **insert** (int i, [QCPBars](#) *bars)
- void **remove** ([QCPBars](#) *bars)

Protected Member Functions

- void **registerBars** ([QCPBars](#) *bars)
- void **unregisterBars** ([QCPBars](#) *bars)
- double **keyPixelOffset** (const [QCPBars](#) *bars, double keyCoord)
- double **getPixelSpacing** (const [QCPBars](#) *bars, double keyCoord)

Protected Attributes

- [QCustomPlot](#) * **mParentPlot**
- [SpacingType](#) **mSpacingType**
- double **mSpacing**
- [QList](#)< [QCPBars](#) * > **mBars**

Friends

- class **QCPBars**

6.24.1 Detailed Description

Groups multiple [QCPBars](#) together so they appear side by side.

When showing multiple [QCPBars](#) in one plot which have bars at identical keys, it may be desirable to have them appearing next to each other at each key. This is what adding the respective [QCPBars](#) plottables to a [QCPBarsGroup](#) achieves. (An alternative approach is to stack them on top of each other, see [QCPBars::moveAbove](#).)

6.24.2 Usage

To add a [QCPBars](#) plottable to the group, create a new group and then add the respective bars instances:

Alternatively to appending to the group like shown above, you can also set the group on the [QCPBars](#) plottable via [QCPBars::setBarsGroup](#).

The spacing between the bars can be configured via [setSpacingType](#) and [setSpacing](#). The bars in this group appear in the plot in the order they were appended. To insert a bars plottable at a certain index position, or to reposition a bars plottable which is already in the group, use [insert](#).

To remove specific bars from the group, use either [remove](#) or call [QCPBars::setBarsGroup\(0\)](#) on the respective bars plottable.

To clear the entire group, call [clear](#), or simply delete the group.

6.24.3 Example

The image above is generated with the following code:

6.24.4 Member Enumeration Documentation

6.24.4.1 SpacingType

```
enum QCPBarsGroup::SpacingType
```

Defines the ways the spacing between bars in the group can be specified. Thus it defines what the number passed to [setSpacing](#) actually means.

See also

[setSpacingType](#), [setSpacing](#)

Enumerator

stAbsolute	Bar spacing is in absolute pixels.
stAxisRectRatio	Bar spacing is given by a fraction of the axis rect size.
stPlotCoords	Bar spacing is in key coordinates and thus scales with the key axis range.

6.24.5 Constructor & Destructor Documentation

6.24.5.1 QCPBarsGroup()

```
QCPBarsGroup::QCPBarsGroup (
    QCustomPlot * parentPlot )
```

Constructs a new bars group for the specified [QCustomPlot](#) instance.

6.24.6 Member Function Documentation

6.24.6.1 append()

```
void QCPBarsGroup::append (
    QCPBars * bars )
```

Adds the specified *bars* plottable to this group. Alternatively, you can also use [QCPBars::setBarsGroup](#) on the *bars* instance.

See also

[insert](#), [remove](#)

6.24.6.2 bars() [1/2]

```
QList< QCPBars * > QCPBarsGroup::bars ( ) const [inline]
```

Returns all bars currently in this group.

See also

[bars\(int index\)](#)

6.24.6.3 bars() [2/2]

```
QCPBars * QCPBarsGroup::bars (
    int index ) const
```

Returns the [QCPBars](#) instance with the specified *index* in this group. If no such [QCPBars](#) exists, returns 0.

See also

[bars\(\)](#), [size](#)

6.24.6.4 clear()

```
void QCPBarsGroup::clear ( )
```

Removes all [QCPBars](#) plottables from this group.

See also

[isEmpty](#)

6.24.6.5 contains()

```
bool QCPBarsGroup::contains (
    QCPBars * bars ) const [inline]
```

Returns whether the specified *bars* plottable is part of this group.

6.24.6.6 insert()

```
void QCPBarsGroup::insert (
    int i,
    QCPBars * bars )
```

Inserts the specified *bars* plottable into this group at the specified index position *i*. This gives you full control over the ordering of the bars.

bars may already be part of this group. In that case, *bars* is just moved to the new index position.

See also

[append](#), [remove](#)

6.24.6.7 isEmpty()

```
bool QCPBarsGroup::isEmpty ( ) const [inline]
```

Returns whether this bars group is empty.

See also

[size](#)

6.24.6.8 remove()

```
void QCPBarsGroup::remove (
    QCPBars * bars )
```

Removes the specified *bars* plottable from this group.

See also

[contains](#), [clear](#)

6.24.6.9 setSpacing()

```
void QCPBarsGroup::setSpacing (
    double spacing )
```

Sets the spacing between adjacent bars. What the number passed as *spacing* actually means, is defined by the current [SpacingType](#), which can be set with [setSpacingType](#).

See also

[setSpacingType](#)

6.24.6.10 setSpacingType()

```
void QCPBarsGroup::setSpacingType (
    SpacingType spacingType )
```

Sets how the spacing between adjacent bars is interpreted. See [SpacingType](#).

The actual spacing can then be specified with [setSpacing](#).

See also

[setSpacing](#)

6.24.6.11 size()

```
int QCPBarsGroup::size ( ) const [inline]
```

Returns the number of [QCPBars](#) plottables that are part of this group.

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

- [/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h](#)↔
- [/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp](#)↔

6.25 QCPColorGradient Class Reference

Defines a color gradient for use with e.g. [QCPColorMap](#).

Public Types

- enum [ColorInterpolation](#) { ciRGB, ciHSV }
- enum [GradientPreset](#) {
gpGrayscale, gpHot, gpCold, gpNight,
gpCandy, gpGeography, gpIon, gpThermal,
gpPolar, gpSpectrum, gpJet, gpHues }

Public Member Functions

- [QCPColorGradient](#) ()
- [QCPColorGradient](#) ([GradientPreset](#) preset)
- bool **operator==** (const [QCPColorGradient](#) &other) const
- bool **operator!=** (const [QCPColorGradient](#) &other) const
- int **levelCount** () const
- QMap< double, QColor > **colorStops** () const
- [ColorInterpolation](#) **colorInterpolation** () const
- bool **periodic** () const
- void **setLevelCount** (int n)
- void **setColorStops** (const QMap< double, QColor > &colorStops)
- void **setColorStopAt** (double position, const QColor &color)
- void **setColorInterpolation** ([ColorInterpolation](#) interpolation)
- void **setPeriodic** (bool enabled)
- void **colorize** (const double *data, const [QCPRange](#) &range, QRgb *scanLine, int n, int dataIndexFactor=1, bool logarithmic=false)
- void **colorize** (const double *data, const unsigned char *alpha, const [QCPRange](#) &range, QRgb *scanLine, int n, int dataIndexFactor=1, bool logarithmic=false)
- QRgb **color** (double position, const [QCPRange](#) &range, bool logarithmic=false)
- void **loadPreset** ([GradientPreset](#) preset)
- void **clearColorStops** ()
- [QCPColorGradient](#) **inverted** () const

Protected Member Functions

- bool **stopsUseAlpha** () const
- void **updateColorBuffer** ()

Protected Attributes

- int **mLevelCount**
- QMap< double, QColor > **mColorStops**
- [ColorInterpolation](#) **mColorInterpolation**
- bool **mPeriodic**
- QVector< QRgb > **mColorBuffer**
- bool **mColorBufferInvalidated**

6.25.1 Detailed Description

Defines a color gradient for use with e.g. [QCPColorMap](#).

This class describes a color gradient which can be used to encode data with color. For example, [QCPColorMap](#) and [QCPColorScale](#) have [setGradient](#) methods which take an instance of this class. Colors are set with [setColor↵StopAt\(double position, const QColor &color\)](#) with a *position* from 0 to 1. In between these defined color positions, the color will be interpolated linearly either in RGB or HSV space, see [setColorInterpolation](#).

Alternatively, load one of the preset color gradients shown in the image below, with [loadPreset](#), or by directly specifying the preset in the constructor.

Apart from red, green and blue components, the gradient also interpolates the alpha values of the configured color stops. This allows to display some portions of the data range as transparent in the plot.

The [ructor](#) allows directly converting a [GradientPreset](#) to a [QCPColorGradient](#). This means that you can directly pass [GradientPreset](#) to all the *setGradient* methods, e.g.:

The total number of levels used in the gradient can be set with [setLevelCount](#). Whether the color gradient shall be applied periodically (wrapping around) to data values that lie outside the data range specified on the plottable instance can be controlled with [setPeriodic](#).

6.25.2 Member Enumeration Documentation

6.25.2.1 ColorInterpolation

```
enum QCPColorGradient::ColorInterpolation
```

Defines the color spaces in which color interpolation between gradient stops can be performed.

See also

[setColorInterpolation](#)

Enumerator

ciRGB	Color channels red, green and blue are linearly interpolated.
ciHSV	Color channels hue, saturation and value are linearly interpolated (The hue is interpolated over the shortest angle distance)

6.25.2.2 GradientPreset

```
enum QCPColorGradient::GradientPreset
```

Defines the available presets that can be loaded with [loadPreset](#). See the documentation there for an image of the presets.

Enumerator

gpGrayscale	Continuous lightness from black to white (suited for non-biased data representation)
gpHot	Continuous lightness from black over firey colors to white (suited for non-biased data representation)
gpCold	Continuous lightness from black over icy colors to white (suited for non-biased data representation)
gpNight	Continuous lightness from black over weak blueish colors to white (suited for non-biased data representation)
gpCandy	Blue over pink to white.
gpGeography	Colors suitable to represent different elevations on geographical maps.
gpIon	Half hue spectrum from black over purple to blue and finally green (creates banding illusion but allows more precise magnitude estimates)
gpThermal	Colors suitable for thermal imaging, ranging from dark blue over purple to orange, yellow and white.
gpPolar	Colors suitable to emphasize polarity around the center, with blue for negative, black in the middle and red for positive values.
gpSpectrum	An approximation of the visible light spectrum (creates banding illusion but allows more precise magnitude estimates)
gpJet	Hue variation similar to a spectrum, often used in numerical visualization (creates banding illusion but allows more precise magnitude estimates)
gpHues	Full hue cycle, with highest and lowest color red (suitable for periodic data, such as angles and phases, see setPeriodic)

6.25.3 Constructor & Destructor Documentation

6.25.3.1 QCPCColorGradient() [1/2]

```
QCPCColorGradient::QCPCColorGradient ( )
```

Constructs a new, empty [QCPCColorGradient](#) with no predefined color stops. You can add own color stops with [setColorStopAt](#).

The color level count is initialized to 350.

6.25.3.2 QCPCColorGradient() [2/2]

```
QCPCColorGradient::QCPCColorGradient (
    GradientPreset preset )
```

Constructs a new [QCPCColorGradient](#) initialized with the colors and color interpolation according to *preset*.

The color level count is initialized to 350.

6.25.4 Member Function Documentation

6.25.4.1 clearColorStops()

```
void QCPColorGradient::clearColorStops ( )
```

Clears all color stops.

See also

[setColorStops](#), [setColorStopAt](#)

6.25.4.2 colorize() [1/2]

```
void QCPColorGradient::colorize (
    const double * data,
    const QCPRange & range,
    QRgb * scanLine,
    int n,
    int dataIndexFactor = 1,
    bool logarithmic = false )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

This method is used to quickly convert a *data* array to colors. The colors will be output in the array *scanLine*. Both *data* and *scanLine* must have the length *n* when passed to this function. The data range that shall be used for mapping the data value to the gradient is passed in *range*. *logarithmic* indicates whether the data values shall be mapped to colors logarithmically.

if *data* actually contains 2D-data linearized via `[row*columnCount + column]`, you can set *dataIndexFactor* to `columnCount` to convert a column instead of a row of the data array, in *scanLine*. *scanLine* will remain a regular (1D) array. This works because *data* is addressed `data[i*dataIndexFactor]`.

Use the overloaded method to additionally provide alpha map data.

The QRgb values that are placed in *scanLine* have their r, g and b components premultiplied with alpha (see [QImage::Format_ARGB32_Premultiplied](#)).

6.25.4.3 colorize() [2/2]

```
void QCPColorGradient::colorize (
    const double * data,
    const unsigned char * alpha,
    const QCPRange & range,
    QRgb * scanLine,
    int n,
    int dataIndexFactor = 1,
    bool logarithmic = false )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Additionally to the other overload of [colorize](#), this method takes the array *alpha*, which has the same size and structure as *data* and encodes the alpha information per data point.

The QRgb values that are placed in *scanLine* have their r, g and b components premultiplied with alpha (see [QImage::Format_ARGB32_Premultiplied](#)).

6.25.4.4 `inverted()`

```
QCPColorGradient QCPColorGradient::inverted ( ) const
```

Returns an inverted gradient. The inverted gradient has all properties as this [QCPColorGradient](#), but the order of the color stops is inverted.

See also

[setColorStops](#), [setColorStopAt](#)

6.25.4.5 `loadPreset()`

```
void QCPColorGradient::loadPreset (
    GradientPreset preset )
```

Clears the current color stops and loads the specified *preset*. A preset consists of predefined color stops and the corresponding color interpolation method.

The available presets are:

6.25.4.6 `setColorInterpolation()`

```
void QCPColorGradient::setColorInterpolation (
    QCPColorGradient::ColorInterpolation interpolation )
```

Sets whether the colors in between the configured color stops (see [setColorStopAt](#)) shall be interpolated linearly in RGB or in HSV color space.

For example, a sweep in RGB space from red to green will have a muddy brown intermediate color, whereas in HSV space the intermediate color is yellow.

6.25.4.7 `setColorStopAt()`

```
void QCPColorGradient::setColorStopAt (
    double position,
    const QColor & color )
```

Sets the *color* the gradient will have at the specified *position* (from 0 to 1). In between these color stops, the color is interpolated according to [setColorInterpolation](#).

See also

[setColorStops](#), [clearColorStops](#)

6.25.4.8 setColorStops()

```
void QCPColorGradient::setColorStops (
    const QMap< double, QColor > & colorStops )
```

Sets at which positions from 0 to 1 which color shall occur. The positions are the keys, the colors are the values of the passed QMap *colorStops*. In between these color stops, the color is interpolated according to [setColor↔Interpolation](#).

A more convenient way to create a custom gradient may be to clear all color stops with [clearColorStops](#) (or creating a new, empty [QCPColorGradient](#)) and then adding them one by one with [setColorStopAt](#).

See also

[clearColorStops](#)

6.25.4.9 setLevelCount()

```
void QCPColorGradient::setLevelCount (
    int n )
```

Sets the number of discretization levels of the color gradient to *n*. The default is 350 which is typically enough to create a smooth appearance. The minimum number of levels is 2.

6.25.4.10 setPeriodic()

```
void QCPColorGradient::setPeriodic (
    bool enabled )
```

Sets whether data points that are outside the configured data range (e.g. [QCPColorMap::setDataRange](#)) are colored by periodically repeating the color gradient or whether they all have the same color, corresponding to the respective gradient boundary color.

As shown in the image above, gradients that have the same start and end color are especially suitable for a periodic gradient mapping, since they produce smooth color transitions throughout the color map. A preset that has this property is [gpHues](#).

In practice, using periodic color gradients makes sense when the data corresponds to a periodic dimension, such as an angle or a phase. If this is not the case, the color encoding might become ambiguous, because multiple different data values are shown as the same color.

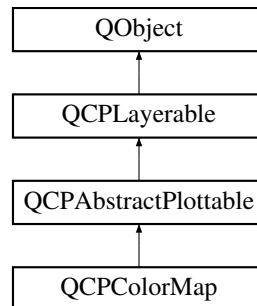
The documentation for this class was generated from the following files:

- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.↔h
- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.↔cpp

6.26 QCPCoMap Class Reference

A plottable representing a two-dimensional color map in a plot.

Inheritance diagram for QCPCoMap:



Signals

- void [dataRangeChanged](#) (const [QCPRange](#) &newRange)
- void [dataScaleTypeChanged](#) ([QCPAxis::ScaleType](#) scaleType)
- void [gradientChanged](#) (const [QCPCoColorGradient](#) &newGradient)

Public Member Functions

- [QCPCoMap](#) ([QCPAxis](#) *keyAxis, [QCPAxis](#) *valueAxis)
- [QCPCoMapData](#) * [data](#) () const
- [QCPRange](#) [dataRange](#) () const
- [QCPAxis::ScaleType](#) [dataScaleType](#) () const
- bool [interpolate](#) () const
- bool [tightBoundary](#) () const
- [QCPCoColorGradient](#) [gradient](#) () const
- [QCPCoColorScale](#) * [colorScale](#) () const
- void [setData](#) ([QCPCoMapData](#) *data, bool copy=false)
- Q_SLOT void [setDataRange](#) (const [QCPRange](#) &dataRange)
- Q_SLOT void [setDataScaleType](#) ([QCPAxis::ScaleType](#) scaleType)
- Q_SLOT void [setGradient](#) (const [QCPCoColorGradient](#) &gradient)
- void [setInterpolate](#) (bool enabled)
- void [setTightBoundary](#) (bool enabled)
- void [setColorScale](#) ([QCPCoColorScale](#) *colorScale)
- void [rescaleDataRange](#) (bool recalculateDataBounds=false)
- Q_SLOT void [updateLegendIcon](#) (Qt::TransformationMode transformMode=Qt::SmoothTransformation, const QSize &thumbSize=QSize(32, 18))
- virtual double [selectTest](#) (const QPointF &pos, bool onlySelectable, QVariant *details=0) const Q_DECL_OVERRIDE
- virtual [QCPRange](#) [getKeyRange](#) (bool &foundRange, [QCP::SignDomain](#) inSignDomain=[QCP::sdBoth](#)) const Q_DECL_OVERRIDE
- virtual [QCPRange](#) [getValueRange](#) (bool &foundRange, [QCP::SignDomain](#) inSignDomain=[QCP::sdBoth](#), const [QCPRange](#) &inKeyRange=[QCPRange](#)()) const Q_DECL_OVERRIDE

Protected Member Functions

- virtual void **updateMapImage** ()
- virtual void **draw** ([QCPPainter](#) *painter) Q_DECL_OVERRIDE
- virtual void **drawLegendIcon** ([QCPPainter](#) *painter, const QRectF &rect) const Q_DECL_OVERRIDE

Protected Attributes

- [QCPRange](#) **mDataRange**
- [QCPAxis::ScaleType](#) **mDataScaleType**
- [QCPCoMapData](#) * **mMapData**
- [QCPCoMapGradient](#) **mGradient**
- bool **mInterpolate**
- bool **mTightBoundary**
- QPointer< [QCPCoMapScale](#) > **mColorScale**
- QImage **mMapImage**
- QImage **mUndersampledMapImage**
- QPixmap **mLegendIcon**
- bool **mMapImageInvalidated**

Friends

- class **QCustomPlot**
- class **QCPLegend**

6.26.1 Detailed Description

A plottable representing a two-dimensional color map in a plot.

The data is stored in the class [QCPCoMapData](#), which can be accessed via the [data\(\)](#) method.

A color map has three dimensions to represent a data point: The *key* dimension, the *value* dimension and the *data* dimension. As with other plottables such as graphs, *key* and *value* correspond to two orthogonal axes on the [QCustomPlot](#) surface that you specify in the [QCPCoMap](#) constructor. The *data* dimension however is encoded as the color of the point at (*key*, *value*).

Set the number of points (or *cells*) in the key/value dimension via [QCPCoMapData::setSize](#). The plot coordinate range over which these points will be displayed is specified via [QCPCoMapData::setRange](#). The first cell will be centered on the lower range boundary and the last cell will be centered on the upper range boundary. The data can be set by either accessing the cells directly with [QCPCoMapData::setCell](#) or by addressing the cells via their plot coordinates with [QCPCoMapData::setData](#). If possible, you should prefer [setCell](#), since it doesn't need to do any coordinate transformation and thus performs a bit better.

The cell with index (0, 0) is at the bottom left, if the color map uses normal (i.e. not reversed) key and value axes.

To show the user which colors correspond to which *data* values, a [QCPCoMapScale](#) is typically placed to the right of the axis rect. See the documentation there for details on how to add and use a color scale.

6.26.2 Changing the appearance

The central part of the appearance is the color gradient, which can be specified via [setGradient](#). See the documentation of [QCPCColorGradient](#) for details on configuring a color gradient.

The *data* range that is mapped to the colors of the gradient can be specified with [setDataRange](#). To make the data range encompass the whole data set minimum to maximum, call [rescaleDataRange](#).

6.26.3 Transparency

Transparency in color maps can be achieved by two mechanisms. On one hand, you can specify alpha values for color stops of the [QCPCColorGradient](#), via the regular QColor interface. This will cause the color map data which gets mapped to colors around those color stops to appear with the accordingly interpolated transparency.

On the other hand you can also directly apply an alpha value to each cell independent of its data, by using the alpha map feature of [QCPCColorMapData](#). The relevant methods are [QCPCColorMapData::setAlpha](#), [QCPCColorMapData::fillAlpha](#) and [QCPCColorMapData::clearAlpha\(\)](#).

The two transparencies will be joined together in the plot and otherwise not interfere with each other. They are mixed in a multiplicative matter, so an alpha of e.g. 50% (128/255) in both modes simultaneously, will result in a total transparency of 25% (64/255).

6.26.4 Usage

Like all data representing objects in [QCustomPlot](#), the [QCPCColorMap](#) is a plottable ([QCPAbstractPlottable](#)). So the plottable-interface of [QCustomPlot](#) applies ([QCustomPlot::plottable](#), [QCustomPlot::removePlottable](#), etc.)

Usually, you first create an instance:

which registers it with the [QCustomPlot](#) instance of the passed axes. Note that this [QCustomPlot](#) instance takes ownership of the plottable, so do not delete it manually but use [QCustomPlot::removePlottable\(\)](#) instead. The newly created plottable can be modified, e.g.:

Note

The [QCPCColorMap](#) always displays the data at equal key/value intervals, even if the key or value axis is set to a logarithmic scaling. If you want to use [QCPCColorMap](#) with logarithmic axes, you shouldn't use the [QCPCColorMapData::setData](#) method as it uses a linear transformation to determine the cell index. Rather directly access the cell index with [QCPCColorMapData::setCell](#).

6.26.5 Constructor & Destructor Documentation

6.26.5.1 QCPColorMap()

```
QCPColorMap::QCPColorMap (
    QCPAxis * keyAxis,
    QCPAxis * valueAxis ) [explicit]
```

Constructs a color map with the specified *keyAxis* and *valueAxis*.

The created [QCPColorMap](#) is automatically registered with the [QCustomPlot](#) instance inferred from *keyAxis*. This [QCustomPlot](#) instance takes ownership of the [QCPColorMap](#), so do not delete it manually but use [QCustomPlot::removePlottable\(\)](#) instead.

6.26.6 Member Function Documentation

6.26.6.1 data()

```
QCPColorMapData * QCPColorMap::data ( ) const [inline]
```

Returns a pointer to the internal data storage of type [QCPColorMapData](#). Access this to modify data points (cells) and the color map key/value range.

See also

[setData](#)

6.26.6.2 dataRangeChanged

```
void QCPColorMap::dataRangeChanged (
    const QCPRange & newRange ) [signal]
```

This signal is emitted when the data range changes.

See also

[setDataRange](#)

6.26.6.3 dataScaleTypeChanged

```
void QCPColorMap::dataScaleTypeChanged (
    QCPAxis::ScaleType scaleType ) [signal]
```

This signal is emitted when the data scale type changes.

See also

[setDataScaleType](#)

6.26.6.4 getKeyRange()

```
QCPRange QCPColorMap::getKeyRange (
    bool & foundRange,
    QCP::SignDomain inSignDomain = QCP::sdBoth ) const [virtual]
```

Returns the coordinate range that all data in this plottable span in the key axis dimension. For logarithmic plots, one can set *inSignDomain* to either [QCP::sdNegative](#) or [QCP::sdPositive](#) in order to restrict the returned range to that sign domain. E.g. when only negative range is wanted, set *inSignDomain* to [QCP::sdNegative](#) and all positive points will be ignored for range calculation. For no restriction, just set *inSignDomain* to [QCP::sdBoth](#) (default). *foundRange* is an output parameter that indicates whether a range could be found or not. If this is false, you shouldn't use the returned range (e.g. no points in data).

Note that *foundRange* is not the same as [QCPRange::validRange](#), since the range returned by this function may have size zero (e.g. when there is only one data point). In this case *foundRange* would return true, but the returned range is not a valid range in terms of [QCPRange::validRange](#).

See also

[rescaleAxes](#), [getValueRange](#)

Implements [QCPAbstractPlottable](#).

6.26.6.5 getValueRange()

```
QCPRange QCPColorMap::getValueRange (
    bool & foundRange,
    QCP::SignDomain inSignDomain = QCP::sdBoth,
    const QCPRange & inKeyRange = QCPRange() ) const [virtual]
```

Returns the coordinate range that the data points in the specified key range (*inKeyRange*) span in the value axis dimension. For logarithmic plots, one can set *inSignDomain* to either [QCP::sdNegative](#) or [QCP::sdPositive](#) in order to restrict the returned range to that sign domain. E.g. when only negative range is wanted, set *inSignDomain* to [QCP::sdNegative](#) and all positive points will be ignored for range calculation. For no restriction, just set *inSignDomain* to [QCP::sdBoth](#) (default). *foundRange* is an output parameter that indicates whether a range could be found or not. If this is false, you shouldn't use the returned range (e.g. no points in data).

If *inKeyRange* has both lower and upper bound set to zero (is equal to [QCPRange\(\)](#)), all data points are considered, without any restriction on the keys.

Note that *foundRange* is not the same as [QCPRange::validRange](#), since the range returned by this function may have size zero (e.g. when there is only one data point). In this case *foundRange* would return true, but the returned range is not a valid range in terms of [QCPRange::validRange](#).

See also

[rescaleAxes](#), [getKeyRange](#)

Implements [QCPAbstractPlottable](#).

6.26.6.6 gradientChanged

```
void QCPCoLorMap::gradientChanged (
    const QCPCoLorGradient & newGradient ) [signal]
```

This signal is emitted when the gradient changes.

See also

[setGradient](#)

6.26.6.7 rescaleDataRange()

```
void QCPCoLorMap::rescaleDataRange (
    bool recalculateDataBounds = false )
```

Sets the data range ([setDataRange](#)) to span the minimum and maximum values that occur in the current data set. This corresponds to the [rescaleKeyAxis](#) or [rescaleValueAxis](#) methods, only for the third data dimension of the color map.

The minimum and maximum values of the data set are buffered in the internal [QCPCoLorMapData](#) instance ([data](#)). As data is updated via its [QCPCoLorMapData::setCell](#) or [QCPCoLorMapData::setData](#), the buffered minimum and maximum values are updated, too. For performance reasons, however, they are only updated in an expanding fashion. So the buffered maximum can only increase and the buffered minimum can only decrease. In consequence, changes to the data that actually lower the maximum of the data set (by overwriting the cell holding the current maximum with a smaller value), aren't recognized and the buffered maximum overestimates the true maximum of the data set. The same happens for the buffered minimum. To recalculate the true minimum and maximum by explicitly looking at each cell, the method [QCPCoLorMapData::recalculateDataBounds](#) can be used. For convenience, setting the parameter *recalculateDataBounds* calls this method before setting the data range to the buffered minimum and maximum.

See also

[setDataRange](#)

6.26.6.8 selectTest()

```
double QCPCoLorMap::selectTest (
    const QPointF & pos,
    bool onlySelectable,
    QVariant * details = 0 ) const [virtual]
```

This function is used to decide whether a click hits a layerable object or not.

pos is a point in pixel coordinates on the [QCustomPlot](#) surface. This function returns the shortest pixel distance of this point to the object. If the object is either invisible or the distance couldn't be determined, -1.0 is returned. Further, if *onlySelectable* is true and the object is not selectable, -1.0 is returned, too.

If the object is represented not by single lines but by an area like a [QCItemText](#) or the bars of a [QCPBars](#) plottable, a click inside the area should also be considered a hit. In these cases this function thus returns a constant value greater zero but still below the parent plot's selection tolerance. (typically the selectionTolerance multiplied by 0.99).

Providing a constant value for area objects allows selecting line objects even when they are obscured by such area objects, by clicking close to the lines (i.e. closer than $0.99 \cdot \text{selectionTolerance}$).

The actual setting of the selection state is not done by this function. This is handled by the parent [QCustomPlot](#) when the `mouseReleaseEvent` occurs, and the finally selected object is notified via the `selectEvent/deselectEvent` methods.

details is an optional output parameter. Every layerable subclass may place any information in *details*. This information will be passed to `selectEvent` when the parent [QCustomPlot](#) decides on the basis of this `selectTest` call, that the object was successfully selected. The subsequent call to `selectEvent` will carry the *details*. This is useful for multi-part objects (like [QCPAxis](#)). This way, a possibly complex calculation to decide which part was clicked is only done once in `selectTest`. The result (i.e. the actually clicked part) can then be placed in *details*. So in the subsequent `selectEvent`, the decision which part was selected doesn't have to be done a second time for a single selection operation.

You may pass 0 as *details* to indicate that you are not interested in those selection details.

See also

`selectEvent`, `deselectEvent`, [mousePressEvent](#), [wheelEvent](#), [QCustomPlot::setInteractions](#)

Implements [QCPAbstractPlottable](#).

6.26.6.9 setColorScale()

```
void QCPColorMap::setColorScale (
    QCPColorScale * colorScale )
```

Associates the color scale *colorScale* with this color map.

This means that both the color scale and the color map synchronize their gradient, data range and data scale type ([setGradient](#), [setDataRange](#), [setDataScaleType](#)). Multiple color maps can be associated with one single color scale. This causes the color maps to also synchronize those properties, via the mutual color scale.

This function causes the color map to adopt the current color gradient, data range and data scale type of *colorScale*. After this call, you may change these properties at either the color map or the color scale, and the setting will be applied to both.

Pass 0 as *colorScale* to disconnect the color scale from this color map again.

6.26.6.10 setData()

```
void QCPColorMap::setData (
    QCPColorMapData * data,
    bool copy = false )
```

Replaces the current [data](#) with the provided *data*.

If *copy* is set to true, the *data* object will only be copied. If false, the color map takes ownership of the passed data and replaces the internal data pointer with it. This is significantly faster than copying for large datasets.

6.26.6.11 setDataRange()

```
void QCPColorMap::setDataRange (
    const QCPRange & dataRange )
```

Sets the data range of this color map to *dataRange*. The data range defines which data values are mapped to the color gradient.

To make the data range span the full range of the data set, use [rescaleDataRange](#).

See also

[QCPColorScale::setDataRange](#)

6.26.6.12 setDataScaleType()

```
void QCPColorMap::setDataScaleType (
    QCPAxis::ScaleType scaleType )
```

Sets whether the data is correlated with the color gradient linearly or logarithmically.

See also

[QCPColorScale::setDataScaleType](#)

6.26.6.13 setGradient()

```
void QCPColorMap::setGradient (
    const QCPColorGradient & gradient )
```

Sets the color gradient that is used to represent the data. For more details on how to create an own gradient or use one of the preset gradients, see [QCPColorGradient](#).

The colors defined by the gradient will be used to represent data values in the currently set data range, see [setDataRange](#). Data points that are outside this data range will either be colored uniformly with the respective gradient boundary color, or the gradient will repeat, depending on [QCPColorGradient::setPeriodic](#).

See also

[QCPColorScale::setGradient](#)

6.26.6.14 `setInterpolate()`

```
void QCPColormap::setInterpolate (
    bool enabled )
```

Sets whether the color map image shall use bicubic interpolation when displaying the color map shrunk or expanded, and not at a 1:1 pixel-to-data scale.

6.26.6.15 `setTightBoundary()`

```
void QCPColormap::setTightBoundary (
    bool enabled )
```

Sets whether the outer most data rows and columns are clipped to the specified key and value range (see [QCPColorMapData::setKeyRange](#), [QCPColorMapData::setValueRange](#)).

if *enabled* is set to false, the data points at the border of the color map are drawn with the same width and height as all other data points. Since the data points are represented by rectangles of one color centered on the data coordinate, this means that the shown color map extends by half a data point over the specified key/value range in each direction.

6.26.6.16 `updateLegendIcon()`

```
void QCPColormap::updateLegendIcon (
    Qt::TransformationMode transformMode = Qt::SmoothTransformation,
    const QSize & thumbSize = QSize(32, 18) )
```

Takes the current appearance of the color map and updates the legend icon, which is used to represent this color map in the legend (see [QCPLegend](#)).

The *transformMode* specifies whether the rescaling is done by a faster, low quality image scaling algorithm (Qt::FastTransformation) or by a slower, higher quality algorithm (Qt::SmoothTransformation).

The current color map appearance is scaled down to *thumbSize*. Ideally, this should be equal to the size of the legend icon (see [QCPLegend::setIconSize](#)). If it isn't exactly the configured legend icon size, the thumb will be rescaled during drawing of the legend item.

See also

[setDataRange](#)

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

- `/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h`
- `/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp`

6.27 QCPColormapData Class Reference

Holds the two-dimensional data of a [QCPColormap](#) plottable.

Public Member Functions

- [QCPColormapData](#) (int keySize, int valueSize, const [QCPRange](#) &keyRange, const [QCPRange](#) &valueRange)
- [QCPColormapData](#) (const [QCPColormapData](#) &other)
- [QCPColormapData](#) & operator= (const [QCPColormapData](#) &other)
- int **keySize** () const
- int **valueSize** () const
- [QCPRange](#) **keyRange** () const
- [QCPRange](#) **valueRange** () const
- [QCPRange](#) **dataBounds** () const
- double **data** (double key, double value)
- double **cell** (int keyIndex, int valueIndex)
- unsigned char **alpha** (int keyIndex, int valueIndex)
- void **setSize** (int keySize, int valueSize)
- void **setKeySize** (int keySize)
- void **setValueSize** (int valueSize)
- void **setRange** (const [QCPRange](#) &keyRange, const [QCPRange](#) &valueRange)
- void **setKeyRange** (const [QCPRange](#) &keyRange)
- void **setValueRange** (const [QCPRange](#) &valueRange)
- void **setData** (double key, double value, double z)
- void **setCell** (int keyIndex, int valueIndex, double z)
- void **setAlpha** (int keyIndex, int valueIndex, unsigned char [alpha](#))
- void **recalculateDataBounds** ()
- void **clear** ()
- void **clearAlpha** ()
- void **fill** (double z)
- void **fillAlpha** (unsigned char [alpha](#))
- bool **isEmpty** () const
- void **coordToCell** (double key, double value, int *keyIndex, int *valueIndex) const
- void **cellToCoord** (int keyIndex, int valueIndex, double *key, double *value) const

Protected Member Functions

- bool **createAlpha** (bool initializeOpaque=true)

Protected Attributes

- int **mKeySize**
- int **mValueSize**
- [QCPRange](#) **mKeyRange**
- [QCPRange](#) **mValueRange**
- bool **mIsEmpty**
- double * **mData**
- unsigned char * **mAlpha**
- [QCPRange](#) **mDataBounds**
- bool **mDataModified**

Friends

- class **QCPCColorMap**

6.27.1 Detailed Description

Holds the two-dimensional data of a [QCPCColorMap](#) plottable.

This class is a data storage for [QCPCColorMap](#). It holds a two-dimensional array, which [QCPCColorMap](#) then displays as a 2D image in the plot, where the array values are represented by a color, depending on the value.

The size of the array can be controlled via [setSize](#) (or [setKeySize](#), [setValueSize](#)). Which plot coordinates these cells correspond to can be configured with [setRange](#) (or [setKeyRange](#), [setValueRange](#)).

The data cells can be accessed in two ways: They can be directly addressed by an integer index with [setCell](#). This is the fastest method. Alternatively, they can be addressed by their plot coordinate with [setData](#). plot coordinate to cell index transformations and vice versa are provided by the functions [coordToCell](#) and [cellToCoord](#).

A [QCPCColorMapData](#) also holds an on-demand two-dimensional array of alpha values which (if allocated) has the same size as the data map. It can be accessed via [setAlpha](#), [fillAlpha](#) and [clearAlpha](#). The memory for the alpha map is only allocated if needed, i.e. on the first call of [setAlpha](#). [clearAlpha](#) restores full opacity and frees the alpha map.

This class also buffers the minimum and maximum values that are in the data set, to provide [QCPCColorMap::rescaleDataRange](#) with the necessary information quickly. Setting a cell to a value that is greater than the current maximum increases this maximum to the new value. However, setting the cell that currently holds the maximum value to a smaller value doesn't decrease the maximum again, because finding the true new maximum would require going through the entire data array, which might be time consuming. The same holds for the data minimum. This functionality is given by [recalculateDataBounds](#), such that you can decide when it is sensible to find the true current minimum and maximum. The method [QCPCColorMap::rescaleDataRange](#) offers a convenience parameter *recalculateDataBounds* which may be set to true to automatically call [recalculateDataBounds](#) internally.

6.27.2 Constructor & Destructor Documentation

6.27.2.1 QCPCColorMapData() [1/2]

```
QCPCColorMapData::QCPCColorMapData (
    int keySize,
    int valueSize,
    const QCPRange & keyRange,
    const QCPRange & valueRange )
```

Constructs a new [QCPCColorMapData](#) instance. The instance has *keySize* cells in the key direction and *valueSize* cells in the value direction. These cells will be displayed by the [QCPCColorMap](#) at the coordinates *keyRange* and *valueRange*.

See also

[setSize](#), [setKeySize](#), [setValueSize](#), [setRange](#), [setKeyRange](#), [setValueRange](#)

6.27.2.2 QCPColormapData() [2/2]

```
QCPColormapData::QCPColormapData (
    const QCPColormapData & other )
```

Constructs a new [QCPColormapData](#) instance copying the data and range of *other*.

6.27.3 Member Function Documentation

6.27.3.1 alpha()

```
unsigned char QCPColormapData::alpha (
    int keyIndex,
    int valueIndex )
```

Returns the alpha map value of the cell with the indices *keyIndex* and *valueIndex*.

If this color map data doesn't have an alpha map (because [setAlpha](#) was never called after creation or after a call to [clearAlpha](#)), returns 255, which corresponds to full opacity.

See also

[setAlpha](#)

6.27.3.2 cellToCoord()

```
void QCPColormapData::cellToCoord (
    int keyIndex,
    int valueIndex,
    double * key,
    double * value ) const
```

Transforms cell indices given by *keyIndex* and *valueIndex* to cell indices of this [QCPColormapData](#) instance. The resulting coordinates are returned via the output parameters *key* and *value*.

If you are only interested in a key or value coordinate, you may pass 0 as *key* or *value*.

Note

The [QCPColormap](#) always displays the data at equal key/value intervals, even if the key or value axis is set to a logarithmic scaling. If you want to use [QCPColormap](#) with logarithmic axes, you shouldn't use the [QCPColormapData::cellToCoord](#) method as it uses a linear transformation to determine the cell index.

See also

[coordToCell](#), [QCPColormap::pixelToCoord](#)

6.27.3.3 clear()

```
void QCPColormapData::clear ( )
```

Frees the internal data memory.

This is equivalent to calling [setSize\(0, 0\)](#).

6.27.3.4 clearAlpha()

```
void QCPColormapData::clearAlpha ( )
```

Frees the internal alpha map. The color map will have full opacity again.

6.27.3.5 coordToCell()

```
void QCPColormapData::coordToCell (
    double key,
    double value,
    int * keyIndex,
    int * valueIndex ) const
```

Transforms plot coordinates given by *key* and *value* to cell indices of this [QCPColormapData](#) instance. The resulting cell indices are returned via the output parameters *keyIndex* and *valueIndex*.

The retrieved key/value cell indices can then be used for example with [setCell](#).

If you are only interested in a key or value index, you may pass 0 as *valueIndex* or *keyIndex*.

Note

The [QCPColormap](#) always displays the data at equal key/value intervals, even if the key or value axis is set to a logarithmic scaling. If you want to use [QCPColormap](#) with logarithmic axes, you shouldn't use the [QCPColormapData::coordToCell](#) method as it uses a linear transformation to determine the cell index.

See also

[cellToCoord](#), [QCPColormap::coordToPixel](#)

6.27.3.6 fill()

```
void QCPColormapData::fill (
    double z )
```

Sets all cells to the value *z*.

6.27.3.7 fillAlpha()

```
void QCPColorMapData::fillAlpha (
    unsigned char alpha )
```

Sets the opacity of all color map cells to *alpha*. A value of 0 for *alpha* results in a fully transparent color map, and a value of 255 results in a fully opaque color map.

If you wish to restore opacity to 100% and free any used memory for the alpha map, rather use [clearAlpha](#).

See also

[setAlpha](#)

6.27.3.8 isEmpty()

```
bool QCPColorMapData::isEmpty ( ) const [inline]
```

Returns whether this instance carries no data. This is equivalent to having a size where at least one of the dimensions is 0 (see [setSize](#)).

6.27.3.9 operator=()

```
QCPColorMapData & QCPColorMapData::operator= (
    const QCPColorMapData & other )
```

Overwrites this color map data instance with the data stored in *other*. The alpha map state is transferred, too.

6.27.3.10 recalculateDataBounds()

```
void QCPColorMapData::recalculateDataBounds ( )
```

Goes through the data and updates the buffered minimum and maximum data values.

Calling this method is only advised if you are about to call [QCPColorMap::rescaleDataRange](#) and can not guarantee that the cells holding the maximum or minimum data haven't been overwritten with a smaller or larger value respectively, since the buffered maximum/minimum values have been updated the last time. Why this is the case is explained in the class description ([QCPColorMapData](#)).

Note that the method [QCPColorMap::rescaleDataRange](#) provides a parameter *recalculateDataBounds* for convenience. Setting this to true will call this method for you, before doing the rescale.

6.27.3.11 setAlpha()

```
void QCPColorMapData::setAlpha (
    int keyIndex,
    int valueIndex,
    unsigned char alpha )
```

Sets the alpha of the color map cell given by *keyIndex* and *valueIndex* to *alpha*. A value of 0 for *alpha* results in a fully transparent cell, and a value of 255 results in a fully opaque cell.

If an alpha map doesn't exist yet for this color map data, it will be created here. If you wish to restore full opacity and free any allocated memory of the alpha map, call [clearAlpha](#).

Note that the cell-wise alpha which can be configured here is independent of any alpha configured in the color map's gradient ([QCPColorGradient](#)). If a cell is affected both by the cell-wise and gradient alpha, the alpha values will be blended accordingly during rendering of the color map.

See also

[fillAlpha](#), [clearAlpha](#)

6.27.3.12 setCell()

```
void QCPColorMapData::setCell (
    int keyIndex,
    int valueIndex,
    double z )
```

Sets the data of the cell with indices *keyIndex* and *valueIndex* to *z*. The indices enumerate the cells starting from zero, up to the map's size-1 in the respective dimension (see [setSize](#)).

In the standard plot configuration (horizontal key axis and vertical value axis, both not range-reversed), the cell with indices (0, 0) is in the bottom left corner and the cell with indices (keySize-1, valueSize-1) is in the top right corner of the color map.

See also

[setData](#), [setSize](#)

6.27.3.13 setData()

```
void QCPColorMapData::setData (
    double key,
    double value,
    double z )
```

Sets the data of the cell, which lies at the plot coordinates given by *key* and *value*, to *z*.

Note

The [QCPColorMap](#) always displays the data at equal key/value intervals, even if the key or value axis is set to a logarithmic scaling. If you want to use [QCPColorMap](#) with logarithmic axes, you shouldn't use the [QCPColorMapData::setData](#) method as it uses a linear transformation to determine the cell index. Rather directly access the cell index with [QCPColorMapData::setCell](#).

See also

[setCell](#), [setRange](#)

6.27.3.14 setKeyRange()

```
void QCPColorMapData::setKeyRange (
    const QCPRange & keyRange )
```

Sets the coordinate range the data shall be distributed over in the key dimension. Together with the value range, This defines the rectangular area covered by the color map in plot coordinates.

The outer cells will be centered on the range boundaries given to this function. For example, if the key size ([setKeySize](#)) is 3 and *keyRange* is set to [QCPRange \(2, 3\)](#) there will be cells centered on the key coordinates 2, 2.5 and 3.

See also

[setRange](#), [setValueRange](#), [setSize](#)

6.27.3.15 setKeySize()

```
void QCPColorMapData::setKeySize (
    int keySize )
```

Resizes the data array to have *keySize* cells in the key dimension.

The current data is discarded and the map cells are set to 0, unless the map had already the requested size.

Setting *keySize* to zero frees the internal data array and [isEmpty](#) returns true.

See also

[setKeyRange](#), [setSize](#), [setValueSize](#)

6.27.3.16 setRange()

```
void QCPColorMapData::setRange (
    const QCPRange & keyRange,
    const QCPRange & valueRange )
```

Sets the coordinate ranges the data shall be distributed over. This defines the rectangular area covered by the color map in plot coordinates.

The outer cells will be centered on the range boundaries given to this function. For example, if the key size ([setKeySize](#)) is 3 and *keyRange* is set to [QCPRange \(2, 3\)](#) there will be cells centered on the key coordinates 2, 2.5 and 3.

See also

[setSize](#)

6.27.3.17 setSize()

```
void QCPColormapData::setSize (
    int keySize,
    int valueSize )
```

Resizes the data array to have *keySize* cells in the key dimension and *valueSize* cells in the value dimension.

The current data is discarded and the map cells are set to 0, unless the map had already the requested size.

Setting at least one of *keySize* or *valueSize* to zero frees the internal data array and [isEmpty](#) returns true.

See also

[setRange](#), [setKeySize](#), [setValueSize](#)

6.27.3.18 setValueRange()

```
void QCPColormapData::setValueRange (
    const QCPRange & valueRange )
```

Sets the coordinate range the data shall be distributed over in the value dimension. Together with the key range, This defines the rectangular area covered by the color map in plot coordinates.

The outer cells will be centered on the range boundaries given to this function. For example, if the value size ([set↵ValueSize](#)) is 3 and *valueRange* is set to [QCPRange](#) (2, 3) there will be cells centered on the value coordinates 2, 2.5 and 3.

See also

[setRange](#), [setKeyRange](#), [setSize](#)

6.27.3.19 setValueSize()

```
void QCPColormapData::setValueSize (
    int valueSize )
```

Resizes the data array to have *valueSize* cells in the value dimension.

The current data is discarded and the map cells are set to 0, unless the map had already the requested size.

Setting *valueSize* to zero frees the internal data array and [isEmpty](#) returns true.

See also

[setValueRange](#), [setSize](#), [setKeySize](#)

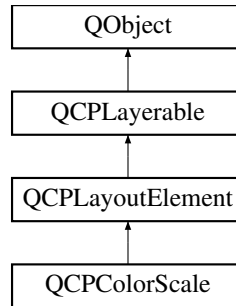
The documentation for this class was generated from the following files:

- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.↵h
- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.↵cpp

6.28 QCPCColorScale Class Reference

A color scale for use with color coding data such as [QCPCColorMap](#).

Inheritance diagram for QCPCColorScale:



Signals

- void [dataRangeChanged](#) (const [QCPCRange](#) &newRange)
- void [dataScaleTypeChanged](#) ([QCPCAxis::ScaleType](#) scaleType)
- void [gradientChanged](#) (const [QCPCColorGradient](#) &newGradient)

Public Member Functions

- [QCPCColorScale](#) ([QCustomPlot](#) *parentPlot)
- [QCPCAxis](#) * [axis](#) () const
- [QCPCAxis::AxisType](#) [type](#) () const
- [QCPCRange](#) [dataRange](#) () const
- [QCPCAxis::ScaleType](#) [dataScaleType](#) () const
- [QCPCColorGradient](#) [gradient](#) () const
- [QString](#) [label](#) () const
- int [barWidth](#) () const
- bool [rangeDrag](#) () const
- bool [rangeZoom](#) () const
- void [setType](#) ([QCPCAxis::AxisType](#) type)
- Q_SLOT void [setDataRange](#) (const [QCPCRange](#) &dataRange)
- Q_SLOT void [setDataScaleType](#) ([QCPCAxis::ScaleType](#) scaleType)
- Q_SLOT void [setGradient](#) (const [QCPCColorGradient](#) &gradient)
- void [setLabel](#) (const [QString](#) &str)
- void [setBarWidth](#) (int width)
- void [setRangeDrag](#) (bool enabled)
- void [setRangeZoom](#) (bool enabled)
- [QList](#)< [QCPCColorMap](#) * > [colorMaps](#) () const
- void [rescaleDataRange](#) (bool onlyVisibleMaps)
- virtual void [update](#) ([UpdatePhase](#) phase) Q_DECL_OVERRIDE

Protected Member Functions

- virtual void [applyDefaultAntialiasingHint](#) ([QCPPainter](#) *painter) const Q_DECL_OVERRIDE
- virtual void [mousePressEvent](#) ([QMouseEvent](#) *event, const [QVariant](#) &details) Q_DECL_OVERRIDE
- virtual void [mouseMoveEvent](#) ([QMouseEvent](#) *event, const [QPointF](#) &startPos) Q_DECL_OVERRIDE
- virtual void [mouseReleaseEvent](#) ([QMouseEvent](#) *event, const [QPointF](#) &startPos) Q_DECL_OVERRIDE
- virtual void [wheelEvent](#) ([QWheelEvent](#) *event) Q_DECL_OVERRIDE

Protected Attributes

- [QCPAxis::AxisType](#) **mType**
- [QCPRange](#) **mDataRange**
- [QCPAxis::ScaleType](#) **mDataScaleType**
- [QCPColorGradient](#) **mGradient**
- **int mBarWidth**
- [QPointer< QCPColorScaleAxisRectPrivate >](#) **mAxisRect**
- [QPointer< QCPAxis >](#) **mColorAxis**

Friends

- class **QCPColorScaleAxisRectPrivate**

Additional Inherited Members

6.28.1 Detailed Description

A color scale for use with color coding data such as [QCPColorMap](#).

This layout element can be placed on the plot to correlate a color gradient with data values. It is usually used in combination with one or multiple [QCPColorMaps](#).

The color scale can be either horizontal or vertical, as shown in the image above. The orientation and the side where the numbers appear is controlled with [setType](#).

Use [QCPColorMap::setColorScale](#) to connect a color map with a color scale. Once they are connected, they share their gradient, data range and data scale type ([setGradient](#), [setDataRange](#), [setDataScaleType](#)). Multiple color maps may be associated with a single color scale, to make them all synchronize these properties.

To have finer control over the number display and axis behaviour, you can directly access the [axis](#). See the documentation of [QCPAxis](#) for details about configuring axes. For example, if you want to change the number of automatically generated ticks, call

Placing a color scale next to the main axis rect works like with any other layout element:

In this case we have placed it to the right of the default axis rect, so it wasn't necessary to call [setType](#), since [QCPAxis::atRight](#) is already the default. The text next to the color scale can be set with [setLabel](#).

For optimum appearance (like in the image above), it may be desirable to line up the axis rect and the borders of the color scale. Use a [QCPMarginGroup](#) to achieve this:

Color scales are initialized with a non-zero minimum top and bottom margin ([setMinimumMargins](#)), because vertical color scales are most common and the minimum top/bottom margin makes sure it keeps some distance to the top/bottom widget border. So if you change to a horizontal color scale by setting [setType](#) to [QCPAxis::atBottom](#) or [QCPAxis::atTop](#), you might want to also change the minimum margins accordingly, e.g. `setMinimumMargins(QMargins(6, 0, 6, 0))`.

6.28.2 Constructor & Destructor Documentation

6.28.2.1 QCPColorScale()

```
QCPColorScale::QCPColorScale (
    QCustomPlot * parentPlot ) [explicit]
```

Constructs a new [QCPColorScale](#).

6.28.3 Member Function Documentation

6.28.3.1 axis()

```
QCPAxis * QCPColorScale::axis ( ) const [inline]
```

Returns the internal [QCPAxis](#) instance of this color scale. You can access it to alter the appearance and behaviour of the axis. [QCPColorScale](#) duplicates some properties in its interface for convenience. Those are [setDataRange](#) ([QCPAxis::setRange](#)), [setDataScaleType](#) ([QCPAxis::setScaleType](#)), and the method [setLabel](#) ([QCPAxis::setLabel](#)). As they each are connected, it does not matter whether you use the method on the [QCPColorScale](#) or on its [QCPAxis](#).

If the type of the color scale is changed with [setType](#), the axis returned by this method will change, too, to either the left, right, bottom or top axis, depending on which type was set.

6.28.3.2 colorMaps()

```
QList< QCPColorMap * > QCPColorScale::colorMaps ( ) const
```

Returns a list of all the color maps associated with this color scale.

6.28.3.3 dataRangeChanged

```
void QCPColorScale::dataRangeChanged (
    const QCPRange & newRange ) [signal]
```

This signal is emitted when the data range changes.

See also

[setDataRange](#)

6.28.3.4 dataScaleTypeChanged

```
void QCPColorScale::dataScaleTypeChanged (
    QCPAxis::ScaleType scaleType ) [signal]
```

This signal is emitted when the data scale type changes.

See also

[setDataScaleType](#)

6.28.3.5 gradientChanged

```
void QCPColorScale::gradientChanged (
    const QCPColorGradient & newGradient ) [signal]
```

This signal is emitted when the gradient changes.

See also

[setGradient](#)

6.28.3.6 mouseMoveEvent()

```
void QCPColorScale::mouseMoveEvent (
    QMouseEvent * event,
    const QPointF & startPos ) [protected], [virtual]
```

This event gets called when the user moves the mouse while holding a mouse button, after this layerable has become the mouse grabber by accepting the preceding [mousePressEvent](#).

The current pixel position of the cursor on the [QCustomPlot](#) widget is accessible via `event->pos()`. The parameter *startPos* indicates the position where the initial [mousePressEvent](#) occurred, that started the mouse interaction.

The default implementation does nothing.

See also

[mousePressEvent](#), [mouseReleaseEvent](#), [mouseDoubleClickEvent](#), [wheelEvent](#)

Reimplemented from [QCPLayerable](#).

6.28.3.7 mousePressEvent()

```
void QCPColorScale::mousePressEvent (
    QMouseEvent * event,
    const QVariant & details ) [protected], [virtual]
```

This event gets called when the user presses a mouse button while the cursor is over the layerable. Whether a cursor is over the layerable is decided by a preceding call to [selectTest](#).

The current pixel position of the cursor on the [QCustomPlot](#) widget is accessible via `event->pos()`. The parameter *details* contains layerable-specific details about the hit, which were generated in the previous call to [selectTest](#). For example, One-dimensional plottables like [QCPGraph](#) or [QCPBars](#) convey the clicked data point in the *details* parameter, as [QCPDataSelection](#) packed as `QVariant`. Multi-part objects convey the specific `SelectablePart` that was hit (e.g. [QCPAxis::SelectablePart](#) in the case of axes).

[QCustomPlot](#) uses an event propagation system that works the same as Qt's system. If your layerable doesn't reimplement the [mousePressEvent](#) or explicitly calls `event->ignore()` in its reimplementation, the event will be propagated to the next layerable in the stacking order.

Once a layerable has accepted the [mousePressEvent](#), it is considered the mouse grabber and will receive all following calls to [mouseMoveEvent](#) or [mouseReleaseEvent](#) for this mouse interaction (a "mouse interaction" in this context ends with the release).

The default implementation does nothing except explicitly ignoring the event with `event->ignore()`.

See also

[mouseMoveEvent](#), [mouseReleaseEvent](#), [mouseDoubleClickEvent](#), [wheelEvent](#)

Reimplemented from [QCPLayerable](#).

6.28.3.8 mouseReleaseEvent()

```
void QCPColorScale::mouseReleaseEvent (
    QMouseEvent * event,
    const QPointF & startPos ) [protected], [virtual]
```

This event gets called when the user releases the mouse button, after this layerable has become the mouse grabber by accepting the preceding [mousePressEvent](#).

The current pixel position of the cursor on the [QCustomPlot](#) widget is accessible via `event->pos()`. The parameter *startPos* indicates the position where the initial [mousePressEvent](#) occurred, that started the mouse interaction.

The default implementation does nothing.

See also

[mousePressEvent](#), [mouseMoveEvent](#), [mouseDoubleClickEvent](#), [wheelEvent](#)

Reimplemented from [QCPLayerable](#).

6.28.3.9 rescaleDataRange()

```
void QCPColorScale::rescaleDataRange (
    bool onlyVisibleMaps )
```

Changes the data range such that all color maps associated with this color scale are fully mapped to the gradient in the data dimension.

See also

[setDataRange](#)

6.28.3.10 setBarWidth()

```
void QCPColorScale::setBarWidth (
    int width )
```

Sets the width (or height, for horizontal color scales) the bar where the gradient is displayed will have.

6.28.3.11 setDataRange()

```
void QCPColorScale::setDataRange (
    const QCPRange & dataRange )
```

Sets the range spanned by the color gradient and that is shown by the axis in the color scale.

It is equivalent to calling [QCPColorMap::setDataRange](#) on any of the connected color maps. It is also equivalent to directly accessing the [axis](#) and setting its range with [QCPAxis::setRange](#).

See also

[setDataScaleType](#), [setGradient](#), [rescaleDataRange](#)

6.28.3.12 setDataScaleType()

```
void QCPColorScale::setDataScaleType (
    QCPAxis::ScaleType scaleType )
```

Sets the scale type of the color scale, i.e. whether values are linearly associated with colors or logarithmically.

It is equivalent to calling [QCPColorMap::setDataScaleType](#) on any of the connected color maps. It is also equivalent to directly accessing the [axis](#) and setting its scale type with [QCPAxis::setScaleType](#).

See also

[setDataRange](#), [setGradient](#)

6.28.3.13 setGradient()

```
void QCPColorScale::setGradient (
    const QCPColorGradient & gradient )
```

Sets the color gradient that will be used to represent data values.

It is equivalent to calling [QCPColorMap::setGradient](#) on any of the connected color maps.

See also

[setDataRange](#), [setDataScaleType](#)

6.28.3.14 setLabel()

```
void QCPColorScale::setLabel (
    const QString & str )
```

Sets the axis label of the color scale. This is equivalent to calling [QCPAxis::setLabel](#) on the internal [axis](#).

6.28.3.15 setRangeDrag()

```
void QCPColorScale::setRangeDrag (
    bool enabled )
```

Sets whether the user can drag the data range ([setDataRange](#)).

Note that [QCP::iRangeDrag](#) must be in the [QCustomPlot](#)'s interactions ([QCustomPlot::setInteractions](#)) to allow range dragging.

6.28.3.16 setRangeZoom()

```
void QCPColorScale::setRangeZoom (
    bool enabled )
```

Sets whether the user can zoom the data range ([setDataRange](#)) by scrolling the mouse wheel.

Note that [QCP::iRangeZoom](#) must be in the [QCustomPlot](#)'s interactions ([QCustomPlot::setInteractions](#)) to allow range dragging.

6.28.3.17 setType()

```
void QCPColorScale::setType (
    QCPAxis::AxisType type )
```

Sets at which side of the color scale the axis is placed, and thus also its orientation.

Note that after setting *type* to a different value, the axis returned by [axis\(\)](#) will be a different one. The new axis will adopt the following properties from the previous axis: The range, scale type, label and ticker (the latter will be shared and not copied).

6.28.3.18 update()

```
void QCPCColorScale::update (
    UpdatePhase phase ) [virtual]
```

Updates the layout element and sub-elements. This function is automatically called before every replot by the parent layout element. It is called multiple times, once for every [UpdatePhase](#). The phases are run through in the order of the enum values. For details about what happens at the different phases, see the documentation of [UpdatePhase](#).

Layout elements that have child elements should call the [update](#) method of their child elements, and pass the current *phase* unchanged.

The default implementation executes the automatic margin mechanism in the [upMargins](#) phase. Subclasses should make sure to call the base class implementation.

Reimplemented from [QCPLayoutElement](#).

6.28.3.19 wheelEvent()

```
void QCPCColorScale::wheelEvent (
    QWheelEvent * event ) [protected], [virtual]
```

This event gets called when the user turns the mouse scroll wheel while the cursor is over the layerable. Whether a cursor is over the layerable is decided by a preceding call to [selectTest](#).

The current pixel position of the cursor on the [QCustomPlot](#) widget is accessible via `event->pos()`.

The `event->delta()` indicates how far the mouse wheel was turned, which is usually +/- 120 for single rotation steps. However, if the mouse wheel is turned rapidly, multiple steps may accumulate to one event, making `event->delta()` larger. On the other hand, if the wheel has very smooth steps or none at all, the delta may be smaller.

The default implementation does nothing.

See also

[mousePressEvent](#), [mouseMoveEvent](#), [mouseReleaseEvent](#), [mouseDoubleClickEvent](#)

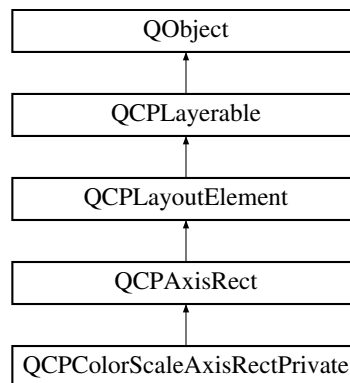
Reimplemented from [QCPLayerable](#).

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

- `/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h`↔
- `/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp`↔

6.29 QCPCColorScaleAxisRectPrivate Class Reference

Inheritance diagram for QCPCColorScaleAxisRectPrivate:



Public Member Functions

- [QCPCColorScaleAxisRectPrivate](#) ([QCPCColorScale](#) *parentColorScale)

Protected Member Functions

- virtual void **draw** ([QCPPainter](#) *painter)
- void **updateGradientImage** ()
- Q_SLOT void **axisSelectionChanged** (QCPAxis::SelectableParts selectedParts)
- Q_SLOT void **axisSelectableChanged** (QCPAxis::SelectableParts selectableParts)

Protected Attributes

- [QCPCColorScale](#) * **mParentColorScale**
- QImage **mGradientImage**
- bool **mGradientImageInvalidated**

Friends

- class **QCPCColorScale**

Additional Inherited Members

6.29.1 Constructor & Destructor Documentation

6.29.1.1 QCPColorScaleAxisRectPrivate()

```
QCPColorScaleAxisRectPrivate::QCPColorScaleAxisRectPrivate (
    QCPColorScale * parentColorScale ) [explicit]
```

Creates a new instance, as a child of *parentColorScale*.

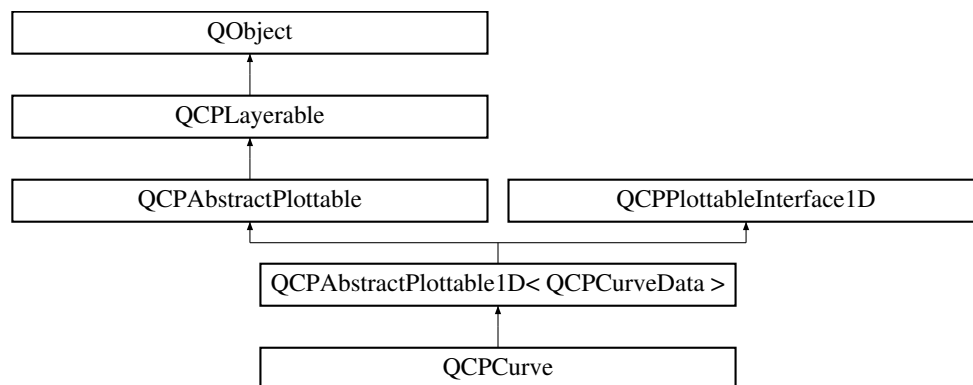
The documentation for this class was generated from the following files:

- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h
- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp

6.30 QCPCurve Class Reference

A plottable representing a parametric curve in a plot.

Inheritance diagram for QCPCurve:



Public Types

- enum [LineStyle](#) { [IsNone](#), [IsLine](#) }

Public Member Functions

- [QCPCurve](#) ([QCPAxis](#) *keyAxis, [QCPAxis](#) *valueAxis)
- [QSharedPointer](#)< [QCPCurveDataContainer](#) > [data](#) () const
- [QCPScatterStyle](#) [scatterStyle](#) () const
- int [scatterSkip](#) () const
- [LineStyle](#) [lineStyle](#) () const
- void [setData](#) ([QSharedPointer](#)< [QCPCurveDataContainer](#) > [data](#))
- void [setData](#) (const [QVector](#)< double > &t, const [QVector](#)< double > &keys, const [QVector](#)< double > &values, bool alreadySorted=false)
- void [setData](#) (const [QVector](#)< double > &keys, const [QVector](#)< double > &values)
- void [setScatterStyle](#) (const [QCPScatterStyle](#) &style)
- void [setScatterSkip](#) (int skip)

- void [setLineStyle](#) ([LineStyle](#) style)
- void [addData](#) (const QVector< double > &t, const QVector< double > &keys, const QVector< double > &values, bool alreadySorted=false)
- void [addData](#) (const QVector< double > &keys, const QVector< double > &values)
- void [addData](#) (double t, double key, double value)
- void [addData](#) (double key, double value)
- virtual double [selectTest](#) (const QPointF &pos, bool onlySelectable, QVariant *details=0) const Q_DECL_OVERRIDE
- virtual [QCPRange](#) [getKeyRange](#) (bool &foundRange, [QCP::SignDomain](#) inSignDomain=[QCP::sdBoth](#)) const Q_DECL_OVERRIDE
- virtual [QCPRange](#) [getValueRange](#) (bool &foundRange, [QCP::SignDomain](#) inSignDomain=[QCP::sdBoth](#), const [QCPRange](#) &inKeyRange=[QCPRange](#)()) const Q_DECL_OVERRIDE

Protected Member Functions

- virtual void **draw** ([QCPPainter](#) *painter) Q_DECL_OVERRIDE
- virtual void **drawLegendIcon** ([QCPPainter](#) *painter, const QRectF &rect) const Q_DECL_OVERRIDE
- virtual void **drawCurveLine** ([QCPPainter](#) *painter, const QVector< QPointF > &lines) const
- virtual void **drawScatterPlot** ([QCPPainter](#) *painter, const QVector< QPointF > &points, const [QCPScatterStyle](#) &style) const
- void **getCurveLines** (QVector< QPointF > *lines, const [QCPDataRange](#) &dataRange, double penWidth) const
- void **getScatters** (QVector< QPointF > *scatters, const [QCPDataRange](#) &dataRange, double scatterWidth) const
- int **getRegion** (double key, double value, double keyMin, double valueMax, double keyMax, double valueMin) const
- QPointF **getOptimizedPoint** (int prevRegion, double prevKey, double prevValue, double key, double value, double keyMin, double valueMax, double keyMax, double valueMin) const
- QVector< QPointF > **getOptimizedCornerPoints** (int prevRegion, int currentRegion, double prevKey, double prevValue, double key, double value, double keyMin, double valueMax, double keyMax, double valueMin) const
- bool **mayTraverse** (int prevRegion, int currentRegion) const
- bool **getTraverse** (double prevKey, double prevValue, double key, double value, double keyMin, double valueMax, double keyMax, double valueMin, QPointF &crossA, QPointF &crossB) const
- void **getTraverseCornerPoints** (int prevRegion, int currentRegion, double keyMin, double valueMax, double keyMax, double valueMin, QVector< QPointF > &beforeTraverse, QVector< QPointF > &afterTraverse) const
- double **pointDistance** (const QPointF &pixelPoint, QCPCurveDataContainer::const_iterator &closestData) const

Protected Attributes

- [QCPScatterStyle](#) mScatterStyle
- int mScatterSkip
- [LineStyle](#) mLineStyle

Friends

- class **QCustomPlot**
- class **QCPLegend**

Additional Inherited Members

6.30.1 Detailed Description

A plottable representing a parametric curve in a plot.

Unlike [QCPGraph](#), plottables of this type may have multiple points with the same key coordinate, so their visual representation can have *loops*. This is realized by introducing a third coordinate t , which defines the order of the points described by the other two coordinates x and y .

To plot data, assign it with the [setData](#) or [addData](#) functions. Alternatively, you can also access and modify the curve's data via the [data](#) method, which returns a pointer to the internal `QCPCurveDataContainer`.

Gaps in the curve can be created by adding data points with NaN as key and value (`qQNaN()` or `std::numeric_limits<double>::quiet_NaN()`) in between the two data points that shall be separated.

6.30.2 Changing the appearance

The appearance of the curve is determined by the pen and the brush ([setPen](#), [setBrush](#)).

6.30.3 Usage

Like all data representing objects in [QCustomPlot](#), the [QCPCurve](#) is a plottable ([QCPAbstractPlottable](#)). So the plottable-interface of [QCustomPlot](#) applies ([QCustomPlot::plottable](#), [QCustomPlot::removePlottable](#), etc.)

Usually, you first create an instance:

which registers it with the [QCustomPlot](#) instance of the passed axes. Note that this [QCustomPlot](#) instance takes ownership of the plottable, so do not delete it manually but use [QCustomPlot::removePlottable\(\)](#) instead. The newly created plottable can be modified, e.g.:

6.30.4 Member Enumeration Documentation

6.30.4.1 LineStyle

```
enum QCPCurve::LineStyle
```

Defines how the curve's line is represented visually in the plot. The line is drawn with the current pen of the curve ([setPen](#)).

See also

[setLineStyle](#)

Enumerator

IsNone	No line is drawn between data points (e.g. only scatters)
IsLine	Data points are connected with a straight line.

6.30.5 Constructor & Destructor Documentation

6.30.5.1 QCPCurve()

```
QCPCurve::QCPCurve (
    QCPAxis * keyAxis,
    QCPAxis * valueAxis ) [explicit]
```

Constructs a curve which uses *keyAxis* as its key axis ("x") and *valueAxis* as its value axis ("y"). *keyAxis* and *valueAxis* must reside in the same [QCustomPlot](#) instance and not have the same orientation. If either of these restrictions is violated, a corresponding message is printed to the debug output (qDebug), the construction is not aborted, though.

The created [QCPCurve](#) is automatically registered with the [QCustomPlot](#) instance inferred from *keyAxis*. This [QCustomPlot](#) instance takes ownership of the [QCPCurve](#), so do not delete it manually but use [QCustomPlot::removePlottable\(\)](#) instead.

6.30.6 Member Function Documentation

6.30.6.1 addData() [1/4]

```
void QCPCurve::addData (
    const QVector< double > & t,
    const QVector< double > & keys,
    const QVector< double > & values,
    bool alreadySorted = false )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Adds the provided points in *t*, *keys* and *values* to the current data. The provided vectors should have equal length. Else, the number of added points will be the size of the smallest vector.

If you can guarantee that the passed data points are sorted by *keys* in ascending order, you can set *alreadySorted* to true, to improve performance by saving a sorting run.

Alternatively, you can also access and modify the data directly via the [data](#) method, which returns a pointer to the internal data container.

6.30.6.2 addData() [2/4]

```
void QCPCurve::addData (
    const QVector< double > & keys,
    const QVector< double > & values )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Adds the provided points in *keys* and *values* to the current data. The provided vectors should have equal length. Else, the number of added points will be the size of the smallest vector.

The *t* parameter of each data point will be set to the integer index of the respective key/value pair.

Alternatively, you can also access and modify the data directly via the [data](#) method, which returns a pointer to the internal data container.

6.30.6.3 addData() [3/4]

```
void QCPCurve::addData (
    double t,
    double key,
    double value )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts. Adds the provided data point as *t*, *key* and *value* to the current data.

Alternatively, you can also access and modify the data directly via the [data](#) method, which returns a pointer to the internal data container.

6.30.6.4 addData() [4/4]

```
void QCPCurve::addData (
    double key,
    double value )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Adds the provided data point as *key* and *value* to the current data.

The *t* parameter is generated automatically by increments of 1 for each point, starting at the highest *t* of previously existing data or 0, if the curve data is empty.

Alternatively, you can also access and modify the data directly via the [data](#) method, which returns a pointer to the internal data container.

6.30.6.5 data()

```
QSharedPointer< QCPCurveDataContainer > QCPCurve::data ( ) const [inline]
```

Returns a shared pointer to the internal data storage of type `QCPCurveDataContainer`. You may use it to directly manipulate the data, which may be more convenient and faster than using the regular [setData](#) or [addData](#) methods.

6.30.6.6 getKeyRange()

```
QCPRange QCPCurve::getKeyRange (
    bool & foundRange,
    QCP::SignDomain inSignDomain = QCP::sdBoth ) const [virtual]
```

Returns the coordinate range that all data in this plottable span in the key axis dimension. For logarithmic plots, one can set *inSignDomain* to either [QCP::sdNegative](#) or [QCP::sdPositive](#) in order to restrict the returned range to that sign domain. E.g. when only negative range is wanted, set *inSignDomain* to [QCP::sdNegative](#) and all positive points will be ignored for range calculation. For no restriction, just set *inSignDomain* to [QCP::sdBoth](#) (default). *foundRange* is an output parameter that indicates whether a range could be found or not. If this is false, you shouldn't use the returned range (e.g. no points in data).

Note that *foundRange* is not the same as [QCPRange::validRange](#), since the range returned by this function may have size zero (e.g. when there is only one data point). In this case *foundRange* would return true, but the returned range is not a valid range in terms of [QCPRange::validRange](#).

See also

[rescaleAxes](#), [getValueRange](#)

Implements [QCPAbstractPlottable](#).

6.30.6.7 getValueRange()

```
QCPRange QCPCurve::getValueRange (
    bool & foundRange,
    QCP::SignDomain inSignDomain = QCP::sdBoth,
    const QCPRange & inKeyRange = QCPRange() ) const [virtual]
```

Returns the coordinate range that the data points in the specified key range (*inKeyRange*) span in the value axis dimension. For logarithmic plots, one can set *inSignDomain* to either [QCP::sdNegative](#) or [QCP::sdPositive](#) in order to restrict the returned range to that sign domain. E.g. when only negative range is wanted, set *inSignDomain* to [QCP::sdNegative](#) and all positive points will be ignored for range calculation. For no restriction, just set *inSignDomain* to [QCP::sdBoth](#) (default). *foundRange* is an output parameter that indicates whether a range could be found or not. If this is false, you shouldn't use the returned range (e.g. no points in data).

If *inKeyRange* has both lower and upper bound set to zero (is equal to [QCPRange\(\)](#)), all data points are considered, without any restriction on the keys.

Note that *foundRange* is not the same as [QCPRange::validRange](#), since the range returned by this function may have size zero (e.g. when there is only one data point). In this case *foundRange* would return true, but the returned range is not a valid range in terms of [QCPRange::validRange](#).

See also

[rescaleAxes](#), [getKeyRange](#)

Implements [QCPAbstractPlottable](#).

6.30.6.8 selectTest()

```
double QCPCurve::selectTest (
    const QPointF & pos,
    bool onlySelectable,
    QVariant * details = 0 ) const [virtual]
```

Implements a point-selection algorithm assuming the data (accessed via the 1D data interface) is point-like. Most subclasses will want to reimplement this method again, to provide a more accurate hit test based on the true data visualization geometry.

Reimplemented from [QCPAbstractPlottable1D< QCPCurveData >](#).

6.30.6.9 setData() [1/3]

```
void QCPCurve::setData (
    QSharedPointer< QCPCurveDataContainer > data )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Replaces the current data container with the provided *data* container.

Since a QSharedPointer is used, multiple QCPCurves may share the same data container safely. Modifying the data in the container will then affect all curves that share the container. Sharing can be achieved by simply exchanging the data containers wrapped in shared pointers:

If you do not wish to share containers, but create a copy from an existing container, rather use the [QCPDataContainer<DataType>::set](#) method on the curve's data container directly:

See also

[addData](#)

6.30.6.10 setData() [2/3]

```
void QCPCurve::setData (
    const QVector< double > & t,
    const QVector< double > & keys,
    const QVector< double > & values,
    bool alreadySorted = false )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Replaces the current data with the provided points in *t*, *keys* and *values*. The provided vectors should have equal length. Else, the number of added points will be the size of the smallest vector.

If you can guarantee that the passed data points are sorted by *t* in ascending order, you can set *alreadySorted* to true, to improve performance by saving a sorting run.

See also

[addData](#)

6.30.6.11 setData() [3/3]

```
void QCPCurve::setData (
    const QVector< double > & keys,
    const QVector< double > & values )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Replaces the current data with the provided points in *keys* and *values*. The provided vectors should have equal length. Else, the number of added points will be the size of the smallest vector.

The *t* parameter of each data point will be set to the integer index of the respective key/value pair.

See also

[addData](#)

6.30.6.12 setLineStyle()

```
void QCPCurve::setLineStyle (
    QCPCurve::LineStyle style )
```

Sets how the single data points are connected in the plot or how they are represented visually apart from the scatter symbol. For scatter-only plots, set *style* to [lsNone](#) and [setScatterStyle](#) to the desired scatter style.

See also

[setScatterStyle](#)

6.30.6.13 setScatterSkip()

```
void QCPCurve::setScatterSkip (
    int skip )
```

If scatters are displayed (scatter style not [QCPScatterStyle::ssNone](#)), *skip* number of scatter points are skipped/not drawn after every drawn scatter point.

This can be used to make the data appear sparser while for example still having a smooth line, and to improve performance for very high density plots.

If *skip* is set to 0 (default), all scatter points are drawn.

See also

[setScatterStyle](#)

6.30.6.14 setScatterStyle()

```
void QCPCurve::setScatterStyle (
    const QCPScatterStyle & style )
```

Sets the visual appearance of single data points in the plot. If set to [QCPScatterStyle::ssNone](#), no scatter points are drawn (e.g. for line-only plots with appropriate line style).

See also

[QCPScatterStyle](#), [setLineStyle](#)

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

- [/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h](#)↔
- [/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp](#)↔

6.31 QCPCurveData Class Reference

Holds the data of one single data point for [QCPCurve](#).

Public Member Functions

- [QCPCurveData](#) ()
- [QCPCurveData](#) (double t, double key, double value)
- double [sortKey](#) () const
- double [mainKey](#) () const
- double [mainValue](#) () const
- [QCPRange](#) [valueRange](#) () const

Static Public Member Functions

- static [QCPCurveData](#) [fromSortKey](#) (double [sortKey](#))
- static bool [sortKeysIsMainKey](#) ()

Public Attributes

- double **t**
- double **key**
- double **value**

6.31.1 Detailed Description

Holds the data of one single data point for [QCPCurve](#).

The stored data is:

- *t*: the free ordering parameter of this curve point, like in the mathematical vector $(x(t), y(t))$. (This is the *sortKey*)
- *key*: coordinate on the key axis of this curve point (this is the *mainKey*)
- *value*: coordinate on the value axis of this curve point (this is the *mainValue*)

The container for storing multiple data points is [QCPCurveDataContainer](#). It is a typedef for [QCPDataContainer](#) with [QCPCurveData](#) as the `DataType` template parameter. See the documentation there for an explanation regarding the data type's generic methods.

See also

[QCPCurveDataContainer](#)

6.31.2 Constructor & Destructor Documentation

6.31.2.1 QCPCurveData() [1/2]

```
QCPCurveData::QCPCurveData ( )
```

Constructs a curve data point with *t*, *key* and *value* set to zero.

6.31.2.2 QCPCurveData() [2/2]

```
QCPCurveData::QCPCurveData (
    double t,
    double key,
    double value )
```

Constructs a curve data point with the specified *t*, *key* and *value*.

6.31.3 Member Function Documentation

6.31.3.1 fromSortKey()

```
static QCPCurveData QCPCurveData::fromSortKey (
    double sortKey ) [inline], [static]
```

Returns a data point with the specified *sortKey* (assigned to the data point's *t* member). All other members are set to zero.

For a general explanation of what this method is good for in the context of the data container, see the documentation of [QCPCDataContainer](#).

6.31.3.2 mainKey()

```
double QCPCurveData::mainKey ( ) const [inline]
```

Returns the *key* member of this data point.

For a general explanation of what this method is good for in the context of the data container, see the documentation of [QCPCDataContainer](#).

6.31.3.3 mainValue()

```
double QCPCurveData::mainValue ( ) const [inline]
```

Returns the *value* member of this data point.

For a general explanation of what this method is good for in the context of the data container, see the documentation of [QCPCDataContainer](#).

6.31.3.4 sortKey()

```
double QCPCurveData::sortKey ( ) const [inline]
```

Returns the *t* member of this data point.

For a general explanation of what this method is good for in the context of the data container, see the documentation of [QCPCDataContainer](#).

6.31.3.5 sortKeyIsMainKey()

```
static static bool QCPCurveData::sortKeyIsMainKey ( ) [inline], [static]
```

Since the member *key* is the data point key coordinate and the member *t* is the data ordering parameter, this method returns false.

For a general explanation of what this method is good for in the context of the data container, see the documentation of [QCPCDataContainer](#).

6.31.3.6 valueRange()

`QCPRange` QCPCurveData::valueRange () const [inline]

Returns a `QCPRange` with both lower and upper boundary set to *value* of this data point.

For a general explanation of what this method is good for in the context of the data container, see the documentation of `QCPDataContainer`.

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

- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h
- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp

6.32 QCPDataContainer< DataType > Class Template Reference

The generic data container for one-dimensional plottables.

```
#include <qcustomplot.h>
```

Public Types

- typedef QVector< DataType >::const_iterator **const_iterator**
- typedef QVector< DataType >::iterator **iterator**

Public Member Functions

- `QCPDataContainer` ()
- int `size` () const
- bool `isEmpty` () const
- bool `autoSqueeze` () const
- void `setAutoSqueeze` (bool enabled)
- void `set` (const `QCPDataContainer`< DataType > &data)
- void `set` (const QVector< DataType > &data, bool alreadySorted=false)
- void `add` (const `QCPDataContainer`< DataType > &data)
- void `add` (const QVector< DataType > &data, bool alreadySorted=false)
- void `add` (const DataType &data)
- void `removeBefore` (double sortKey)
- void `removeAfter` (double sortKey)
- void `remove` (double sortKeyFrom, double sortKeyTo)
- void `remove` (double sortKey)
- void `clear` ()
- void `sort` ()
- void `squeeze` (bool preAllocation=true, bool postAllocation=true)
- const_iterator `constBegin` () const
- const_iterator `constEnd` () const
- iterator `begin` ()
- iterator `end` ()
- const_iterator `findBegin` (double sortKey, bool expandedRange=true) const
- const_iterator `findEnd` (double sortKey, bool expandedRange=true) const
- const_iterator `at` (int index) const
- `QCPRange` `keyRange` (bool &foundRange, `QCP::SignDomain` signDomain=`QCP::sdBoth`)
- `QCPRange` `valueRange` (bool &foundRange, `QCP::SignDomain` signDomain=`QCP::sdBoth`, const `QCPRange` &inKeyRange=`QCPRange`())
- `QCPDataRange` `dataRange` () const
- void `limitIteratorsToDataRange` (const_iterator &begin, const_iterator &end, const `QCPDataRange` &dataRange) const

Protected Member Functions

- void **preallocateGrow** (int minimumPreallocSize)
- void **performAutoSqueeze** ()

Protected Attributes

- bool **mAutoSqueeze**
- QVector< DataType > **mData**
- int **mPreallocSize**
- int **mPreallocIteration**

Related Functions

(Note that these are not member functions.)

- template<class DataType >
bool **qcpLessThanSortKey** (const DataType &a, const DataType &b)

6.32.1 Detailed Description

```
template<class DataType>
class QCPDataContainer< DataType >
```

The generic data container for one-dimensional plottables.

This class template provides a fast container for data storage of one-dimensional data. The data type is specified as template parameter (called *DataType* in the following) and must provide some methods as described in the [next section](#).

The data is stored in a sorted fashion, which allows very quick lookups by the sorted key as well as retrieval of ranges (see [findBegin](#), [findEnd](#), [keyRange](#)) using binary search. The container uses a preallocation and a postallocation scheme, such that appending and prepending data (with respect to the sort key) is very fast and minimizes reallocations. If data is added which needs to be inserted between existing keys, the merge usually can be done quickly too, using the fact that existing data is always sorted. The user can further improve performance by specifying that added data is already itself sorted by key, if he can guarantee that this is the case (see for example [add\(const QVector<DataType> &data, bool alreadySorted\)](#)).

The data can be accessed with the provided const iterators ([constBegin](#), [constEnd](#)). If it is necessary to alter existing data in-place, the non-const iterators can be used ([begin](#), [end](#)). Changing data members that are not the sort key (for most data types called *key*) is safe from the container's perspective.

Great care must be taken however if the sort key is modified through the non-const iterators. For performance reasons, the iterators don't automatically cause a re-sorting upon their manipulation. It is thus the responsibility of the user to leave the container in a sorted state when finished with the data manipulation, before calling any other methods on the container. A complete re-sort (e.g. after finishing all sort key manipulation) can be done by calling [sort](#). Failing to do so can not be detected by the container efficiently and will cause both rendering artifacts and potential data loss.

Implementing one-dimensional plottables that make use of a [QCPDataContainer<T>](#) is usually done by subclassing from [QCPAbstractPlottable1D<T>](#), which introduces an according *mDataContainer* member and some convenience methods.

6.32.2 Requirements for the DataType template parameter

The template parameter `DataType` is the type of the stored data points. It must be trivially copyable and have the following public methods, preferably inline:

- `double sortKey() const`
Returns the member variable of this data point that is the sort key, defining the ordering in the container. Often this variable is simply called *key*.
- `static DataType fromSortKey(double sortKey)`
Returns a new instance of the data type initialized with its sort key set to *sortKey*.
- `static bool sortKeyIsMainKey()`
Returns true if the sort key is equal to the main key (see method `mainKey` below). For most plottables this is the case. It is not the case for example for [QCPCurve](#), which uses *t* as sort key and *key* as main key. This is the reason why [QCPCurve](#) unlike [QCPGraph](#) can display parametric curves with loops.
- `double mainKey() const`
Returns the variable of this data point considered the main key. This is commonly the variable that is used as the coordinate of this data point on the key axis of the plottable. This method is used for example when determining the automatic axis rescaling of key axes ([QCPAxis::rescale](#)).
- `double mainValue() const`
Returns the variable of this data point considered the main value. This is commonly the variable that is used as the coordinate of this data point on the value axis of the plottable.
- `QCPRange valueRange() const`
Returns the range this data point spans in the value axis coordinate. If the data is single-valued (e.g. [QCPGraphData](#)), this is simply a range with both lower and upper set to the main data point value. However if the data points can represent multiple values at once (e.g. [QCPFinancialData](#) with its *high*, *low*, *open* and *close* values at each *key*) this method should return the range those values span. This method is used for example when determining the automatic axis rescaling of value axes ([QCPAxis::rescale](#)).

6.32.3 Constructor & Destructor Documentation

6.32.3.1 QCPDataContainer()

```
template<class DataType >
QCPDataContainer< DataType >::QCPDataContainer ( )
```

Constructs a [QCPDataContainer](#) used for plottable classes that represent a series of key-sorted data

6.32.4 Member Function Documentation

6.32.4.1 `add()` [1/3]

```
template<class DataType >
void QCPDataContainer< DataType >::add (
    const QCPDataContainer< DataType > & data )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Adds the provided *data* to the current data in this container.

See also

[set](#), [remove](#)

6.32.4.2 `add()` [2/3]

```
template<class DataType >
void QCPDataContainer< DataType >::add (
    const QVector< DataType > & data,
    bool alreadySorted = false )
```

Adds the provided data points in *data* to the current data.

If you can guarantee that the data points in *data* have ascending order with respect to the *DataType*'s sort key, set *alreadySorted* to true to avoid an unnecessary sorting run.

See also

[set](#), [remove](#)

6.32.4.3 `add()` [3/3]

```
template<class DataType >
void QCPDataContainer< DataType >::add (
    const DataType & data )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Adds the provided single data point to the current data.

See also

[remove](#)

6.32.4.4 at()

```
template<class DataType>
QCPDataContainer::const_iterator QCPDataContainer< DataType >::at (
    int index ) const [inline]
```

Returns a const iterator to the element with the specified *index*. If *index* points beyond the available elements in this container, returns [constEnd](#), i.e. an iterator past the last valid element.

You can use this method to easily obtain iterators from a [QCPDataRange](#), see the data selection page for an example.

6.32.4.5 begin()

```
template<class DataType>
QCPDataContainer::iterator QCPDataContainer< DataType >::begin ( ) [inline]
```

Returns a non-const iterator to the first data point in this container.

You can manipulate the data points in-place through the non-const iterators, but great care must be taken when manipulating the sort key of a data point, see [sort](#), or the detailed description of this class.

6.32.4.6 clear()

```
template<class DataType >
void QCPDataContainer< DataType >::clear ( )
```

Removes all data points.

See also

[remove](#), [removeAfter](#), [removeBefore](#)

6.32.4.7 constBegin()

```
template<class DataType>
QCPDataContainer::const_iterator QCPDataContainer< DataType >::constBegin ( ) const [inline]
```

Returns a const iterator to the first data point in this container.

6.32.4.8 constEnd()

```
template<class DataType>
QCPDataContainer::const_iterator QCPDataContainer< DataType >::constEnd ( ) const [inline]
```

Returns a const iterator to the element past the last data point in this container.

6.32.4.9 `dataRange()`

```
template<class DataType>
QCPDataRange QCPDataContainer< DataType >::dataRange ( ) const [inline]
```

Returns a [QCPDataRange](#) encompassing the entire data set of this container. This means the begin index of the returned range is 0, and the end index is [size](#).

6.32.4.10 `end()`

```
template<class DataType>
QCPDataContainer::iterator QCPDataContainer< DataType >::end ( ) [inline]
```

Returns a non-const iterator to the element past the last data point in this container.

You can manipulate the data points in-place through the non-const iterators, but great care must be taken when manipulating the sort key of a data point, see [sort](#), or the detailed description of this class.

6.32.4.11 `findBegin()`

```
template<class DataType >
QCPDataContainer< DataType >::const_iterator QCPDataContainer< DataType >::findBegin (
    double sortKey,
    bool expandedRange = true ) const
```

Returns an iterator to the data point with a (sort-)key that is equal to, just below, or just above *sortKey*. If *expandedRange* is true, the data point just below *sortKey* will be considered, otherwise the one just above.

This can be used in conjunction with [findEnd](#) to iterate over data points within a given key range, including or excluding the bounding data points that are just beyond the specified range.

If *expandedRange* is true but there are no data points below *sortKey*, [constBegin](#) is returned.

If the container is empty, returns [constEnd](#).

See also

[findEnd](#), [QCPPlottableInterface1D::findBegin](#)

6.32.4.12 `findEnd()`

```
template<class DataType >
QCPDataContainer< DataType >::const_iterator QCPDataContainer< DataType >::findEnd (
    double sortKey,
    bool expandedRange = true ) const
```

Returns an iterator to the element after the data point with a (sort-)key that is equal to, just above or just below *sortKey*. If *expandedRange* is true, the data point just above *sortKey* will be considered, otherwise the one just below.

This can be used in conjunction with [findBegin](#) to iterate over data points within a given key range, including the bounding data points that are just below and above the specified range.

If *expandedRange* is true but there are no data points above *sortKey*, [constEnd](#) is returned.

If the container is empty, [constEnd](#) is returned.

See also

[findBegin](#), [QCPPlottableInterface1D::findEnd](#)

6.32.4.13 isEmpty()

```
template<class DataType>
bool QCPDataContainer< DataType >::isEmpty ( ) const [inline]
```

Returns whether this container holds no data points.

6.32.4.14 keyRange()

```
template<class DataType >
QCPRange QCPDataContainer< DataType >::keyRange (
    bool & foundRange,
    QCP::SignDomain signDomain = QCP::sdBoth )
```

Returns the range encompassed by the (main-)key coordinate of all data points. The output parameter *foundRange* indicates whether a sensible range was found. If this is false, you should not use the returned [QCPRange](#) (e.g. the data container is empty or all points have the same key).

Use *signDomain* to control which sign of the key coordinates should be considered. This is relevant e.g. for logarithmic plots which can mathematically only display one sign domain at a time.

If the *DataType* reports that its main key is equal to the sort key (*sortKeyIsMainKey*), as is the case for most plottables, this method uses this fact and finds the range very quickly.

See also

[valueRange](#)

6.32.4.15 limitIteratorsToDataRange()

```
template<class DataType >
void QCPDataContainer< DataType >::limitIteratorsToDataRange (
    const_iterator & begin,
    const_iterator & end,
    const QCPDataRange & dataRange ) const
```

Makes sure *begin* and *end* mark a data range that is both within the bounds of this data container's data, as well as within the specified *dataRange*.

This function doesn't require for *dataRange* to be within the bounds of this data container's valid range.

6.32.4.16 remove() [1/2]

```
template<class DataType >
void QCPDataContainer< DataType >::remove (
    double sortKeyFrom,
    double sortKeyTo )
```

Removes all data points with (sort-)keys between *sortKeyFrom* and *sortKeyTo*. If *sortKeyFrom* is greater or equal to *sortKeyTo*, the function does nothing. To remove a single data point with known (sort-)key, use [remove\(double sortKey\)](#).

See also

[removeBefore](#), [removeAfter](#), [clear](#)

6.32.4.17 `remove()` [2/2]

```
template<class DataType >
void QCPDataContainer< DataType >::remove (
    double sortKey )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Removes a single data point at *sortKey*. If the position is not known with absolute (binary) precision, consider using [remove\(double sortKeyFrom, double sortKeyTo\)](#) with a small fuzziness interval around the suspected position, depending on the precision with which the (sort-)key is known.

See also

[removeBefore](#), [removeAfter](#), [clear](#)

6.32.4.18 `removeAfter()`

```
template<class DataType >
void QCPDataContainer< DataType >::removeAfter (
    double sortKey )
```

Removes all data points with (sort-)keys greater than or equal to *sortKey*.

See also

[removeBefore](#), [remove](#), [clear](#)

6.32.4.19 `removeBefore()`

```
template<class DataType >
void QCPDataContainer< DataType >::removeBefore (
    double sortKey )
```

Removes all data points with (sort-)keys smaller than or equal to *sortKey*.

See also

[removeAfter](#), [remove](#), [clear](#)

6.32.4.20 `set()` [1/2]

```
template<class DataType >
void QCPDataContainer< DataType >::set (
    const QCPDataContainer< DataType > & data )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Replaces the current data in this container with the provided *data*.

See also

[add](#), [remove](#)

6.32.4.21 `set()` [2/2]

```
template<class DataType >
void QCPDataContainer< DataType >::set (
    const QVector< DataType > & data,
    bool alreadySorted = false )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Replaces the current data in this container with the provided *data*

If you can guarantee that the data points in *data* have ascending order with respect to the DataType's sort key, set *alreadySorted* to true to avoid an unnecessary sorting run.

See also

[add](#), [remove](#)

6.32.4.22 `setAutoSqueeze()`

```
template<class DataType >
void QCPDataContainer< DataType >::setAutoSqueeze (
    bool enabled )
```

Sets whether the container automatically decides when to release memory from its post- and preallocation pools when data points are removed. By default this is enabled and for typical applications shouldn't be changed.

If auto squeeze is disabled, you can manually decide when to release pre-/postallocation with [squeeze](#).

6.32.4.23 size()

```
template<class DataType>
int QCPDataContainer< DataType >::size ( ) const [inline]
```

Returns the number of data points in the container.

6.32.4.24 sort()

```
template<class DataType >
void QCPDataContainer< DataType >::sort ( )
```

Re-sorts all data points in the container by their sort key.

When setting, adding or removing points using the [QCPDataContainer](#) interface ([set](#), [add](#), [remove](#), etc.), the container makes sure to always stay in a sorted state such that a full resort is never necessary. However, if you choose to directly manipulate the sort key on data points by accessing and modifying it through the non-const iterators ([begin](#), [end](#)), it is your responsibility to bring the container back into a sorted state before any other methods are called on it. This can be achieved by calling this method immediately after finishing the sort key manipulation.

6.32.4.25 squeeze()

```
template<class DataType >
void QCPDataContainer< DataType >::squeeze (
    bool preAllocation = true,
    bool postAllocation = true )
```

Frees all unused memory that is currently in the preallocation and postallocation pools.

Note that [QCPDataContainer](#) automatically decides whether squeezing is necessary, if [setAutoSqueeze](#) is left enabled. It should thus not be necessary to use this method for typical applications.

The parameters *preAllocation* and *postAllocation* control whether pre- and/or post allocation should be freed, respectively.

6.32.4.26 valueRange()

```
template<class DataType >
QCPRange QCPDataContainer< DataType >::valueRange (
    bool & foundRange,
    QCP::SignDomain signDomain = QCP::sdBoth,
    const QCPRange & inKeyRange = QCPRange() )
```

Returns the range encompassed by the value coordinates of the data points in the specified key range (*inKeyRange*), using the full *DataType::valueRange* reported by the data points. The output parameter *foundRange* indicates whether a sensible range was found. If this is false, you should not use the returned [QCPRange](#) (e.g. the data container is empty or all points have the same value).

If *inKeyRange* has both lower and upper bound set to zero (is equal to [QCPRange\(\)](#)), all data points are considered, without any restriction on the keys.

Use *signDomain* to control which sign of the value coordinates should be considered. This is relevant e.g. for logarithmic plots which can mathematically only display one sign domain at a time.

See also

[keyRange](#)

6.32.5 Friends And Related Function Documentation

6.32.5.1 qcpLessThanSortKey()

```
template<class DataType >
bool qcpLessThanSortKey (
    const DataType & a,
    const DataType & b ) [related]
```

Returns whether the sort key of *a* is less than the sort key of *b*.

See also

[QCPDataContainer::sort](#)

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

- [/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h](#)

6.33 QCPDataRange Class Reference

Describes a data range given by begin and end index.

Public Member Functions

- [QCPDataRange \(\)](#)
- [QCPDataRange \(int begin, int end\)](#)
- **bool operator==** (const [QCPDataRange](#) &other) const
- **bool operator!=** (const [QCPDataRange](#) &other) const
- **int begin** () const
- **int end** () const
- **int size** () const
- **int length** () const
- **void setBegin** (int begin)
- **void setEnd** (int end)
- **bool isValid** () const
- **bool isEmpty** () const
- **QCPDataRange bounded** (const [QCPDataRange](#) &other) const
- **QCPDataRange expanded** (const [QCPDataRange](#) &other) const
- **QCPDataRange intersection** (const [QCPDataRange](#) &other) const
- **QCPDataRange adjusted** (int changeBegin, int changeEnd) const
- **bool intersects** (const [QCPDataRange](#) &other) const
- **bool contains** (const [QCPDataRange](#) &other) const

Related Functions

(Note that these are not member functions.)

- `QDebug operator<<` (`QDebug d`, `const QCPDataRange &dataRange`)

6.33.1 Detailed Description

Describes a data range given by begin and end index.

`QCPDataRange` holds two integers describing the begin (`setBegin`) and end (`setEnd`) index of a contiguous set of data points. The end index points to the data point above the last data point that's part of the data range, similarly to the nomenclature used in standard iterators.

Data Ranges are not bound to a certain plottable, thus they can be freely exchanged, created and modified. If a non-contiguous data set shall be described, the class `QCPDataSelection` is used, which holds and manages multiple instances of `QCPDataRange`. In most situations, `QCPDataSelection` is thus used.

Both `QCPDataRange` and `QCPDataSelection` offer convenience methods to work with them, e.g. `bounded`, `expanded`, `intersects`, `intersection`, `adjusted`, `contains`. Further, addition and subtraction operators (defined in `QCPDataSelection`) can be used to join/subtract data ranges and data selections (or mixtures), to retrieve a corresponding `QCPDataSelection`.

`QCustomPlot`'s data selection mechanism is based on `QCPDataSelection` and `QCPDataRange`.

Note

Do not confuse `QCPDataRange` with `QCPRange`. A `QCPRange` describes an interval in floating point plot coordinates, e.g. the current axis range.

6.33.2 Constructor & Destructor Documentation

6.33.2.1 `QCPDataRange()` [1/2]

```
QCPDataRange::QCPDataRange ( )
```

Creates an empty `QCPDataRange`, with begin and end set to 0.

6.33.2.2 `QCPDataRange()` [2/2]

```
QCPDataRange::QCPDataRange (
    int begin,
    int end )
```

Creates a `QCPDataRange`, initialized with the specified *begin* and *end*.

No checks or corrections are made to ensure the resulting range is valid (`isValid`).

6.33.3 Member Function Documentation

6.33.3.1 adjusted()

```
QCPDataRange QCPDataRange::adjusted (
    int changeBegin,
    int changeEnd ) const [inline]
```

Returns a data range where *changeBegin* and *changeEnd* were added to the begin and end indices, respectively.

6.33.3.2 bounded()

```
QCPDataRange QCPDataRange::bounded (
    const QCPDataRange & other ) const
```

Returns a data range that matches this data range, except that parts exceeding *other* are excluded.

This method is very similar to [intersection](#), with one distinction: If this range and the *other* range share no intersection, the returned data range will be empty with begin and end set to the respective boundary side of *other*, at which this range is residing. ([intersection](#) would just return a range with begin and end set to 0.)

6.33.3.3 contains()

```
bool QCPDataRange::contains (
    const QCPDataRange & other ) const
```

Returns whether all data points described by this data range are also in *other*.

See also

[intersects](#)

6.33.3.4 expanded()

```
QCPDataRange QCPDataRange::expanded (
    const QCPDataRange & other ) const
```

Returns a data range that contains both this data range as well as *other*.

6.33.3.5 intersection()

```
QCPDataRange QCPDataRange::intersection (
    const QCPDataRange & other ) const
```

Returns the data range which is contained in both this data range and *other*.

This method is very similar to [bounded](#), with one distinction: If this range and the *other* range share no intersection, the returned data range will be empty with begin and end set to 0. ([bounded](#) would return a range with begin and end set to one of the boundaries of *other*, depending on which side this range is on.)

See also

[QCPDataSelection::intersection](#)

6.33.3.6 intersects()

```
bool QCPDataRange::intersects (
    const QCPDataRange & other ) const
```

Returns whether this data range and *other* share common data points.

See also

[intersection](#), [contains](#)

6.33.3.7 isEmpty()

```
bool QCPDataRange::isEmpty ( ) const [inline]
```

Returns whether this range is empty, i.e. whether its begin index equals its end index.

See also

[size](#), [length](#)

6.33.3.8 isValid()

```
bool QCPDataRange::isValid ( ) const [inline]
```

Returns whether this range is valid. A valid range has a begin index greater or equal to 0, and an end index greater or equal to the begin index.

Note

Invalid ranges should be avoided and are never the result of any of [QCustomPlot](#)'s methods (unless they are themselves fed with invalid ranges). Do not pass invalid ranges to [QCustomPlot](#)'s methods. The invalid range is not inherently prevented in [QCPDataRange](#), to allow temporary invalid begin/end values while manipulating the range. An invalid range is not necessarily empty ([isEmpty](#)), since its [length](#) can be negative and thus non-zero.

6.33.3.9 length()

```
int QCPDataRange::length ( ) const [inline]
```

Returns the number of data points described by this data range. Equivalent to [size](#).

6.33.3.10 setBegin()

```
void QCPDataRange::setBegin (
    int begin ) [inline]
```

Sets the begin of this data range. The *begin* index points to the first data point that is part of the data range.

No checks or corrections are made to ensure the resulting range is valid ([isValid](#)).

See also

[setEnd](#)

6.33.3.11 setEnd()

```
void QCPDataRange::setEnd (
    int end ) [inline]
```

Sets the end of this data range. The *end* index points to the data point just above the last data point that is part of the data range.

No checks or corrections are made to ensure the resulting range is valid ([isValid](#)).

See also

[setBegin](#)

6.33.3.12 size()

```
int QCPDataRange::size ( ) const [inline]
```

Returns the number of data points described by this data range. This is equal to the end index minus the begin index.

See also

[length](#)

6.33.4 Friends And Related Function Documentation

6.33.4.1 operator<<()

```
QDebug operator<< (
    QDebug d,
    const QCPDataRange & dataRange ) [related]
```

Prints *dataRange* in a human readable format to the qDebug output.

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

- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h↵
- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp↵

6.34 QCPDataSelection Class Reference

Describes a data set by holding multiple [QCPDataRange](#) instances.

Public Member Functions

- [QCPDataSelection](#) ()
- [QCPDataSelection](#) (const [QCPDataRange](#) &range)
- bool [operator==](#) (const [QCPDataSelection](#) &other) const
- bool [operator!=](#) (const [QCPDataSelection](#) &other) const
- [QCPDataSelection](#) & [operator+=](#) (const [QCPDataSelection](#) &other)
- [QCPDataSelection](#) & [operator+=](#) (const [QCPDataRange](#) &other)
- [QCPDataSelection](#) & [operator-=](#) (const [QCPDataSelection](#) &other)
- [QCPDataSelection](#) & [operator-=](#) (const [QCPDataRange](#) &other)
- int [dataRangeCount](#) () const
- int [dataPointCount](#) () const
- [QCPDataRange](#) [dataRange](#) (int index=0) const
- QList< [QCPDataRange](#) > [dataRanges](#) () const
- [QCPDataRange](#) [span](#) () const
- void [addDataRange](#) (const [QCPDataRange](#) &dataRange, bool [simplify](#)=true)
- void [clear](#) ()
- bool [isEmpty](#) () const
- void [simplify](#) ()
- void [enforceType](#) ([QCP::SelectionType](#) type)
- bool [contains](#) (const [QCPDataSelection](#) &other) const
- [QCPDataSelection](#) [intersection](#) (const [QCPDataRange](#) &other) const
- [QCPDataSelection](#) [intersection](#) (const [QCPDataSelection](#) &other) const
- [QCPDataSelection](#) [inverse](#) (const [QCPDataRange](#) &outerRange) const

Friends

- const [QCPDataSelection operator+](#) (const [QCPDataSelection](#) &a, const [QCPDataSelection](#) &b)
- const [QCPDataSelection operator+](#) (const [QCPDataRange](#) &a, const [QCPDataSelection](#) &b)
- const [QCPDataSelection operator+](#) (const [QCPDataSelection](#) &a, const [QCPDataRange](#) &b)
- const [QCPDataSelection operator+](#) (const [QCPDataRange](#) &a, const [QCPDataRange](#) &b)
- const [QCPDataSelection operator-](#) (const [QCPDataSelection](#) &a, const [QCPDataSelection](#) &b)
- const [QCPDataSelection operator-](#) (const [QCPDataRange](#) &a, const [QCPDataSelection](#) &b)
- const [QCPDataSelection operator-](#) (const [QCPDataSelection](#) &a, const [QCPDataRange](#) &b)
- const [QCPDataSelection operator-](#) (const [QCPDataRange](#) &a, const [QCPDataRange](#) &b)

Related Functions

(Note that these are not member functions.)

- [QDebug operator<<](#) ([QDebug](#) d, const [QCPDataSelection](#) &selection)

6.34.1 Detailed Description

Describes a data set by holding multiple [QCPDataRange](#) instances.

[QCPDataSelection](#) manages multiple instances of [QCPDataRange](#) in order to represent any (possibly disjoint) set of data selection.

The data selection can be modified with addition and subtraction operators which take [QCPDataSelection](#) and [QCPDataRange](#) instances, as well as methods such as [addDataRange](#) and [clear](#). Read access is provided by [dataRange](#), [dataRanges](#), [dataRangeCount](#), etc.

The method [simplify](#) is used to join directly adjacent or even overlapping [QCPDataRange](#) instances. [QCPDataSelection](#) automatically simplifies when using the addition/subtraction operators. The only case when [simplify](#) is left to the user, is when calling [addDataRange](#), with the parameter *simplify* explicitly set to false. This is useful if many data ranges will be added to the selection successively and the overhead for simplifying after each iteration shall be avoided. In this case, you should make sure to call [simplify](#) after completing the operation.

Use [enforceType](#) to bring the data selection into a state complying with the constraints for selections defined in [QCP::SelectionType](#).

QCustomPlot's data selection mechanism is based on [QCPDataSelection](#) and [QCPDataRange](#).

6.34.2 Iterating over a data selection

As an example, the following code snippet calculates the average value of a graph's data [selection](#):

6.34.3 Constructor & Destructor Documentation

6.34.3.1 QCPDataSelection() [1/2]

```
QCPDataSelection::QCPDataSelection ( ) [explicit]
```

Creates an empty [QCPDataSelection](#).

6.34.3.2 QCPDataSelection() [2/2]

```
QCPDataSelection::QCPDataSelection (
    const QCPDataRange & range ) [explicit]
```

Creates a [QCPDataSelection](#) containing the provided *range*.

6.34.4 Member Function Documentation

6.34.4.1 addDataRange()

```
void QCPDataSelection::addDataRange (
    const QCPDataRange & dataRange,
    bool simplify = true )
```

Adds the given *dataRange* to this data selection. This is equivalent to the += operator but allows disabling immediate simplification by setting *simplify* to false. This can improve performance if adding a very large amount of data ranges successively. In this case, make sure to call [simplify](#) manually, after the operation.

6.34.4.2 clear()

```
void QCPDataSelection::clear ( )
```

Removes all data ranges. The data selection then contains no data points.

[isEmpty](#)

6.34.4.3 contains()

```
bool QCPDataSelection::contains (
    const QCPDataSelection & other ) const
```

Returns true if the data selection *other* is contained entirely in this data selection, i.e. all data point indices that are in *other* are also in this data selection.

See also

[QCPDataRange::contains](#)

6.34.4.4 dataPointCount()

```
int QCPDataSelection::dataPointCount ( ) const
```

Returns the total number of data points contained in all data ranges that make up this data selection.

6.34.4.5 dataRange()

```
QCPDataRange QCPDataSelection::dataRange (
    int index = 0 ) const
```

Returns the data range with the specified *index*.

If the data selection is simplified (the usual state of the selection, see [simplify](#)), the ranges are sorted by ascending data point index.

See also

[dataRangeCount](#)

6.34.4.6 dataRangeCount()

```
int QCPDataSelection::dataRangeCount ( ) const [inline]
```

Returns the number of ranges that make up the data selection. The ranges can be accessed by [dataRange](#) via their index.

See also

[dataRange](#), [dataPointCount](#)

6.34.4.7 dataRanges()

```
QList< QCPDataRange > QCPDataSelection::dataRanges ( ) const [inline]
```

Returns all data ranges that make up the data selection. If the data selection is simplified (the usual state of the selection, see [simplify](#)), the ranges are sorted by ascending data point index.

See also

[dataRange](#)

6.34.4.8 enforceType()

```
void QCPDataSelection::enforceType (
    QCP::SelectionType type )
```

Makes sure this data selection conforms to the specified *type* selection type. Before the type is enforced, [simplify](#) is called.

Depending on *type*, enforcing means adding new data points that were previously not part of the selection, or removing data points from the selection. If the current selection already conforms to *type*, the data selection is not changed.

See also

[QCP::SelectionType](#)

6.34.4.9 intersection() [1/2]

```
QCPDataSelection QCPDataSelection::intersection (
    const QCPDataRange & other ) const
```

Returns a data selection containing the points which are both in this data selection and in the data range *other*.

A common use case is to limit an unknown data selection to the valid range of a data container, using [QCPDataContainer::dataRange](#) as *other*. One can then safely iterate over the returned data selection without exceeding the data container's bounds.

6.34.4.10 intersection() [2/2]

```
QCPDataSelection QCPDataSelection::intersection (
    const QCPDataSelection & other ) const
```

Returns a data selection containing the points which are both in this data selection and in the data selection *other*.

6.34.4.11 inverse()

```
QCPDataSelection QCPDataSelection::inverse (
    const QCPDataRange & outerRange ) const
```

Returns a data selection which is the exact inverse of this data selection, with *outerRange* defining the base range on which to invert. If *outerRange* is smaller than the [span](#) of this data selection, it is expanded accordingly.

For example, this method can be used to retrieve all unselected segments by setting *outerRange* to the full data range of the plottable, and calling this method on a data selection holding the selected segments.

6.34.4.12 isEmpty()

```
bool QCPDataSelection::isEmpty ( ) const [inline]
```

Returns true if there are no data ranges, and thus no data points, in this [QCPDataSelection](#) instance.

See also

[dataRangeCount](#)

6.34.4.13 operator+=() [1/2]

```
QCPDataSelection & QCPDataSelection::operator+= (
    const QCPDataSelection & other )
```

Adds the data selection of *other* to this data selection, and then simplifies this data selection (see [simplify](#)).

6.34.4.14 operator+=() [2/2]

```
QCPDataSelection & QCPDataSelection::operator+= (
    const QCPDataRange & other )
```

Adds the data range *other* to this data selection, and then simplifies this data selection (see [simplify](#)).

6.34.4.15 operator-= () [1/2]

```
QCPDataSelection & QCPDataSelection::operator-= (
    const QCPDataSelection & other )
```

Removes all data point indices that are described by *other* from this data range.

6.34.4.16 operator-= () [2/2]

```
QCPDataSelection & QCPDataSelection::operator-= (
    const QCPDataRange & other )
```

Removes all data point indices that are described by *other* from this data range.

6.34.4.17 operator==()

```
bool QCPDataSelection::operator==(
    const QCPDataSelection & other ) const
```

Returns true if this selection is identical (contains the same data ranges with the same begin and end indices) to *other*.

Note that both data selections must be in simplified state (the usual state of the selection, see [simplify](#)) for this operator to return correct results.

6.34.4.18 `simplify()`

```
void QCPDataSelection::simplify ( )
```

Sorts all data ranges by range begin index in ascending order, and then joins directly adjacent or overlapping ranges. This can reduce the number of individual data ranges in the selection, and prevents possible double-counting when iterating over the data points held by the data ranges.

This method is automatically called when using the addition/subtraction operators. The only case when `simplify` is left to the user, is when calling `addDataRange`, with the parameter `simplify` explicitly set to false.

6.34.4.19 `span()`

```
QCPDataRange QCPDataSelection::span ( ) const
```

Returns a `QCPDataRange` which spans the entire data selection, including possible intermediate segments which are not part of the original data selection.

6.34.5 Friends And Related Function Documentation

6.34.5.1 `operator+` [1/4]

```
const QCPDataSelection operator+ (
    const QCPDataSelection & a,
    const QCPDataSelection & b ) [friend]
```

Return a `QCPDataSelection` with the data points in *a* joined with the data points in *b*. The resulting data selection is already simplified (see `QCPDataSelection::simplify`).

6.34.5.2 `operator+` [2/4]

```
const QCPDataSelection operator+ (
    const QCPDataRange & a,
    const QCPDataSelection & b ) [friend]
```

Return a `QCPDataSelection` with the data points in *a* joined with the data points in *b*. The resulting data selection is already simplified (see `QCPDataSelection::simplify`).

6.34.5.3 `operator+` [3/4]

```
const QCPDataSelection operator+ (
    const QCPDataSelection & a,
    const QCPDataRange & b ) [friend]
```

Return a `QCPDataSelection` with the data points in *a* joined with the data points in *b*. The resulting data selection is already simplified (see `QCPDataSelection::simplify`).

6.34.5.4 `operator+` [4/4]

```
const QCPDataSelection operator+ (
    const QCPDataRange & a,
    const QCPDataRange & b ) [friend]
```

Return a [QCPDataSelection](#) with the data points in *a* joined with the data points in *b*. The resulting data selection is already simplified (see [QCPDataSelection::simplify](#)).

6.34.5.5 `operator-` [1/4]

```
const QCPDataSelection operator- (
    const QCPDataSelection & a,
    const QCPDataSelection & b ) [friend]
```

Return a [QCPDataSelection](#) with the data points which are in *a* but not in *b*.

6.34.5.6 `operator-` [2/4]

```
const QCPDataSelection operator- (
    const QCPDataRange & a,
    const QCPDataSelection & b ) [friend]
```

Return a [QCPDataSelection](#) with the data points which are in *a* but not in *b*.

6.34.5.7 `operator-` [3/4]

```
const QCPDataSelection operator- (
    const QCPDataSelection & a,
    const QCPDataRange & b ) [friend]
```

Return a [QCPDataSelection](#) with the data points which are in *a* but not in *b*.

6.34.5.8 `operator-` [4/4]

```
const QCPDataSelection operator- (
    const QCPDataRange & a,
    const QCPDataRange & b ) [friend]
```

Return a [QCPDataSelection](#) with the data points which are in *a* but not in *b*.

6.34.5.9 `operator<<()`

```
QDebug operator<< (
    QDebug d,
    const QCPDataSelection & selection ) [related]
```

Prints *selection* in a human readable format to the qDebug output.

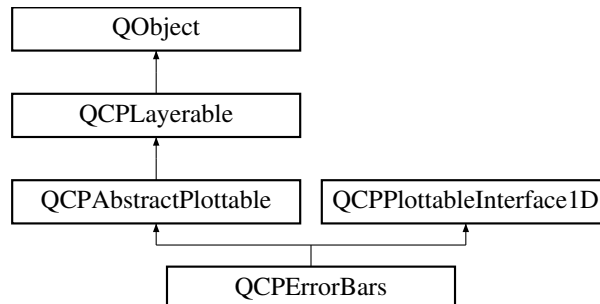
The documentation for this class was generated from the following files:

- `/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h`↔
- `/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp`↔

6.35 QCPErrorBars Class Reference

A plottable that adds a set of error bars to other plottables.

Inheritance diagram for QCPErrorBars:



Public Types

- enum [ErrorType](#) { [etKeyError](#), [etValueError](#) }

Public Member Functions

- [QCPErrorBars](#) ([QCPAxis](#) *keyAxis, [QCPAxis](#) *valueAxis)
- [QSharedPointer](#)< [QCPErrorBarsDataContainer](#) > [data](#) () const
- [QCPAbstractPlottable](#) * [dataPlottable](#) () const
- [ErrorType](#) [errorType](#) () const
- double [whiskerWidth](#) () const
- double [symbolGap](#) () const
- void [setData](#) ([QSharedPointer](#)< [QCPErrorBarsDataContainer](#) > [data](#))
- void [setData](#) (const [QVector](#)< double > &error)
- void [setData](#) (const [QVector](#)< double > &errorMinus, const [QVector](#)< double > &errorPlus)
- void [setDataPlottable](#) ([QCPAbstractPlottable](#) *plottable)
- void [setErrorType](#) ([ErrorType](#) type)
- void [setWhiskerWidth](#) (double pixels)
- void [setSymbolGap](#) (double pixels)
- void [addData](#) (const [QVector](#)< double > &error)
- void [addData](#) (const [QVector](#)< double > &errorMinus, const [QVector](#)< double > &errorPlus)
- void [addData](#) (double error)
- void [addData](#) (double errorMinus, double errorPlus)
- virtual int [dataCount](#) () const
- virtual double [dataMainKey](#) (int index) const
- virtual double [dataSortKey](#) (int index) const
- virtual double [dataMainValue](#) (int index) const
- virtual [QCPRange](#) [dataValueRange](#) (int index) const
- virtual [QPointF](#) [dataPixelPosition](#) (int index) const
- virtual bool [sortKeysIsMainKey](#) () const
- virtual [QCPDataSelection](#) [selectTestRect](#) (const [QRectF](#) &rect, bool onlySelectable) const
- virtual int [findBegin](#) (double sortKey, bool expandedRange=true) const
- virtual int [findEnd](#) (double sortKey, bool expandedRange=true) const
- virtual double [selectTest](#) (const [QPointF](#) &pos, bool onlySelectable, [QVariant](#) *details=0) const Q_DECL_OVERRIDE
- virtual [QCPPlottableInterface1D](#) * [interface1D](#) () Q_DECL_OVERRIDE

Protected Member Functions

- virtual void **draw** (QCPPainter *painter) Q_DECL_OVERRIDE
- virtual void **drawLegendIcon** (QCPPainter *painter, const QRectF &rect) const Q_DECL_OVERRIDE
- virtual [QCPRange](#) **getKeyRange** (bool &foundRange, [QCP::SignDomain](#) inSignDomain=[QCP::sdBoth](#)) const Q_DECL_OVERRIDE
- virtual [QCPRange](#) **getValueRange** (bool &foundRange, [QCP::SignDomain](#) inSignDomain=[QCP::sdBoth](#), const [QCPRange](#) &inKeyRange=[QCPRange](#)()) const Q_DECL_OVERRIDE
- void **getErrorBarLines** (QCErrorBarsDataContainer::const_iterator it, QVector< QLineF > &backbones, QVector< QLineF > &whiskers) const
- void **getVisibleDataBounds** (QCErrorBarsDataContainer::const_iterator &begin, QCErrorBarsDataContainer::const_iterator &end, const [QCPDataRange](#) &rangeRestriction) const
- double **pointDistance** (const QPointF &pixelPoint, QCErrorBarsDataContainer::const_iterator &closestData) const
- void **getDataSegments** (QList< [QCPDataRange](#) > &selectedSegments, QList< [QCPDataRange](#) > &unselectedSegments) const
- bool **errorBarVisible** (int index) const
- bool **rectIntersectsLine** (const QRectF &pixelRect, const QLineF &line) const

Protected Attributes

- QSharedPointer< QCErrorBarsDataContainer > **mDataContainer**
- QPointer< [QCPAbstractPlottable](#) > **mDataPlottable**
- [ErrorType](#) **mErrorType**
- double **mWhiskerWidth**
- double **mSymbolGap**

Friends

- class **QCustomPlot**
- class **QCPLegend**

Additional Inherited Members

6.35.1 Detailed Description

A plottable that adds a set of error bars to other plottables.

The [QCErrorBars](#) plottable can be attached to other one-dimensional plottables (e.g. [QCPGraph](#), [QCPCurve](#), [QCPBars](#), etc.) and equips them with error bars.

Use [setDataPlottable](#) to define for which plottable the [QCErrorBars](#) shall display the error bars. The orientation of the error bars can be controlled with [setErrorType](#).

By using [setData](#), you can supply the actual error data, either as symmetric error or plus/minus asymmetric errors. [QCErrorBars](#) only stores the error data. The absolute key/value position of each error bar will be adopted from the configured data plottable. The error data of the [QCErrorBars](#) are associated one-to-one via their index to the data points of the data plottable. You can directly access and manipulate the error bar data via [data](#).

Set either of the plus/minus errors to NaN ([qQNaN\(\)](#) or [std::numeric_limits<double>::quiet_NaN\(\)](#)) to not show the respective error bar on the data point at that index.

6.35.2 Changing the appearance

The appearance of the error bars is defined by the pen ([setPen](#)), and the width of the whiskers ([setWhiskerWidth](#)). Further, the error bar backbones may leave a gap around the data point center to prevent that error bars are drawn too close to or even through scatter points. This gap size can be controlled via [setSymbolGap](#).

6.35.3 Member Enumeration Documentation

6.35.3.1 ErrorType

enum [QCPErrorBars::ErrorType](#)

Defines in which orientation the error bars shall appear. If your data needs both error dimensions, create two [QCPErrorBars](#) with different [ErrorType](#).

See also

[setErrorType](#)

Enumerator

etKeyError	The errors are for the key dimension (bars appear parallel to the key axis)
etValueError	The errors are for the value dimension (bars appear parallel to the value axis)

6.35.4 Constructor & Destructor Documentation

6.35.4.1 QCPErrorBars()

```
QCPErrorBars::QCPErrorBars (
    QCPAxis * keyAxis,
    QCPAxis * valueAxis ) [explicit]
```

Constructs an error bars plottable which uses *keyAxis* as its key axis ("x") and *valueAxis* as its value axis ("y"). *keyAxis* and *valueAxis* must reside in the same [QCustomPlot](#) instance and not have the same orientation. If either of these restrictions is violated, a corresponding message is printed to the debug output (qDebug), the construction is not aborted, though.

It is also important that the *keyAxis* and *valueAxis* are the same for the error bars plottable and the data plottable that the error bars shall be drawn on ([setDataPlottable](#)).

The created [QCPErrorBars](#) is automatically registered with the [QCustomPlot](#) instance inferred from *keyAxis*. This [QCustomPlot](#) instance takes ownership of the [QCPErrorBars](#), so do not delete it manually but use [QCustomPlot::removePlottable\(\)](#) instead.

6.35.5 Member Function Documentation

6.35.5.1 `addData()` [1/4]

```
void QCPErrorBars::addData (
    const QVector< double > & error )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Adds symmetrical error values as specified in *error*. The errors will be associated one-to-one by the data point index to the associated data plottable ([setDataPlottable](#)).

You can directly access and manipulate the error bar data via [data](#).

See also

[setData](#)

6.35.5.2 `addData()` [2/4]

```
void QCPErrorBars::addData (
    const QVector< double > & errorMinus,
    const QVector< double > & errorPlus )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Adds asymmetrical errors as specified in *errorMinus* and *errorPlus*. The errors will be associated one-to-one by the data point index to the associated data plottable ([setDataPlottable](#)).

You can directly access and manipulate the error bar data via [data](#).

See also

[setData](#)

6.35.5.3 `addData()` [3/4]

```
void QCPErrorBars::addData (
    double error )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Adds a single symmetrical error bar as specified in *error*. The errors will be associated one-to-one by the data point index to the associated data plottable ([setDataPlottable](#)).

You can directly access and manipulate the error bar data via [data](#).

See also

[setData](#)

6.35.5.4 `addData()` [4/4]

```
void QCPErrors::addData (
    double errorMinus,
    double errorPlus )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Adds a single asymmetrical error bar as specified in *errorMinus* and *errorPlus*. The errors will be associated one-to-one by the data point index to the associated data plottable ([setDataPlottable](#)).

You can directly access and manipulate the error bar data via [data](#).

See also

[setData](#)

6.35.5.5 `data()`

```
QSharedPointer< QCPErrorsDataContainer > QCPErrors::data ( ) const [inline]
```

Returns a shared pointer to the internal data storage of type `QCPErrorsDataContainer`. You may use it to directly manipulate the error values, which may be more convenient and faster than using the regular [setData](#) methods.

6.35.5.6 `dataCount()`

```
int QCPErrors::dataCount ( ) const [virtual]
```

Returns the number of data points of the plottable.

Implements [QCPPlottableInterface1D](#).

6.35.5.7 `dataMainKey()`

```
double QCPErrors::dataMainKey (
    int index ) const [virtual]
```

Returns the main key of the data point at the given *index*.

What the main key is, is defined by the plottable's data type. See the [QCPDataContainer DataType](#) documentation for details about this naming convention.

Implements [QCPPlottableInterface1D](#).

6.35.5.8 dataMainValue()

```
double QCPErrorBars::dataMainValue (
    int index ) const [virtual]
```

Returns the main value of the data point at the given *index*.

What the main value is, is defined by the plottable's data type. See the [QCPDataContainer DataType](#) documentation for details about this naming convention.

Implements [QCPPlottableInterface1D](#).

6.35.5.9 dataPixelPosition()

```
QPointF QCPErrorBars::dataPixelPosition (
    int index ) const [virtual]
```

Returns the pixel position on the widget surface at which the data point at the given *index* appears.

Usually this corresponds to the point of [dataMainKey/dataMainValue](#), in pixel coordinates. However, depending on the plottable, this might be a different apparent position than just a coord-to-pixel transform of those values. For example, [QCPBars](#) apparent data values can be shifted depending on their stacking, bar grouping or configured base value.

Implements [QCPPlottableInterface1D](#).

6.35.5.10 dataSortKey()

```
double QCPErrorBars::dataSortKey (
    int index ) const [virtual]
```

Returns the sort key of the data point at the given *index*.

What the sort key is, is defined by the plottable's data type. See the [QCPDataContainer DataType](#) documentation for details about this naming convention.

Implements [QCPPlottableInterface1D](#).

6.35.5.11 dataValueRange()

```
QCPRange QCPErrorBars::dataValueRange (
    int index ) const [virtual]
```

Returns the value range of the data point at the given *index*.

What the value range is, is defined by the plottable's data type. See the [QCPDataContainer DataType](#) documentation for details about this naming convention.

Implements [QCPPlottableInterface1D](#).

6.35.5.12 findBegin()

```
int QCPErrors::findBegin (
    double sortKey,
    bool expandedRange = true ) const [virtual]
```

Returns the index of the data point with a (sort-)key that is equal to, just below, or just above *sortKey*. If *expandedRange* is true, the data point just below *sortKey* will be considered, otherwise the one just above.

This can be used in conjunction with [findEnd](#) to iterate over data points within a given key range, including or excluding the bounding data points that are just beyond the specified range.

If *expandedRange* is true but there are no data points below *sortKey*, 0 is returned.

If the container is empty, returns 0 (in that case, [findEnd](#) will also return 0, so a loop using these methods will not iterate over the index 0).

See also

[findEnd](#), [QCPDataContainer::findBegin](#)

Implements [QCPPlottableInterface1D](#).

6.35.5.13 findEnd()

```
int QCPErrors::findEnd (
    double sortKey,
    bool expandedRange = true ) const [virtual]
```

Returns the index one after the data point with a (sort-)key that is equal to, just above, or just below *sortKey*. If *expandedRange* is true, the data point just above *sortKey* will be considered, otherwise the one just below.

This can be used in conjunction with [findBegin](#) to iterate over data points within a given key range, including the bounding data points that are just below and above the specified range.

If *expandedRange* is true but there are no data points above *sortKey*, the index just above the highest data point is returned.

If the container is empty, returns 0.

See also

[findBegin](#), [QCPDataContainer::findEnd](#)

Implements [QCPPlottableInterface1D](#).

6.35.5.14 getKeyRange()

```
QCPRange QCPErrors::getKeyRange (
    bool & foundRange,
    QCP::SignDomain inSignDomain = QCP::sdBoth ) const [protected], [virtual]
```

Returns the coordinate range that all data in this plottable span in the key axis dimension. For logarithmic plots, one can set *inSignDomain* to either [QCP::sdNegative](#) or [QCP::sdPositive](#) in order to restrict the returned range to that sign domain. E.g. when only negative range is wanted, set *inSignDomain* to [QCP::sdNegative](#) and all positive points will be ignored for range calculation. For no restriction, just set *inSignDomain* to [QCP::sdBoth](#) (default). *foundRange* is an output parameter that indicates whether a range could be found or not. If this is false, you shouldn't use the returned range (e.g. no points in data).

Note that *foundRange* is not the same as [QCPRange::validRange](#), since the range returned by this function may have size zero (e.g. when there is only one data point). In this case *foundRange* would return true, but the returned range is not a valid range in terms of [QCPRange::validRange](#).

See also

[rescaleAxes](#), [getValueRange](#)

Implements [QCPAbstractPlottable](#).

6.35.5.15 getValueRange()

```
QCPRange QCPErrors::getValueRange (
    bool & foundRange,
    QCP::SignDomain inSignDomain = QCP::sdBoth,
    const QCPRange & inKeyRange = QCPRange() ) const [protected], [virtual]
```

Returns the coordinate range that the data points in the specified key range (*inKeyRange*) span in the value axis dimension. For logarithmic plots, one can set *inSignDomain* to either [QCP::sdNegative](#) or [QCP::sdPositive](#) in order to restrict the returned range to that sign domain. E.g. when only negative range is wanted, set *inSignDomain* to [QCP::sdNegative](#) and all positive points will be ignored for range calculation. For no restriction, just set *inSignDomain* to [QCP::sdBoth](#) (default). *foundRange* is an output parameter that indicates whether a range could be found or not. If this is false, you shouldn't use the returned range (e.g. no points in data).

If *inKeyRange* has both lower and upper bound set to zero (is equal to [QCPRange\(\)](#)), all data points are considered, without any restriction on the keys.

Note that *foundRange* is not the same as [QCPRange::validRange](#), since the range returned by this function may have size zero (e.g. when there is only one data point). In this case *foundRange* would return true, but the returned range is not a valid range in terms of [QCPRange::validRange](#).

See also

[rescaleAxes](#), [getKeyRange](#)

Implements [QCPAbstractPlottable](#).

6.35.5.16 interface1D()

```
virtual QCPPlottableInterface1D* QCPErrors::interface1D ( ) [inline], [virtual]
```

If this plottable is a one-dimensional plottable, i.e. it implements the [QCPPlottableInterface1D](#), returns the *this* pointer with that type. Otherwise (e.g. in the case of a [QCPColorMap](#)) returns zero.

You can use this method to gain read access to data coordinates while holding a pointer to the abstract base class only.

Reimplemented from [QCPAbstractPlottable](#).

6.35.5.17 selectTest()

```
double QCPErrors::selectTest (
    const QPointF & pos,
    bool onlySelectable,
    QVariant * details = 0 ) const [virtual]
```

This function is used to decide whether a click hits a layerable object or not.

pos is a point in pixel coordinates on the [QCustomPlot](#) surface. This function returns the shortest pixel distance of this point to the object. If the object is either invisible or the distance couldn't be determined, -1.0 is returned. Further, if *onlySelectable* is true and the object is not selectable, -1.0 is returned, too.

If the object is represented not by single lines but by an area like a [QCItemText](#) or the bars of a [QCPBars](#) plottable, a click inside the area should also be considered a hit. In these cases this function thus returns a constant value greater zero but still below the parent plot's selection tolerance. (typically the selectionTolerance multiplied by 0.99).

Providing a constant value for area objects allows selecting line objects even when they are obscured by such area objects, by clicking close to the lines (i.e. closer than 0.99*selectionTolerance).

The actual setting of the selection state is not done by this function. This is handled by the parent [QCustomPlot](#) when the mouseReleaseEvent occurs, and the finally selected object is notified via the selectEvent/deselectEvent methods.

details is an optional output parameter. Every layerable subclass may place any information in *details*. This information will be passed to selectEvent when the parent [QCustomPlot](#) decides on the basis of this selectTest call, that the object was successfully selected. The subsequent call to selectEvent will carry the *details*. This is useful for multi-part objects (like [QCPAxis](#)). This way, a possibly complex calculation to decide which part was clicked is only done once in [selectTest](#). The result (i.e. the actually clicked part) can then be placed in *details*. So in the subsequent selectEvent, the decision which part was selected doesn't have to be done a second time for a single selection operation.

You may pass 0 as *details* to indicate that you are not interested in those selection details.

See also

selectEvent, deselectEvent, [mousePressEvent](#), [wheelEvent](#), [QCustomPlot::setInteractions](#)

Implements [QCPAbstractPlottable](#).

6.35.5.18 selectTestRect()

```
QCPDataSelection QCPErrors::selectTestRect (
    const QRectF & rect,
    bool onlySelectable ) const [virtual]
```

Returns a data selection containing all the data points of this plottable which are contained (or hit by) *rect*. This is used mainly in the selection rect interaction for data selection (data selection mechanism).

If *onlySelectable* is true, an empty [QCPDataSelection](#) is returned if this plottable is not selectable (i.e. if [QCPAbstractPlottable::setSelectable](#) is [QCP::stNone](#)).

Note

rect must be a normalized rect (positive or zero width and height). This is especially important when using the rect of [QCPSelectionRect::accepted](#), which is not necessarily normalized. Use `QRect::normalized()` when passing a rect which might not be normalized.

Implements [QCPPlottableInterface1D](#).

6.35.5.19 setData() [1/3]

```
void QCPErrors::setData (
    QSharedPointer< QCPErrorsDataContainer > data )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Replaces the current data container with the provided *data* container.

Since a `QSharedPointer` is used, multiple [QCPErrors](#) instances may share the same data container safely. Modifying the data in the container will then affect all [QCPErrors](#) instances that share the container. Sharing can be achieved by simply exchanging the data containers wrapped in shared pointers:

If you do not wish to share containers, but create a copy from an existing container, assign the data containers directly:

(This uses different notation compared with other plottables, because the [QCPErrors](#) uses a `QVector<QCPErrorBarsData>` as its data container, instead of a [QCPDataContainer](#).)

See also

[addData](#)

6.35.5.20 setData() [2/3]

```
void QCPErrors::setData (
    const QVector< double > & error )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Sets symmetrical error values as specified in *error*. The errors will be associated one-to-one by the data point index to the associated data plottable ([setDataPlottable](#)).

You can directly access and manipulate the error bar data via [data](#).

See also

[addData](#)

6.35.5.21 setData() [3/3]

```
void QCPErrors::setData (
    const QVector< double > & errorMinus,
    const QVector< double > & errorPlus )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Sets asymmetrical errors as specified in *errorMinus* and *errorPlus*. The errors will be associated one-to-one by the data point index to the associated data plottable ([setDataPlottable](#)).

You can directly access and manipulate the error bar data via [data](#).

See also

[addData](#)

6.35.5.22 setDataPlottable()

```
void QCPErrors::setDataPlottable (
    QCPAbstractPlottable * plottable )
```

Sets the data plottable to which the error bars will be applied. The error values specified e.g. via [setData](#) will be associated one-to-one by the data point index to the data points of *plottable*. This means that the error bars will adopt the key/value coordinates of the data point with the same index.

The passed *plottable* must be a one-dimensional plottable, i.e. it must implement the [QCPPlottableInterface1D](#). Further, it must not be a [QCPErrors](#) instance itself. If either of these restrictions is violated, a corresponding qDebug output is generated, and the data plottable of this [QCPErrors](#) instance is set to zero.

For proper display, care must also be taken that the key and value axes of the *plottable* match those configured for this [QCPErrors](#) instance.

6.35.5.23 setErrorType()

```
void QCPErrorsData::setErrorType (
    ErrorType type )
```

Sets in which orientation the error bars shall appear on the data points. If your data needs both error dimensions, create two [QCPErrorsData](#) with different *type*.

6.35.5.24 setSymbolGap()

```
void QCPErrorsData::setSymbolGap (
    double pixels )
```

Sets the gap diameter around the data points that will be left out when drawing the error bar backbones. This gap prevents that error bars are drawn too close to or even through scatter points.

6.35.5.25 setWhiskerWidth()

```
void QCPErrorsData::setWhiskerWidth (
    double pixels )
```

Sets the width of the whiskers (the short bars at the end of the actual error bar backbones) to *pixels*.

6.35.5.26 sortKeyIsMainKey()

```
bool QCPErrorsData::sortKeyIsMainKey ( ) const [virtual]
```

Returns whether the sort key ([dataSortKey](#)) is identical to the main key ([dataMainKey](#)).

What the sort and main keys are, is defined by the plottable's data type. See the [QCPErrorsData Container DataType](#) documentation for details about this naming convention.

Implements [QCPErrorsDataInterface1D](#).

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

- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h
- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp

6.36 QCPErrorsData Class Reference

Holds the data of one single error bar for [QCPErrorsData](#).

Public Member Functions

- [QCPErrorsData](#) ()
- [QCPErrorsData](#) (double error)
- [QCPErrorsData](#) (double errorMinus, double errorPlus)

Public Attributes

- double **errorMinus**
- double **errorPlus**

6.36.1 Detailed Description

Holds the data of one single error bar for [QCPErrorsBars](#).

The stored data is:

- *errorMinus*: how much the error bar extends towards negative coordinates from the data point position
- *errorPlus*: how much the error bar extends towards positive coordinates from the data point position

The container for storing the error bar information is QCPErrorsBarsDataContainer. It is a typedef for `QVector<QCPErrorsBarsData>`.

See also

[QCPErrorsBarsDataContainer](#)

6.36.2 Constructor & Destructor Documentation

6.36.2.1 QCPErrorsBarsData() [1/3]

```
QCPErrorsBarsData::QCPErrorsBarsData ( )
```

Constructs an error bar with errors set to zero.

6.36.2.2 QCPErrorsBarsData() [2/3]

```
QCPErrorsBarsData::QCPErrorsBarsData (
    double error ) [explicit]
```

Constructs an error bar with equal *error* in both negative and positive direction.

6.36.2.3 QCPErrorsBarsData() [3/3]

```
QCPErrorsBarsData::QCPErrorsBarsData (
    double errorMinus,
    double errorPlus )
```

Constructs an error bar with negative and positive errors set to *errorMinus* and *errorPlus*, respectively.

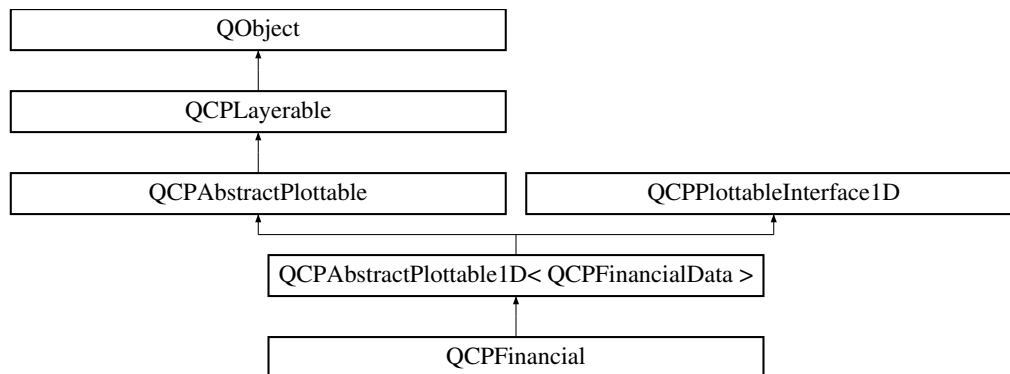
The documentation for this class was generated from the following files:

- [/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h](#)
- [/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp](#)

6.37 QCPFinancial Class Reference

A plottable representing a financial stock chart.

Inheritance diagram for QCPFinancial:



Public Types

- enum [WidthType](#) { [wtAbsolute](#), [wtAxisRectRatio](#), [wtPlotCoords](#) }
- enum [ChartStyle](#) { [csOhlc](#), [csCandlestick](#) }

Public Member Functions

- [QCPFinancial](#) ([QCPAxis](#) *keyAxis, [QCPAxis](#) *valueAxis)
- [QSharedPointer](#)< [QCPFinancialDataContainer](#) > [data](#) () const
- [ChartStyle](#) [chartStyle](#) () const
- double [width](#) () const
- [WidthType](#) [widthType](#) () const
- bool [twoColored](#) () const
- [QBrush](#) [brushPositive](#) () const
- [QBrush](#) [brushNegative](#) () const
- [QPen](#) [penPositive](#) () const
- [QPen](#) [penNegative](#) () const
- void [setData](#) ([QSharedPointer](#)< [QCPFinancialDataContainer](#) > data)
- void [setData](#) (const [QVector](#)< double > &keys, const [QVector](#)< double > &open, const [QVector](#)< double > &high, const [QVector](#)< double > &low, const [QVector](#)< double > &close, bool alreadySorted=false)
- void [setChartStyle](#) ([ChartStyle](#) style)
- void [setWidth](#) (double width)
- void [setWidthType](#) ([WidthType](#) widthType)
- void [setTwoColored](#) (bool twoColored)
- void [setBrushPositive](#) (const [QBrush](#) &brush)
- void [setBrushNegative](#) (const [QBrush](#) &brush)
- void [setPenPositive](#) (const [QPen](#) &pen)
- void [setPenNegative](#) (const [QPen](#) &pen)
- void [addData](#) (const [QVector](#)< double > &keys, const [QVector](#)< double > &open, const [QVector](#)< double > &high, const [QVector](#)< double > &low, const [QVector](#)< double > &close, bool alreadySorted=false)
- void [addData](#) (double key, double open, double high, double low, double close)
- virtual [QCPDataSelection](#) [selectTestRect](#) (const [QRectF](#) &rect, bool onlySelectable) const [Q_DECL_OVERRIDE](#)
- virtual double [selectTest](#) (const [QPointF](#) &pos, bool onlySelectable, [QVariant](#) *details=0) const [Q_DECL_OVERRIDE](#)
- virtual [QCPRange](#) [getKeyRange](#) (bool &foundRange, [QCP::SignDomain](#) inSignDomain=[QCP::sdBoth](#)) const [Q_DECL_OVERRIDE](#)
- virtual [QCPRange](#) [getValueRange](#) (bool &foundRange, [QCP::SignDomain](#) inSignDomain=[QCP::sdBoth](#), const [QCPRange](#) &inKeyRange=[QCPRange](#)()) const [Q_DECL_OVERRIDE](#)

Static Public Member Functions

- static [QCPFinancialDataContainer](#) [timeSeriesToOhlc](#) (const QVector< double > &time, const QVector< double > &value, double timeBinSize, double timeBinOffset=0)

Protected Member Functions

- virtual void **draw** ([QCPPainter](#) *painter) Q_DECL_OVERRIDE
- virtual void **drawLegendIcon** ([QCPPainter](#) *painter, const QRectF &rect) const Q_DECL_OVERRIDE
- void **drawOhlcPlot** ([QCPPainter](#) *painter, const QCPFinancialDataContainer::const_iterator &begin, const QCPFinancialDataContainer::const_iterator &end, bool isSelected)
- void **drawCandlestickPlot** ([QCPPainter](#) *painter, const QCPFinancialDataContainer::const_iterator &begin, const QCPFinancialDataContainer::const_iterator &end, bool isSelected)
- double **getPixelWidth** (double key, double keyPixel) const
- double **ohlcSelectTest** (const QPointF &pos, const QCPFinancialDataContainer::const_iterator &begin, const QCPFinancialDataContainer::const_iterator &end, QCPFinancialDataContainer::const_iterator &closestDataPoint) const
- double **candlestickSelectTest** (const QPointF &pos, const QCPFinancialDataContainer::const_iterator &begin, const QCPFinancialDataContainer::const_iterator &end, QCPFinancialDataContainer::const_iterator &closestDataPoint) const
- void **getVisibleDataBounds** (QCPFinancialDataContainer::const_iterator &begin, QCPFinancialDataContainer::const_iterator &end) const
- QRectF **selectionHitBox** (QCPFinancialDataContainer::const_iterator it) const

Protected Attributes

- [ChartStyle](#) **mChartStyle**
- double **mWidth**
- [WidthType](#) **mWidthType**
- bool **mTwoColored**
- QBrush **mBrushPositive**
- QBrush **mBrushNegative**
- QPen **mPenPositive**
- QPen **mPenNegative**

Friends

- class **QCustomPlot**
- class **QCPLegend**

Additional Inherited Members

6.37.1 Detailed Description

A plottable representing a financial stock chart.

This plottable represents time series data binned to certain intervals, mainly used for stock charts. The two common representations OHLC (Open-High-Low-Close) bars and Candlesticks can be set via [setChartStyle](#).

The data is passed via [setData](#) as a set of open/high/low/close values at certain keys (typically times). This means the data must be already binned appropriately. If data is only available as a series of values (e.g. *price* against *time*), you can use the static convenience function [timeSeriesToOhlc](#) to generate binned OHLC-data which can then be passed to [setData](#).

The width of the OHLC bars/candlesticks can be controlled with [setWidth](#) and [setWidthType](#). A typical choice is to set the width type to [wtPlotCoords](#) (the default) and the width to (or slightly less than) one time bin interval width.

6.37.2 Changing the appearance

Charts can be either single- or two-colored ([setTwoColored](#)). If set to be single-colored, lines are drawn with the plottable's pen ([setPen](#)) and fills with the brush ([setBrush](#)).

If set to two-colored, positive changes of the value during an interval ($close \geq open$) are represented with a different pen and brush than negative changes ($close < open$). These can be configured with [setPenPositive](#), [setPenNegative](#), [setBrushPositive](#), and [setBrushNegative](#). In two-colored mode, the normal plottable pen/brush is ignored. Upon selection however, the normal selected pen/brush (provided by the [selectionDecorator](#)) is used, irrespective of whether the chart is single- or two-colored.

6.37.3 Usage

Like all data representing objects in [QCustomPlot](#), the [QCPFinancial](#) is a plottable ([QCPAbstractPlottable](#)). So the plottable-interface of [QCustomPlot](#) applies ([QCustomPlot::plottable](#), [QCustomPlot::removePlottable](#), etc.)

Usually, you first create an instance:

which registers it with the [QCustomPlot](#) instance of the passed axes. Note that this [QCustomPlot](#) instance takes ownership of the plottable, so do not delete it manually but use [QCustomPlot::removePlottable\(\)](#) instead. The newly created plottable can be modified, e.g.:

Here we have used the static helper method [timeSeriesToOhlc](#), to turn a time-price data series into a 24-hour binned open-high-low-close data series as [QCPFinancial](#) uses.

6.37.4 Member Enumeration Documentation

6.37.4.1 ChartStyle

```
enum QCPFinancial::ChartStyle
```

Defines the possible representations of OHLC data in the plot.

See also

[setChartStyle](#)

Enumerator

csOhlc	Open-High-Low-Close bar representation.
csCandlestick	Candlestick representation.

6.37.4.2 WidthType

enum `QCPFinancial::WidthType`

Defines the ways the width of the financial bar can be specified. Thus it defines what the number passed to [setWidth](#) actually means.

See also

[setWidthType](#), [setWidth](#)

Enumerator

<code>wtAbsolute</code>	width is in absolute pixels
<code>wtAxisRectRatio</code>	width is given by a fraction of the axis rect size
<code>wtPlotCoords</code>	width is in key coordinates and thus scales with the key axis range

6.37.5 Constructor & Destructor Documentation

6.37.5.1 QCPFinancial()

```
QCPFinancial::QCPFinancial (
    QCPAxis * keyAxis,
    QCPAxis * valueAxis ) [explicit]
```

Constructs a financial chart which uses *keyAxis* as its key axis ("x") and *valueAxis* as its value axis ("y"). *keyAxis* and *valueAxis* must reside in the same [QCustomPlot](#) instance and not have the same orientation. If either of these restrictions is violated, a corresponding message is printed to the debug output (qDebug), the construction is not aborted, though.

The created [QCPFinancial](#) is automatically registered with the [QCustomPlot](#) instance inferred from *keyAxis*. This [QCustomPlot](#) instance takes ownership of the [QCPFinancial](#), so do not delete it manually but use [QCustomPlot::removePlottable\(\)](#) instead.

6.37.6 Member Function Documentation

6.37.6.1 `addData()` [1/2]

```
void QCPFinancial::addData (
    const QVector< double > & keys,
    const QVector< double > & open,
    const QVector< double > & high,
    const QVector< double > & low,
    const QVector< double > & close,
    bool alreadySorted = false )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Adds the provided points in *keys*, *open*, *high*, *low* and *close* to the current data. The provided vectors should have equal length. Else, the number of added points will be the size of the smallest vector.

If you can guarantee that the passed data points are sorted by *keys* in ascending order, you can set *alreadySorted* to true, to improve performance by saving a sorting run.

Alternatively, you can also access and modify the data directly via the [data](#) method, which returns a pointer to the internal data container.

See also

[timeSeriesToOhlc](#)

6.37.6.2 `addData()` [2/2]

```
void QCPFinancial::addData (
    double key,
    double open,
    double high,
    double low,
    double close )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Adds the provided data point as *key*, *open*, *high*, *low* and *close* to the current data.

Alternatively, you can also access and modify the data directly via the [data](#) method, which returns a pointer to the internal data container.

See also

[timeSeriesToOhlc](#)

6.37.6.3 data()

```
QCPFinancialDataContainer * QCPFinancial::data ( ) const [inline]
```

Returns a pointer to the internal data storage of type `QCPFinancialDataContainer`. You may use it to directly manipulate the data, which may be more convenient and faster than using the regular `setData` or `addData` methods, in certain situations.

6.37.6.4 getKeyRange()

```
QCPRange QCPFinancial::getKeyRange (
    bool & foundRange,
    QCP::SignDomain inSignDomain = QCP::sdBoth ) const [virtual]
```

Returns the coordinate range that all data in this plottable span in the key axis dimension. For logarithmic plots, one can set *inSignDomain* to either `QCP::sdNegative` or `QCP::sdPositive` in order to restrict the returned range to that sign domain. E.g. when only negative range is wanted, set *inSignDomain* to `QCP::sdNegative` and all positive points will be ignored for range calculation. For no restriction, just set *inSignDomain* to `QCP::sdBoth` (default). *foundRange* is an output parameter that indicates whether a range could be found or not. If this is false, you shouldn't use the returned range (e.g. no points in data).

Note that *foundRange* is not the same as `QCPRange::validRange`, since the range returned by this function may have size zero (e.g. when there is only one data point). In this case *foundRange* would return true, but the returned range is not a valid range in terms of `QCPRange::validRange`.

See also

[rescaleAxes](#), [getValueRange](#)

Implements [QCPAbstractPlottable](#).

6.37.6.5 getValueRange()

```
QCPRange QCPFinancial::getValueRange (
    bool & foundRange,
    QCP::SignDomain inSignDomain = QCP::sdBoth,
    const QCPRange & inKeyRange = QCPRange() ) const [virtual]
```

Returns the coordinate range that the data points in the specified key range (*inKeyRange*) span in the value axis dimension. For logarithmic plots, one can set *inSignDomain* to either `QCP::sdNegative` or `QCP::sdPositive` in order to restrict the returned range to that sign domain. E.g. when only negative range is wanted, set *inSignDomain* to `QCP::sdNegative` and all positive points will be ignored for range calculation. For no restriction, just set *inSignDomain* to `QCP::sdBoth` (default). *foundRange* is an output parameter that indicates whether a range could be found or not. If this is false, you shouldn't use the returned range (e.g. no points in data).

If *inKeyRange* has both lower and upper bound set to zero (is equal to `QCPRange()`), all data points are considered, without any restriction on the keys.

Note that *foundRange* is not the same as `QCPRange::validRange`, since the range returned by this function may have size zero (e.g. when there is only one data point). In this case *foundRange* would return true, but the returned range is not a valid range in terms of `QCPRange::validRange`.

See also

[rescaleAxes](#), [getKeyRange](#)

Implements [QCPAbstractPlottable](#).

6.37.6.6 selectTest()

```
double QCPFinancial::selectTest (
    const QPointF & pos,
    bool onlySelectable,
    QVariant * details = 0 ) const [virtual]
```

Implements a point-selection algorithm assuming the data (accessed via the 1D data interface) is point-like. Most subclasses will want to reimplement this method again, to provide a more accurate hit test based on the true data visualization geometry.

Reimplemented from [QCPAbstractPlottable1D< QCPFinancialData >](#).

6.37.6.7 selectTestRect()

```
QCPDataSelection QCPFinancial::selectTestRect (
    const QRectF & rect,
    bool onlySelectable ) const [virtual]
```

Returns a data selection containing all the data points of this plottable which are contained (or hit by) *rect*. This is used mainly in the selection rect interaction for data selection (data selection mechanism).

If *onlySelectable* is true, an empty [QCPDataSelection](#) is returned if this plottable is not selectable (i.e. if [QCPAbstractPlottable::setSelectable](#) is [QCP::stNone](#)).

Note

rect must be a normalized rect (positive or zero width and height). This is especially important when using the rect of [QCPSelectionRect::accepted](#), which is not necessarily normalized. Use `QRect::normalized()` when passing a rect which might not be normalized.

Reimplemented from [QCPAbstractPlottable1D< QCPFinancialData >](#).

6.37.6.8 setBrushNegative()

```
void QCPFinancial::setBrushNegative (
    const QBrush & brush )
```

If [setTwoColored](#) is set to true, this function controls the brush that is used to draw fills of data points with a negative trend (i.e. bars/candlesticks with close < open).

If *twoColored* is false, the normal plottable's pen and brush are used ([setPen](#), [setBrush](#)).

See also

[setBrushPositive](#), [setPenNegative](#), [setPenPositive](#)

6.37.6.9 `setBrushPositive()`

```
void QCPFinancial::setBrushPositive (
    const QBrush & brush )
```

If [setTwoColored](#) is set to true, this function controls the brush that is used to draw fills of data points with a positive trend (i.e. bars/candlesticks with close >= open).

If *twoColored* is false, the normal plottable's pen and brush are used ([setPen](#), [setBrush](#)).

See also

[setBrushNegative](#), [setPenPositive](#), [setPenNegative](#)

6.37.6.10 `setChartStyle()`

```
void QCPFinancial::setChartStyle (
    QCPFinancial::ChartStyle style )
```

Sets which representation style shall be used to display the OHLC data.

6.37.6.11 `setData()` [1/2]

```
void QCPFinancial::setData (
    QSharedPointer< QCPFinancialDataContainer > data )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Replaces the current data container with the provided *data* container.

Since a QSharedPointer is used, multiple QCPFinancials may share the same data container safely. Modifying the data in the container will then affect all financials that share the container. Sharing can be achieved by simply exchanging the data containers wrapped in shared pointers:

If you do not wish to share containers, but create a copy from an existing container, rather use the [QCPDataContainer<DataType>::set](#) method on the financial's data container directly:

See also

[addData](#), [timeSeriesToOhlc](#)

6.37.6.12 setData() [2/2]

```
void QCPFinancial::setData (
    const QVector< double > & keys,
    const QVector< double > & open,
    const QVector< double > & high,
    const QVector< double > & low,
    const QVector< double > & close,
    bool alreadySorted = false )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Replaces the current data with the provided points in *keys*, *open*, *high*, *low* and *close*. The provided vectors should have equal length. Else, the number of added points will be the size of the smallest vector.

If you can guarantee that the passed data points are sorted by *keys* in ascending order, you can set *alreadySorted* to true, to improve performance by saving a sorting run.

See also

[addData](#), [timeSeriesToOhlc](#)

6.37.6.13 setPenNegative()

```
void QCPFinancial::setPenNegative (
    const QPen & pen )
```

If [setTwoColored](#) is set to true, this function controls the pen that is used to draw outlines of data points with a negative trend (i.e. bars/candlesticks with $\text{close} < \text{open}$).

If *twoColored* is false, the normal plottable's pen and brush are used ([setPen](#), [setBrush](#)).

See also

[setPenPositive](#), [setBrushNegative](#), [setBrushPositive](#)

6.37.6.14 setPenPositive()

```
void QCPFinancial::setPenPositive (
    const QPen & pen )
```

If [setTwoColored](#) is set to true, this function controls the pen that is used to draw outlines of data points with a positive trend (i.e. bars/candlesticks with $\text{close} \geq \text{open}$).

If *twoColored* is false, the normal plottable's pen and brush are used ([setPen](#), [setBrush](#)).

See also

[setPenNegative](#), [setBrushPositive](#), [setBrushNegative](#)

6.37.6.15 setTwoColored()

```
void QCPFinancial::setTwoColored (
    bool twoColored )
```

Sets whether this chart shall contrast positive from negative trends per data point by using two separate colors to draw the respective bars/candlesticks.

If *twoColored* is false, the normal plottable's pen and brush are used ([setPen](#), [setBrush](#)).

See also

[setPenPositive](#), [setPenNegative](#), [setBrushPositive](#), [setBrushNegative](#)

6.37.6.16 setWidth()

```
void QCPFinancial::setWidth (
    double width )
```

Sets the width of the individual bars/candlesticks to *width* in plot key coordinates.

A typical choice is to set it to (or slightly less than) one bin interval width.

6.37.6.17 setWidthType()

```
void QCPFinancial::setWidthType (
    QCPFinancial::WidthType widthType )
```

Sets how the width of the financial bars is defined. See the documentation of [WidthType](#) for an explanation of the possible values for *widthType*.

The default value is [wtPlotCoords](#).

See also

[setWidth](#)

6.37.6.18 timeSeriesToOhlc()

```
QCPFinancialDataContainer QCPFinancial::timeSeriesToOhlc (
    const QVector< double > & time,
    const QVector< double > & value,
    double timeBinSize,
    double timeBinOffset = 0 ) [static]
```

A convenience function that converts time series data (*value* against *time*) to OHLC binned data points. The return value can then be passed on to [QCPFinancialDataContainer::set](#)(const QCPFinancialDataContainer&).

The size of the bins can be controlled with *timeBinSize* in the same units as *time* is given. For example, if the unit of *time* is seconds and single OHLC/Candlesticks should span an hour each, set *timeBinSize* to 3600.

timeBinOffset allows to control precisely at what *time* coordinate a bin should start. The value passed as *timeBinOffset* doesn't need to be in the range encompassed by the *time* keys. It merely defines the mathematical offset/phase of the bins that will be used to process the data.

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

- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h↵
- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp↵

6.38 QCPFinancialData Class Reference

Holds the data of one single data point for [QCPFinancial](#).

Public Member Functions

- [QCPFinancialData](#) ()
- [QCPFinancialData](#) (double key, double open, double high, double low, double close)
- double [sortKey](#) () const
- double [mainKey](#) () const
- double [mainValue](#) () const
- [QCPRange](#) [valueRange](#) () const

Static Public Member Functions

- static [QCPFinancialData](#) [fromSortKey](#) (double [sortKey](#))
- static bool [sortKeysIsMainKey](#) ()

Public Attributes

- double **key**
- double **open**
- double **high**
- double **low**
- double **close**

6.38.1 Detailed Description

Holds the data of one single data point for [QCPFinancial](#).

The stored data is:

- *key*: coordinate on the key axis of this data point (this is the *mainKey* and the *sortKey*)
- *open*: The opening value at the data point (this is the *mainValue*)
- *high*: The high/maximum value at the data point
- *low*: The low/minimum value at the data point
- *close*: The closing value at the data point

The container for storing multiple data points is [QCPFinancialDataContainer](#). It is a typedef for [QCPDataContainer](#) with [QCPFinancialData](#) as the `DataType` template parameter. See the documentation there for an explanation regarding the data type's generic methods.

See also

[QCPFinancialDataContainer](#)

6.38.2 Constructor & Destructor Documentation

6.38.2.1 [QCPFinancialData\(\)](#) [1/2]

```
QCPFinancialData::QCPFinancialData ( )
```

Constructs a data point with *key* and all values set to zero.

6.38.2.2 [QCPFinancialData\(\)](#) [2/2]

```
QCPFinancialData::QCPFinancialData (
    double key,
    double open,
    double high,
    double low,
    double close )
```

Constructs a data point with the specified *key* and OHLC values.

6.38.3 Member Function Documentation

6.38.3.1 fromSortKey()

```
static QCPFinancialData QCPFinancialData::fromSortKey (
    double sortKey ) [inline], [static]
```

Returns a data point with the specified *sortKey*. All other members are set to zero.

For a general explanation of what this method is good for in the context of the data container, see the documentation of [QCPDataContainer](#).

6.38.3.2 mainKey()

```
double QCPFinancialData::mainKey ( ) const [inline]
```

Returns the *key* member of this data point.

For a general explanation of what this method is good for in the context of the data container, see the documentation of [QCPDataContainer](#).

6.38.3.3 mainValue()

```
double QCPFinancialData::mainValue ( ) const [inline]
```

Returns the *open* member of this data point.

For a general explanation of what this method is good for in the context of the data container, see the documentation of [QCPDataContainer](#).

6.38.3.4 sortKey()

```
double QCPFinancialData::sortKey ( ) const [inline]
```

Returns the *key* member of this data point.

For a general explanation of what this method is good for in the context of the data container, see the documentation of [QCPDataContainer](#).

6.38.3.5 sortKeyIsMainKey()

```
static static bool QCPFinancialData::sortKeyIsMainKey ( ) [inline], [static]
```

Since the member *key* is both the data point key coordinate and the data ordering parameter, this method returns true.

For a general explanation of what this method is good for in the context of the data container, see the documentation of [QCPDataContainer](#).

6.38.3.6 valueRange()

```
QCPRange QCPFinancialData::valueRange ( ) const [inline]
```

Returns a [QCPRange](#) spanning from the *low* to the *high* value of this data point.

For a general explanation of what this method is good for in the context of the data container, see the documentation of [QCPDataContainer](#).

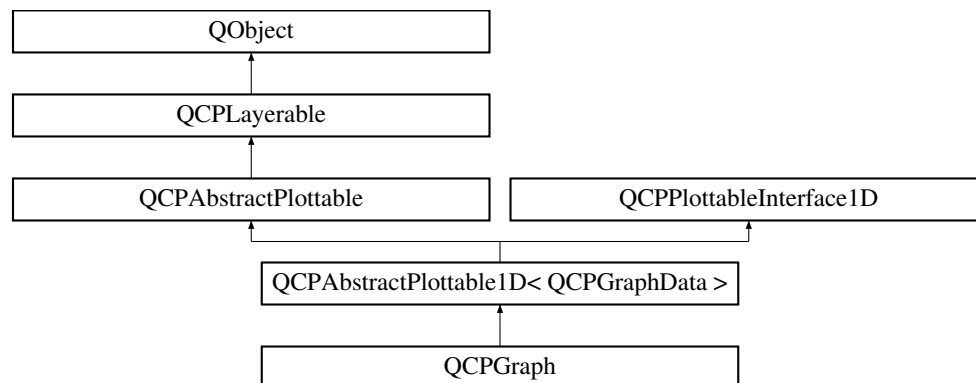
The documentation for this class was generated from the following files:

- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h↵
- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp↵

6.39 QCPGraph Class Reference

A plottable representing a graph in a plot.

Inheritance diagram for QCPGraph:



Public Types

- enum [LineStyle](#) {
[IsNone](#), [IsLine](#), [IsStepLeft](#), [IsStepRight](#),
[IsStepCenter](#), [IsImpulse](#) }

Public Member Functions

- [QCPGraph](#) ([QCPAxis](#) *keyAxis, [QCPAxis](#) *valueAxis)
- [QSharedPointer](#)< [QCPGraphDataContainer](#) > [data](#) () const
- [LineStyle](#) [lineStyle](#) () const
- [QCPScatterStyle](#) [scatterStyle](#) () const
- int [scatterSkip](#) () const
- [QCPGraph](#) * [channelFillGraph](#) () const
- bool [adaptiveSampling](#) () const
- void [setData](#) ([QSharedPointer](#)< [QCPGraphDataContainer](#) > [data](#))
- void [setData](#) (const [QVector](#)< double > &keys, const [QVector](#)< double > &values, bool alreadySorted=false)
- void [setLineStyle](#) ([LineStyle](#) ls)
- void [setScatterStyle](#) (const [QCPScatterStyle](#) &style)
- void [setScatterSkip](#) (int skip)
- void [setChannelFillGraph](#) ([QCPGraph](#) *targetGraph)
- void [setAdaptiveSampling](#) (bool enabled)
- void [addData](#) (const [QVector](#)< double > &keys, const [QVector](#)< double > &values, bool alreadySorted=false)
- void [addData](#) (double key, double value)
- virtual double [selectTest](#) (const [QPointF](#) &pos, bool onlySelectable, [QVariant](#) *details=0) const [Q_DECL_OVERRIDE](#)
- virtual [QCPRange](#) [getKeyRange](#) (bool &foundRange, [QCP::SignDomain](#) inSignDomain=[QCP::sdBoth](#)) const [Q_DECL_OVERRIDE](#)
- virtual [QCPRange](#) [getValueRange](#) (bool &foundRange, [QCP::SignDomain](#) inSignDomain=[QCP::sdBoth](#), const [QCPRange](#) &inKeyRange=[QCPRange](#)()) const [Q_DECL_OVERRIDE](#)

Protected Member Functions

- virtual void [draw](#) ([QCPPainter](#) *painter) [Q_DECL_OVERRIDE](#)
- virtual void [drawLegendIcon](#) ([QCPPainter](#) *painter, const [QRectF](#) &rect) const [Q_DECL_OVERRIDE](#)
- virtual void [drawFill](#) ([QCPPainter](#) *painter, [QVector](#)< [QPointF](#) > *lines) const
- virtual void [drawScatterPlot](#) ([QCPPainter](#) *painter, const [QVector](#)< [QPointF](#) > &scatters, const [QCPScatterStyle](#) &style) const
- virtual void [drawLinePlot](#) ([QCPPainter](#) *painter, const [QVector](#)< [QPointF](#) > &lines) const
- virtual void [drawImpulsePlot](#) ([QCPPainter](#) *painter, const [QVector](#)< [QPointF](#) > &lines) const
- virtual void [getOptimizedLineData](#) ([QVector](#)< [QCPGraphData](#) > *lineData, const [QCPGraphDataContainer::const_iterator](#) &begin, const [QCPGraphDataContainer::const_iterator](#) &end) const
- virtual void [getOptimizedScatterData](#) ([QVector](#)< [QCPGraphData](#) > *scatterData, [QCPGraphDataContainer::const_iterator](#) begin, [QCPGraphDataContainer::const_iterator](#) end) const
- void [getVisibleDataBounds](#) ([QCPGraphDataContainer::const_iterator](#) &begin, [QCPGraphDataContainer::const_iterator](#) &end, const [QCPDataRange](#) &rangeRestriction) const
- void [getLines](#) ([QVector](#)< [QPointF](#) > *lines, const [QCPDataRange](#) &dataRange) const
- void [getScatters](#) ([QVector](#)< [QPointF](#) > *scatters, const [QCPDataRange](#) &dataRange) const
- [QVector](#)< [QPointF](#) > [dataToLines](#) (const [QVector](#)< [QCPGraphData](#) > &data) const
- [QVector](#)< [QPointF](#) > [dataToStepLeftLines](#) (const [QVector](#)< [QCPGraphData](#) > &data) const
- [QVector](#)< [QPointF](#) > [dataToStepRightLines](#) (const [QVector](#)< [QCPGraphData](#) > &data) const
- [QVector](#)< [QPointF](#) > [dataToStepCenterLines](#) (const [QVector](#)< [QCPGraphData](#) > &data) const
- [QVector](#)< [QPointF](#) > [dataToImpulseLines](#) (const [QVector](#)< [QCPGraphData](#) > &data) const
- void [addFillBasePoints](#) ([QVector](#)< [QPointF](#) > *lines) const
- void [removeFillBasePoints](#) ([QVector](#)< [QPointF](#) > *lines) const
- [QPointF](#) [lowerFillBasePoint](#) (double lowerKey) const
- [QPointF](#) [upperFillBasePoint](#) (double upperKey) const
- const [QPolygonF](#) [getChannelFillPolygon](#) (const [QVector](#)< [QPointF](#) > *lines) const
- int [findIndexBelowX](#) (const [QVector](#)< [QPointF](#) > *data, double x) const
- int [findIndexAboveX](#) (const [QVector](#)< [QPointF](#) > *data, double x) const
- int [findIndexBelowY](#) (const [QVector](#)< [QPointF](#) > *data, double y) const
- int [findIndexAboveY](#) (const [QVector](#)< [QPointF](#) > *data, double y) const
- double [pointDistance](#) (const [QPointF](#) &pixelPoint, [QCPGraphDataContainer::const_iterator](#) &closestData) const

Protected Attributes

- [LineStyle](#) **mLineStyle**
- [QCPScatterStyle](#) **mScatterStyle**
- **int mScatterSkip**
- [QPointer< QCPGraph >](#) **mChannelFillGraph**
- **bool mAdaptiveSampling**

Friends

- class **QCustomPlot**
- class **QCPLegend**

Additional Inherited Members

6.39.1 Detailed Description

A plottable representing a graph in a plot.

Usually you create new graphs by calling [QCustomPlot::addGraph](#). The resulting instance can be accessed via [QCustomPlot::graph](#).

To plot data, assign it with the [setData](#) or [addData](#) functions. Alternatively, you can also access and modify the data via the [data](#) method, which returns a pointer to the internal `QCPGraphDataContainer`.

Graphs are used to display single-valued data. Single-valued means that there should only be one data point per unique key coordinate. In other words, the graph can't have *loops*. If you do want to plot non-single-valued curves, rather use the [QCPCurve](#) plottable.

Gaps in the graph line can be created by adding data points with NaN as value (`qQNaN()` or `std::numeric_limits<double>::quiet_NaN()`) in between the two data points that shall be separated.

6.39.2 Changing the appearance

The appearance of the graph is mainly determined by the line style, scatter style, brush and pen of the graph ([setLineStyle](#), [setScatterStyle](#), [setBrush](#), [setPen](#)).

6.39.2.1 Filling under or between graphs

[QCPGraph](#) knows two types of fills: Normal graph fills towards the zero-value-line parallel to the key axis of the graph, and fills between two graphs, called channel fills. To enable a fill, just set a brush with [setBrush](#) which is neither `Qt::NoBrush` nor fully transparent.

By default, a normal fill towards the zero-value-line will be drawn. To set up a channel fill between this graph and another one, call [setChannelFillGraph](#) with the other graph as parameter.

See also

[QCustomPlot::addGraph](#), [QCustomPlot::graph](#)

6.39.3 Member Enumeration Documentation

6.39.3.1 LineStyle

enum `QCPGraph::LineStyle`

Defines how the graph's line is represented visually in the plot. The line is drawn with the current pen of the graph ([setPen](#)).

See also

[setLineStyle](#)

Enumerator

<code>IsNone</code>	data points are not connected with any lines (e.g. data only represented with symbols according to the scatter style, see setScatterStyle)
<code>IsLine</code>	data points are connected by a straight line
<code>IsStepLeft</code>	line is drawn as steps where the step height is the value of the left data point
<code>IsStepRight</code>	line is drawn as steps where the step height is the value of the right data point
<code>IsStepCenter</code>	line is drawn as steps where the step is in between two data points
<code>IsImpulse</code>	each data point is represented by a line parallel to the value axis, which reaches from the data point to the zero-value-line

6.39.4 Constructor & Destructor Documentation

6.39.4.1 QCPGraph()

```
QCPGraph::QCPGraph (
    QCPAxis * keyAxis,
    QCPAxis * valueAxis ) [explicit]
```

Constructs a graph which uses *keyAxis* as its key axis ("x") and *valueAxis* as its value axis ("y"). *keyAxis* and *valueAxis* must reside in the same [QCustomPlot](#) instance and not have the same orientation. If either of these restrictions is violated, a corresponding message is printed to the debug output (qDebug), the construction is not aborted, though.

The created [QCPGraph](#) is automatically registered with the [QCustomPlot](#) instance inferred from *keyAxis*. This [QCustomPlot](#) instance takes ownership of the [QCPGraph](#), so do not delete it manually but use [QCustomPlot::removePlottable\(\)](#) instead.

To directly create a graph inside a plot, you can also use the simpler [QCustomPlot::addGraph](#) function.

6.39.5 Member Function Documentation

6.39.5.1 `addData()` [1/2]

```
void QCPGraph::addData (
    const QVector< double > & keys,
    const QVector< double > & values,
    bool alreadySorted = false )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Adds the provided points in *keys* and *values* to the current data. The provided vectors should have equal length. Else, the number of added points will be the size of the smallest vector.

If you can guarantee that the passed data points are sorted by *keys* in ascending order, you can set *alreadySorted* to true, to improve performance by saving a sorting run.

Alternatively, you can also access and modify the data directly via the [data](#) method, which returns a pointer to the internal data container.

6.39.5.2 `addData()` [2/2]

```
void QCPGraph::addData (
    double key,
    double value )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Adds the provided data point as *key* and *value* to the current data.

Alternatively, you can also access and modify the data directly via the [data](#) method, which returns a pointer to the internal data container.

6.39.5.3 `data()`

```
QSharedPointer< QCPGraphDataContainer > QCPGraph::data ( ) const [inline]
```

Returns a shared pointer to the internal data storage of type `QCPGraphDataContainer`. You may use it to directly manipulate the data, which may be more convenient and faster than using the regular [setData](#) or [addData](#) methods.

6.39.5.4 getKeyRange()

```
QCPRange QCPGraph::getKeyRange (
    bool & foundRange,
    QCP::SignDomain inSignDomain = QCP::sdBoth ) const [virtual]
```

Returns the coordinate range that all data in this plottable span in the key axis dimension. For logarithmic plots, one can set *inSignDomain* to either [QCP::sdNegative](#) or [QCP::sdPositive](#) in order to restrict the returned range to that sign domain. E.g. when only negative range is wanted, set *inSignDomain* to [QCP::sdNegative](#) and all positive points will be ignored for range calculation. For no restriction, just set *inSignDomain* to [QCP::sdBoth](#) (default). *foundRange* is an output parameter that indicates whether a range could be found or not. If this is false, you shouldn't use the returned range (e.g. no points in data).

Note that *foundRange* is not the same as [QCPRange::validRange](#), since the range returned by this function may have size zero (e.g. when there is only one data point). In this case *foundRange* would return true, but the returned range is not a valid range in terms of [QCPRange::validRange](#).

See also

[rescaleAxes](#), [getValueRange](#)

Implements [QCPAbstractPlottable](#).

6.39.5.5 getValueRange()

```
QCPRange QCPGraph::getValueRange (
    bool & foundRange,
    QCP::SignDomain inSignDomain = QCP::sdBoth,
    const QCPRange & inKeyRange = QCPRange() ) const [virtual]
```

Returns the coordinate range that the data points in the specified key range (*inKeyRange*) span in the value axis dimension. For logarithmic plots, one can set *inSignDomain* to either [QCP::sdNegative](#) or [QCP::sdPositive](#) in order to restrict the returned range to that sign domain. E.g. when only negative range is wanted, set *inSignDomain* to [QCP::sdNegative](#) and all positive points will be ignored for range calculation. For no restriction, just set *inSignDomain* to [QCP::sdBoth](#) (default). *foundRange* is an output parameter that indicates whether a range could be found or not. If this is false, you shouldn't use the returned range (e.g. no points in data).

If *inKeyRange* has both lower and upper bound set to zero (is equal to [QCPRange\(\)](#)), all data points are considered, without any restriction on the keys.

Note that *foundRange* is not the same as [QCPRange::validRange](#), since the range returned by this function may have size zero (e.g. when there is only one data point). In this case *foundRange* would return true, but the returned range is not a valid range in terms of [QCPRange::validRange](#).

See also

[rescaleAxes](#), [getKeyRange](#)

Implements [QCPAbstractPlottable](#).

6.39.5.6 `setVisibleDataBounds()`

```
void QCPGraph::setVisibleDataBounds (
    QCPGraphDataContainer::const_iterator & begin,
    QCPGraphDataContainer::const_iterator & end,
    const QCPDataRange & rangeRestriction ) const [protected]
```

This method outputs the currently visible data range via *begin* and *end*. The returned range will also never exceed *rangeRestriction*.

This method takes into account that the drawing of data lines at the axis rect border always requires the points just outside the visible axis range. So *begin* and *end* may actually indicate a range that contains one additional data point to the left and right of the visible axis range.

6.39.5.7 `selectTest()`

```
double QCPGraph::selectTest (
    const QPointF & pos,
    bool onlySelectable,
    QVariant * details = 0 ) const [virtual]
```

Implements a point-selection algorithm assuming the data (accessed via the 1D data interface) is point-like. Most subclasses will want to reimplement this method again, to provide a more accurate hit test based on the true data visualization geometry.

Reimplemented from [QCPAbstractPlottable1D< QCPGraphData >](#).

6.39.5.8 `setAdaptiveSampling()`

```
void QCPGraph::setAdaptiveSampling (
    bool enabled )
```

Sets whether adaptive sampling shall be used when plotting this graph. [QCustomPlot](#)'s adaptive sampling technique can drastically improve the replot performance for graphs with a larger number of points (e.g. above 10,000), without notably changing the appearance of the graph.

By default, adaptive sampling is enabled. Even if enabled, [QCustomPlot](#) decides whether adaptive sampling shall actually be used on a per-graph basis. So leaving adaptive sampling enabled has no disadvantage in almost all cases.

As can be seen, line plots experience no visual degradation from adaptive sampling. Outliers are reproduced reliably, as well as the overall shape of the data set. The replot time reduces dramatically though. This allows [QCustomPlot](#) to display large amounts of data in realtime.

Care must be taken when using high-density scatter plots in combination with adaptive sampling. The adaptive sampling algorithm treats scatter plots more carefully than line plots which still gives a significant reduction of replot times, but not quite as much as for line plots. This is because scatter plots inherently need more data points to be preserved in order to still resemble the original, non-adaptive-sampling plot. As shown above, the results still aren't quite identical, as banding occurs for the outer data points. This is in fact intentional, such that the boundaries of the data cloud stay visible to the viewer. How strong the banding appears, depends on the point density, i.e. the number of points in the plot.

For some situations with scatter plots it might thus be desirable to manually turn adaptive sampling off. For example, when saving the plot to disk. This can be achieved by setting *enabled* to false before issuing a command like [QCustomPlot::savePng](#), and setting *enabled* back to true afterwards.

6.39.5.9 setChannelFillGraph()

```
void QCPGraph::setChannelFillGraph (
    QCPGraph * targetGraph )
```

Sets the target graph for filling the area between this graph and *targetGraph* with the current brush ([setBrush](#)).

When *targetGraph* is set to 0, a normal graph fill to the zero-value-line will be shown. To disable any filling, set the brush to Qt::NoBrush.

See also

[setBrush](#)

6.39.5.10 setData() [1/2]

```
void QCPGraph::setData (
    QSharedPointer< QCPGraphDataContainer > data )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Replaces the current data container with the provided *data* container.

Since a QSharedPointer is used, multiple QCPGraphs may share the same data container safely. Modifying the data in the container will then affect all graphs that share the container. Sharing can be achieved by simply exchanging the data containers wrapped in shared pointers:

If you do not wish to share containers, but create a copy from an existing container, rather use the [QCPDataContainer<DataType>::set](#) method on the graph's data container directly:

See also

[addData](#)

6.39.5.11 setData() [2/2]

```
void QCPGraph::setData (
    const QVector< double > & keys,
    const QVector< double > & values,
    bool alreadySorted = false )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Replaces the current data with the provided points in *keys* and *values*. The provided vectors should have equal length. Else, the number of added points will be the size of the smallest vector.

If you can guarantee that the passed data points are sorted by *keys* in ascending order, you can set *alreadySorted* to true, to improve performance by saving a sorting run.

See also

[addData](#)

6.39.5.12 setLineStyle()

```
void QCPGraph::setLineStyle (
    LineStyle ls )
```

Sets how the single data points are connected in the plot. For scatter-only plots, set *ls* to [lsNone](#) and [setScatterStyle](#) to the desired scatter style.

See also

[setScatterStyle](#)

6.39.5.13 setScatterSkip()

```
void QCPGraph::setScatterSkip (
    int skip )
```

If scatters are displayed (scatter style not [QCPScatterStyle::ssNone](#)), *skip* number of scatter points are skipped/not drawn after every drawn scatter point.

This can be used to make the data appear sparser while for example still having a smooth line, and to improve performance for very high density plots.

If *skip* is set to 0 (default), all scatter points are drawn.

See also

[setScatterStyle](#)

6.39.5.14 setScatterStyle()

```
void QCPGraph::setScatterStyle (
    const QCPScatterStyle & style )
```

Sets the visual appearance of single data points in the plot. If set to [QCPScatterStyle::ssNone](#), no scatter points are drawn (e.g. for line-only-plots with appropriate line style).

See also

[QCPScatterStyle](#), [setLineStyle](#)

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

- [/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h](#)↔
- [/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp](#)↔

6.40 QCPGraphData Class Reference

Holds the data of one single data point for [QCPGraph](#).

Public Member Functions

- [QCPGraphData](#) ()
- [QCPGraphData](#) (double key, double value)
- double [sortKey](#) () const
- double [mainKey](#) () const
- double [mainValue](#) () const
- [QCPRange valueRange](#) () const

Static Public Member Functions

- static [QCPGraphData fromSortKey](#) (double [sortKey](#))
- static bool [sortKeysIsMainKey](#) ()

Public Attributes

- double **key**
- double **value**

6.40.1 Detailed Description

Holds the data of one single data point for [QCPGraph](#).

The stored data is:

- *key*: coordinate on the key axis of this data point (this is the *mainKey* and the *sortKey*)
- *value*: coordinate on the value axis of this data point (this is the *mainValue*)

The container for storing multiple data points is [QCPGraphDataContainer](#). It is a typedef for [QCPDataContainer](#) with [QCPGraphData](#) as the `DataType` template parameter. See the documentation there for an explanation regarding the data type's generic methods.

See also

[QCPGraphDataContainer](#)

6.40.2 Constructor & Destructor Documentation

6.40.2.1 [QCPGraphData\(\)](#) [1/2]

```
QCPGraphData::QCPGraphData ( )
```

Constructs a data point with *key* and *value* set to zero.

6.40.2.2 [QCPGraphData\(\)](#) [2/2]

```
QCPGraphData::QCPGraphData (
    double key,
    double value )
```

Constructs a data point with the specified *key* and *value*.

6.40.3 Member Function Documentation

6.40.3.1 [fromSortKey\(\)](#)

```
static QCPGraphData QCPGraphData::fromSortKey (
    double sortKey ) [inline], [static]
```

Returns a data point with the specified *sortKey*. All other members are set to zero.

For a general explanation of what this method is good for in the context of the data container, see the documentation of [QCPDataContainer](#).

6.40.3.2 mainKey()

```
double QCPGraphData::mainKey ( ) const [inline]
```

Returns the *key* member of this data point.

For a general explanation of what this method is good for in the context of the data container, see the documentation of [QCPDataContainer](#).

6.40.3.3 mainValue()

```
double QCPGraphData::mainValue ( ) const [inline]
```

Returns the *value* member of this data point.

For a general explanation of what this method is good for in the context of the data container, see the documentation of [QCPDataContainer](#).

6.40.3.4 sortKey()

```
double QCPGraphData::sortKey ( ) const [inline]
```

Returns the *key* member of this data point.

For a general explanation of what this method is good for in the context of the data container, see the documentation of [QCPDataContainer](#).

6.40.3.5 sortKeyIsMainKey()

```
static static bool QCPGraphData::sortKeyIsMainKey ( ) [inline], [static]
```

Since the member *key* is both the data point key coordinate and the data ordering parameter, this method returns true.

For a general explanation of what this method is good for in the context of the data container, see the documentation of [QCPDataContainer](#).

6.40.3.6 valueRange()

```
QCPRange QCPGraphData::valueRange ( ) const [inline]
```

Returns a [QCPRange](#) with both lower and upper boundary set to *value* of this data point.

For a general explanation of what this method is good for in the context of the data container, see the documentation of [QCPDataContainer](#).

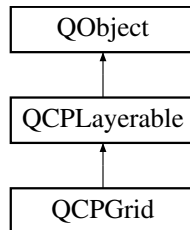
The documentation for this class was generated from the following files:

- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h↵
- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp↵

6.41 QCPGrid Class Reference

Responsible for drawing the grid of a [QCPAxis](#).

Inheritance diagram for QCPGrid:



Public Member Functions

- [QCPGrid](#) ([QCPAxis](#) *parentAxis)
- bool **subGridVisible** () const
- bool **antialiasedSubGrid** () const
- bool **antialiasedZeroLine** () const
- QPen **pen** () const
- QPen **subGridPen** () const
- QPen **zeroLinePen** () const
- void [setSubGridVisible](#) (bool visible)
- void [setAntialiasedSubGrid](#) (bool enabled)
- void [setAntialiasedZeroLine](#) (bool enabled)
- void [setPen](#) (const QPen &pen)
- void [setSubGridPen](#) (const QPen &pen)
- void [setZeroLinePen](#) (const QPen &pen)

Protected Member Functions

- virtual void **applyDefaultAntialiasingHint** ([QCPPainter](#) *painter) const Q_DECL_OVERRIDE
- virtual void **draw** ([QCPPainter](#) *painter) Q_DECL_OVERRIDE
- void **drawGridLines** ([QCPPainter](#) *painter) const
- void **drawSubGridLines** ([QCPPainter](#) *painter) const

Protected Attributes

- bool **mSubGridVisible**
- bool **mAntialiasedSubGrid**
- bool **mAntialiasedZeroLine**
- QPen **mPen**
- QPen **mSubGridPen**
- QPen **mZeroLinePen**
- [QCPAxis](#) * **mParentAxis**

Friends

- class **QCPAxis**

Additional Inherited Members

6.41.1 Detailed Description

Responsible for drawing the grid of a [QCPAxis](#).

This class is tightly bound to [QCPAxis](#). Every axis owns a grid instance and uses it to draw the grid lines, sub grid lines and zero-line. You can interact with the grid of an axis via [QCPAxis::grid](#). Normally, you don't need to create an instance of [QCPGrid](#) yourself.

The axis and grid drawing was split into two classes to allow them to be placed on different layers (both [QCPAxis](#) and [QCPGrid](#) inherit from [QCPLayerable](#)). Thus it is possible to have the grid in the background and the axes in the foreground, and any plottables/items in between. This described situation is the default setup, see the [QCPLayer](#) documentation.

6.41.2 Constructor & Destructor Documentation

6.41.2.1 QCPGrid()

```
QCPGrid::QCPGrid (
    QCPAxis * parentAxis ) [explicit]
```

Creates a [QCPGrid](#) instance and sets default values.

You shouldn't instantiate grids on their own, since every [QCPAxis](#) brings its own [QCPGrid](#).

6.41.3 Member Function Documentation

6.41.3.1 setAntialiasedSubGrid()

```
void QCPGrid::setAntialiasedSubGrid (
    bool enabled )
```

Sets whether sub grid lines are drawn antialiased.

6.41.3.2 setAntialiasedZeroLine()

```
void QCPGrid::setAntialiasedZeroLine (
    bool enabled )
```

Sets whether zero lines are drawn antialiased.

6.41.3.3 setPen()

```
void QCPGrid::setPen (
    const QPen & pen )
```

Sets the pen with which (major) grid lines are drawn.

6.41.3.4 setSubGridPen()

```
void QCPGrid::setSubGridPen (
    const QPen & pen )
```

Sets the pen with which sub grid lines are drawn.

6.41.3.5 setSubGridVisible()

```
void QCPGrid::setSubGridVisible (
    bool visible )
```

Sets whether grid lines at sub tick marks are drawn.

See also

[setSubGridPen](#)

6.41.3.6 setZeroLinePen()

```
void QCPGrid::setZeroLinePen (
    const QPen & pen )
```

Sets the pen with which zero lines are drawn.

Zero lines are lines at value coordinate 0 which may be drawn with a different pen than other grid lines. To disable zero lines and just draw normal grid lines at zero, set *pen* to Qt::NoPen.

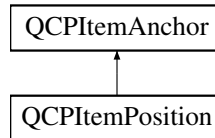
The documentation for this class was generated from the following files:

- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.↵h
- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.↵cpp

6.42 QCItemAnchor Class Reference

An anchor of an item to which positions can be attached to.

Inheritance diagram for QCItemAnchor:



Public Member Functions

- [QCItemAnchor](#) ([QCustomPlot](#) *parentPlot, [QCPAbstractItem](#) *parentItem, const QString &name, int anchorId=-1)
- QString **name** () const
- virtual QPointF [pixelPosition](#) () const

Protected Member Functions

- virtual [QCItemPosition](#) * [toQCItemPosition](#) ()
- void **addChildX** ([QCItemPosition](#) *pos)
- void **removeChildX** ([QCItemPosition](#) *pos)
- void **addChildY** ([QCItemPosition](#) *pos)
- void **removeChildY** ([QCItemPosition](#) *pos)

Protected Attributes

- QString **mName**
- [QCustomPlot](#) * **mParentPlot**
- [QCPAbstractItem](#) * **mParentItem**
- int **mAnchorId**
- QSet< [QCItemPosition](#) * > **mChildrenX**
- QSet< [QCItemPosition](#) * > **mChildrenY**

Friends

- class **QCItemPosition**

6.42.1 Detailed Description

An anchor of an item to which positions can be attached to.

An item ([QCPAbstractItem](#)) may have one or more anchors. Unlike [QCItemPosition](#), an anchor doesn't control anything on its item, but provides a way to tie other items via their positions to the anchor.

For example, a [QCItemRect](#) is defined by its positions *topLeft* and *bottomRight*. Additionally it has various anchors like *top*, *topRight* or *bottomLeft* etc. So you can attach the *start* (which is a [QCItemPosition](#)) of a [QCItemLine](#) to one of the anchors by calling [QCItemPosition::setParentAnchor](#) on *start*, passing the wanted anchor of the [QCItemRect](#). This way the start of the line will now always follow the respective anchor location on the rect item.

Note that [QCItemPosition](#) derives from [QCItemAnchor](#), so every position can also serve as an anchor to other positions.

To learn how to provide anchors in your own item subclasses, see the subclassing section of the [QCPAbstractItem](#) documentation.

6.42.2 Constructor & Destructor Documentation

6.42.2.1 QCItemAnchor()

```
QCItemAnchor::QCItemAnchor (
    QCustomPlot * parentPlot,
    QCPAbstractItem * parentItem,
    const QString & name,
    int anchorId = -1 )
```

Creates a new [QCItemAnchor](#). You shouldn't create [QCItemAnchor](#) instances directly, even if you want to make a new item subclass. Use [QCPAbstractItem::createAnchor](#) instead, as explained in the subclassing section of the [QCPAbstractItem](#) documentation.

6.42.3 Member Function Documentation

6.42.3.1 pixelPosition()

```
QPointF QCItemAnchor::pixelPosition ( ) const [virtual]
```

Returns the final absolute pixel position of the [QCItemAnchor](#) on the [QCustomPlot](#) surface.

The pixel information is internally retrieved via [QCPAbstractItem::anchorPixelPosition](#) of the parent item, [QCItemAnchor](#) is just an intermediary.

Reimplemented in [QCItemPosition](#).

6.42.3.2 toQCItemPosition()

```
QCItemPosition * QCItemAnchor::toQCItemPosition ( ) [inline], [protected], [virtual]
```

Returns 0 if this instance is merely a [QCItemAnchor](#), and a valid pointer of type [QCItemPosition*](#) if it actually is a [QCItemPosition](#) (which is a subclass of [QCItemAnchor](#)).

This safe downcast functionality could also be achieved with a `dynamic_cast`. However, [QCustomPlot](#) avoids `dynamic_cast` to work with projects that don't have RTTI support enabled (e.g. `-fno-rtti` flag with gcc compiler).

Reimplemented in [QCItemPosition](#).

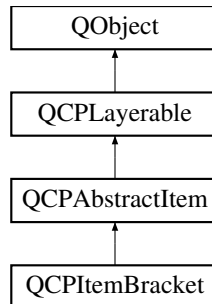
The documentation for this class was generated from the following files:

- `/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h`
- `/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp`

6.43 QCItemBracket Class Reference

A bracket for referencing/highlighting certain parts in the plot.

Inheritance diagram for QCItemBracket:



Public Types

- enum [BracketStyle](#) { [bsSquare](#), [bsRound](#), [bsCurly](#), [bsCalligraphic](#) }

Public Member Functions

- [QCItemBracket](#) ([QCustomPlot](#) *parentPlot)
- [QPen](#) **pen** () const
- [QPen](#) **selectedPen** () const
- double **length** () const
- [BracketStyle](#) **style** () const
- void [setPen](#) (const [QPen](#) &pen)
- void [setSelectedPen](#) (const [QPen](#) &pen)
- void [setLength](#) (double length)
- void [setStyle](#) ([BracketStyle](#) style)
- virtual double [selectTest](#) (const [QPointF](#) &pos, bool onlySelectable, [QVariant](#) *details=0) const [Q_DECL_OVERRIDE](#)

Public Attributes

- [QCItemPosition](#) *const **left**
- [QCItemPosition](#) *const **right**
- [QCItemAnchor](#) *const **center**

Protected Types

- enum **AnchorIndex** { **aiCenter** }

Protected Member Functions

- virtual void **draw** ([QCPPainter](#) *painter) [Q_DECL_OVERRIDE](#)
- virtual [QPointF](#) **anchorPixelPosition** (int anchorId) const [Q_DECL_OVERRIDE](#)
- [QPen](#) **mainPen** () const

Protected Attributes

- QPen **mPen**
- QPen **mSelectedPen**
- double **mLength**
- [BracketStyle](#) **mStyle**

Additional Inherited Members

6.43.1 Detailed Description

A bracket for referencing/highlighting certain parts in the plot.

It has two positions, *left* and *right*, which define the span of the bracket. If *left* is actually farther to the left than *right*, the bracket is opened to the bottom, as shown in the example image.

The bracket supports multiple styles via [setStyle](#). The length, i.e. how far the bracket stretches away from the embraced span, can be controlled with [setLength](#).

Demonstrating the effect of different values for [setLength](#), for styles [bsCalligraphic](#) and [bsSquare](#). Anchors and positions are displayed for reference.

It provides an anchor *center*, to allow connection of other items, e.g. an arrow ([QCPIItemLine](#) or [QCPIItemCurve](#)) or a text label ([QCPIItemText](#)), to the bracket.

6.43.2 Member Enumeration Documentation

6.43.2.1 BracketStyle

enum [QCPIItemBracket::BracketStyle](#)

Defines the various visual shapes of the bracket item. The appearance can be further modified by [setLength](#) and [setPen](#).

See also

[setStyle](#)

Enumerator

bsSquare	A brace with angled edges.
bsRound	A brace with round edges.
bsCurly	A curly brace.
bsCalligraphic	A curly brace with varying stroke width giving a calligraphic impression.

6.43.3 Constructor & Destructor Documentation

6.43.3.1 QCPItemBracket()

```
QCPItemBracket::QCPItemBracket (
    QCustomPlot * parentPlot ) [explicit]
```

Creates a bracket item and sets default values.

The created item is automatically registered with *parentPlot*. This [QCustomPlot](#) instance takes ownership of the item, so do not delete it manually but use [QCustomPlot::removeItem\(\)](#) instead.

6.43.4 Member Function Documentation

6.43.4.1 selectTest()

```
double QCPItemBracket::selectTest (
    const QPointF & pos,
    bool onlySelectable,
    QVariant * details = 0 ) const [virtual]
```

This function is used to decide whether a click hits a layerable object or not.

pos is a point in pixel coordinates on the [QCustomPlot](#) surface. This function returns the shortest pixel distance of this point to the object. If the object is either invisible or the distance couldn't be determined, -1.0 is returned. Further, if *onlySelectable* is true and the object is not selectable, -1.0 is returned, too.

If the object is represented not by single lines but by an area like a [QCPItemText](#) or the bars of a [QCPBars](#) plottable, a click inside the area should also be considered a hit. In these cases this function thus returns a constant value greater zero but still below the parent plot's selection tolerance. (typically the selectionTolerance multiplied by 0.99).

Providing a constant value for area objects allows selecting line objects even when they are obscured by such area objects, by clicking close to the lines (i.e. closer than $0.99 \cdot \text{selectionTolerance}$).

The actual setting of the selection state is not done by this function. This is handled by the parent [QCustomPlot](#) when the `mouseReleaseEvent` occurs, and the finally selected object is notified via the `selectEvent/deselectEvent` methods.

details is an optional output parameter. Every layerable subclass may place any information in *details*. This information will be passed to `selectEvent` when the parent [QCustomPlot](#) decides on the basis of this `selectTest` call, that the object was successfully selected. The subsequent call to `selectEvent` will carry the *details*. This is useful for multi-part objects (like [QCPAxis](#)). This way, a possibly complex calculation to decide which part was clicked is only done once in [selectTest](#). The result (i.e. the actually clicked part) can then be placed in *details*. So in the subsequent `selectEvent`, the decision which part was selected doesn't have to be done a second time for a single selection operation.

You may pass 0 as *details* to indicate that you are not interested in those selection details.

See also

`selectEvent`, `deselectEvent`, [mousePressEvent](#), [wheelEvent](#), [QCustomPlot::setInteractions](#)

Implements [QCPAbstractItem](#).

6.43.4.2 `setLength()`

```
void QCPItemBracket::setLength (
    double length )
```

Sets the *length* in pixels how far the bracket extends in the direction towards the embraced span of the bracket (i.e. perpendicular to the *left-right*-direction)

Demonstrating the effect of different values for [setLength](#), for styles [bsCalligraphic](#) and [bsSquare](#). Anchors and positions are displayed for reference.

6.43.4.3 `setPen()`

```
void QCPItemBracket::setPen (
    const QPen & pen )
```

Sets the pen that will be used to draw the bracket.

Note that when the style is [bsCalligraphic](#), only the color will be taken from the pen, the stroke and width are ignored. To change the apparent stroke width of a calligraphic bracket, use [setLength](#), which has a similar effect.

See also

[setSelectedPen](#)

6.43.4.4 `setSelectedPen()`

```
void QCPItemBracket::setSelectedPen (
    const QPen & pen )
```

Sets the pen that will be used to draw the bracket when selected

See also

[setPen](#), [setSelected](#)

6.43.4.5 `setStyle()`

```
void QCPItemBracket::setStyle (
    QCPItemBracket::BracketStyle style )
```

Sets the style of the bracket, i.e. the shape/visual appearance.

See also

[setPen](#)

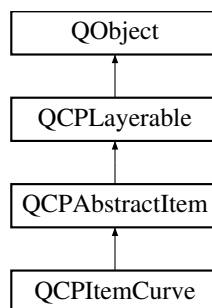
The documentation for this class was generated from the following files:

- [/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h](#)
- [/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp](#)

6.44 QCPItemCurve Class Reference

A curved line from one point to another.

Inheritance diagram for QCPItemCurve:



Public Member Functions

- [QCPItemCurve](#) ([QCustomPlot](#) *parentPlot)
- `QPen` **pen** () const
- `QPen` **selectedPen** () const
- [QCPLineEnding](#) **head** () const
- [QCPLineEnding](#) **tail** () const
- void [setPen](#) (const `QPen` &pen)
- void [setSelectedPen](#) (const `QPen` &pen)
- void [setHead](#) (const [QCPLineEnding](#) &head)
- void [setTail](#) (const [QCPLineEnding](#) &tail)
- virtual double [selectTest](#) (const `QPointF` &pos, bool onlySelectable, `QVariant` *details=0) const `Q_DECL_OVERRIDE`

Public Attributes

- [QCItemPosition](#) *const **start**
- [QCItemPosition](#) *const **startDir**
- [QCItemPosition](#) *const **endDir**
- [QCItemPosition](#) *const **end**

Protected Member Functions

- virtual void **draw** ([QCPPainter](#) *painter) Q_DECL_OVERRIDE
- QPen **mainPen** () const

Protected Attributes

- QPen **mPen**
- QPen **mSelectedPen**
- [QCLineEnding](#) **mHead**
- [QCLineEnding](#) **mTail**

Additional Inherited Members

6.44.1 Detailed Description

A curved line from one point to another.

It has four positions, *start* and *end*, which define the end points of the line, and two control points which define the direction the line exits from the start and the direction from which it approaches the end: *startDir* and *endDir*.

With [setHead](#) and [setTail](#) you may set different line ending styles, e.g. to create an arrow.

Often it is desirable for the control points to stay at fixed relative positions to the start/end point. This can be achieved by setting the parent anchor e.g. of *startDir* simply to *start*, and then specify the desired pixel offset with [QCItemPosition::setCoords](#) on *startDir*.

6.44.2 Constructor & Destructor Documentation

6.44.2.1 QCItemCurve()

```
QCItemCurve::QCItemCurve (
    QCustomPlot * parentPlot ) [explicit]
```

Creates a curve item and sets default values.

The created item is automatically registered with *parentPlot*. This [QCustomPlot](#) instance takes ownership of the item, so do not delete it manually but use [QCustomPlot::removeItem\(\)](#) instead.

6.44.3 Member Function Documentation

6.44.3.1 selectTest()

```
double QCItemCurve::selectTest (
    const QPointF & pos,
    bool onlySelectable,
    QVariant * details = 0 ) const [virtual]
```

This function is used to decide whether a click hits a layerable object or not.

pos is a point in pixel coordinates on the [QCustomPlot](#) surface. This function returns the shortest pixel distance of this point to the object. If the object is either invisible or the distance couldn't be determined, -1.0 is returned. Further, if *onlySelectable* is true and the object is not selectable, -1.0 is returned, too.

If the object is represented not by single lines but by an area like a [QCItemText](#) or the bars of a [QCPBars](#) plottable, a click inside the area should also be considered a hit. In these cases this function thus returns a constant value greater zero but still below the parent plot's selection tolerance. (typically the selectionTolerance multiplied by 0.99).

Providing a constant value for area objects allows selecting line objects even when they are obscured by such area objects, by clicking close to the lines (i.e. closer than 0.99*selectionTolerance).

The actual setting of the selection state is not done by this function. This is handled by the parent [QCustomPlot](#) when the mouseReleaseEvent occurs, and the finally selected object is notified via the selectEvent/deselectEvent methods.

details is an optional output parameter. Every layerable subclass may place any information in *details*. This information will be passed to selectEvent when the parent [QCustomPlot](#) decides on the basis of this selectTest call, that the object was successfully selected. The subsequent call to selectEvent will carry the *details*. This is useful for multi-part objects (like [QCPAxis](#)). This way, a possibly complex calculation to decide which part was clicked is only done once in [selectTest](#). The result (i.e. the actually clicked part) can then be placed in *details*. So in the subsequent selectEvent, the decision which part was selected doesn't have to be done a second time for a single selection operation.

You may pass 0 as *details* to indicate that you are not interested in those selection details.

See also

selectEvent, deselectEvent, [mousePressEvent](#), [wheelEvent](#), [QCustomPlot::setInteractions](#)

Implements [QCPAbstractItem](#).

6.44.3.2 setHead()

```
void QCItemCurve::setHead (
    const QCPLLineEnding & head )
```

Sets the line ending style of the head. The head corresponds to the *end* position.

Note that due to the overloaded [QCPLLineEnding](#) constructor, you may directly specify a [QCPLLineEnding::EndingStyle](#) here, e.g.

```
setHead(QCPLLineEnding::esSpikeArrow)
```

See also

[setTail](#)

6.44.3.3 setPen()

```
void QCItemCurve::setPen (
    const QPen & pen )
```

Sets the pen that will be used to draw the line

See also

[setSelectedPen](#)

6.44.3.4 setSelectedPen()

```
void QCItemCurve::setSelectedPen (
    const QPen & pen )
```

Sets the pen that will be used to draw the line when selected

See also

[setPen](#), [setSelected](#)

6.44.3.5 setTail()

```
void QCItemCurve::setTail (
    const QCPLLineEnding & tail )
```

Sets the line ending style of the tail. The tail corresponds to the *start* position.

Note that due to the overloaded [QCPLLineEnding](#) constructor, you may directly specify a [QCPLLineEnding::Ending↵](#) [Style](#) here, e.g.

```
setTail(QCPLLineEnding::esSpikeArrow)
```

See also

[setHead](#)

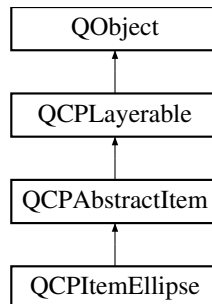
The documentation for this class was generated from the following files:

- [/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.↵](#)
h
- [/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.↵](#)
cpp

6.45 QCPItemEllipse Class Reference

An ellipse.

Inheritance diagram for QCPItemEllipse:



Public Member Functions

- [QCPItemEllipse](#) ([QCustomPlot](#) *parentPlot)
- [QPen](#) **pen** () const
- [QPen](#) **selectedPen** () const
- [QBrush](#) **brush** () const
- [QBrush](#) **selectedBrush** () const
- void [setPen](#) (const [QPen](#) &pen)
- void [setSelectedPen](#) (const [QPen](#) &pen)
- void [setBrush](#) (const [QBrush](#) &brush)
- void [setSelectedBrush](#) (const [QBrush](#) &brush)
- virtual double [selectTest](#) (const [QPointF](#) &pos, bool onlySelectable, [QVariant](#) *details=0) const [Q_DECL_OVERRIDE](#)

Public Attributes

- [QCPItemPosition](#) *const **topLeft**
- [QCPItemPosition](#) *const **bottomRight**
- [QCPItemAnchor](#) *const **topLeftRim**
- [QCPItemAnchor](#) *const **top**
- [QCPItemAnchor](#) *const **topRightRim**
- [QCPItemAnchor](#) *const **right**
- [QCPItemAnchor](#) *const **bottomRightRim**
- [QCPItemAnchor](#) *const **bottom**
- [QCPItemAnchor](#) *const **bottomLeftRim**
- [QCPItemAnchor](#) *const **left**
- [QCPItemAnchor](#) *const **center**

Protected Types

- enum **AnchorIndex** {
[aiTopLeftRim](#), [aiTop](#), [aiTopRightRim](#), [aiRight](#),
[aiBottomRightRim](#), [aiBottom](#), [aiBottomLeftRim](#), [aiLeft](#),
[aiCenter](#) }

Protected Member Functions

- virtual void **draw** ([QCPPainter](#) *painter) Q_DECL_OVERRIDE
- virtual QPointF **anchorPixelPosition** (int anchorId) const Q_DECL_OVERRIDE
- QPen **mainPen** () const
- QBrush **mainBrush** () const

Protected Attributes

- QPen **mPen**
- QPen **mSelectedPen**
- QBrush **mBrush**
- QBrush **mSelectedBrush**

Additional Inherited Members

6.45.1 Detailed Description

An ellipse.

It has two positions, *topLeft* and *bottomRight*, which define the rect the ellipse will be drawn in.

6.45.2 Constructor & Destructor Documentation

6.45.2.1 QCPItemEllipse()

```
QCPItemEllipse::QCPItemEllipse (  
    QCustomPlot * parentPlot ) [explicit]
```

Creates an ellipse item and sets default values.

The created item is automatically registered with *parentPlot*. This [QCustomPlot](#) instance takes ownership of the item, so do not delete it manually but use [QCustomPlot::removeItem\(\)](#) instead.

6.45.3 Member Function Documentation

6.45.3.1 selectTest()

```
double QCItemEllipse::selectTest (
    const QPointF & pos,
    bool onlySelectable,
    QVariant * details = 0 ) const [virtual]
```

This function is used to decide whether a click hits a layerable object or not.

pos is a point in pixel coordinates on the [QCustomPlot](#) surface. This function returns the shortest pixel distance of this point to the object. If the object is either invisible or the distance couldn't be determined, -1.0 is returned. Further, if *onlySelectable* is true and the object is not selectable, -1.0 is returned, too.

If the object is represented not by single lines but by an area like a [QCItemText](#) or the bars of a [QCPBars](#) plottable, a click inside the area should also be considered a hit. In these cases this function thus returns a constant value greater zero but still below the parent plot's selection tolerance. (typically the selectionTolerance multiplied by 0.99).

Providing a constant value for area objects allows selecting line objects even when they are obscured by such area objects, by clicking close to the lines (i.e. closer than $0.99 \cdot \text{selectionTolerance}$).

The actual setting of the selection state is not done by this function. This is handled by the parent [QCustomPlot](#) when the `mouseReleaseEvent` occurs, and the finally selected object is notified via the `selectEvent/deselectEvent` methods.

details is an optional output parameter. Every layerable subclass may place any information in *details*. This information will be passed to `selectEvent` when the parent [QCustomPlot](#) decides on the basis of this `selectTest` call, that the object was successfully selected. The subsequent call to `selectEvent` will carry the *details*. This is useful for multi-part objects (like [QCPAxis](#)). This way, a possibly complex calculation to decide which part was clicked is only done once in `selectTest`. The result (i.e. the actually clicked part) can then be placed in *details*. So in the subsequent `selectEvent`, the decision which part was selected doesn't have to be done a second time for a single selection operation.

You may pass 0 as *details* to indicate that you are not interested in those selection details.

See also

`selectEvent`, `deselectEvent`, [mousePressEvent](#), [wheelEvent](#), [QCustomPlot::setInteractions](#)

Implements [QCPAbstractItem](#).

6.45.3.2 setBrush()

```
void QCItemEllipse::setBrush (
    const QBrush & brush )
```

Sets the brush that will be used to fill the ellipse. To disable filling, set *brush* to `Qt::NoBrush`.

See also

[setSelectedBrush](#), [setPen](#)

6.45.3.3 setPen()

```
void QCPIItemEllipse::setPen (
    const QPen & pen )
```

Sets the pen that will be used to draw the line of the ellipse

See also

[setSelectedPen](#), [setBrush](#)

6.45.3.4 setSelectedBrush()

```
void QCPIItemEllipse::setSelectedBrush (
    const QBrush & brush )
```

Sets the brush that will be used to fill the ellipse when selected. To disable filling, set *brush* to Qt::NoBrush.

See also

[setBrush](#)

6.45.3.5 setSelectedPen()

```
void QCPIItemEllipse::setSelectedPen (
    const QPen & pen )
```

Sets the pen that will be used to draw the line of the ellipse when selected

See also

[setPen](#), [setSelected](#)

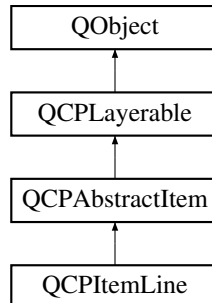
The documentation for this class was generated from the following files:

- [/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h](#)↔
- [/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp](#)↔

6.46 QCPLItemLine Class Reference

A line from one point to another.

Inheritance diagram for QCPLItemLine:



Public Member Functions

- [QCPLItemLine](#) ([QCustomPlot](#) *parentPlot)
- [QPen](#) **pen** () const
- [QPen](#) **selectedPen** () const
- [QCPLineEnding](#) **head** () const
- [QCPLineEnding](#) **tail** () const
- void **setPen** (const [QPen](#) &pen)
- void **setSelectedPen** (const [QPen](#) &pen)
- void **setHead** (const [QCPLineEnding](#) &head)
- void **setTail** (const [QCPLineEnding](#) &tail)
- virtual double **selectTest** (const [QPointF](#) &pos, bool onlySelectable, [QVariant](#) *details=0) const [Q_DECL_OVERRIDE](#)

Public Attributes

- [QCPLItemPosition](#) *const **start**
- [QCPLItemPosition](#) *const **end**

Protected Member Functions

- virtual void **draw** ([QCPPainter](#) *painter) [Q_DECL_OVERRIDE](#)
- [QLineF](#) **getRectClippedLine** (const [QCPVector2D](#) &start, const [QCPVector2D](#) &end, const [QRect](#) &rect) const
- [QPen](#) **mainPen** () const

Protected Attributes

- [QPen](#) **mPen**
- [QPen](#) **mSelectedPen**
- [QCPLineEnding](#) **mHead**
- [QCPLineEnding](#) **mTail**

Additional Inherited Members

6.46.1 Detailed Description

A line from one point to another.

It has two positions, *start* and *end*, which define the end points of the line.

With [setHead](#) and [setTail](#) you may set different line ending styles, e.g. to create an arrow.

6.46.2 Constructor & Destructor Documentation

6.46.2.1 QCPItemLine()

```
QCPItemLine::QCPItemLine (
    QCustomPlot * parentPlot ) [explicit]
```

Creates a line item and sets default values.

The created item is automatically registered with *parentPlot*. This [QCustomPlot](#) instance takes ownership of the item, so do not delete it manually but use [QCustomPlot::removeItem\(\)](#) instead.

6.46.3 Member Function Documentation

6.46.3.1 selectTest()

```
double QCPItemLine::selectTest (
    const QPointF & pos,
    bool onlySelectable,
    QVariant * details = 0 ) const [virtual]
```

This function is used to decide whether a click hits a layerable object or not.

pos is a point in pixel coordinates on the [QCustomPlot](#) surface. This function returns the shortest pixel distance of this point to the object. If the object is either invisible or the distance couldn't be determined, -1.0 is returned. Further, if *onlySelectable* is true and the object is not selectable, -1.0 is returned, too.

If the object is represented not by single lines but by an area like a [QCPItemText](#) or the bars of a [QCPBars](#) plottable, a click inside the area should also be considered a hit. In these cases this function thus returns a constant value greater zero but still below the parent plot's selection tolerance. (typically the selectionTolerance multiplied by 0.99).

Providing a constant value for area objects allows selecting line objects even when they are obscured by such area objects, by clicking close to the lines (i.e. closer than 0.99*selectionTolerance).

The actual setting of the selection state is not done by this function. This is handled by the parent [QCustomPlot](#) when the `mouseReleaseEvent` occurs, and the finally selected object is notified via the `selectEvent/deselectEvent` methods.

details is an optional output parameter. Every layerable subclass may place any information in *details*. This information will be passed to `selectEvent` when the parent [QCustomPlot](#) decides on the basis of this `selectTest` call, that the object was successfully selected. The subsequent call to `selectEvent` will carry the *details*. This is useful for multi-part objects (like [QCPAxis](#)). This way, a possibly complex calculation to decide which part was clicked is only done once in `selectTest`. The result (i.e. the actually clicked part) can then be placed in *details*. So in the subsequent `selectEvent`, the decision which part was selected doesn't have to be done a second time for a single selection operation.

You may pass 0 as *details* to indicate that you are not interested in those selection details.

See also

[selectEvent](#), [deselectEvent](#), [mousePressEvent](#), [wheelEvent](#), [QCustomPlot::setInteractions](#)

Implements [QCPLAbstractItem](#).

6.46.3.2 setHead()

```
void QCPLItemLine::setHead (
    const QCPLLineEnding & head )
```

Sets the line ending style of the head. The head corresponds to the *end* position.

Note that due to the overloaded [QCPLLineEnding](#) constructor, you may directly specify a [QCPLLineEnding::Ending↵Style](#) here, e.g.

```
setHead(QCPLLineEnding::esSpikeArrow)
```

See also

[setTail](#)

6.46.3.3 setPen()

```
void QCPLItemLine::setPen (
    const QPen & pen )
```

Sets the pen that will be used to draw the line

See also

[setSelectedPen](#)

6.46.3.4 setSelectedPen()

```
void QCPLItemLine::setSelectedPen (
    const QPen & pen )
```

Sets the pen that will be used to draw the line when selected

See also

[setPen](#), [setSelected](#)

6.46.3.5 setTail()

```
void QCPLItemLine::setTail (
    const QCPLLineEnding & tail )
```

Sets the line ending style of the tail. The tail corresponds to the *start* position.

Note that due to the overloaded [QCPLLineEnding](#) constructor, you may directly specify a [QCPLLineEnding::EndingStyle](#) here, e.g.

```
setTail(QCPLLineEnding::esSpikeArrow)
```

See also

[setHead](#)

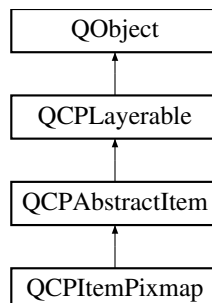
The documentation for this class was generated from the following files:

- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h
- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp

6.47 QCPLItemPixmap Class Reference

An arbitrary pixmap.

Inheritance diagram for QCPLItemPixmap:



Public Member Functions

- [QCPLItemPixmap](#) ([QCustomPlot](#) *parentPlot)
- QPixmap **pixmap** () const
- bool **scaled** () const
- Qt::AspectRatioMode **aspectRatioMode** () const
- Qt::TransformationMode **transformationMode** () const
- QPen **pen** () const
- QPen **selectedPen** () const
- void [setPixmap](#) (const QPixmap &pixmap)
- void [setScale](#) (bool scaled, Qt::AspectRatioMode aspectRatioMode=Qt::KeepAspectRatio, Qt::TransformationMode transformationMode=Qt::SmoothTransformation)
- void [setPen](#) (const QPen &pen)
- void [setSelectedPen](#) (const QPen &pen)
- virtual double [selectTest](#) (const QPointF &pos, bool onlySelectable, QVariant *details=0) const Q_DECL_OVERRIDE

Public Attributes

- [QCItemPosition](#) *const **topLeft**
- [QCItemPosition](#) *const **bottomRight**
- [QCItemAnchor](#) *const **top**
- [QCItemAnchor](#) *const **topRight**
- [QCItemAnchor](#) *const **right**
- [QCItemAnchor](#) *const **bottom**
- [QCItemAnchor](#) *const **bottomLeft**
- [QCItemAnchor](#) *const **left**

Protected Types

- enum **AnchorIndex** {
 aiTop, **aiTopRight**, **aiRight**, **aiBottom**,
 aiBottomLeft, **aiLeft** }

Protected Member Functions

- virtual void **draw** ([QCPPainter](#) *painter) Q_DECL_OVERRIDE
- virtual QPointF **anchorPixelPosition** (int anchorId) const Q_DECL_OVERRIDE
- void **updateScaledPixmap** (QRect finalRect=QRect(), bool flipHorz=false, bool flipVert=false)
- QRect **getFinalRect** (bool *flippedHorz=0, bool *flippedVert=0) const
- QPen **mainPen** () const

Protected Attributes

- QPixmap **mPixmap**
- QPixmap **mScaledPixmap**
- bool **mScaled**
- bool **mScaledPixmapInvalidated**
- Qt::AspectRatioMode **mAspectRatioMode**
- Qt::TransformationMode **mTransformationMode**
- QPen **mPen**
- QPen **mSelectedPen**

Additional Inherited Members

6.47.1 Detailed Description

An arbitrary pixmap.

It has two positions, *topLeft* and *bottomRight*, which define the rectangle the pixmap will be drawn in. Depending on the scale setting ([setScaled](#)), the pixmap will be either scaled to fit the rectangle or be drawn aligned to the *topLeft* position.

If scaling is enabled and *topLeft* is further to the bottom/right than *bottomRight* (as shown on the right side of the example image), the pixmap will be flipped in the respective orientations.

6.47.2 Constructor & Destructor Documentation

6.47.2.1 QCPItemPixmap()

```
QCPItemPixmap::QCPItemPixmap (
    QCustomPlot * parentPlot ) [explicit]
```

Creates a rectangle item and sets default values.

The created item is automatically registered with *parentPlot*. This [QCustomPlot](#) instance takes ownership of the item, so do not delete it manually but use [QCustomPlot::removeItem\(\)](#) instead.

6.47.3 Member Function Documentation

6.47.3.1 selectTest()

```
double QCPItemPixmap::selectTest (
    const QPointF & pos,
    bool onlySelectable,
    QVariant * details = 0 ) const [virtual]
```

This function is used to decide whether a click hits a layerable object or not.

pos is a point in pixel coordinates on the [QCustomPlot](#) surface. This function returns the shortest pixel distance of this point to the object. If the object is either invisible or the distance couldn't be determined, -1.0 is returned. Further, if *onlySelectable* is true and the object is not selectable, -1.0 is returned, too.

If the object is represented not by single lines but by an area like a [QCPItemText](#) or the bars of a [QCPBars](#) plottable, a click inside the area should also be considered a hit. In these cases this function thus returns a constant value greater zero but still below the parent plot's selection tolerance. (typically the selectionTolerance multiplied by 0.99).

Providing a constant value for area objects allows selecting line objects even when they are obscured by such area objects, by clicking close to the lines (i.e. closer than $0.99 \cdot \text{selectionTolerance}$).

The actual setting of the selection state is not done by this function. This is handled by the parent [QCustomPlot](#) when the `mouseReleaseEvent` occurs, and the finally selected object is notified via the `selectEvent/deselectEvent` methods.

details is an optional output parameter. Every layerable subclass may place any information in *details*. This information will be passed to `selectEvent` when the parent [QCustomPlot](#) decides on the basis of this `selectTest` call, that the object was successfully selected. The subsequent call to `selectEvent` will carry the *details*. This is useful for multi-part objects (like [QCPAxis](#)). This way, a possibly complex calculation to decide which part was clicked is only done once in [selectTest](#). The result (i.e. the actually clicked part) can then be placed in *details*. So in the subsequent `selectEvent`, the decision which part was selected doesn't have to be done a second time for a single selection operation.

You may pass 0 as *details* to indicate that you are not interested in those selection details.

See also

`selectEvent`, `deselectEvent`, [mousePressEvent](#), [wheelEvent](#), [QCustomPlot::setInteractions](#)

Implements [QCPAbstractItem](#).

6.47.3.2 setPen()

```
void QCPItemPixmap::setPen (
    const QPen & pen )
```

Sets the pen that will be used to draw a border around the pixmap.

See also

[setSelectedPen](#), [setBrush](#)

6.47.3.3 setPixmap()

```
void QCPItemPixmap::setPixmap (
    const QPixmap & pixmap )
```

Sets the pixmap that will be displayed.

6.47.3.4 setScaled()

```
void QCPItemPixmap::setScaled (
    bool scaled,
    Qt::AspectRatioMode aspectRatioMode = Qt::KeepAspectRatio,
    Qt::TransformationMode transformationMode = Qt::SmoothTransformation )
```

Sets whether the pixmap will be scaled to fit the rectangle defined by the *topLeft* and *bottomRight* positions.

6.47.3.5 setSelectedPen()

```
void QCPItemPixmap::setSelectedPen (
    const QPen & pen )
```

Sets the pen that will be used to draw a border around the pixmap when selected

See also

[setPen](#), [setSelected](#)

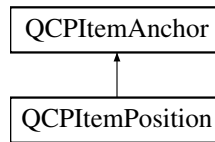
The documentation for this class was generated from the following files:

- `/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h`
- `/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp`

6.48 QCItemPosition Class Reference

Manages the position of an item.

Inheritance diagram for QCItemPosition:



Public Types

- enum [PositionType](#) { [ptAbsolute](#), [ptViewportRatio](#), [ptAxisRectRatio](#), [ptPlotCoords](#) }

Public Member Functions

- [QCItemPosition](#) ([QCustomPlot](#) *parentPlot, [QCPAbstractItem](#) *parentItem, const QString &name)
- [PositionType](#) type () const
- [PositionType](#) **typeX** () const
- [PositionType](#) **typeY** () const
- [QCItemAnchor](#) * [parentAnchor](#) () const
- [QCItemAnchor](#) * **parentAnchorX** () const
- [QCItemAnchor](#) * **parentAnchorY** () const
- double **key** () const
- double **value** () const
- QPointF **coords** () const
- [QCPAxis](#) * **keyAxis** () const
- [QCPAxis](#) * **valueAxis** () const
- [QCPAxisRect](#) * **axisRect** () const
- virtual QPointF [pixelPosition](#) () const
- void [setType](#) ([PositionType](#) type)
- void [setTypeX](#) ([PositionType](#) type)
- void [setTypeY](#) ([PositionType](#) type)
- bool [setParentAnchor](#) ([QCItemAnchor](#) *parentAnchor, bool keepPixelPosition=false)
- bool [setParentAnchorX](#) ([QCItemAnchor](#) *parentAnchor, bool keepPixelPosition=false)
- bool [setParentAnchorY](#) ([QCItemAnchor](#) *parentAnchor, bool keepPixelPosition=false)
- void [setCoords](#) (double key, double value)
- void [setCoords](#) (const QPointF &coords)
- void [setAxes](#) ([QCPAxis](#) *keyAxis, [QCPAxis](#) *valueAxis)
- void [setAxisRect](#) ([QCPAxisRect](#) *axisRect)
- void [setPixelPosition](#) (const QPointF &pixelPosition)

Protected Member Functions

- virtual [QCItemPosition](#) * [toQCItemPosition](#) () Q_DECL_OVERRIDE

Protected Attributes

- [PositionType](#) **mPositionTypeX**
- [PositionType](#) **mPositionTypeY**
- [QPointer< QCPAxis >](#) **mKeyAxis**
- [QPointer< QCPAxis >](#) **mValueAxis**
- [QPointer< QCPAxisRect >](#) **mAxisRect**
- double **mKey**
- double **mValue**
- [QCItemAnchor](#) * **mParentAnchorX**
- [QCItemAnchor](#) * **mParentAnchorY**

6.48.1 Detailed Description

Manages the position of an item.

Every item has at least one public [QCItemPosition](#) member pointer which provides ways to position the item on the [QCustomPlot](#) surface. Some items have multiple positions, for example [QCItemRect](#) has two: *topLeft* and *bottomRight*.

[QCItemPosition](#) has a type ([PositionType](#)) that can be set with [setType](#). This type defines how coordinates passed to [setCoords](#) are to be interpreted, e.g. as absolute pixel coordinates, as plot coordinates of certain axes, etc. For more advanced plots it is also possible to assign different types per X/Y coordinate of the position (see [setTypeX](#), [setTypeY](#)). This way an item could be positioned at a fixed pixel distance from the top in the Y direction, while following a plot coordinate in the X direction.

A [QCItemPosition](#) may have a parent [QCItemAnchor](#), see [setParentAnchor](#). This way you can tie multiple items together. If the [QCItemPosition](#) has a parent, its coordinates ([setCoords](#)) are considered to be absolute pixels in the reference frame of the parent anchor, where (0, 0) means directly ontop of the parent anchor. For example, You could attach the *start* position of a [QCItemLine](#) to the *bottom* anchor of a [QCItemText](#) to make the starting point of the line always be centered under the text label, no matter where the text is moved to. For more advanced plots, it is possible to assign different parent anchors per X/Y coordinate of the position, see [setParentAnchorX](#), [setParentAnchorY](#). This way an item could follow another item in the X direction but stay at a fixed position in the Y direction. Or even follow item A in X, and item B in Y.

Note that every [QCItemPosition](#) inherits from [QCItemAnchor](#) and thus can itself be used as parent anchor for other positions.

To set the apparent pixel position on the [QCustomPlot](#) surface directly, use [setPixelPosition](#). This works no matter what type this [QCItemPosition](#) is or what parent-child situation it is in, as [setPixelPosition](#) transforms the coordinates appropriately, to make the position appear at the specified pixel values.

6.48.2 Member Enumeration Documentation

6.48.2.1 PositionType

```
enum QCItemPosition::PositionType
```

Defines the ways an item position can be specified. Thus it defines what the numbers passed to [setCoords](#) actually mean.

See also

[setType](#)

Enumerator

ptAbsolute	Static positioning in pixels, starting from the top left corner of the viewport/widget.
ptViewportRatio	Static positioning given by a fraction of the viewport size. For example, if you call <code>setCoords(0, 0)</code> , the position will be at the top left corner of the viewport/widget. <code>setCoords(1, 1)</code> will be at the bottom right corner, <code>setCoords(0.5, 0)</code> will be horizontally centered and vertically at the top of the viewport/widget, etc.
ptAxisRectRatio	Static positioning given by a fraction of the axis rect size (see setAxisRect). For example, if you call <code>setCoords(0, 0)</code> , the position will be at the top left corner of the axis rect. <code>setCoords(1, 1)</code> will be at the bottom right corner, <code>setCoords(0.5, 0)</code> will be horizontally centered and vertically at the top of the axis rect, etc. You can also go beyond the axis rect by providing negative coordinates or coordinates larger than 1.
ptPlotCoords	Dynamic positioning at a plot coordinate defined by two axes (see setAxes).

6.48.3 Constructor & Destructor Documentation

6.48.3.1 QCPItemPosition()

```
QCPItemPosition::QCPItemPosition (
    QCustomPlot * parentPlot,
    QCPAbstractItem * parentItem,
    const QString & name )
```

Creates a new [QCPItemPosition](#). You shouldn't create [QCPItemPosition](#) instances directly, even if you want to make a new item subclass. Use `QCPAbstractItem::createPosition` instead, as explained in the subclassing section of the [QCPAbstractItem](#) documentation.

6.48.4 Member Function Documentation

6.48.4.1 parentAnchor()

```
QCPItemAnchor * QCPItemPosition::parentAnchor ( ) const [inline]
```

Returns the current parent anchor.

If different parent anchors were set for X and Y ([setParentAnchorX](#), [setParentAnchorY](#)), this method returns the parent anchor of the Y coordinate. In that case rather use `parentAnchorX()` and `parentAnchorY()`.

See also

[setParentAnchor](#)

6.48.4.2 pixelPosition()

```
QPointF QCItemPosition::pixelPosition ( ) const [virtual]
```

Returns the final absolute pixel position of the [QCItemPosition](#) on the [QCustomPlot](#) surface. It includes all effects of type ([setType](#)) and possible parent anchors ([setParentAnchor](#)).

See also

[setPixelPosition](#)

Reimplemented from [QCItemAnchor](#).

6.48.4.3 setAxes()

```
void QCItemPosition::setAxes (
    QCPAxis * keyAxis,
    QCPAxis * valueAxis )
```

When [setType](#) is [ptPlotCoords](#), this function may be used to specify the axes the coordinates set with [setCoords](#) relate to. By default they are set to the initial [xAxis](#) and [yAxis](#) of the [QCustomPlot](#).

6.48.4.4 setAxisRect()

```
void QCItemPosition::setAxisRect (
    QCPAxisRect * axisRect )
```

When [setType](#) is [ptAxisRectRatio](#), this function may be used to specify the axis rect the coordinates set with [setCoords](#) relate to. By default this is set to the main axis rect of the [QCustomPlot](#).

6.48.4.5 setCoords() [1/2]

```
void QCItemPosition::setCoords (
    double key,
    double value )
```

Sets the coordinates of this [QCItemPosition](#). What the coordinates mean, is defined by the type ([setType](#), [setTypeX](#), [setTypeY](#)).

For example, if the type is [ptAbsolute](#), *key* and *value* mean the x and y pixel position on the [QCustomPlot](#) surface. In that case the origin (0, 0) is in the top left corner of the [QCustomPlot](#) viewport. If the type is [ptPlotCoords](#), *key* and *value* mean a point in the plot coordinate system defined by the axes set by [setAxes](#). By default those are the [QCustomPlot](#)'s [xAxis](#) and [yAxis](#). See the documentation of [setType](#) for other available coordinate types and their meaning.

If different types were configured for X and Y ([setTypeX](#), [setTypeY](#)), *key* and *value* must also be provided in the different coordinate systems. Here, the X type refers to *key*, and the Y type refers to *value*.

See also

[setPixelPosition](#)

6.48.4.6 `setCoords()` [2/2]

```
void QCPItemPosition::setCoords (
    const QPointF & pos )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Sets the coordinates as a `QPointF` *pos* where *pos.x* has the meaning of *key* and *pos.y* the meaning of *value* of the [setCoords\(double key, double value\)](#) method.

6.48.4.7 `setParentAnchor()`

```
bool QCPItemPosition::setParentAnchor (
    QCPItemAnchor * parentAnchor,
    bool keepPixelPosition = false )
```

Sets the parent of this [QCPItemPosition](#) to *parentAnchor*. This means the position will now follow any position changes of the anchor. The local coordinate system of positions with a parent anchor always is absolute pixels, with (0, 0) being exactly on top of the parent anchor. (Hence the type shouldn't be set to [ptPlotCoords](#) for positions with parent anchors.)

if *keepPixelPosition* is true, the current pixel position of the [QCPItemPosition](#) is preserved during reparenting. If it's set to false, the coordinates are set to (0, 0), i.e. the position will be exactly on top of the parent anchor.

To remove this [QCPItemPosition](#) from any parent anchor, set *parentAnchor* to 0.

If the [QCPItemPosition](#) previously had no parent and the type is [ptPlotCoords](#), the type is set to [ptAbsolute](#), to keep the position in a valid state.

This method sets the parent anchor for both X and Y directions. It is also possible to set different parents for X and Y, see [setParentAnchorX](#), [setParentAnchorY](#).

6.48.4.8 `setParentAnchorX()`

```
bool QCPItemPosition::setParentAnchorX (
    QCPItemAnchor * parentAnchor,
    bool keepPixelPosition = false )
```

This method sets the parent anchor of the X coordinate to *parentAnchor*.

For a detailed description of what a parent anchor is, see the documentation of [setParentAnchor](#).

See also

[setParentAnchor](#), [setParentAnchorY](#)

6.48.4.9 setParentAnchorY()

```
bool QCPItemPosition::setParentAnchorY (
    QCPItemAnchor * parentAnchor,
    bool keepPixelPosition = false )
```

This method sets the parent anchor of the Y coordinate to *parentAnchor*.

For a detailed description of what a parent anchor is, see the documentation of [setParentAnchor](#).

See also

[setParentAnchor](#), [setParentAnchorX](#)

6.48.4.10 setPixelPosition()

```
void QCPItemPosition::setPixelPosition (
    const QPointF & pixelPosition )
```

Sets the apparent pixel position. This works no matter what type ([setType](#)) this [QCPItemPosition](#) is or what parent-child situation it is in, as coordinates are transformed appropriately, to make the position finally appear at the specified pixel values.

Only if the type is [ptAbsolute](#) and no parent anchor is set, this function's effect is identical to that of [setCoords](#).

See also

[pixelPosition](#), [setCoords](#)

6.48.4.11 setType()

```
void QCPItemPosition::setType (
    QCPItemPosition::PositionType type )
```

Sets the type of the position. The type defines how the coordinates passed to [setCoords](#) should be handled and how the [QCPItemPosition](#) should behave in the plot.

The possible values for *type* can be separated in two main categories:

- The position is regarded as a point in plot coordinates. This corresponds to [ptPlotCoords](#) and requires two axes that define the plot coordinate system. They can be specified with [setAxes](#). By default, the [QCustomPlot](#)'s x- and yAxis are used.
- The position is fixed on the [QCustomPlot](#) surface, i.e. independent of axis ranges. This corresponds to all other types, i.e. [ptAbsolute](#), [ptViewportRatio](#) and [ptAxisRectRatio](#). They differ only in the way the absolute position is described, see the documentation of [PositionType](#) for details. For [ptAxisRectRatio](#), note that you can specify the axis rect with [setAxisRect](#). By default this is set to the main axis rect.

Note that the position type [ptPlotCoords](#) is only available (and sensible) when the position has no parent anchor ([setParentAnchor](#)).

If the type is changed, the apparent pixel position on the plot is preserved. This means the coordinates as retrieved with [coords\(\)](#) and set with [setCoords](#) may change in the process.

This method sets the type for both X and Y directions. It is also possible to set different types for X and Y, see [setTypeX](#), [setTypeY](#).

6.48.4.12 setTypeX()

```
void QCItemPosition::setTypeX (
    QCItemPosition::PositionType type )
```

This method sets the position type of the X coordinate to *type*.

For a detailed description of what a position type is, see the documentation of [setType](#).

See also

[setType](#), [setTypeY](#)

6.48.4.13 setTypeY()

```
void QCItemPosition::setTypeY (
    QCItemPosition::PositionType type )
```

This method sets the position type of the Y coordinate to *type*.

For a detailed description of what a position type is, see the documentation of [setType](#).

See also

[setType](#), [setTypeX](#)

6.48.4.14 toQCItemPosition()

```
virtual QCItemPosition* QCItemPosition::toQCItemPosition ( ) [inline], [protected], [virtual]
```

Returns 0 if this instance is merely a [QCItemAnchor](#), and a valid pointer of type [QCItemPosition*](#) if it actually is a [QCItemPosition](#) (which is a subclass of [QCItemAnchor](#)).

This safe downcast functionality could also be achieved with a `dynamic_cast`. However, [QCustomPlot](#) avoids `dynamic_cast` to work with projects that don't have RTTI support enabled (e.g. `-fno-rtti` flag with gcc compiler).

Reimplemented from [QCItemAnchor](#).

6.48.4.15 type()

```
QCItemPosition::PositionType * QCItemPosition::type ( ) const [inline]
```

Returns the current position type.

If different types were set for X and Y ([setTypeX](#), [setTypeY](#)), this method returns the type of the X coordinate. In that case rather use [typeX\(\)](#) and [typeY\(\)](#).

See also

[setType](#)

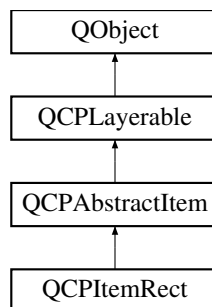
The documentation for this class was generated from the following files:

- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h↵
- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp↵

6.49 QCItemRect Class Reference

A rectangle.

Inheritance diagram for QCItemRect:



Public Member Functions

- [QCItemRect](#) ([QCustomPlot](#) *parentPlot)
- QPen **pen** () const
- QPen **selectedPen** () const
- QBrush **brush** () const
- QBrush **selectedBrush** () const
- void [setPen](#) (const QPen &pen)
- void [setSelectedPen](#) (const QPen &pen)
- void [setBrush](#) (const QBrush &brush)
- void [setSelectedBrush](#) (const QBrush &brush)
- virtual double [selectTest](#) (const QPointF &pos, bool onlySelectable, QVariant *details=0) const Q_DECL_OVERRIDE↵

Public Attributes

- [QCItemPosition](#) *const **topLeft**
- [QCItemPosition](#) *const **bottomRight**
- [QCItemAnchor](#) *const **top**
- [QCItemAnchor](#) *const **topRight**
- [QCItemAnchor](#) *const **right**
- [QCItemAnchor](#) *const **bottom**
- [QCItemAnchor](#) *const **bottomLeft**
- [QCItemAnchor](#) *const **left**

Protected Types

- enum **AnchorIndex** {
 aiTop, **aiTopRight**, **aiRight**, **aiBottom**,
 aiBottomLeft, **aiLeft** }

Protected Member Functions

- virtual void **draw** ([QCPPainter](#) *painter) Q_DECL_OVERRIDE
- virtual QPointF **anchorPixelPosition** (int anchorId) const Q_DECL_OVERRIDE
- QPen **mainPen** () const
- QBrush **mainBrush** () const

Protected Attributes

- QPen **mPen**
- QPen **mSelectedPen**
- QBrush **mBrush**
- QBrush **mSelectedBrush**

Additional Inherited Members

6.49.1 Detailed Description

A rectangle.

It has two positions, *topLeft* and *bottomRight*, which define the rectangle.

6.49.2 Constructor & Destructor Documentation

6.49.2.1 QCPItemRect()

```
QCPItemRect::QCPItemRect (
    QCustomPlot * parentPlot ) [explicit]
```

Creates a rectangle item and sets default values.

The created item is automatically registered with *parentPlot*. This [QCustomPlot](#) instance takes ownership of the item, so do not delete it manually but use [QCustomPlot::removeItem\(\)](#) instead.

6.49.3 Member Function Documentation

6.49.3.1 selectTest()

```
double QCPItemRect::selectTest (
    const QPointF & pos,
    bool onlySelectable,
    QVariant * details = 0 ) const [virtual]
```

This function is used to decide whether a click hits a layerable object or not.

pos is a point in pixel coordinates on the [QCustomPlot](#) surface. This function returns the shortest pixel distance of this point to the object. If the object is either invisible or the distance couldn't be determined, -1.0 is returned. Further, if *onlySelectable* is true and the object is not selectable, -1.0 is returned, too.

If the object is represented not by single lines but by an area like a [QCPItemText](#) or the bars of a [QCPBars](#) plottable, a click inside the area should also be considered a hit. In these cases this function thus returns a constant value greater zero but still below the parent plot's selection tolerance. (typically the selectionTolerance multiplied by 0.99).

Providing a constant value for area objects allows selecting line objects even when they are obscured by such area objects, by clicking close to the lines (i.e. closer than 0.99*selectionTolerance).

The actual setting of the selection state is not done by this function. This is handled by the parent [QCustomPlot](#) when the mouseReleaseEvent occurs, and the finally selected object is notified via the selectEvent/deselectEvent methods.

details is an optional output parameter. Every layerable subclass may place any information in *details*. This information will be passed to selectEvent when the parent [QCustomPlot](#) decides on the basis of this selectTest call, that the object was successfully selected. The subsequent call to selectEvent will carry the *details*. This is useful for multi-part objects (like [QCPAxis](#)). This way, a possibly complex calculation to decide which part was clicked is only done once in [selectTest](#). The result (i.e. the actually clicked part) can then be placed in *details*. So in the subsequent selectEvent, the decision which part was selected doesn't have to be done a second time for a single selection operation.

You may pass 0 as *details* to indicate that you are not interested in those selection details.

See also

selectEvent, deselectEvent, [mousePressEvent](#), [wheelEvent](#), [QCustomPlot::setInteractions](#)

Implements [QCPAbstractItem](#).

6.49.3.2 setBrush()

```
void QCPItemRect::setBrush (
    const QBrush & brush )
```

Sets the brush that will be used to fill the rectangle. To disable filling, set *brush* to Qt::NoBrush.

See also

[setSelectedBrush](#), [setPen](#)

6.49.3.3 setPen()

```
void QCPItemRect::setPen (
    const QPen & pen )
```

Sets the pen that will be used to draw the line of the rectangle

See also

[setSelectedPen](#), [setBrush](#)

6.49.3.4 setSelectedBrush()

```
void QCPItemRect::setSelectedBrush (
    const QBrush & brush )
```

Sets the brush that will be used to fill the rectangle when selected. To disable filling, set *brush* to Qt::NoBrush.

See also

[setBrush](#)

6.49.3.5 setSelectedPen()

```
void QCPItemRect::setSelectedPen (
    const QPen & pen )
```

Sets the pen that will be used to draw the line of the rectangle when selected

See also

[setPen](#), [setSelected](#)

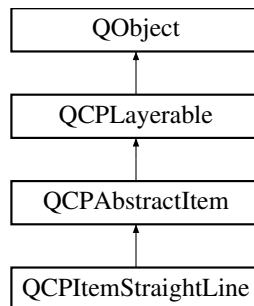
The documentation for this class was generated from the following files:

- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h↵
- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp↵

6.50 QCPLItemStraightLine Class Reference

A straight line that spans infinitely in both directions.

Inheritance diagram for QCPLItemStraightLine:



Public Member Functions

- [QCPLItemStraightLine](#) ([QCustomPlot](#) *parentPlot)
- `QPen pen () const`
- `QPen selectedPen () const`
- `void setPen (const QPen &pen)`
- `void setSelectedPen (const QPen &pen)`
- virtual double [selectTest](#) (const QPointF &pos, bool onlySelectable, QVariant *details=0) const Q_DECL_OVERRIDE

Public Attributes

- [QCPLItemPosition](#) *const **point1**
- [QCPLItemPosition](#) *const **point2**

Protected Member Functions

- virtual void **draw** ([QCPPainter](#) *painter) Q_DECL_OVERRIDE
- `QLineF getRectClippedStraightLine (const QCPVector2D &point1, const QCPVector2D &vec, const QRect &rect) const`
- `QPen mainPen () const`

Protected Attributes

- `QPen mPen`
- `QPen mSelectedPen`

Additional Inherited Members

6.50.1 Detailed Description

A straight line that spans infinitely in both directions.

It has two positions, *point1* and *point2*, which define the straight line.

6.50.2 Constructor & Destructor Documentation

6.50.2.1 QCPItemStraightLine()

```
QCPItemStraightLine::QCPItemStraightLine (
    QCustomPlot * parentPlot ) [explicit]
```

Creates a straight line item and sets default values.

The created item is automatically registered with *parentPlot*. This [QCustomPlot](#) instance takes ownership of the item, so do not delete it manually but use [QCustomPlot::removeItem\(\)](#) instead.

6.50.3 Member Function Documentation

6.50.3.1 selectTest()

```
double QCPItemStraightLine::selectTest (
    const QPointF & pos,
    bool onlySelectable,
    QVariant * details = 0 ) const [virtual]
```

This function is used to decide whether a click hits a layerable object or not.

pos is a point in pixel coordinates on the [QCustomPlot](#) surface. This function returns the shortest pixel distance of this point to the object. If the object is either invisible or the distance couldn't be determined, -1.0 is returned. Further, if *onlySelectable* is true and the object is not selectable, -1.0 is returned, too.

If the object is represented not by single lines but by an area like a [QCPItemText](#) or the bars of a [QCPBars](#) plottable, a click inside the area should also be considered a hit. In these cases this function thus returns a constant value greater zero but still below the parent plot's selection tolerance. (typically the selectionTolerance multiplied by 0.99).

Providing a constant value for area objects allows selecting line objects even when they are obscured by such area objects, by clicking close to the lines (i.e. closer than $0.99 \cdot \text{selectionTolerance}$).

The actual setting of the selection state is not done by this function. This is handled by the parent [QCustomPlot](#) when the `mouseReleaseEvent` occurs, and the finally selected object is notified via the `selectEvent/deselectEvent` methods.

details is an optional output parameter. Every layerable subclass may place any information in *details*. This information will be passed to `selectEvent` when the parent [QCustomPlot](#) decides on the basis of this `selectTest` call, that the object was successfully selected. The subsequent call to `selectEvent` will carry the *details*. This is useful for multi-part objects (like [QCPAxis](#)). This way, a possibly complex calculation to decide which part was clicked is only done once in [selectTest](#). The result (i.e. the actually clicked part) can then be placed in *details*. So in the subsequent `selectEvent`, the decision which part was selected doesn't have to be done a second time for a single selection operation.

You may pass 0 as *details* to indicate that you are not interested in those selection details.

See also

`selectEvent`, `deselectEvent`, [mousePressEvent](#), [wheelEvent](#), [QCustomPlot::setInteractions](#)

Implements [QCPAbstractItem](#).

6.50.3.2 setPen()

```
void QCPItemStraightLine::setPen (
    const QPen & pen )
```

Sets the pen that will be used to draw the line

See also

[setSelectedPen](#)

6.50.3.3 setSelectedPen()

```
void QCPItemStraightLine::setSelectedPen (
    const QPen & pen )
```

Sets the pen that will be used to draw the line when selected

See also

[setPen](#), [setSelected](#)

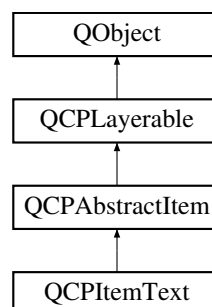
The documentation for this class was generated from the following files:

- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.↵h
- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.↵cpp

6.51 QCPItemText Class Reference

A text label.

Inheritance diagram for QCPItemText:



Public Member Functions

- [QCItemText](#) ([QCustomPlot](#) *parentPlot)
- QColor **color** () const
- QColor **selectedColor** () const
- QPen **pen** () const
- QPen **selectedPen** () const
- QBrush **brush** () const
- QBrush **selectedBrush** () const
- QFont **font** () const
- QFont **selectedFont** () const
- QString **text** () const
- Qt::Alignment **positionAlignment** () const
- Qt::Alignment **textAlignment** () const
- double **rotation** () const
- QMargins **padding** () const
- void [setColor](#) (const QColor &color)
- void [setSelectedColor](#) (const QColor &color)
- void [setPen](#) (const QPen &pen)
- void [setSelectedPen](#) (const QPen &pen)
- void [setBrush](#) (const QBrush &brush)
- void [setSelectedBrush](#) (const QBrush &brush)
- void [setFont](#) (const QFont &font)
- void [setSelectedFont](#) (const QFont &font)
- void [setText](#) (const QString &text)
- void [setPositionAlignment](#) (Qt::Alignment alignment)
- void [setTextAlignment](#) (Qt::Alignment alignment)
- void [setRotation](#) (double degrees)
- void [setPadding](#) (const QMargins &padding)
- virtual double [selectTest](#) (const QPointF &pos, bool onlySelectable, QVariant *details=0) const Q_DECL_OVERRIDE

Public Attributes

- [QCItemPosition](#) *const **position**
- [QCItemAnchor](#) *const **topLeft**
- [QCItemAnchor](#) *const **top**
- [QCItemAnchor](#) *const **topRight**
- [QCItemAnchor](#) *const **right**
- [QCItemAnchor](#) *const **bottomRight**
- [QCItemAnchor](#) *const **bottom**
- [QCItemAnchor](#) *const **bottomLeft**
- [QCItemAnchor](#) *const **left**

Protected Types

- enum **AnchorIndex** {
aiTopLeft, **aiTop**, **aiTopRight**, **aiRight**,
aiBottomRight, **aiBottom**, **aiBottomLeft**, **aiLeft** }

Protected Member Functions

- virtual void **draw** ([QCPPainter](#) *painter) Q_DECL_OVERRIDE
- virtual QPointF **anchorPixelPosition** (int anchorId) const Q_DECL_OVERRIDE
- QPointF **getTextDrawPoint** (const QPointF &pos, const QRectF &rect, Qt::Alignment positionAlignment) const
- QFont **mainFont** () const
- QColor **mainColor** () const
- QPen **mainPen** () const
- QBrush **mainBrush** () const

Protected Attributes

- QColor **mColor**
- QColor **mSelectedColor**
- QPen **mPen**
- QPen **mSelectedPen**
- QBrush **mBrush**
- QBrush **mSelectedBrush**
- QFont **mFont**
- QFont **mSelectedFont**
- QString **mText**
- Qt::Alignment **mPositionAlignment**
- Qt::Alignment **mTextAlignment**
- double **mRotation**
- QMargins **mPadding**

Additional Inherited Members

6.51.1 Detailed Description

A text label.

Its position is defined by the member *position* and the setting of [setPositionAlignment](#). The latter controls which part of the text rect shall be aligned with *position*.

The text alignment itself (i.e. left, center, right) can be controlled with [setTextAlignment](#).

The text may be rotated around the *position* point with [setRotation](#).

6.51.2 Constructor & Destructor Documentation

6.51.2.1 QCPItemText()

```
QCPItemText::QCPItemText (
    QCustomPlot * parentPlot ) [explicit]
```

Creates a text item and sets default values.

The created item is automatically registered with *parentPlot*. This [QCustomPlot](#) instance takes ownership of the item, so do not delete it manually but use [QCustomPlot::removeItem\(\)](#) instead.

6.51.3 Member Function Documentation

6.51.3.1 selectTest()

```
double QCPItemText::selectTest (
    const QPointF & pos,
    bool onlySelectable,
    QVariant * details = 0 ) const [virtual]
```

This function is used to decide whether a click hits a layerable object or not.

pos is a point in pixel coordinates on the [QCustomPlot](#) surface. This function returns the shortest pixel distance of this point to the object. If the object is either invisible or the distance couldn't be determined, -1.0 is returned. Further, if *onlySelectable* is true and the object is not selectable, -1.0 is returned, too.

If the object is represented not by single lines but by an area like a [QCPItemText](#) or the bars of a [QCPBars](#) plottable, a click inside the area should also be considered a hit. In these cases this function thus returns a constant value greater zero but still below the parent plot's selection tolerance. (typically the selectionTolerance multiplied by 0.99).

Providing a constant value for area objects allows selecting line objects even when they are obscured by such area objects, by clicking close to the lines (i.e. closer than 0.99*selectionTolerance).

The actual setting of the selection state is not done by this function. This is handled by the parent [QCustomPlot](#) when the mouseReleaseEvent occurs, and the finally selected object is notified via the selectEvent/deselectEvent methods.

details is an optional output parameter. Every layerable subclass may place any information in *details*. This information will be passed to selectEvent when the parent [QCustomPlot](#) decides on the basis of this selectTest call, that the object was successfully selected. The subsequent call to selectEvent will carry the *details*. This is useful for multi-part objects (like [QCPAxis](#)). This way, a possibly complex calculation to decide which part was clicked is only done once in [selectTest](#). The result (i.e. the actually clicked part) can then be placed in *details*. So in the subsequent selectEvent, the decision which part was selected doesn't have to be done a second time for a single selection operation.

You may pass 0 as *details* to indicate that you are not interested in those selection details.

See also

selectEvent, deselectEvent, [mousePressEvent](#), [wheelEvent](#), [QCustomPlot::setInteractions](#)

Implements [QCPAbstractItem](#).

6.51.3.2 setBrush()

```
void QCPItemText::setBrush (
    const QBrush & brush )
```

Sets the brush that will be used to fill the background of the text. To disable the background, set *brush* to [Qt::NoBrush](#).

See also

[setSelectedBrush](#), [setPen](#), [setPadding](#)

6.51.3.3 setColor()

```
void QCPItemText::setColor (
    const QColor & color )
```

Sets the color of the text.

6.51.3.4 setFont()

```
void QCPItemText::setFont (
    const QFont & font )
```

Sets the font of the text.

See also

[setSelectedFont](#), [setColor](#)

6.51.3.5 setPadding()

```
void QCPItemText::setPadding (
    const QMargins & padding )
```

Sets the distance between the border of the text rectangle and the text. The appearance (and visibility) of the text rectangle can be controlled with [setPen](#) and [setBrush](#).

6.51.3.6 setPen()

```
void QCPItemText::setPen (
    const QPen & pen )
```

Sets the pen that will be used to draw a rectangular border around the text. To disable the border, set *pen* to `Qt::NoPen`.

See also

[setSelectedPen](#), [setBrush](#), [setPadding](#)

6.51.3.7 setPositionAlignment()

```
void QCPItemText::setPositionAlignment (
    Qt::Alignment alignment )
```

Sets which point of the text rect shall be aligned with *position*.

Examples:

- If *alignment* is `Qt::AlignHCenter` | `Qt::AlignTop`, the text will be positioned such that the top of the text rect will be horizontally centered on *position*.
- If *alignment* is `Qt::AlignLeft` | `Qt::AlignBottom`, *position* will indicate the bottom left corner of the text rect.

If you want to control the alignment of (multi-lined) text within the text rect, use [setTextAlignment](#).

6.51.3.8 setRotation()

```
void QCPItemText::setRotation (
    double degrees )
```

Sets the angle in degrees by which the text (and the text rectangle, if visible) will be rotated around *position*.

6.51.3.9 setSelectedBrush()

```
void QCPItemText::setSelectedBrush (
    const QBrush & brush )
```

Sets the brush that will be used to fill the background of the text, when the item is selected. To disable the background, set *brush* to Qt::NoBrush.

See also

[setBrush](#)

6.51.3.10 setSelectedColor()

```
void QCPItemText::setSelectedColor (
    const QColor & color )
```

Sets the color of the text that will be used when the item is selected.

6.51.3.11 setSelectedFont()

```
void QCPItemText::setSelectedFont (
    const QFont & font )
```

Sets the font of the text that will be used when the item is selected.

See also

[setFont](#)

6.51.3.12 setSelectedPen()

```
void QCPItemText::setSelectedPen (
    const QPen & pen )
```

Sets the pen that will be used to draw a rectangular border around the text, when the item is selected. To disable the border, set *pen* to Qt::NoPen.

See also

[setPen](#)

6.51.3.13 setText()

```
void QCPItemText::setText (
    const QString & text )
```

Sets the text that will be displayed. Multi-line texts are supported by inserting a line break character, e.g. '
'.

See also

[setFont](#), [setColor](#), [setTextAlignment](#)

6.51.3.14 setTextAlignment()

```
void QCPItemText::setTextAlignment (
    Qt::Alignment alignment )
```

Controls how (multi-lined) text is aligned inside the text rect (typically Qt::AlignLeft, Qt::AlignCenter or Qt::Align↵Right).

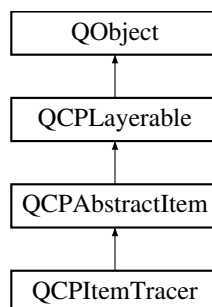
The documentation for this class was generated from the following files:

- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.↵h
- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.↵cpp

6.52 QCPItemTracer Class Reference

Item that sticks to [QCPGraph](#) data points.

Inheritance diagram for QCPItemTracer:



Public Types

- enum [TracerStyle](#) {
[tsNone](#), [tsPlus](#), [tsCrosshair](#), [tsCircle](#),
[tsSquare](#) }

Public Member Functions

- [QCPlotItemTracer](#) ([QCustomPlot](#) *parentPlot)
- QPen **pen** () const
- QPen **selectedPen** () const
- QBrush **brush** () const
- QBrush **selectedBrush** () const
- double **size** () const
- [TracerStyle](#) **style** () const
- [QCPGraph](#) * **graph** () const
- double **graphKey** () const
- bool **interpolating** () const
- void **setPen** (const QPen &pen)
- void **setSelectedPen** (const QPen &pen)
- void **setBrush** (const QBrush &brush)
- void **setSelectedBrush** (const QBrush &brush)
- void **setSize** (double size)
- void **setStyle** ([TracerStyle](#) style)
- void **setGraph** ([QCPGraph](#) *graph)
- void **setGraphKey** (double key)
- void **setInterpolating** (bool enabled)
- virtual double **selectTest** (const QPointF &pos, bool onlySelectable, QVariant *details=0) const Q_DECL_OVERRIDE↵
- void **updatePosition** ()

Public Attributes

- [QCPlotItemPosition](#) *const **position**

Protected Member Functions

- virtual void **draw** ([QCPPainter](#) *painter) Q_DECL_OVERRIDE
- QPen **mainPen** () const
- QBrush **mainBrush** () const

Protected Attributes

- QPen **mPen**
- QPen **mSelectedPen**
- QBrush **mBrush**
- QBrush **mSelectedBrush**
- double **mSize**
- [TracerStyle](#) **mStyle**
- [QCPGraph](#) * **mGraph**
- double **mGraphKey**
- bool **mInterpolating**

Additional Inherited Members

6.52.1 Detailed Description

Item that sticks to [QCPGraph](#) data points.

The tracer can be connected with a [QCPGraph](#) via [setGraph](#). Then it will automatically adopt the coordinate axes of the graph and update its *position* to be on the graph's data. This means the key stays controllable via [setGraphKey](#), but the value will follow the graph data. If a [QCPGraph](#) is connected, note that setting the coordinates of the tracer item directly via *position* will have no effect because they will be overridden in the next redraw (this is when the coordinate update happens).

If the specified key in [setGraphKey](#) is outside the key bounds of the graph, the tracer will stay at the corresponding end of the graph.

With [setInterpolating](#) you may specify whether the tracer may only stay exactly on data points or whether it interpolates data points linearly, if given a key that lies between two data points of the graph.

The tracer has different visual styles, see [setStyle](#). It is also possible to make the tracer have no own visual appearance (set the style to [tsNone](#)), and just connect other item positions to the tracer *position* (used as an anchor) via [QCPItemPosition::setParentAnchor](#).

Note

The tracer position is only automatically updated upon redraws. So when the data of the graph changes and immediately afterwards (without a redraw) the position coordinates of the tracer are retrieved, they will not reflect the updated data of the graph. In this case [updatePosition](#) must be called manually, prior to reading the tracer coordinates.

6.52.2 Member Enumeration Documentation

6.52.2.1 TracerStyle

```
enum QCPItemTracer::TracerStyle
```

The different visual appearances a tracer item can have. Some styles size may be controlled with [setSize](#).

See also

[setStyle](#)

Enumerator

tsNone	The tracer is not visible.
tsPlus	A plus shaped crosshair with limited size.
tsCrosshair	A plus shaped crosshair which spans the complete axis rect.
tsCircle	A circle.
tsSquare	A square.

6.52.3 Constructor & Destructor Documentation

6.52.3.1 QCPItemTracer()

```
QCPItemTracer::QCPItemTracer (
    QCustomPlot * parentPlot ) [explicit]
```

Creates a tracer item and sets default values.

The created item is automatically registered with *parentPlot*. This [QCustomPlot](#) instance takes ownership of the item, so do not delete it manually but use [QCustomPlot::removeItem\(\)](#) instead.

6.52.4 Member Function Documentation

6.52.4.1 selectTest()

```
double QCPItemTracer::selectTest (
    const QPointF & pos,
    bool onlySelectable,
    QVariant * details = 0 ) const [virtual]
```

This function is used to decide whether a click hits a layerable object or not.

pos is a point in pixel coordinates on the [QCustomPlot](#) surface. This function returns the shortest pixel distance of this point to the object. If the object is either invisible or the distance couldn't be determined, -1.0 is returned. Further, if *onlySelectable* is true and the object is not selectable, -1.0 is returned, too.

If the object is represented not by single lines but by an area like a [QCPItemText](#) or the bars of a [QCPBars](#) plottable, a click inside the area should also be considered a hit. In these cases this function thus returns a constant value greater zero but still below the parent plot's selection tolerance. (typically the selectionTolerance multiplied by 0.99).

Providing a constant value for area objects allows selecting line objects even when they are obscured by such area objects, by clicking close to the lines (i.e. closer than $0.99 \cdot \text{selectionTolerance}$).

The actual setting of the selection state is not done by this function. This is handled by the parent [QCustomPlot](#) when the `mouseReleaseEvent` occurs, and the finally selected object is notified via the `selectEvent/deselectEvent` methods.

details is an optional output parameter. Every layerable subclass may place any information in *details*. This information will be passed to `selectEvent` when the parent [QCustomPlot](#) decides on the basis of this `selectTest` call, that the object was successfully selected. The subsequent call to `selectEvent` will carry the *details*. This is useful for multi-part objects (like [QCPAxis](#)). This way, a possibly complex calculation to decide which part was clicked is only done once in [selectTest](#). The result (i.e. the actually clicked part) can then be placed in *details*. So in the subsequent `selectEvent`, the decision which part was selected doesn't have to be done a second time for a single selection operation.

You may pass 0 as *details* to indicate that you are not interested in those selection details.

See also

`selectEvent`, `deselectEvent`, [mousePressEvent](#), [wheelEvent](#), [QCustomPlot::setInteractions](#)

Implements [QCPAbstractItem](#).

6.52.4.2 setBrush()

```
void QCPItemTracer::setBrush (
    const QBrush & brush )
```

Sets the brush that will be used to draw any fills of the tracer

See also

[setSelectedBrush](#), [setPen](#)

6.52.4.3 setGraph()

```
void QCPItemTracer::setGraph (
    QCPGraph * graph )
```

Sets the [QCPGraph](#) this tracer sticks to. The tracer *position* will be set to type [QCPItemPosition::ptPlotCoords](#) and the axes will be set to the axes of *graph*.

To free the tracer from any graph, set *graph* to 0. The tracer *position* can then be placed freely like any other item position. This is the state the tracer will assume when its graph gets deleted while still attached to it.

See also

[setGraphKey](#)

6.52.4.4 setGraphKey()

```
void QCPItemTracer::setGraphKey (
    double key )
```

Sets the key of the graph's data point the tracer will be positioned at. This is the only free coordinate of a tracer when attached to a graph.

Depending on [setInterpolating](#), the tracer will be either positioned on the data point closest to *key*, or will stay exactly at *key* and interpolate the value linearly.

See also

[setGraph](#), [setInterpolating](#)

6.52.4.5 setInterpolating()

```
void QCPItemTracer::setInterpolating (
    bool enabled )
```

Sets whether the value of the graph's data points shall be interpolated, when positioning the tracer.

If *enabled* is set to false and a key is given with [setGraphKey](#), the tracer is placed on the data point of the graph which is closest to the key, but which is not necessarily exactly there. If *enabled* is true, the tracer will be positioned exactly at the specified key, and the appropriate value will be interpolated from the graph's data points linearly.

See also

[setGraph](#), [setGraphKey](#)

6.52.4.6 setPen()

```
void QCPItemTracer::setPen (
    const QPen & pen )
```

Sets the pen that will be used to draw the line of the tracer

See also

[setSelectedPen](#), [setBrush](#)

6.52.4.7 setSelectedBrush()

```
void QCPItemTracer::setSelectedBrush (
    const QBrush & brush )
```

Sets the brush that will be used to draw any fills of the tracer, when selected.

See also

[setBrush](#), [setSelected](#)

6.52.4.8 setSelectedPen()

```
void QCPItemTracer::setSelectedPen (
    const QPen & pen )
```

Sets the pen that will be used to draw the line of the tracer when selected

See also

[setPen](#), [setSelected](#)

6.52.4.9 setSize()

```
void QCPLayerTracer::setSize (
    double size )
```

Sets the size of the tracer in pixels, if the style supports setting a size (e.g. [tsSquare](#) does, [tsCrosshair](#) does not).

6.52.4.10 setStyle()

```
void QCPLayerTracer::setStyle (
    QCPLayerTracer::TracerStyle style )
```

Sets the style/visual appearance of the tracer.

If you only want to use the tracer *position* as an anchor for other items, set *style* to [tsNone](#).

6.52.4.11 updatePosition()

```
void QCPLayerTracer::updatePosition ( )
```

If the tracer is connected with a graph ([setGraph](#)), this function updates the tracer's *position* to reside on the graph data, depending on the configured key ([setGraphKey](#)).

It is called automatically on every redraw and normally doesn't need to be called manually. One exception is when you want to read the tracer coordinates via *position* and are not sure that the graph's data (or the tracer key with [setGraphKey](#)) hasn't changed since the last redraw. In that situation, call this function before accessing *position*, to make sure you don't get out-of-date coordinates.

If there is no graph set on this tracer, this function does nothing.

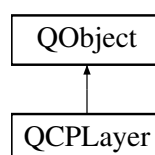
The documentation for this class was generated from the following files:

- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h
- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp

6.53 QCPLayer Class Reference

A layer that may contain objects, to control the rendering order.

Inheritance diagram for QCPLayer:



Public Types

- enum [LayerMode](#) { [ImLogical](#), [ImBuffered](#) }

Public Member Functions

- [QCPLayer](#) ([QCustomPlot](#) *parentPlot, const QString &layerName)
- [QCustomPlot](#) * **parentPlot** () const
- QString **name** () const
- int **index** () const
- QList< [QCPLayerable](#) * > **children** () const
- bool **visible** () const
- [LayerMode](#) **mode** () const
- void **setVisible** (bool visible)
- void **setMode** ([LayerMode](#) mode)
- void **replot** ()

Protected Member Functions

- void **draw** ([QCPPainter](#) *painter)
- void **drawToPaintBuffer** ()
- void **addChild** ([QCPLayerable](#) *layerable, bool prepend)
- void **removeChild** ([QCPLayerable](#) *layerable)

Protected Attributes

- [QCustomPlot](#) * **mParentPlot**
- QString **mName**
- int **mIndex**
- QList< [QCPLayerable](#) * > **mChildren**
- bool **mVisible**
- [LayerMode](#) **mMode**
- QWeakPointer< [QCPAbstractPaintBuffer](#) > **mPaintBuffer**

Friends

- class **QCustomPlot**
- class **QCPLayerable**

6.53.1 Detailed Description

A layer that may contain objects, to control the rendering order.

The Layering system of [QCustomPlot](#) is the mechanism to control the rendering order of the elements inside the plot.

It is based on the two classes [QCPLayer](#) and [QCPLayerable](#). [QCustomPlot](#) holds an ordered list of one or more instances of [QCPLayer](#) (see [QCustomPlot::addLayer](#), [QCustomPlot::layer](#), [QCustomPlot::moveLayer](#), etc.). When replotting, [QCustomPlot](#) goes through the list of layers bottom to top and successively draws the layerables of the layers into the paint buffer(s).

A [QCPLayer](#) contains an ordered list of [QCPLayerable](#) instances. [QCPLayerable](#) is an abstract base class from which almost all visible objects derive, like axes, grids, graphs, items, etc.

6.53.2 Default layers

Initially, `QCustomPlot` has six layers: "background", "grid", "main", "axes", "legend" and "overlay" (in that order). On top is the "overlay" layer, which only contains the `QCustomPlot`'s selection rect (`QCustomPlot::selectionRect`). The next two layers "axes" and "legend" contain the default axes and legend, so they will be drawn above plottables. In the middle, there is the "main" layer. It is initially empty and set as the current layer (see `QCustomPlot::setCurrentLayer`). This means, all new plottables, items etc. are created on this layer by default. Then comes the "grid" layer which contains the `QCPGrid` instances (which belong tightly to `QCPAxis`, see `QCPAxis::grid`). The Axis rect background shall be drawn behind everything else, thus the default `QCPAxisRect` instance is placed on the "background" layer. Of course, the layer affiliation of the individual objects can be changed as required (`QCPLayerable::setLayer`).

6.53.3 Controlling the rendering order via layers

Controlling the ordering of layerables in the plot is easy: Create a new layer in the position you want the layerable to be in, e.g. above "main", with `QCustomPlot::addLayer`. Then set the current layer with `QCustomPlot::setCurrentLayer` to that new layer and finally create the objects normally. They will be placed on the new layer automatically, due to the current layer setting. Alternatively you could have also ignored the current layer setting and just moved the objects with `QCPLayerable::setLayer` to the desired layer after creating them.

It is also possible to move whole layers. For example, If you want the grid to be shown in front of all plottables/items on the "main" layer, just move it above "main" with `QCustomPlot::moveLayer`.

The rendering order within one layer is simply by order of creation or insertion. The item created last (or added last to the layer), is drawn on top of all other objects on that layer.

When a layer is deleted, the objects on it are not deleted with it, but fall on the layer below the deleted layer, see `QCustomPlot::removeLayer`.

6.53.4 Replotting only a specific layer

If the layer mode (`setMode`) is set to `ImBuffered`, you can replot only this specific layer by calling `replot`. In certain situations this can provide better replot performance, compared with a full replot of all layers. Upon creation of a new layer, the layer mode is initialized to `ImLogical`. The only layer that is set to `ImBuffered` in a new `QCustomPlot` instance is the "overlay" layer, containing the selection rect.

6.53.5 Member Enumeration Documentation

6.53.5.1 LayerMode

```
enum QCPLayer::LayerMode
```

Defines the different rendering modes of a layer. Depending on the mode, certain layers can be replotted individually, without the need to replot (possibly complex) layerables on other layers.

See also

[`setMode`](#)

Enumerator

ImLogical	Layer is used only for rendering order, and shares paint buffer with all other adjacent logical layers.
ImBuffered	Layer has its own paint buffer and may be replotted individually (see replot).

6.53.6 Constructor & Destructor Documentation

6.53.6.1 QCPLayer()

```
QCPLayer::QCPLayer (
    QCustomPlot * parentPlot,
    const QString & layerName )
```

Creates a new [QCPLayer](#) instance.

Normally you shouldn't directly instantiate layers, use [QCustomPlot::addLayer](#) instead.

Warning

It is not checked that *layerName* is actually a unique layer name in *parentPlot*. This check is only performed by [QCustomPlot::addLayer](#).

6.53.7 Member Function Documentation

6.53.7.1 children()

```
QList< QCPLayerable * > QCPLayer::children ( ) const [inline]
```

Returns a list of all layerables on this layer. The order corresponds to the rendering order: layerables with higher indices are drawn above layerables with lower indices.

6.53.7.2 index()

```
int QCPLayer::index ( ) const [inline]
```

Returns the index this layer has in the [QCustomPlot](#). The index is the integer number by which this layer can be accessed via [QCustomPlot::layer](#).

Layers with higher indices will be drawn above layers with lower indices.

6.53.7.3 replot()

```
void QCPLayer::replot ( )
```

If the layer mode ([setMode](#)) is set to [lmBuffered](#), this method allows replotting only the layerables on this specific layer, without the need to replot all other layers (as a call to [QCustomPlot::replot](#) would do).

If the layer mode is [lmLogical](#) however, this method simply calls [QCustomPlot::replot](#) on the parent [QCustomPlot](#) instance.

[QCustomPlot](#) also makes sure to replot all layers instead of only this one, if the layer ordering has changed since the last full replot and the other paint buffers were thus invalidated.

See also

[draw](#)

6.53.7.4 setMode()

```
void QCPLayer::setMode (
    QCPLayer::LayerMode mode )
```

Sets the rendering mode of this layer.

If *mode* is set to [lmBuffered](#) for a layer, it will be given a dedicated paint buffer by the parent [QCustomPlot](#) instance. This means it may be replotted individually by calling [QCPLayer::replot](#), without needing to replot all other layers.

Layers which are set to [lmLogical](#) (the default) are used only to define the rendering order and can't be replotted individually.

Note that each layer which is set to [lmBuffered](#) requires additional paint buffers for the layers below, above and for the layer itself. This increases the memory consumption and (slightly) decreases the repainting speed because multiple paint buffers need to be joined. So you should carefully choose which layers benefit from having their own paint buffer. A typical example would be a layer which contains certain layerables (e.g. items) that need to be changed and thus replotted regularly, while all other layerables on other layers stay static. By default, only the topmost layer called "overlay" is in mode [lmBuffered](#), and contains the selection rect.

See also

[replot](#)

6.53.7.5 setVisible()

```
void QCPLayer::setVisible (
    bool visible )
```

Sets whether this layer is visible or not. If *visible* is set to false, all layerables on this layer will be invisible.

This function doesn't change the visibility property of the layerables ([QCPLayerable::setVisible](#)), but the [QCP->Layerable::realVisibility](#) of each layerable takes the visibility of the parent layer into account.

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

- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h↵
- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp↵

Protected Attributes

- bool **mVisible**
- [QCustomPlot](#) * **mParentPlot**
- [QPointer](#)< [QCPLayerable](#) > **mParentLayerable**
- [QCPLayer](#) * **mLayer**
- bool **mAntialiased**

Friends

- class **QCustomPlot**
- class **QCPLayer**
- class **QCPLayerRect**

6.54.1 Detailed Description

Base class for all drawable objects.

This is the abstract base class most visible objects derive from, e.g. plottables, axes, grid etc.

Every layerable is on a layer ([QCPLayer](#)) which allows controlling the rendering order by stacking the layers accordingly.

For details about the layering mechanism, see the [QCPLayer](#) documentation.

6.54.2 Constructor & Destructor Documentation

6.54.2.1 QCPLayerable()

```
QCPLayerable::QCPLayerable (
    QCustomPlot * plot,
    QString targetLayer = QString(),
    QCPLayerable * parentLayerable = 0 )
```

Creates a new [QCPLayerable](#) instance.

Since [QCPLayerable](#) is an abstract base class, it can't be instantiated directly. Use one of the derived classes.

If *plot* is provided, it automatically places itself on the layer named *targetLayer*. If *targetLayer* is an empty string, it places itself on the current layer of the plot (see [QCustomPlot::setCurrentLayer](#)).

It is possible to provide 0 as *plot*. In that case, you should assign a parent plot at a later time with `initializeParentPlot`.

The layerable's parent layerable is set to *parentLayerable*, if provided. Direct layerable parents are mainly used to control visibility in a hierarchy of layerables. This means a layerable is only drawn, if all its ancestor layerables are also visible. Note that *parentLayerable* does not become the QObject-parent (for memory management) of this layerable, *plot* does. It is not uncommon to set the QObject-parent to something else in the constructors of [QCPLayerable](#) subclasses, to guarantee a working destruction hierarchy.

6.54.3 Member Function Documentation

6.54.3.1 layerChanged

```
void QCPLayerable::layerChanged (
    QCPLayer * newLayer ) [signal]
```

This signal is emitted when the layer of this layerable changes, i.e. this layerable is moved to a different layer.

See also

[setLayer](#)

6.54.3.2 mouseDoubleClickEvent()

```
void QCPLayerable::mouseDoubleClickEvent (
    QMouseEvent * event,
    const QVariant & details ) [protected], [virtual]
```

This event gets called when the user presses the mouse button a second time in a double-click, while the cursor is over the layerable. Whether a cursor is over the layerable is decided by a preceding call to [selectTest](#).

The [mouseDoubleClickEvent](#) is called instead of the second [mousePressEvent](#). So in the case of a double-click, the event succession is *pressEvent* – *releaseEvent* – *doubleClickEvent* – *releaseEvent*.

The current pixel position of the cursor on the [QCustomPlot](#) widget is accessible via `event->pos()`. The parameter *details* contains layerable-specific details about the hit, which were generated in the previous call to [selectTest](#). For example, One-dimensional plottables like [QCPGraph](#) or [QCPBars](#) convey the clicked data point in the *details* parameter, as [QCPDataSelection](#) packed as `QVariant`. Multi-part objects convey the specific `SelectablePart` that was hit (e.g. [QCPAxis::SelectablePart](#) in the case of axes).

Similarly to [mousePressEvent](#), once a layerable has accepted the [mouseDoubleClickEvent](#), it is considered the mouse grabber and will receive all following calls to [mouseMoveEvent](#) and [mouseReleaseEvent](#) for this mouse interaction (a "mouse interaction" in this context ends with the release).

The default implementation does nothing except explicitly ignoring the event with `event->ignore()`.

See also

[mousePressEvent](#), [mouseMoveEvent](#), [mouseReleaseEvent](#), [wheelEvent](#)

Reimplemented in [QCPLTextElement](#).

6.54.3.3 mouseMoveEvent()

```
void QCPLayerable::mouseMoveEvent (
    QMouseEvent * event,
    const QPointF & startPos ) [protected], [virtual]
```

This event gets called when the user moves the mouse while holding a mouse button, after this layerable has become the mouse grabber by accepting the preceding [mousePressEvent](#).

The current pixel position of the cursor on the [QCustomPlot](#) widget is accessible via `event->pos()`. The parameter *startPos* indicates the position where the initial [mousePressEvent](#) occurred, that started the mouse interaction.

The default implementation does nothing.

See also

[mousePressEvent](#), [mouseReleaseEvent](#), [mouseDoubleClickEvent](#), [wheelEvent](#)

Reimplemented in [QCPColorScale](#), and [QCPAxisRect](#).

6.54.3.4 mousePressEvent()

```
void QCPLayerable::mousePressEvent (
    QMouseEvent * event,
    const QVariant & details ) [protected], [virtual]
```

This event gets called when the user presses a mouse button while the cursor is over the layerable. Whether a cursor is over the layerable is decided by a preceding call to [selectTest](#).

The current pixel position of the cursor on the [QCustomPlot](#) widget is accessible via `event->pos()`. The parameter *details* contains layerable-specific details about the hit, which were generated in the previous call to [selectTest](#). For example, One-dimensional plottables like [QCPGraph](#) or [QCPBars](#) convey the clicked data point in the *details* parameter, as [QCPDataSelection](#) packed as `QVariant`. Multi-part objects convey the specific `SelectablePart` that was hit (e.g. [QCPAxis::SelectablePart](#) in the case of axes).

[QCustomPlot](#) uses an event propagation system that works the same as Qt's system. If your layerable doesn't reimplement the [mousePressEvent](#) or explicitly calls `event->ignore()` in its reimplementation, the event will be propagated to the next layerable in the stacking order.

Once a layerable has accepted the [mousePressEvent](#), it is considered the mouse grabber and will receive all following calls to [mouseMoveEvent](#) or [mouseReleaseEvent](#) for this mouse interaction (a "mouse interaction" in this context ends with the release).

The default implementation does nothing except explicitly ignoring the event with `event->ignore()`.

See also

[mouseMoveEvent](#), [mouseReleaseEvent](#), [mouseDoubleClickEvent](#), [wheelEvent](#)

Reimplemented in [QCPColorScale](#), [QCPTextElement](#), and [QCPAxisRect](#).

6.54.3.5 mouseReleaseEvent()

```
void QCPLayerable::mouseReleaseEvent (
    QMouseEvent * event,
    const QPointF & startPos ) [protected], [virtual]
```

This event gets called when the user releases the mouse button, after this layerable has become the mouse grabber by accepting the preceding [mousePressEvent](#).

The current pixel position of the cursor on the [QCustomPlot](#) widget is accessible via `event->pos()`. The parameter *startPos* indicates the position where the initial [mousePressEvent](#) occurred, that started the mouse interaction.

The default implementation does nothing.

See also

[mousePressEvent](#), [mouseMoveEvent](#), [mouseDoubleClickEvent](#), [wheelEvent](#)

Reimplemented in [QCPCColorScale](#), [QCPTTextElement](#), and [QCPAxisRect](#).

6.54.3.6 parentLayerable()

```
QCPLayerable * QCPLayerable::parentLayerable ( ) const [inline]
```

Returns the parent layerable of this layerable. The parent layerable is used to provide visibility hierarchies in conjunction with the method [realVisibility](#). This way, layerables only get drawn if their parent layerables are visible, too.

Note that a parent layerable is not necessarily also the QObject parent for memory management. Further, a layerable doesn't always have a parent layerable, so this function may return 0.

A parent layerable is set implicitly when placed inside layout elements and doesn't need to be set manually by the user.

6.54.3.7 realVisibility()

```
bool QCPLayerable::realVisibility ( ) const
```

Returns whether this layerable is visible, taking the visibility of the layerable parent and the visibility of this layerable's layer into account. This is the method that is consulted to decide whether a layerable shall be drawn or not.

If this layerable has a direct layerable parent (usually set via hierarchies implemented in subclasses, like in the case of [QCPLayoutElement](#)), this function returns true only if this layerable has its visibility set to true and the parent layerable's [realVisibility](#) returns true.

6.54.3.8 selectTest()

```
double QCPLayerable::selectTest (
    const QPointF & pos,
    bool onlySelectable,
    QVariant * details = 0 ) const [virtual]
```

This function is used to decide whether a click hits a layerable object or not.

pos is a point in pixel coordinates on the [QCustomPlot](#) surface. This function returns the shortest pixel distance of this point to the object. If the object is either invisible or the distance couldn't be determined, -1.0 is returned. Further, if *onlySelectable* is true and the object is not selectable, -1.0 is returned, too.

If the object is represented not by single lines but by an area like a [QCPLItemText](#) or the bars of a [QCPBars](#) plottable, a click inside the area should also be considered a hit. In these cases this function thus returns a constant value greater zero but still below the parent plot's selection tolerance. (typically the selectionTolerance multiplied by 0.99).

Providing a constant value for area objects allows selecting line objects even when they are obscured by such area objects, by clicking close to the lines (i.e. closer than $0.99 \cdot \text{selectionTolerance}$).

The actual setting of the selection state is not done by this function. This is handled by the parent [QCustomPlot](#) when the `mouseReleaseEvent` occurs, and the finally selected object is notified via the `selectEvent/deselectEvent` methods.

details is an optional output parameter. Every layerable subclass may place any information in *details*. This information will be passed to `selectEvent` when the parent [QCustomPlot](#) decides on the basis of this `selectTest` call, that the object was successfully selected. The subsequent call to `selectEvent` will carry the *details*. This is useful for multi-part objects (like [QCPAxis](#)). This way, a possibly complex calculation to decide which part was clicked is only done once in `selectTest`. The result (i.e. the actually clicked part) can then be placed in *details*. So in the subsequent `selectEvent`, the decision which part was selected doesn't have to be done a second time for a single selection operation.

You may pass 0 as *details* to indicate that you are not interested in those selection details.

See also

`selectEvent`, `deselectEvent`, [mousePressEvent](#), [wheelEvent](#), [QCustomPlot::setInteractions](#)

Reimplemented in [QCPLItemBracket](#), [QCPLItemTracer](#), [QCPLItemPixmap](#), [QCPLItemEllipse](#), [QCPLItemText](#), [QCPLItemRect](#), [QCPLItemCurve](#), [QCPLItemLine](#), [QCPLItemStraightLine](#), [QCPErrorBars](#), [QCPFinancial](#), [QCPColorMap](#), [QCPStatisticalBox](#), [QCPBars](#), [QCPCurve](#), [QCPGraph](#), [QCPLegend](#), [QCPAbstractLegendItem](#), [QCPAbstractPlottable1D< DataType >](#), [QCPAbstractPlottable1D< QCPFinancialData >](#), [QCPAbstractPlottable1D< QCPStatisticalBoxData >](#), [QCPAbstractPlottable1D< QCPGraphData >](#), [QCPAbstractPlottable1D< QCPBarsData >](#), [QCPAbstractPlottable1D< QCPCurveData >](#), [QCPAbstractItem](#), [QCPAbstractPlottable](#), [QCPAxis](#), [QCPLayoutInset](#), and [QCPLayoutElement](#).

6.54.3.9 setAntialiased()

```
void QCPLayerable::setAntialiased (
    bool enabled )
```

Sets whether this object will be drawn antialiased or not.

Note that antialiasing settings may be overridden by [QCustomPlot::setAntialiasedElements](#) and [QCustomPlot::setNotAntialiasedElements](#).

6.54.3.10 setLayer() [1/2]

```
bool QCPLayerable::setLayer (
    QCPLayer * layer )
```

Sets the *layer* of this layerable object. The object will be placed on top of the other objects already on *layer*.

If *layer* is 0, this layerable will not be on any layer and thus not appear in the plot (or interact/receive events).

Returns true if the layer of this layerable was successfully changed to *layer*.

6.54.3.11 setLayer() [2/2]

```
bool QCPLayerable::setLayer (
    const QString & layerName )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts. Sets the layer of this layerable object by name

Returns true on success, i.e. if *layerName* is a valid layer name.

6.54.3.12 setVisible()

```
void QCPLayerable::setVisible (
    bool on )
```

Sets the visibility of this layerable object. If an object is not visible, it will not be drawn on the [QCustomPlot](#) surface, and user interaction with it (e.g. click and selection) is not possible.

6.54.3.13 wheelEvent()

```
void QCPLayerable::wheelEvent (
    QWheelEvent * event ) [protected], [virtual]
```

This event gets called when the user turns the mouse scroll wheel while the cursor is over the layerable. Whether a cursor is over the layerable is decided by a preceding call to [selectTest](#).

The current pixel position of the cursor on the [QCustomPlot](#) widget is accessible via `event->pos()`.

The `event->delta()` indicates how far the mouse wheel was turned, which is usually +/- 120 for single rotation steps. However, if the mouse wheel is turned rapidly, multiple steps may accumulate to one event, making `event->delta()` larger. On the other hand, if the wheel has very smooth steps or none at all, the delta may be smaller.

The default implementation does nothing.

See also

[mousePressEvent](#), [mouseMoveEvent](#), [mouseReleaseEvent](#), [mouseDoubleClickEvent](#)

Reimplemented in [QCPCColorScale](#), and [QCPAxisRect](#).

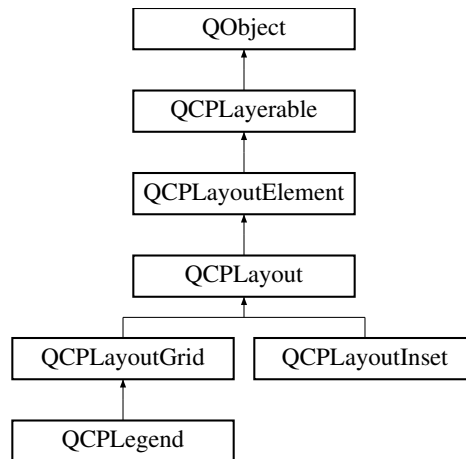
The documentation for this class was generated from the following files:

- [/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h](#)
- [/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp](#)

6.55 QCPLayout Class Reference

The abstract base class for layouts.

Inheritance diagram for QCPLayout:



Public Member Functions

- [QCPLayout](#) ()
- virtual void [update](#) ([UpdatePhase](#) phase) Q_DECL_OVERRIDE
- virtual QList< [QCPLayoutElement](#) * > [elements](#) (bool recursive) const Q_DECL_OVERRIDE
- virtual int [elementCount](#) () const =0
- virtual [QCPLayoutElement](#) * [elementAt](#) (int index) const =0
- virtual [QCPLayoutElement](#) * [takeAt](#) (int index)=0
- virtual bool [take](#) ([QCPLayoutElement](#) *element)=0
- virtual void [simplify](#) ()
- bool [removeAt](#) (int index)
- bool [remove](#) ([QCPLayoutElement](#) *element)
- void [clear](#) ()

Protected Member Functions

- virtual void **[updateLayout](#)** ()
- void [sizeConstraintsChanged](#) () const
- void **[adoptElement](#)** ([QCPLayoutElement](#) *el)
- void **[releaseElement](#)** ([QCPLayoutElement](#) *el)
- QVector< int > **[getSectionSizes](#)** (QVector< int > maxSizes, QVector< int > minSizes, QVector< double > stretchFactors, int totalSize) const

Friends

- class **[QCPLayoutElement](#)**

Additional Inherited Members

6.55.1 Detailed Description

The abstract base class for layouts.

This is an abstract base class for layout elements whose main purpose is to define the position and size of other child layout elements. In most cases, layouts don't draw anything themselves (but there are exceptions to this, e.g. [QCPLegend](#)).

[QCPLayout](#) derives from [QCPLayoutElement](#), and thus can itself be nested in other layouts.

[QCPLayout](#) introduces a common interface for accessing and manipulating the child elements. Those functions are most notably [elementCount](#), [elementAt](#), [takeAt](#), [take](#), [simplify](#), [removeAt](#), [remove](#) and [clear](#). Individual subclasses may add more functions to this interface which are more specialized to the form of the layout. For example, [QCPLayoutGrid](#) adds functions that take row and column indices to access cells of the layout grid more conveniently.

Since this is an abstract base class, you can't instantiate it directly. Rather use one of its subclasses like [QCPLayoutGrid](#) or [QCPLayoutInset](#).

For a general introduction to the layout system, see the dedicated documentation page [The Layout System](#).

6.55.2 Constructor & Destructor Documentation

6.55.2.1 QCPLayout()

```
QCPLayout::QCPLayout ( ) [explicit]
```

Creates an instance of [QCPLayout](#) and sets default values. Note that since [QCPLayout](#) is an abstract base class, it can't be instantiated directly.

6.55.3 Member Function Documentation

6.55.3.1 clear()

```
void QCPLayout::clear ( )
```

Removes and deletes all layout elements in this layout. Finally calls [simplify](#) to make sure all empty cells are collapsed.

See also

[remove](#), [removeAt](#)

6.55.3.2 elementAt()

```
QCPLayoutElement * QCPLayout::elementAt (
    int index ) const [pure virtual]
```

Returns the element in the cell with the given *index*. If *index* is invalid, returns 0.

Note that even if *index* is valid, the respective cell may be empty in some layouts (e.g. [QCPLayoutGrid](#)), so this function may return 0 in those cases. You may use this function to check whether a cell is empty or not.

See also

[elements](#), [elementCount](#), [takeAt](#)

Implemented in [QCPLayoutInset](#), and [QCPLayoutGrid](#).

6.55.3.3 elementCount()

```
int QCPLayout::elementCount ( ) const [pure virtual]
```

Returns the number of elements/cells in the layout.

See also

[elements](#), [elementAt](#)

Implemented in [QCPLayoutInset](#), and [QCPLayoutGrid](#).

6.55.3.4 elements()

```
QList< QCPLayoutElement * > QCPLayout::elements (
    bool recursive ) const [virtual]
```

Returns a list of all child elements in this layout element. If *recursive* is true, all sub-child elements are included in the list, too.

Warning

There may be entries with value 0 in the returned list. (For example, [QCPLayoutGrid](#) may have empty cells which yield 0 at the respective index.)

Reimplemented from [QCPLayoutElement](#).

Reimplemented in [QCPLayoutGrid](#).

6.55.3.5 `remove()`

```
bool QCPLayout::remove (
    QCPLayoutElement * element )
```

Removes and deletes the provided *element*. Returns true on success. If *element* is not in the layout, returns false.

This function internally uses [takeAt](#) to remove the element from the layout and then deletes the element. Note that some layouts don't remove the respective cell right away but leave an empty cell after successful removal of the layout element. To collapse empty cells, use [simplify](#).

See also

[removeAt](#), [take](#)

6.55.3.6 `removeAt()`

```
bool QCPLayout::removeAt (
    int index )
```

Removes and deletes the element at the provided *index*. Returns true on success. If *index* is invalid or points to an empty cell, returns false.

This function internally uses [takeAt](#) to remove the element from the layout and then deletes the returned element. Note that some layouts don't remove the respective cell right away but leave an empty cell after successful removal of the layout element. To collapse empty cells, use [simplify](#).

See also

[remove](#), [takeAt](#)

6.55.3.7 `simplify()`

```
void QCPLayout::simplify ( ) [virtual]
```

Simplifies the layout by collapsing empty cells. The exact behavior depends on subclasses, the default implementation does nothing.

Not all layouts need simplification. For example, [QCPLayoutInset](#) doesn't use explicit simplification while [QCPLayoutGrid](#) does.

Reimplemented in [QCPLayoutInset](#), and [QCPLayoutGrid](#).

6.55.3.8 sizeConstraintsChanged()

```
void QCPLLayout::sizeConstraintsChanged ( ) const [protected]
```

Subclasses call this method to report changed (minimum/maximum) size constraints.

If the parent of this layout is again a [QCPLLayout](#), forwards the call to the parent's [sizeConstraintsChanged](#). If the parent is a [QWidget](#) (i.e. is the [QCustomPlot::plotLayout](#) of [QCustomPlot](#)), calls [QWidget::updateGeometry](#), so if the [QCustomPlot](#) widget is inside a Qt [QLayout](#), it may update itself and resize cells accordingly.

6.55.3.9 take()

```
bool QCPLLayout::take (
    QCPLLayoutElement * element ) [pure virtual]
```

Removes the specified *element* from the layout and returns true on success.

If the *element* isn't in this layout, returns false.

Note that some layouts don't remove the respective cell right away but leave an empty cell after successful removal of the layout element. To collapse empty cells, use [simplify](#).

See also

[takeAt](#)

Implemented in [QCPLLayoutInset](#), and [QCPLLayoutGrid](#).

6.55.3.10 takeAt()

```
QCPLLayoutElement * QCPLLayout::takeAt (
    int index ) [pure virtual]
```

Removes the element with the given *index* from the layout and returns it.

If the *index* is invalid or the cell with that index is empty, returns 0.

Note that some layouts don't remove the respective cell right away but leave an empty cell after successful removal of the layout element. To collapse empty cells, use [simplify](#).

See also

[elementAt](#), [take](#)

Implemented in [QCPLLayoutInset](#), and [QCPLLayoutGrid](#).

6.55.3.11 update()

```
void QCPLayout::update (
    UpdatePhase phase ) [virtual]
```

First calls the [QCPLayoutElement::update](#) base class implementation to update the margins on this layout.

Then calls `updateLayout` which subclasses reimplement to reposition and resize their cells.

Finally, `update` is called on all child elements.

Reimplemented from [QCPLayoutElement](#).

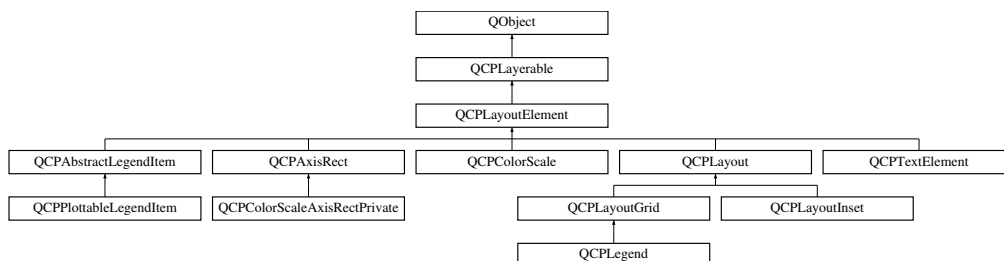
The documentation for this class was generated from the following files:

- `/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h`
- `/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp`

6.56 QCPLayoutElement Class Reference

The abstract base class for all objects that form the layout system.

Inheritance diagram for QCPLayoutElement:



Public Types

- enum [UpdatePhase](#) { `upPreparation`, `upMargins`, `upLayout` }

Public Member Functions

- [QCPLayOutElement](#) ([QCustomPlot](#) *parentPlot=0)
- [QCPLayOut](#) * [layout](#) () const
- [QRect](#) [rect](#) () const
- [QRect](#) [outerRect](#) () const
- [QMargins](#) [margins](#) () const
- [QMargins](#) [minimumMargins](#) () const
- [QCP::MarginSides](#) [autoMargins](#) () const
- [QSize](#) [minimumSize](#) () const
- [QSize](#) [maximumSize](#) () const
- [QCPMarginGroup](#) * [marginGroup](#) ([QCP::MarginSide](#) side) const
- [QHash](#)< [QCP::MarginSide](#), [QCPMarginGroup](#) * > [marginGroups](#) () const
- void [setOuterRect](#) (const [QRect](#) &rect)
- void [setMargins](#) (const [QMargins](#) &margins)
- void [setMinimumMargins](#) (const [QMargins](#) &margins)
- void [setAutoMargins](#) ([QCP::MarginSides](#) sides)
- void [setMinimumSize](#) (const [QSize](#) &size)
- void [setMinimumSize](#) (int width, int height)
- void [setMaximumSize](#) (const [QSize](#) &size)
- void [setMaximumSize](#) (int width, int height)
- void [setMarginGroup](#) ([QCP::MarginSides](#) sides, [QCPMarginGroup](#) *group)
- virtual void [update](#) ([UpdatePhase](#) phase)
- virtual [QSize](#) [minimumSizeHint](#) () const
- virtual [QSize](#) [maximumSizeHint](#) () const
- virtual [QList](#)< [QCPLayOutElement](#) * > [elements](#) (bool recursive) const
- virtual double [selectTest](#) (const [QPointF](#) &pos, bool onlySelectable, [QVariant](#) *details=0) const [Q_DECL_OVERRIDE](#)

Protected Member Functions

- virtual int [calculateAutoMargin](#) ([QCP::MarginSide](#) side)
- virtual void [layoutChanged](#) ()
- virtual void [applyDefaultAntialiasingHint](#) ([QCPPainter](#) *painter) const [Q_DECL_OVERRIDE](#)
- virtual void [draw](#) ([QCPPainter](#) *painter) [Q_DECL_OVERRIDE](#)
- virtual void [parentPlotInitialized](#) ([QCustomPlot](#) *parentPlot) [Q_DECL_OVERRIDE](#)

Protected Attributes

- [QCPLayOut](#) * [mParentLayout](#)
- [QSize](#) [mMinimumSize](#)
- [QSize](#) [mMaximumSize](#)
- [QRect](#) [mRect](#)
- [QRect](#) [mOuterRect](#)
- [QMargins](#) [mMargins](#)
- [QMargins](#) [mMinimumMargins](#)
- [QCP::MarginSides](#) [mAutoMargins](#)
- [QHash](#)< [QCP::MarginSide](#), [QCPMarginGroup](#) * > [mMarginGroups](#)

Friends

- class [QCustomPlot](#)
- class [QCPLayOut](#)
- class [QCPMarginGroup](#)

Additional Inherited Members

6.56.1 Detailed Description

The abstract base class for all objects that form the layout system.

This is an abstract base class. As such, it can't be instantiated directly, rather use one of its subclasses.

A Layout element is a rectangular object which can be placed in layouts. It has an outer rect ([QCPLayoutElement::outerRect](#)) and an inner rect ([QCPLayoutElement::rect](#)). The difference between outer and inner rect is called its margin. The margin can either be set to automatic or manual ([setAutoMargins](#)) on a per-side basis. If a side is set to manual, that margin can be set explicitly with [setMargins](#) and will stay fixed at that value. If it's set to automatic, the layout element subclass will control the value itself (via [calculateAutoMargin](#)).

Layout elements can be placed in layouts (base class [QCPLayout](#)) like [QCPLayoutGrid](#). The top level layout is reachable via [QCustomPlot::plotLayout](#), and is a [QCPLayoutGrid](#). Since [QCPLayout](#) itself derives from [QCPLayoutElement](#), layouts can be nested.

Thus in [QCustomPlot](#) one can divide layout elements into two categories: The ones that are invisible by themselves, because they don't draw anything. Their only purpose is to manage the position and size of other layout elements. This category of layout elements usually use [QCPLayout](#) as base class. Then there is the category of layout elements which actually draw something. For example, [QCPAxisRect](#), [QCPLegend](#) and [QCPTextElement](#) are of this category. This does not necessarily mean that the latter category can't have child layout elements. [QCPLegend](#) for instance, actually derives from [QCPLayoutGrid](#) and the individual legend items are child layout elements in the grid layout.

6.56.2 Member Enumeration Documentation

6.56.2.1 UpdatePhase

enum [QCPLayoutElement::UpdatePhase](#)

Defines the phases of the update process, that happens just before a replot. At each phase, [update](#) is called with the according UpdatePhase value.

Enumerator

upPreparation	Phase used for any type of preparation that needs to be done before margin calculation and layout.
upMargins	Phase in which the margins are calculated and set.
upLayout	Final phase in which the layout system places the rects of the elements.

6.56.3 Constructor & Destructor Documentation

6.56.3.1 QCPLayOutElement()

```
QCPLayOutElement::QCPLayOutElement (
    QCustomPlot * parentPlot = 0 ) [explicit]
```

Creates an instance of [QCPLayOutElement](#) and sets default values.

6.56.4 Member Function Documentation

6.56.4.1 elements()

```
QList< QCPLayOutElement * > QCPLayOutElement::elements (
    bool recursive ) const [virtual]
```

Returns a list of all child elements in this layout element. If *recursive* is true, all sub-child elements are included in the list, too.

Warning

There may be entries with value 0 in the returned list. (For example, [QCPLayOutGrid](#) may have empty cells which yield 0 at the respective index.)

Reimplemented in [QCPAxisRect](#), [QCPLayOutGrid](#), and [QCPLayOut](#).

6.56.4.2 layout()

```
QCPLayOut * QCPLayOutElement::layout ( ) const [inline]
```

Returns the parent layout of this layout element.

6.56.4.3 maximumSizeHint()

```
QSize QCPLayOutElement::maximumSizeHint ( ) const [virtual]
```

Returns the maximum size this layout element (the inner [rect](#)) may be expanded to.

if a maximum size ([setMaximumSize](#)) was not set manually, parent layouts consult this function to determine the maximum allowed size of this layout element. (A manual maximum size is considered set if it is smaller than Qt's QWIDGETSIZE_MAX.)

Reimplemented in [QCPLayOutTextElement](#), and [QCPLayOutGrid](#).

6.56.4.4 `minimumSizeHint()`

```
QSize QCPLayoutElement::minimumSizeHint ( ) const [virtual]
```

Returns the minimum size this layout element (the inner [rect](#)) may be compressed to.

if a minimum size ([setMinimumSize](#)) was not set manually, parent layouts consult this function to determine the minimum allowed size of this layout element. (A manual minimum size is considered set if it is non-zero.)

Reimplemented in [QCPTTextElement](#), [QCPPlottableLegendItem](#), and [QCPLayoutGrid](#).

6.56.4.5 `rect()`

```
QRect QCPLayoutElement::rect ( ) const [inline]
```

Returns the inner rect of this layout element. The inner rect is the outer rect ([setOuterRect](#)) shrunk by the margins ([setMargins](#), [setAutoMargins](#)).

In some cases, the area between outer and inner rect is left blank. In other cases the margin area is used to display peripheral graphics while the main content is in the inner rect. This is where automatic margin calculation becomes interesting because it allows the layout element to adapt the margins to the peripheral graphics it wants to draw. For example, [QCPAxisRect](#) draws the axis labels and tick labels in the margin area, thus needs to adjust the margins (if [setAutoMargins](#) is enabled) according to the space required by the labels of the axes.

6.56.4.6 `selectTest()`

```
double QCPLayoutElement::selectTest (
    const QPointF & pos,
    bool onlySelectable,
    QVariant * details = 0 ) const [virtual]
```

Layout elements are sensitive to events inside their outer rect. If *pos* is within the outer rect, this method returns a value corresponding to 0.99 times the parent plot's selection tolerance. However, layout elements are not selectable by default. So if *onlySelectable* is true, -1.0 is returned.

See [QCPLayerable::selectTest](#) for a general explanation of this virtual method.

[QCPLayoutElement](#) subclasses may reimplement this method to provide more specific selection test behaviour.

Reimplemented from [QCPLayerable](#).

Reimplemented in [QCPTTextElement](#), [QCPLegend](#), [QCPAbstractLegendItem](#), and [QCPLayoutInset](#).

6.56.4.7 setAutoMargins()

```
void QCPLayoutElement::setAutoMargins (
    QCP::MarginSides sides )
```

Sets on which sides the margin shall be calculated automatically. If a side is calculated automatically, a minimum margin value may be provided with [setMinimumMargins](#). If a side is set to be controlled manually, the value may be specified with [setMargins](#).

Margin sides that are under automatic control may participate in a [QCPMarginGroup](#) (see [setMarginGroup](#)), to synchronize (align) it with other layout elements in the plot.

See also

[setMinimumMargins](#), [setMargins](#), [QCP::MarginSide](#)

6.56.4.8 setMarginGroup()

```
void QCPLayoutElement::setMarginGroup (
    QCP::MarginSides sides,
    QCPMarginGroup * group )
```

Sets the margin *group* of the specified margin *sides*.

Margin groups allow synchronizing specified margins across layout elements, see the documentation of [QCPMarginGroup](#).

To unset the margin group of *sides*, set *group* to 0.

Note that margin groups only work for margin sides that are set to automatic ([setAutoMargins](#)).

See also

[QCP::MarginSide](#)

6.56.4.9 setMargins()

```
void QCPLayoutElement::setMargins (
    const QMargins & margins )
```

Sets the margins of this layout element. If [setAutoMargins](#) is disabled for some or all sides, this function is used to manually set the margin on those sides. Sides that are still set to be handled automatically are ignored and may have any value in *margins*.

The margin is the distance between the outer rect (controlled by the parent layout via [setOuterRect](#)) and the inner [rect](#) (which usually contains the main content of this layout element).

See also

[setAutoMargins](#)

6.56.4.10 `setMaximumSize()` [1/2]

```
void QCPLayoutElement::setMaximumSize (
    const QSize & size )
```

Sets the maximum size for the inner [rect](#) of this layout element. A parent layout tries to respect the *size* here by changing row/column sizes in the layout accordingly.

6.56.4.11 `setMaximumSize()` [2/2]

```
void QCPLayoutElement::setMaximumSize (
    int width,
    int height )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Sets the maximum size for the inner [rect](#) of this layout element.

6.56.4.12 `setMinimumMargins()`

```
void QCPLayoutElement::setMinimumMargins (
    const QMargins & margins )
```

If [setAutoMargins](#) is enabled on some or all margins, this function is used to provide minimum values for those margins.

The minimum values are not enforced on margin sides that were set to be under manual control via [setAutoMargins](#).

See also

[setAutoMargins](#)

6.56.4.13 `setMinimumSize()` [1/2]

```
void QCPLayoutElement::setMinimumSize (
    const QSize & size )
```

Sets the minimum size for the inner [rect](#) of this layout element. A parent layout tries to respect the *size* here by changing row/column sizes in the layout accordingly.

If the parent layout size is not sufficient to satisfy all minimum size constraints of its child layout elements, the layout may set a size that is actually smaller than *size*. [QCustomPlot](#) propagates the layout's size constraints to the outside by setting its own minimum `QWidget` size accordingly, so violations of *size* should be exceptions.

6.56.4.14 `setMinimumSize()` [2/2]

```
void QCPLayoutElement::setMinimumSize (
    int width,
    int height )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Sets the minimum size for the inner [rect](#) of this layout element.

6.56.4.15 `setOuterRect()`

```
void QCPLayoutElement::setOuterRect (
    const QRect & rect )
```

Sets the outer rect of this layout element. If the layout element is inside a layout, the layout sets the position and size of this layout element using this function.

Calling this function externally has no effect, since the layout will overwrite any changes to the outer rect upon the next replot.

The layout element will adapt its inner [rect](#) by applying the margins inward to the outer rect.

See also

[rect](#)

6.56.4.16 `update()`

```
void QCPLayoutElement::update (
    UpdatePhase phase ) [virtual]
```

Updates the layout element and sub-elements. This function is automatically called before every replot by the parent layout element. It is called multiple times, once for every [UpdatePhase](#). The phases are run through in the order of the enum values. For details about what happens at the different phases, see the documentation of [UpdatePhase](#).

Layout elements that have child elements should call the [update](#) method of their child elements, and pass the current *phase* unchanged.

The default implementation executes the automatic margin mechanism in the [upMargins](#) phase. Subclasses should make sure to call the base class implementation.

Reimplemented in [QCPCColorScale](#), [QCPAxisRect](#), and [QCPLayout](#).

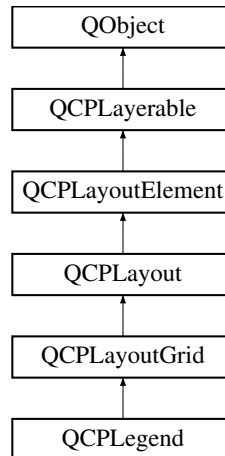
The documentation for this class was generated from the following files:

- [/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h](#)↔
- [/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp](#)↔

6.57 QCPLayoutGrid Class Reference

A layout that arranges child elements in a grid.

Inheritance diagram for QCPLayoutGrid:



Public Types

- enum `FillOrder` { `foRowsFirst`, `foColumnsFirst` }

Public Member Functions

- `QCPLayoutGrid ()`
- `int rowCount () const`
- `int columnCount () const`
- `QList< double > columnStretchFactors () const`
- `QList< double > rowStretchFactors () const`
- `int columnSpacing () const`
- `int rowSpacing () const`
- `int wrap () const`
- `FillOrder fillOrder () const`
- `void setColumnStretchFactor (int column, double factor)`
- `void setColumnStretchFactors (const QList< double > &factors)`
- `void setRowStretchFactor (int row, double factor)`
- `void setRowStretchFactors (const QList< double > &factors)`
- `void setColumnSpacing (int pixels)`
- `void setRowSpacing (int pixels)`
- `void setWrap (int count)`
- `void setFillOrder (FillOrder order, bool rearrange=true)`
- `virtual void updateLayout () Q_DECL_OVERRIDE`
- `virtual int elementCount () const Q_DECL_OVERRIDE`
- `virtual QCPLayoutElement * elementAt (int index) const Q_DECL_OVERRIDE`
- `virtual QCPLayoutElement * takeAt (int index) Q_DECL_OVERRIDE`
- `virtual bool take (QCPLayoutElement *element) Q_DECL_OVERRIDE`
- `virtual QList< QCPLayoutElement * > elements (bool recursive) const Q_DECL_OVERRIDE`
- `virtual void simplify () Q_DECL_OVERRIDE`
- `virtual QSize minimumSizeHint () const Q_DECL_OVERRIDE`

- virtual QSize [maximumSizeHint](#) () const Q_DECL_OVERRIDE
- [QCPLayouElement](#) * [element](#) (int row, int column) const
- bool [addElement](#) (int row, int column, [QCPLayouElement](#) *[element](#))
- bool [addElement](#) ([QCPLayouElement](#) *[element](#))
- bool [hasElement](#) (int row, int column)
- void [expandTo](#) (int newRowCount, int newColumnCount)
- void [insertRow](#) (int newIndex)
- void [insertColumn](#) (int newIndex)
- int [rowColToIndex](#) (int row, int column) const
- void [indexToRowCol](#) (int index, int &row, int &column) const

Protected Member Functions

- void [getMinimumRowColSizes](#) (QVector< int > *minColWidths, QVector< int > *minRowHeights) const
- void [getMaximumRowColSizes](#) (QVector< int > *maxColWidths, QVector< int > *maxRowHeights) const

Protected Attributes

- QList< QList< [QCPLayouElement](#) * > > [mElements](#)
- QList< double > [mColumnStretchFactors](#)
- QList< double > [mRowStretchFactors](#)
- int [mColumnSpacing](#)
- int [mRowSpacing](#)
- int [mWrap](#)
- [FillOrder](#) [mFillOrder](#)

Additional Inherited Members

6.57.1 Detailed Description

A layout that arranges child elements in a grid.

Elements are laid out in a grid with configurable stretch factors ([setColumnStretchFactor](#), [setRowStretchFactor](#)) and spacing ([setColumnSpacing](#), [setRowSpacing](#)).

Elements can be added to cells via [addElement](#). The grid is expanded if the specified row or column doesn't exist yet. Whether a cell contains a valid layout element can be checked with [hasElement](#), that element can be retrieved with [element](#). If rows and columns that only have empty cells shall be removed, call [simplify](#). Removal of elements is either done by just adding the element to a different layout or by using the [QCPLayou](#) interface [take](#) or [remove](#).

If you use [addElement\(QCPLayouElement*\)](#) without explicit parameters for *row* and *column*, the grid layout will choose the position according to the current [setFillOrder](#) and the wrapping ([setWrap](#)).

Row and column insertion can be performed with [insertRow](#) and [insertColumn](#).

6.57.2 Member Enumeration Documentation

6.57.2.1 FillOrder

```
enum QCPLayouGrid::FillOrder
```

Defines in which direction the grid is filled when using [addElement\(QCPLayouElement*\)](#). The column/row at which wrapping into the next row/column occurs can be specified with [setWrap](#).

See also

[setFillOrder](#)

Enumerator

foRowsFirst	Rows are filled first, and a new element is wrapped to the next column if the row count would exceed setWrap .
foColumnsFirst	Columns are filled first, and a new element is wrapped to the next row if the column count would exceed setWrap .

6.57.3 Constructor & Destructor Documentation

6.57.3.1 QCPLayoutGrid()

```
QCPLayoutGrid::QCPLayoutGrid ( ) [explicit]
```

Creates an instance of [QCPLayoutGrid](#) and sets default values.

6.57.4 Member Function Documentation

6.57.4.1 addElement() [1/2]

```
bool QCPLayoutGrid::addElement (
    int row,
    int column,
    QCPLayoutElement * element )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Adds the *element* to cell with *row* and *column*. If *element* is already in a layout, it is first removed from there. If *row* or *column* don't exist yet, the layout is expanded accordingly.

Returns true if the element was added successfully, i.e. if the cell at *row* and *column* didn't already have an element.

Use the overload of this method without explicit row/column index to place the element according to the configured fill order and wrapping settings.

See also

[element](#), [hasElement](#), [take](#), [remove](#)

6.57.4.2 addElement() [2/2]

```
bool QCPLayoutGrid::addElement (
    QCPLayoutElement * element )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Adds the *element* to the next empty cell according to the current fill order ([setFillOrder](#)) and wrapping ([setWrap](#)). If *element* is already in a layout, it is first removed from there. If necessary, the layout is expanded to hold the new element.

Returns true if the element was added successfully.

See also

[setFillOrder](#), [setWrap](#), [element](#), [hasElement](#), [take](#), [remove](#)

6.57.4.3 columnCount()

```
int QCPLayoutGrid::columnCount ( ) const [inline]
```

Returns the number of columns in the layout.

See also

[rowCount](#)

6.57.4.4 element()

```
QCPLayoutElement * QCPLayoutGrid::element (
    int row,
    int column ) const
```

Returns the element in the cell in *row* and *column*.

Returns 0 if either the row/column is invalid or if the cell is empty. In those cases, a qDebug message is printed. To check whether a cell exists and isn't empty, use [hasElement](#).

See also

[addElement](#), [hasElement](#)

6.57.4.5 `elementAt()`

```
QCPLayoutElement * QCPLayoutGrid::elementAt (
    int index ) const [virtual]
```

Note that the association of the linear *index* to the row/column based cells depends on the current setting of [setFillOrder](#).

See also

[rowColToIndex](#)

Implements [QCPLayout](#).

6.57.4.6 `elementCount()`

```
virtual int QCPLayoutGrid::elementCount ( ) const [inline], [virtual]
```

Returns the number of elements/cells in the layout.

See also

[elements](#), [elementAt](#)

Implements [QCPLayout](#).

6.57.4.7 `elements()`

```
QList< QCPLayoutElement * > QCPLayoutGrid::elements (
    bool recursive ) const [virtual]
```

Returns a list of all child elements in this layout element. If *recursive* is true, all sub-child elements are included in the list, too.

Warning

There may be entries with value 0 in the returned list. (For example, [QCPLayoutGrid](#) may have empty cells which yield 0 at the respective index.)

Reimplemented from [QCPLayout](#).

6.57.4.8 `expandTo()`

```
void QCPLayoutGrid::expandTo (
    int newRowCount,
    int newColumnCount )
```

Expands the layout to have *newRowCount* rows and *newColumnCount* columns. So the last valid row index will be *newRowCount-1*, the last valid column index will be *newColumnCount-1*.

If the current column/row count is already larger or equal to *newColumnCount/newRowCount*, this function does nothing in that dimension.

Newly created cells are empty, new rows and columns have the stretch factor 1.

Note that upon a call to [addElement](#), the layout is expanded automatically to contain the specified row and column, using this function.

See also

[simplify](#)

6.57.4.9 `hasElement()`

```
bool QCPLayoutGrid::hasElement (
    int row,
    int column )
```

Returns whether the cell at *row* and *column* exists and contains a valid element, i.e. isn't empty.

See also

[element](#)

6.57.4.10 `indexToRowCol()`

```
void QCPLayoutGrid::indexToRowCol (
    int index,
    int & row,
    int & column ) const
```

Converts the linear index to row and column indices and writes the result to *row* and *column*.

The way the cells are indexed depends on [setFillOrder](#). If it is [foRowsFirst](#), the indices increase left to right and then top to bottom. If it is [foColumnsFirst](#), the indices increase top to bottom and then left to right.

If there are no cells (i.e. column or row count is zero), sets *row* and *column* to -1.

For the retrieved *row* and *column* to be valid, the passed *index* must be valid itself, i.e. greater or equal to zero and smaller than the current [elementCount](#).

See also

[rowColToIndex](#)

6.57.4.11 insertColumn()

```
void QCPLayoutGrid::insertColumn (
    int newIndex )
```

Inserts a new column with empty cells at the column index *newIndex*. Valid values for *newIndex* range from 0 (inserts a row at the left) to *rowCount* (appends a row at the right).

See also

[insertRow](#)

6.57.4.12 insertRow()

```
void QCPLayoutGrid::insertRow (
    int newIndex )
```

Inserts a new row with empty cells at the row index *newIndex*. Valid values for *newIndex* range from 0 (inserts a row at the top) to *rowCount* (appends a row at the bottom).

See also

[insertColumn](#)

6.57.4.13 maximumSizeHint()

```
QSize QCPLayoutGrid::maximumSizeHint ( ) const [virtual]
```

Returns the maximum size this layout element (the inner [rect](#)) may be expanded to.

if a maximum size ([setMaximumSize](#)) was not set manually, parent layouts consult this function to determine the maximum allowed size of this layout element. (A manual maximum size is considered set if it is smaller than Qt's QWIDGETSIZE_MAX.)

Reimplemented from [QCPLayoutElement](#).

6.57.4.14 minimumSizeHint()

```
QSize QCPLayoutGrid::minimumSizeHint ( ) const [virtual]
```

Returns the minimum size this layout element (the inner [rect](#)) may be compressed to.

if a minimum size ([setMinimumSize](#)) was not set manually, parent layouts consult this function to determine the minimum allowed size of this layout element. (A manual minimum size is considered set if it is non-zero.)

Reimplemented from [QCPLayoutElement](#).

6.57.4.15 rowColToIndex()

```
int QCPLayoutGrid::rowColToIndex (
    int row,
    int column ) const
```

Converts the given *row* and *column* to the linear index used by some methods of [QCPLayoutGrid](#) and [QCPLayout](#).

The way the cells are indexed depends on [setFillOrder](#). If it is [foRowsFirst](#), the indices increase left to right and then top to bottom. If it is [foColumnsFirst](#), the indices increase top to bottom and then left to right.

For the returned index to be valid, *row* and *column* must be valid indices themselves, i.e. greater or equal to zero and smaller than the current [rowCount](#)/[columnCount](#).

See also

[indexToRowCol](#)

6.57.4.16 rowCount()

```
int QCPLayoutGrid::rowCount ( ) const [inline]
```

Returns the number of rows in the layout.

See also

[columnCount](#)

6.57.4.17 setColumnSpacing()

```
void QCPLayoutGrid::setColumnSpacing (
    int pixels )
```

Sets the gap that is left blank between columns to *pixels*.

See also

[setRowSpacing](#)

6.57.4.18 `setColumnStretchFactor()`

```
void QCPLayoutGrid::setColumnStretchFactor (
    int column,
    double factor )
```

Sets the stretch *factor* of *column*.

Stretch factors control the relative sizes of rows and columns. Cells will not be resized beyond their minimum and maximum widths/heights ([QCPLayoutElement::setMinimumSize](#), [QCPLayoutElement::setMaximumSize](#)), regardless of the stretch factor.

The default stretch factor of newly created rows/columns is 1.

See also

[setColumnStretchFactors](#), [setRowStretchFactor](#)

6.57.4.19 `setColumnStretchFactors()`

```
void QCPLayoutGrid::setColumnStretchFactors (
    const QList< double > & factors )
```

Sets the stretch *factors* of all columns. *factors* must have the size [columnCount](#).

Stretch factors control the relative sizes of rows and columns. Cells will not be resized beyond their minimum and maximum widths/heights ([QCPLayoutElement::setMinimumSize](#), [QCPLayoutElement::setMaximumSize](#)), regardless of the stretch factor.

The default stretch factor of newly created rows/columns is 1.

See also

[setColumnStretchFactor](#), [setRowStretchFactors](#)

6.57.4.20 `setFillOrder()`

```
void QCPLayoutGrid::setFillOrder (
    FillOrder order,
    bool rearrange = true )
```

Sets the filling order and wrapping behaviour that is used when adding new elements with the method [addElement\(QCPLayoutElement*\)](#).

The specified *order* defines whether rows or columns are filled first. Using [setWrap](#), you can control at which row/column count wrapping into the next column/row will occur. If you set it to zero, no wrapping will ever occur. Changing the fill order also changes the meaning of the linear index used e.g. in [elementAt](#) and [takeAt](#).

If you want to have all current elements arranged in the new order, set *rearrange* to true. The elements will be rearranged in a way that tries to preserve their linear index. However, empty cells are skipped during build-up of the new cell order, which shifts the succeeding element's index. The rearranging is performed even if the specified *order* is already the current fill order. Thus this method can be used to re-wrap the current elements.

If *rearrange* is false, the current element arrangement is not changed, which means the linear indexes change (because the linear index is dependent on the fill order).

Note that the method [addElement\(int row, int column, QCPLayoutElement *element\)](#) with explicitly stated row and column is not subject to wrapping and can place elements even beyond the specified wrapping point.

See also

[setWrap](#), [addElement\(QCPLayoutElement*\)](#)

6.57.4.21 setRowSpacing()

```
void QCPLayoutGrid::setRowSpacing (
    int pixels )
```

Sets the gap that is left blank between rows to *pixels*.

See also

[setColumnSpacing](#)

6.57.4.22 setRowStretchFactor()

```
void QCPLayoutGrid::setRowStretchFactor (
    int row,
    double factor )
```

Sets the stretch *factor* of *row*.

Stretch factors control the relative sizes of rows and columns. Cells will not be resized beyond their minimum and maximum widths/heights ([QCPLayoutElement::setMinimumSize](#), [QCPLayoutElement::setMaximumSize](#)), regardless of the stretch factor.

The default stretch factor of newly created rows/columns is 1.

See also

[setColumnStretchFactors](#), [setRowStretchFactor](#)

6.57.4.23 setRowStretchFactors()

```
void QCPLayoutGrid::setRowStretchFactors (
    const QList< double > & factors )
```

Sets the stretch *factors* of all rows. *factors* must have the size [rowCount](#).

Stretch factors control the relative sizes of rows and columns. Cells will not be resized beyond their minimum and maximum widths/heights ([QCPLayoutElement::setMinimumSize](#), [QCPLayoutElement::setMaximumSize](#)), regardless of the stretch factor.

The default stretch factor of newly created rows/columns is 1.

See also

[setRowStretchFactor](#), [setColumnStretchFactors](#)

6.57.4.24 setWrap()

```
void QCPLayoutGrid::setWrap (
    int count )
```

Sets the maximum number of columns or rows that are used, before new elements added with [addElement\(QCPLLayoutElement*\)](#) will start to fill the next row or column, respectively. It depends on [setFillOrder](#), whether rows or columns are wrapped.

If *count* is set to zero, no wrapping will ever occur.

If you wish to re-wrap the elements currently in the layout, call [setFillOrder](#) with *rearrange* set to true (the actual fill order doesn't need to be changed for the rearranging to be done).

Note that the method [addElement\(int row, int column, QCPLLayoutElement *element\)](#) with explicitly stated row and column is not subject to wrapping and can place elements even beyond the specified wrapping point.

See also

[setFillOrder](#)

6.57.4.25 simplify()

```
void QCPLayoutGrid::simplify ( ) [virtual]
```

Simplifies the layout by collapsing rows and columns which only contain empty cells.

Reimplemented from [QCPLLayout](#).

6.57.4.26 take()

```
bool QCPLayoutGrid::take (
    QCPLLayoutElement * element ) [virtual]
```

Removes the specified *element* from the layout and returns true on success.

If the *element* isn't in this layout, returns false.

Note that some layouts don't remove the respective cell right away but leave an empty cell after successful removal of the layout element. To collapse empty cells, use [simplify](#).

See also

[takeAt](#)

Implements [QCPLLayout](#).

6.57.4.27 takeAt()

```
QCPLayoutElement * QCPLayoutGrid::takeAt (
    int index ) [virtual]
```

Note that the association of the linear *index* to the row/column based cells depends on the current setting of [setFillOrder](#).

See also

[rowColToIndex](#)

Implements [QCPLayout](#).

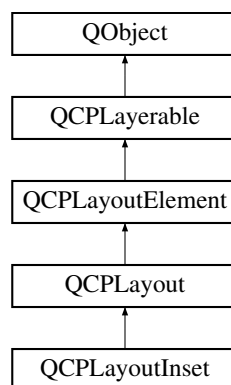
The documentation for this class was generated from the following files:

- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h
- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp

6.58 QCPLayoutInset Class Reference

A layout that places child elements aligned to the border or arbitrarily positioned.

Inheritance diagram for QCPLayoutInset:



Public Types

- enum [InsetPlacement](#) { [ipFree](#), [ipBorderAligned](#) }

Public Member Functions

- [QCPLayoutInset](#) ()
- [InsetPlacement insetPlacement](#) (int index) const
- [Qt::Alignment insetAlignment](#) (int index) const
- [QRectF insetRect](#) (int index) const
- void [setInsetPlacement](#) (int index, [InsetPlacement](#) placement)
- void [setInsetAlignment](#) (int index, [Qt::Alignment](#) alignment)
- void [setInsetRect](#) (int index, const [QRectF](#) &rect)
- virtual void **updateLayout** () Q_DECL_OVERRIDE
- virtual int [elementCount](#) () const Q_DECL_OVERRIDE
- virtual [QCPLayoutElement](#) * [elementAt](#) (int index) const Q_DECL_OVERRIDE
- virtual [QCPLayoutElement](#) * [takeAt](#) (int index) Q_DECL_OVERRIDE
- virtual bool [take](#) ([QCPLayoutElement](#) *element) Q_DECL_OVERRIDE
- virtual void [simplify](#) () Q_DECL_OVERRIDE
- virtual double [selectTest](#) (const [QPointF](#) &pos, bool onlySelectable, [QVariant](#) *details=0) const Q_DECL_OVERRIDE
- void [addElement](#) ([QCPLayoutElement](#) *element, [Qt::Alignment](#) alignment)
- void [addElement](#) ([QCPLayoutElement](#) *element, const [QRectF](#) &rect)

Protected Attributes

- [QList](#)< [QCPLayoutElement](#) * > **mElements**
- [QList](#)< [InsetPlacement](#) > **mInsetPlacement**
- [QList](#)< [Qt::Alignment](#) > **mInsetAlignment**
- [QList](#)< [QRectF](#) > **mInsetRect**

Additional Inherited Members

6.58.1 Detailed Description

A layout that places child elements aligned to the border or arbitrarily positioned.

Elements are placed either aligned to the border or at arbitrary position in the area of the layout. Which placement applies is controlled with the [InsetPlacement](#) ([setInsetPlacement](#)).

Elements are added via [addElement\(QCPLayoutElement *element, Qt::Alignment alignment\)](#) or [addElement\(QCPLayoutElement *element, const QRectF &rect\)](#). If the first method is used, the inset placement will default to [ipBorderAligned](#) and the element will be aligned according to the *alignment* parameter. The second method defaults to [ipFree](#) and allows placing elements at arbitrary position and size, defined by *rect*.

The alignment or rect can be set via [setInsetAlignment](#) or [setInsetRect](#), respectively.

This is the layout that every [QCPAxisRect](#) has as [QCPAxisRect::insetLayout](#).

6.58.2 Member Enumeration Documentation

6.58.2.1 InsetPlacement

```
enum QCPLayoutInset::InsetPlacement
```

Defines how the placement and sizing is handled for a certain element in a [QCPLayoutInset](#).

Enumerator

ipFree	The element may be positioned/sized arbitrarily, see setInsetRect .
ipBorderAligned	The element is aligned to one of the layout sides, see setInsetAlignment .

6.58.3 Constructor & Destructor Documentation

6.58.3.1 QCPLayoutInset()

```
QCPLayoutInset::QCPLayoutInset ( ) [explicit]
```

Creates an instance of [QCPLayoutInset](#) and sets default values.

6.58.4 Member Function Documentation

6.58.4.1 addElement() [1/2]

```
void QCPLayoutInset::addElement (
    QCPLayoutElement * element,
    Qt::Alignment alignment )
```

Adds the specified *element* to the layout as an inset aligned at the border ([setInsetAlignment](#) is initialized with [ipBorderAligned](#)). The alignment is set to *alignment*.

alignment is an or combination of the following alignment flags: Qt::AlignLeft, Qt::AlignHCenter, Qt::AlignRight, Qt::AlignTop, Qt::AlignVCenter, Qt::AlignBottom. Any other alignment flags will be ignored.

See also

[addElement\(QCPLayoutElement *element, const QRectF &rect\)](#)

6.58.4.2 addElement() [2/2]

```
void QCPLayoutInset::addElement (
    QCPLayoutElement * element,
    const QRectF & rect )
```

Adds the specified *element* to the layout as an inset with free positioning/sizing ([setInsetAlignment](#) is initialized with [ipFree](#)). The position and size is set to *rect*.

rect is given in fractions of the whole inset layout rect. So an inset with rect (0, 0, 1, 1) will span the entire layout. An inset with rect (0.6, 0.1, 0.35, 0.35) will be in the top right corner of the layout, with 35% width and height of the parent layout.

See also

[addElement\(QCPLayoutElement *element, Qt::Alignment alignment\)](#)

6.58.4.3 `elementAt()`

```
QCLayoutElement * QCLayoutInset::elementAt (
    int index ) const [virtual]
```

Returns the element in the cell with the given *index*. If *index* is invalid, returns 0.

Note that even if *index* is valid, the respective cell may be empty in some layouts (e.g. [QCLayoutGrid](#)), so this function may return 0 in those cases. You may use this function to check whether a cell is empty or not.

See also

[elements](#), [elementCount](#), [takeAt](#)

Implements [QCLayout](#).

6.58.4.4 `elementCount()`

```
int QCLayoutInset::elementCount ( ) const [virtual]
```

Returns the number of elements/cells in the layout.

See also

[elements](#), [elementAt](#)

Implements [QCLayout](#).

6.58.4.5 `insetAlignment()`

```
Qt::Alignment QCLayoutInset::insetAlignment (
    int index ) const
```

Returns the alignment of the element with the specified *index*. The alignment only has a meaning, if the inset placement ([setInsetPlacement](#)) is [ipBorderAligned](#).

6.58.4.6 `insetPlacement()`

```
QCLayoutInset::InsetPlacement QCLayoutInset::insetPlacement (
    int index ) const
```

Returns the placement type of the element with the specified *index*.

6.58.4.7 insetRect()

```
QRectF QCPLayoutInset::insetRect (
    int index ) const
```

Returns the rect of the element with the specified *index*. The rect only has a meaning, if the inset placement ([setInsetPlacement](#)) is [ipFree](#).

6.58.4.8 selectTest()

```
double QCPLayoutInset::selectTest (
    const QPointF & pos,
    bool onlySelectable,
    QVariant * details = 0 ) const [virtual]
```

The inset layout is sensitive to events only at areas where its (visible) child elements are sensitive. If the `selectTest` method of any of the child elements returns a positive number for *pos*, this method returns a value corresponding to 0.99 times the parent plot's selection tolerance. The inset layout is not selectable itself by default. So if *onlySelectable* is true, -1.0 is returned.

See [QCPLayerable::selectTest](#) for a general explanation of this virtual method.

Reimplemented from [QCPLayoutElement](#).

6.58.4.9 setInsetAlignment()

```
void QCPLayoutInset::setInsetAlignment (
    int index,
    Qt::Alignment alignment )
```

If the inset placement ([setInsetPlacement](#)) is [ipBorderAligned](#), this function is used to set the alignment of the element with the specified *index* to *alignment*.

alignment is an or combination of the following alignment flags: `Qt::AlignLeft`, `Qt::AlignHCenter`, `Qt::AlignRight`, `Qt::AlignTop`, `Qt::AlignVCenter`, `Qt::AlignBottom`. Any other alignment flags will be ignored.

6.58.4.10 setInsetPlacement()

```
void QCPLayoutInset::setInsetPlacement (
    int index,
    QCPLayoutInset::InsetPlacement placement )
```

Sets the inset placement type of the element with the specified *index* to *placement*.

See also

[InsetPlacement](#)

6.58.4.11 `setInsetRect()`

```
void QCPLayoutInset::setInsetRect (
    int index,
    const QRectF & rect )
```

If the inset placement ([setInsetPlacement](#)) is [ipFree](#), this function is used to set the position and size of the element with the specified *index* to *rect*.

rect is given in fractions of the whole inset layout rect. So an inset with rect (0, 0, 1, 1) will span the entire layout. An inset with rect (0.6, 0.1, 0.35, 0.35) will be in the top right corner of the layout, with 35% width and height of the parent layout.

Note that the minimum and maximum sizes of the embedded element ([QCPLayoutElement::setMinimumSize](#), [QCPLayoutElement::setMaximumSize](#)) are enforced.

6.58.4.12 `simplify()`

```
void QCPLayoutInset::simplify ( ) [inline], [virtual]
```

The QCPLInsetLayout does not need simplification since it can never have empty cells due to its linear index structure. This method does nothing.

Reimplemented from [QCPLayout](#).

6.58.4.13 `take()`

```
bool QCPLayoutInset::take (
    QCPLayoutElement * element ) [virtual]
```

Removes the specified *element* from the layout and returns true on success.

If the *element* isn't in this layout, returns false.

Note that some layouts don't remove the respective cell right away but leave an empty cell after successful removal of the layout element. To collapse empty cells, use [simplify](#).

See also

[takeAt](#)

Implements [QCPLayout](#).

6.58.4.14 takeAt()

```
QCPLayoutElement * QCPLayoutInset::takeAt (
    int index ) [virtual]
```

Removes the element with the given *index* from the layout and returns it.

If the *index* is invalid or the cell with that index is empty, returns 0.

Note that some layouts don't remove the respective cell right away but leave an empty cell after successful removal of the layout element. To collapse empty cells, use [simplify](#).

See also

[elementAt](#), [take](#)

Implements [QCPLayout](#).

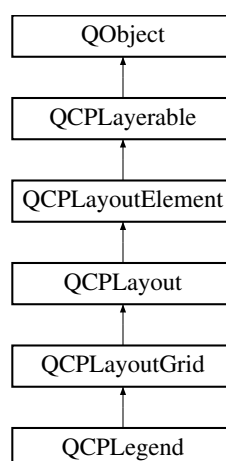
The documentation for this class was generated from the following files:

- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h↵
- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp↵

6.59 QCPLegend Class Reference

Manages a legend inside a [QCustomPlot](#).

Inheritance diagram for QCPLegend:



Public Types

- enum [SelectablePart](#) { [spNone](#) = 0x000, [spLegendBox](#) = 0x001, [splItems](#) = 0x002 }

Signals

- void [selectionChanged](#) (QCPLegend::SelectableParts parts)
- void **selectableChanged** (QCPLegend::SelectableParts parts)

Public Member Functions

- [QCPLegend](#) ()
- QPen **borderPen** () const
- QBrush **brush** () const
- QFont **font** () const
- QColor **textColor** () const
- QSize **iconSize** () const
- int **iconTextPadding** () const
- QPen **iconBorderPen** () const
- SelectableParts **selectableParts** () const
- SelectableParts **selectedParts** () const
- QPen **selectedBorderPen** () const
- QPen **selectedIconBorderPen** () const
- QBrush **selectedBrush** () const
- QFont **selectedFont** () const
- QColor **selectedTextColor** () const
- void [setBorderPen](#) (const QPen &pen)
- void [setBrush](#) (const QBrush &brush)
- void [setFont](#) (const QFont &font)
- void [setTextColor](#) (const QColor &color)
- void [setIconSize](#) (const QSize &size)
- void [setIconSize](#) (int width, int height)
- void [setIconTextPadding](#) (int padding)
- void [setIconBorderPen](#) (const QPen &pen)
- Q_SLOT void [setSelectableParts](#) (const SelectableParts &selectableParts)
- Q_SLOT void [setSelectedParts](#) (const SelectableParts &selectedParts)
- void [setSelectedBorderPen](#) (const QPen &pen)
- void [setSelectedIconBorderPen](#) (const QPen &pen)
- void [setSelectedBrush](#) (const QBrush &brush)
- void [setSelectedFont](#) (const QFont &font)
- void [setSelectedTextColor](#) (const QColor &color)
- virtual double [selectTest](#) (const QPointF &pos, bool onlySelectable, QVariant *details=0) const Q_DECL_OVERRIDE
- QCPAbstractLegendItem * [item](#) (int index) const
- QCPPlottableLegendItem * [itemWithPlottable](#) (const QCPAbstractPlottable *plottable) const
- int [itemCount](#) () const
- bool [hasItem](#) (QCPAbstractLegendItem *item) const
- bool [hasItemWithPlottable](#) (const QCPAbstractPlottable *plottable) const
- bool [addItem](#) (QCPAbstractLegendItem *item)
- bool [removeItem](#) (int index)
- bool [removeItem](#) (QCPAbstractLegendItem *item)
- void [clearItems](#) ()
- QList< QCPAbstractLegendItem * > [selectedItems](#) () const

Protected Member Functions

- virtual void **parentPlotInitialized** ([QCustomPlot](#) *parentPlot) Q_DECL_OVERRIDE
- virtual [QCP::Interaction](#) **selectionCategory** () const Q_DECL_OVERRIDE
- virtual void **applyDefaultAntialiasingHint** ([QCPPainter](#) *painter) const Q_DECL_OVERRIDE
- virtual void **draw** ([QCPPainter](#) *painter) Q_DECL_OVERRIDE
- virtual void **selectEvent** (QMouseEvent *event, bool additive, const QVariant &details, bool *selectionStateChanged) Q_DECL_OVERRIDE
- virtual void **deselectEvent** (bool *selectionStateChanged) Q_DECL_OVERRIDE
- QPen **getBorderPen** () const
- QBrush **getBrush** () const

Protected Attributes

- QPen **mBorderPen**
- QPen **mIconBorderPen**
- QBrush **mBrush**
- QFont **mFont**
- QColor **mTextColor**
- QSize **mIconSize**
- int **mIconTextPadding**
- SelectableParts **mSelectedParts**
- SelectableParts **mSelectableParts**
- QPen **mSelectedBorderPen**
- QPen **mSelectedIconBorderPen**
- QBrush **mSelectedBrush**
- QFont **mSelectedFont**
- QColor **mSelectedTextColor**

Friends

- class [QCustomPlot](#)
- class [QCPAbstractLegendItem](#)

6.59.1 Detailed Description

Manages a legend inside a [QCustomPlot](#).

A legend is a small box somewhere in the plot which lists plottables with their name and icon.

Normally, the legend is populated by calling [QCPAbstractPlottable::addToLegend](#). The respective legend item can be removed with [QCPAbstractPlottable::removeFromLegend](#). However, [QCPLegend](#) also offers an interface to add and manipulate legend items directly: [item](#), [itemWithPlottable](#), [itemCount](#), [addItem](#), [removeItem](#), etc.

Since [QCPLegend](#) derives from [QCPLayoutGrid](#), it can be placed in any position a [QCPLayoutElement](#) may be positioned. The legend items are themselves [QCPLayoutElements](#) which are placed in the grid layout of the legend. [QCPLegend](#) only adds an interface specialized for handling child elements of type [QCPAbstractLegendItem](#), as mentioned above. In principle, any other layout elements may also be added to a legend via the normal [QCPLayoutGrid](#) interface. See the special page about The Layout System for examples on how to add other elements to the legend and move it outside the axis rect.

Use the methods [setFillOrder](#) and [setWrap](#) inherited from [QCPLayoutGrid](#) to control in which order (column first or row first) the legend is filled up when calling [addItem](#), and at which column or row wrapping occurs.

By default, every [QCustomPlot](#) has one legend ([QCustomPlot::legend](#)) which is placed in the inset layout of the main axis rect ([QCPAxisRect::insetLayout](#)). To move the legend to another position inside the axis rect, use the methods of the [QCPLayoutInset](#). To move the legend outside of the axis rect, place it anywhere else with the [QCPLayout/QCPLayoutElement](#) interface.

6.59.2 Member Enumeration Documentation

6.59.2.1 SelectablePart

```
enum QCPLegend::SelectablePart
```

Defines the selectable parts of a legend

See also

[setSelectedParts](#), [setSelectableParts](#)

Enumerator

spNone	0x000 None
spLegendBox	0x001 The legend box (frame)
spItems	0x002 Legend items individually (see selectedItems)

6.59.3 Constructor & Destructor Documentation

6.59.3.1 QCPLegend()

```
QCPLegend::QCPLegend ( ) [explicit]
```

Constructs a new [QCPLegend](#) instance with default values.

Note that by default, [QCustomPlot](#) already contains a legend ready to be used as [QCustomPlot::legend](#)

6.59.4 Member Function Documentation

6.59.4.1 addItem()

```
bool QCPLegend::addItem (
    QCPAbstractLegendItem * item )
```

Adds *item* to the legend, if it's not present already. The element is arranged according to the current fill order ([setFillOrder](#)) and wrapping ([setWrap](#)).

Returns true on success, i.e. if the item wasn't in the list already and has been successfully added.

The legend takes ownership of the item.

See also

[removeItem](#), [item](#), [hasItem](#)

6.59.4.2 clearItems()

```
void QCPLegend::clearItems ( )
```

Removes all items from the legend.

6.59.4.3 hasItem()

```
bool QCPLegend::hasItem (
    QCPAbstractLegendItem * item ) const
```

Returns whether the legend contains *item*.

See also

[hasItemWithPlottable](#)

6.59.4.4 hasItemWithPlottable()

```
bool QCPLegend::hasItemWithPlottable (
    const QCPAbstractPlottable * plottable ) const
```

Returns whether the legend contains a [QCPPlottableLegendItem](#) which is associated with *plottable* (e.g. a [QCPGraph](#)*). If such an item isn't in the legend, returns false.

See also

[itemWithPlottable](#)

6.59.4.5 item()

```
QCPAbstractLegendItem * QCPLegend::item (
    int index ) const
```

Returns the item with index *i*.

Note that the linear index depends on the current fill order ([setFillOrder](#)).

See also

[itemCount](#), [addItem](#), [itemWithPlottable](#)

6.59.4.6 itemCount()

```
int QCPLegend::itemCount ( ) const
```

Returns the number of items currently in the legend.

Note that if empty cells are in the legend (e.g. by calling methods of the [QCPLayoutGrid](#) base class which allows creating empty cells), they are included in the returned count.

See also

[item](#)

6.59.4.7 itemWithPlottable()

```
QCPLottableLegendItem * QCPLegend::itemWithPlottable (
    const QCPLAbstractPlottable * plottable ) const
```

Returns the [QCPLottableLegendItem](#) which is associated with *plottable* (e.g. a [QCPLGraph*](#)). If such an item isn't in the legend, returns 0.

See also

[hasItemWithPlottable](#)

6.59.4.8 removeItem() [1/2]

```
bool QCPLegend::removeItem (
    int index )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Removes the item with the specified *index* from the legend and deletes it.

After successful removal, the legend is reordered according to the current fill order ([setFillOrder](#)) and wrapping ([setWrap](#)), so no empty cell remains where the removed *item* was. If you don't want this, rather use the raw element interface of [QCPLayoutGrid](#).

Returns true, if successful. Unlike [QCPLayoutGrid::removeAt](#), this method only removes elements derived from [QCPLAbstractLegendItem](#).

See also

[itemCount](#), [clearItems](#)

6.59.4.9 `removeItem()` [2/2]

```
bool QCPLegend::removeItem (
    QCPAbstractLegendItem * item )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Removes *item* from the legend and deletes it.

After successful removal, the legend is reordered according to the current fill order ([setFillOrder](#)) and wrapping ([setWrap](#)), so no empty cell remains where the removed *item* was. If you don't want this, rather use the raw element interface of [QCPLayoutGrid](#).

Returns true, if successful.

See also

[clearItems](#)

6.59.4.10 `selectedItems()`

```
QList< QCPAbstractLegendItem * > QCPLegend::selectedItems ( ) const
```

Returns the legend items that are currently selected. If no items are selected, the list is empty.

See also

[QCPAbstractLegendItem::setSelected](#), [setSelectable](#)

6.59.4.11 `selectionChanged`

```
void QCPLegend::selectionChanged (
    QCPLegend::SelectableParts selection ) [signal]
```

This signal is emitted when the selection state of this legend has changed.

See also

[setSelectedParts](#), [setSelectableParts](#)

6.59.4.12 selectTest()

```
double QCPLegend::selectTest (
    const QPointF & pos,
    bool onlySelectable,
    QVariant * details = 0 ) const [virtual]
```

Layout elements are sensitive to events inside their outer rect. If *pos* is within the outer rect, this method returns a value corresponding to 0.99 times the parent plot's selection tolerance. However, layout elements are not selectable by default. So if *onlySelectable* is true, -1.0 is returned.

See [QCPLayerable::selectTest](#) for a general explanation of this virtual method.

[QCPLayoutElement](#) subclasses may reimplement this method to provide more specific selection test behaviour.

Reimplemented from [QCPLayoutElement](#).

6.59.4.13 setBorderPen()

```
void QCPLegend::setBorderPen (
    const QPen & pen )
```

Sets the pen, the border of the entire legend is drawn with.

6.59.4.14 setBrush()

```
void QCPLegend::setBrush (
    const QBrush & brush )
```

Sets the brush of the legend background.

6.59.4.15 setFont()

```
void QCPLegend::setFont (
    const QFont & font )
```

Sets the default font of legend text. Legend items that draw text (e.g. the name of a graph) will use this font by default. However, a different font can be specified on a per-item-basis by accessing the specific legend item.

This function will also set *font* on all already existing legend items.

See also

[QCPLegendItem::setFont](#)

6.59.4.16 setIconBorderPen()

```
void QCPLegend::setIconBorderPen (
    const QPen & pen )
```

Sets the pen used to draw a border around each legend icon. Legend items that draw an icon (e.g. a visual representation of the graph) will use this pen by default.

If no border is wanted, set this to *Qt::NoPen*.

6.59.4.17 setIconSize() [1/2]

```
void QCPLegend::setIconSize (
    const QSize & size )
```

Sets the size of legend icons. Legend items that draw an icon (e.g. a visual representation of the graph) will use this size by default.

6.59.4.18 setIconSize() [2/2]

```
void QCPLegend::setIconSize (
    int width,
    int height )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

6.59.4.19 setIconTextPadding()

```
void QCPLegend::setIconTextPadding (
    int padding )
```

Sets the horizontal space in pixels between the legend icon and the text next to it. Legend items that draw an icon (e.g. a visual representation of the graph) and text (e.g. the name of the graph) will use this space by default.

6.59.4.20 setSelectableParts()

```
void QCPLegend::setSelectableParts (
    const SelectableParts & selectable )
```

Sets whether the user can (de-)select the parts in *selectable* by clicking on the [QCustomPlot](#) surface. (When [QCustomPlot::setInteractions](#) contains [QCP::iSelectLegend](#).)

However, even when *selectable* is set to a value not allowing the selection of a specific part, it is still possible to set the selection of this part manually, by calling [setSelectedParts](#) directly.

See also

[SelectablePart](#), [setSelectedParts](#)

6.59.4.21 setSelectedBorderPen()

```
void QCPLegend::setSelectedBorderPen (
    const QPen & pen )
```

When the legend box is selected, this pen is used to draw the border instead of the normal pen set via [setBorderPen](#).

See also

[setSelectedParts](#), [setSelectableParts](#), [setSelectedBrush](#)

6.59.4.22 setSelectedBrush()

```
void QCPLegend::setSelectedBrush (
    const QBrush & brush )
```

When the legend box is selected, this brush is used to draw the legend background instead of the normal brush set via [setBrush](#).

See also

[setSelectedParts](#), [setSelectableParts](#), [setSelectedBorderPen](#)

6.59.4.23 setSelectedFont()

```
void QCPLegend::setSelectedFont (
    const QFont & font )
```

Sets the default font that is used by legend items when they are selected.

This function will also set *font* on all already existing legend items.

See also

[setFont](#), [QCPAbstractLegendItem::setSelectedFont](#)

6.59.4.24 setSelectedIconBorderPen()

```
void QCPLegend::setSelectedIconBorderPen (
    const QPen & pen )
```

Sets the pen legend items will use to draw their icon borders, when they are selected.

See also

[setSelectedParts](#), [setSelectableParts](#), [setSelectedFont](#)

6.59.4.25 setSelectedParts()

```
void QCPLegend::setSelectedParts (
    const SelectableParts & selected )
```

Sets the selected state of the respective legend parts described by [SelectablePart](#). When a part is selected, it uses a different pen/font and brush. If some legend items are selected and *selected* doesn't contain [splItems](#), those items become deselected.

The entire selection mechanism is handled automatically when [QCustomPlot::setInteractions](#) contains [iSelectLegend](#). You only need to call this function when you wish to change the selection state manually.

This function can change the selection state of a part even when [setSelectableParts](#) was set to a value that actually excludes the part.

emits the [selectionChanged](#) signal when *selected* is different from the previous selection state.

Note that it doesn't make sense to set the selected state [splItems](#) here when it wasn't set before, because there's no way to specify which exact items to newly select. Do this by calling [QCPAbstractLegendItem::setSelected](#) directly on the legend item you wish to select.

See also

[SelectablePart](#), [setSelectableParts](#), [selectTest](#), [setSelectedBorderPen](#), [setSelectedIconBorderPen](#), [setSelectedBrush](#), [setSelectedFont](#)

6.59.4.26 setSelectedTextColor()

```
void QCPLegend::setSelectedTextColor (
    const QColor & color )
```

Sets the default text color that is used by legend items when they are selected.

This function will also set *color* on all already existing legend items.

See also

[setTextColor](#), [QCPAbstractLegendItem::setSelectedTextColor](#)

6.59.4.27 setTextColor()

```
void QCPLegend::setTextColor (
    const QColor & color )
```

Sets the default color of legend text. Legend items that draw text (e.g. the name of a graph) will use this color by default. However, a different colors can be specified on a per-item-basis by accessing the specific legend item.

This function will also set *color* on all already existing legend items.

See also

[QCPAbstractLegendItem::setTextColor](#)

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

- [/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h](#)
- [/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp](#)

6.60 QCPLineEnding Class Reference

Handles the different ending decorations for line-like items.

Public Types

- enum [EndingStyle](#) {
[esNone](#), [esFlatArrow](#), [esSpikeArrow](#), [esLineArrow](#),
[esDisc](#), [esSquare](#), [esDiamond](#), [esBar](#),
[esHalfBar](#), [esSkewedBar](#) }

Public Member Functions

- [QCPLineEnding](#) ()
- [QCPLineEnding](#) ([EndingStyle](#) style, double width=8, double length=10, bool inverted=false)
- [EndingStyle](#) **style** () const
- double **width** () const
- double **length** () const
- bool **inverted** () const
- void **setStyle** ([EndingStyle](#) style)
- void **setWidth** (double width)
- void **setLength** (double length)
- void **setInverted** (bool inverted)
- double **boundingDistance** () const
- double **realLength** () const
- void **draw** ([QCPPainter](#) *painter, const [QCPVector2D](#) &pos, const [QCPVector2D](#) &dir) const
- void **draw** ([QCPPainter](#) *painter, const [QCPVector2D](#) &pos, double angle) const

Protected Attributes

- [EndingStyle](#) **mStyle**
- double **mWidth**
- double **mLength**
- bool **mInverted**

6.60.1 Detailed Description

Handles the different ending decorations for line-like items.

For every ending a line-like item has, an instance of this class exists. For example, [QCPLItemLine](#) has two endings which can be set with [QCPLItemLine::setHead](#) and [QCPLItemLine::setTail](#).

The styles themselves are defined via the enum [QCPLineEnding::EndingStyle](#). Most decorations can be modified regarding width and length, see [setWidth](#) and [setLength](#). The direction of the ending decoration (e.g. direction an arrow is pointing) is controlled by the line-like item. For example, when both endings of a [QCPLItemLine](#) are set to be arrows, they will point to opposite directions, e.g. "outward". This can be changed by [setInverted](#), which would make the respective arrow point inward.

Note that due to the overloaded [QCPLineEnding](#) constructor, you may directly specify a [QCPLineEnding::EndingStyle](#) where actually a [QCPLineEnding](#) is expected, e.g.

6.60.2 Member Enumeration Documentation

6.60.2.1 EndingStyle

```
enum QCPLineEnding::EndingStyle
```

Defines the type of ending decoration for line-like items, e.g. an arrow.

The width and length of these decorations can be controlled with the functions [setWidth](#) and [setLength](#). Some decorations like [esDisc](#), [esSquare](#), [esDiamond](#) and [esBar](#) only support a width, the length property is ignored.

See also

[QCPLItemLine::setHead](#), [QCPLItemLine::setTail](#), [QCPLItemCurve::setHead](#), [QCPLItemCurve::setTail](#), [QCPAxis::setLowerEnding](#), [QCPAxis::setUpperEnding](#)

Enumerator

esNone	No ending decoration.
esFlatArrow	A filled arrow head with a straight/flat back (a triangle)
esSpikeArrow	A filled arrow head with an indented back.
esLineArrow	A non-filled arrow head with open back.
esDisc	A filled circle.
esSquare	A filled square.
esDiamond	A filled diamond (45 degrees rotated square)
esBar	A bar perpendicular to the line.
esHalfBar	A bar perpendicular to the line, pointing out to only one side (to which side can be changed with setInverted)
esSkewedBar	A bar that is skewed (skew controllable via setLength)

6.60.3 Constructor & Destructor Documentation

6.60.3.1 QCPLineEnding() [1/2]

```
QCPLineEnding::QCPLineEnding ( )
```

Creates a [QCPLineEnding](#) instance with default values (style [esNone](#)).

6.60.3.2 QCPLineEnding() [2/2]

```
QCPLineEnding::QCPLineEnding (
    QCPLineEnding::EndingStyle style,
    double width = 8,
    double length = 10,
    bool inverted = false )
```

Creates a [QCPLineEnding](#) instance with the specified values.

6.60.4 Member Function Documentation

6.60.4.1 `realLength()`

```
double QCPLineEnding::realLength ( ) const
```

Starting from the origin of this line ending (which is style specific), returns the length covered by the line ending symbol, in backward direction.

For example, the [esSpikeArrow](#) has a shorter real length than a [esFlatArrow](#), even if both have the same [setLength](#) value, because the spike arrow has an inward curved back, which reduces the length along its center axis (the drawing origin for arrows is at the tip).

This function is used for precise, style specific placement of line endings, for example in QCPAxes.

6.60.4.2 `setInverted()`

```
void QCPLineEnding::setInverted (
    bool inverted )
```

Sets whether the ending decoration shall be inverted. For example, an arrow decoration will point inward when *inverted* is set to true.

Note that also the *width* direction is inverted. For symmetrical ending styles like arrows or discs, this doesn't make a difference. However, asymmetric styles like [esHalfBar](#) are affected by it, which can be used to control to which side the half bar points to.

6.60.4.3 `setLength()`

```
void QCPLineEnding::setLength (
    double length )
```

Sets the length of the ending decoration, if the style supports it. On arrows, for example, the length defines the size in pointing direction.

See also

[setWidth](#)

6.60.4.4 `setStyle()`

```
void QCPLineEnding::setStyle (
    QCPLineEnding::EndingStyle style )
```

Sets the style of the ending decoration.

6.60.4.5 `setWidth()`

```
void QCPLineEnding::setWidth (
    double width )
```

Sets the width of the ending decoration, if the style supports it. On arrows, for example, the width defines the size perpendicular to the arrow's pointing direction.

See also

[setLength](#)

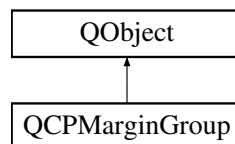
The documentation for this class was generated from the following files:

- `/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h`
- `/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp`

6.61 QCPMarginGroup Class Reference

A margin group allows synchronization of margin sides if working with multiple layout elements.

Inheritance diagram for QCPMarginGroup:



Public Member Functions

- `QCPMarginGroup (QCustomPlot *parentPlot)`
- `QList< QCPLayoutElement * > elements (QCP::MarginSide side) const`
- `bool isEmpty () const`
- `void clear ()`

Protected Member Functions

- `virtual int commonMargin (QCP::MarginSide side) const`
- `void addChild (QCP::MarginSide side, QCPLayoutElement *element)`
- `void removeChild (QCP::MarginSide side, QCPLayoutElement *element)`

Protected Attributes

- `QCustomPlot * mParentPlot`
- `QHash< QCP::MarginSide, QList< QCPLayoutElement * > > mChildren`

Friends

- class **QCPLayoutElement**

6.61.1 Detailed Description

A margin group allows synchronization of margin sides if working with multiple layout elements.

[QCPMarginGroup](#) allows you to tie a margin side of two or more layout elements together, such that they will all have the same size, based on the largest required margin in the group.

In certain situations it is desirable that margins at specific sides are synchronized across layout elements. For example, if one [QCPAxisRect](#) is below another one in a grid layout, it will provide a cleaner look to the user if the left and right margins of the two axis rects are of the same size. The left axis of the top axis rect will then be at the same horizontal position as the left axis of the lower axis rect, making them appear aligned. The same applies for the right axes. This is what [QCPMarginGroup](#) makes possible.

To add/remove a specific side of a layout element to/from a margin group, use the [QCPLayoutElement::setMarginGroup](#) method. To completely break apart the margin group, either call [clear](#), or just delete the margin group.

6.61.2 Example

First create a margin group:

Then set this group on the layout element sides:

Here, we've used the first two axis rects of the plot and synchronized their left margins with each other and their right margins with each other.

6.61.3 Constructor & Destructor Documentation

6.61.3.1 QCPMarginGroup()

```
QCPMarginGroup::QCPMarginGroup (
    QCustomPlot * parentPlot ) [explicit]
```

Creates a new [QCPMarginGroup](#) instance in *parentPlot*.

6.61.4 Member Function Documentation

6.61.4.1 clear()

```
void QCPMarginGroup::clear ( )
```

Clears this margin group. The synchronization of the margin sides that use this margin group is lifted and they will use their individual margin sizes again.

6.61.4.2 elements()

```
QList< QCPLayoutElement * > QCPMarginGroup::elements (
    QCP::MarginSide side ) const [inline]
```

Returns a list of all layout elements that have their margin *side* associated with this margin group.

6.61.4.3 isEmpty()

```
bool QCPMarginGroup::isEmpty ( ) const
```

Returns whether this margin group is empty. If this function returns true, no layout elements use this margin group to synchronize margin sides.

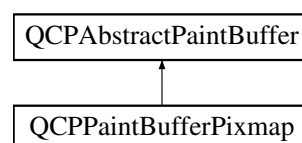
The documentation for this class was generated from the following files:

- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h↵
- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp↵

6.62 QCPPaintBufferPixmap Class Reference

A paint buffer based on QPixmap, using software raster rendering.

Inheritance diagram for QCPPaintBufferPixmap:



Public Member Functions

- [QCPPaintBufferPixmap](#) (const QSize &size, double devicePixelRatio)
- virtual [QCPPainter * startPainting](#) () Q_DECL_OVERRIDE
- virtual void [draw](#) ([QCPPainter *painter](#)) const Q_DECL_OVERRIDE
- void [clear](#) (const QColor &color) Q_DECL_OVERRIDE

Protected Member Functions

- virtual void [reallocateBuffer](#) () Q_DECL_OVERRIDE

Protected Attributes

- QPixmap **mBuffer**

6.62.1 Detailed Description

A paint buffer based on QPixmap, using software raster rendering.

This paint buffer is the default and fall-back paint buffer which uses software rendering and QPixmap as internal buffer. It is used if [QCustomPlot::setOpenGL](#) is false.

6.62.2 Constructor & Destructor Documentation

6.62.2.1 QCPPaintBufferPixmap()

```
QCPPaintBufferPixmap::QCPPaintBufferPixmap (
    const QSize & size,
    double devicePixelRatio ) [explicit]
```

Creates a pixmap paint buffer instancen with the specified *size* and *devicePixelRatio*, if applicable.

6.62.3 Member Function Documentation

6.62.3.1 clear()

```
void QCPPaintBufferPixmap::clear (
    const QColor & color ) [virtual]
```

Fills the entire buffer with the provided *color*. To have an empty transparent buffer, use the named color `Qt::transparent`.

This method must not be called if there is currently a painter (acquired with [startPainting](#)) active.

Implements [QCPAbstractPaintBuffer](#).

6.62.3.2 draw()

```
void QCPPaintBufferPixmap::draw (
    QCPPainter * painter ) const [virtual]
```

Draws the contents of this buffer with the provided *painter*. This is the method that is used to finally join all paint buffers and draw them onto the screen.

Implements [QCPAbstractPaintBuffer](#).

6.62.3.3 reallocateBuffer()

```
void QCPPaintBufferPixmap::reallocateBuffer ( ) [protected], [virtual]
```

Reallocates the internal buffer with the currently configured size ([setSize](#)) and device pixel ratio, if applicable ([setDevicePixelRatio](#)). It is called as soon as any of those properties are changed on this paint buffer.

Note

Subclasses of [QCPAbstractPaintBuffer](#) must call their reimplementation of this method in their constructor, to perform the first allocation (this can not be done by the base class because calling pure virtual methods in base class constructors is not possible).

Implements [QCPAbstractPaintBuffer](#).

6.62.3.4 startPainting()

```
QCPPainter * QCPPaintBufferPixmap::startPainting ( ) [virtual]
```

Returns a [QCPPainter](#) which is ready to draw to this buffer. The ownership and thus the responsibility to delete the painter after the painting operations are complete is given to the caller of this method.

Once you are done using the painter, delete the painter and call [donePainting](#).

While a painter generated with this method is active, you must not call [setSize](#), [setDevicePixelRatio](#) or [clear](#).

This method may return 0, if a painter couldn't be activated on the buffer. This usually indicates a problem with the respective painting backend.

Implements [QCPAbstractPaintBuffer](#).

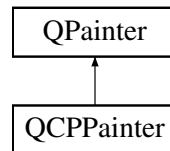
The documentation for this class was generated from the following files:

- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h
- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp

6.63 QCPPainter Class Reference

QPainter subclass used internally.

Inheritance diagram for QCPPainter:



Public Types

- enum [PainterMode](#) { [pmDefault](#) = 0x00, [pmVectorized](#) = 0x01, [pmNoCaching](#) = 0x02, [pmNonCosmetic](#) = 0x04 }

Public Member Functions

- [QCPPainter](#) ()
- [QCPPainter](#) (QPaintDevice *device)
- bool **antialiasing** () const
- PainterModes **modes** () const
- void [setAntialiasing](#) (bool enabled)
- void [setMode](#) ([PainterMode](#) mode, bool enabled=true)
- void [setModes](#) (PainterModes modes)
- bool [begin](#) (QPaintDevice *device)
- void [setPen](#) (const QPen &pen)
- void [setPen](#) (const QColor &color)
- void [setPen](#) (Qt::PenStyle penStyle)
- void [drawLine](#) (const QLineF &line)
- void **drawLine** (const QPointF &p1, const QPointF &p2)
- void [save](#) ()
- void [restore](#) ()
- void [makeNonCosmetic](#) ()

Protected Attributes

- PainterModes **mModes**
- bool **mIsAntialiasing**
- QStack< bool > **mAntialiasingStack**

6.63.1 Detailed Description

QPainter subclass used internally.

This QPainter subclass is used to provide some extended functionality e.g. for tweaking position consistency between antialiased and non-antialiased painting. Further it provides workarounds for QPainter quirks.

Warning

This class intentionally hides non-virtual functions of QPainter, e.g. [setPen](#), [save](#) and [restore](#). So while it is possible to pass a [QCPPainter](#) instance to a function that expects a QPainter pointer, some of the workarounds and tweaks will be unavailable to the function (because it will call the base class implementations of the functions actually hidden by [QCPPainter](#)).

6.63.2 Member Enumeration Documentation

6.63.2.1 PainterMode

enum [QCPPainter::PainterMode](#)

Defines special modes the painter can operate in. They disable or enable certain subsets of features/fixes/workarounds, depending on whether they are wanted on the respective output device.

Enumerator

<code>pmDefault</code>	<code>0x00</code> Default mode for painting on screen devices
<code>pmVectorized</code>	<code>0x01</code> Mode for vectorized painting (e.g. PDF export). For example, this prevents some antialiasing fixes.
<code>pmNoCaching</code>	<code>0x02</code> Mode for all sorts of exports (e.g. PNG, PDF,...). For example, this prevents using cached pixmap labels
<code>pmNonCosmetic</code>	<code>0x04</code> Turns pen widths 0 to 1, i.e. disables cosmetic pens. (A cosmetic pen is always drawn with width 1 pixel in the vector image/pdf viewer, independent of zoom.)

6.63.3 Constructor & Destructor Documentation

6.63.3.1 QCPPainter() [1/2]

```
QCPPainter::QCPPainter ( )
```

Creates a new [QCPPainter](#) instance and sets default values

6.63.3.2 QCPPainter() [2/2]

```
QCPPainter::QCPPainter (
    QPaintDevice * device ) [explicit]
```

Creates a new [QCPPainter](#) instance on the specified paint *device* and sets default values. Just like the analogous QPainter constructor, begins painting on *device* immediately.

Like [begin](#), this method sets QPainter::NonCosmeticDefaultPen in Qt versions before Qt5.

6.63.4 Member Function Documentation

6.63.4.1 begin()

```
bool QCPPainter::begin (
    QPaintDevice * device )
```

Sets the `QPainter::NonCosmeticDefaultPen` in Qt versions before Qt5 after beginning painting on *device*. This is necessary to get cosmetic pen consistency across Qt versions, because since Qt5, all pens are non-cosmetic by default, and in Qt4 this render hint must be set to get that behaviour.

The Constructor `QCPPainter(QPaintDevice *device)` which directly starts painting also sets the render hint as appropriate.

Note

this function hides the non-virtual base class implementation.

6.63.4.2 drawLine()

```
void QCPPainter::drawLine (
    const QLineF & line )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Works around a Qt bug introduced with Qt 4.8 which makes drawing `QLineF` unpredictable when antialiasing is disabled. Thus when antialiasing is disabled, it rounds the *line* to integer coordinates and then passes it to the original `drawLine`.

Note

this function hides the non-virtual base class implementation.

6.63.4.3 makeNonCosmetic()

```
void QCPPainter::makeNonCosmetic ( )
```

Changes the pen width to 1 if it currently is 0. This function is called in the `setPen` overrides when the `pmNonCosmetic` mode is set.

6.63.4.4 restore()

```
void QCPPainter::restore ( )
```

Restores the painter (see `QPainter::restore`). Since `QCPPainter` adds some new internal state to `QPainter`, the save/restore functions are reimplemented to also save/restore those members.

Note

this function hides the non-virtual base class implementation.

See also

[save](#)

6.63.4.5 save()

```
void QCPPainter::save ( )
```

Saves the painter (see `QPainter::save`). Since [QCPPainter](#) adds some new internal state to `QPainter`, the save/restore functions are reimplemented to also save/restore those members.

Note

this function hides the non-virtual base class implementation.

See also

[restore](#)

6.63.4.6 setAntialiasing()

```
void QCPPainter::setAntialiasing (
    bool enabled )
```

Sets whether painting uses antialiasing or not. Use this method instead of using `setRenderHint` with `QPainter::Antialiasing` directly, as it allows [QCPPainter](#) to regain pixel exactness between antialiased and non-antialiased painting (Since Qt < 5.0 uses slightly different coordinate systems for AA/Non-AA painting).

6.63.4.7 setMode()

```
void QCPPainter::setMode (
    QCPPainter::PainterMode mode,
    bool enabled = true )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Sets the mode of the painter. This controls whether the painter shall adjust its fixes/workarounds optimized for certain output devices.

6.63.4.8 setModes()

```
void QCPPainter::setModes (
    PainterModes modes )
```

Sets the mode of the painter. This controls whether the painter shall adjust its fixes/workarounds optimized for certain output devices.

6.63.4.9 setPen() [1/3]

```
void QCPPainter::setPen (
    const QPen & pen )
```

Sets the pen of the painter and applies certain fixes to it, depending on the mode of this [QCPPainter](#).

Note

this function hides the non-virtual base class implementation.

6.63.4.10 setPen() [2/3]

```
void QCPPainter::setPen (
    const QColor & color )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Sets the pen (by color) of the painter and applies certain fixes to it, depending on the mode of this [QCPPainter](#).

Note

this function hides the non-virtual base class implementation.

6.63.4.11 setPen() [3/3]

```
void QCPPainter::setPen (
    Qt::PenStyle penStyle )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Sets the pen (by style) of the painter and applies certain fixes to it, depending on the mode of this [QCPPainter](#).

Note

this function hides the non-virtual base class implementation.

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

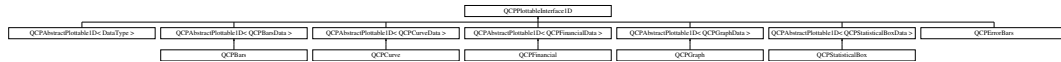
- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.↔h
- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.↔cpp

6.64 QCPPlottableInterface1D Class Reference

Defines an abstract interface for one-dimensional plottables.

```
#include <qcustomplot.h>
```

Inheritance diagram for QCPPlottableInterface1D:



Public Member Functions

- virtual int [dataCount](#) () const =0
- virtual double [dataMainKey](#) (int index) const =0
- virtual double [dataSortKey](#) (int index) const =0
- virtual double [dataMainValue](#) (int index) const =0
- virtual [QCPRange dataValueRange](#) (int index) const =0
- virtual [QPointF dataPixelPosition](#) (int index) const =0
- virtual bool [sortKeysMainKey](#) () const =0
- virtual [QCPDataSelection selectTestRect](#) (const [QRectF](#) &rect, bool onlySelectable) const =0
- virtual int [findBegin](#) (double sortKey, bool expandedRange=true) const =0
- virtual int [findEnd](#) (double sortKey, bool expandedRange=true) const =0

6.64.1 Detailed Description

Defines an abstract interface for one-dimensional plottables.

This class contains only pure virtual methods which define a common interface to the data of one-dimensional plottables.

For example, it is implemented by the template class [QCPAbstractPlottable1D](#) (the preferred base class for one-dimensional plottables). So if you use that template class as base class of your one-dimensional plottable, you won't have to care about implementing the 1d interface yourself.

If your plottable doesn't derive from [QCPAbstractPlottable1D](#) but still wants to provide a 1d interface (e.g. like [QCPErrorBars](#) does), you should inherit from both [QCPAbstractPlottable](#) and [QCPPlottableInterface1D](#) and accordingly reimplement the pure virtual methods of the 1d interface, matching your data container. Also, reimplement [QCPAbstractPlottable::interface1D](#) to return the `this` pointer.

If you have a [QCPAbstractPlottable](#) pointer, you can check whether it implements this interface by calling [QCPAbstractPlottable::interface1D](#) and testing it for a non-zero return value. If it indeed implements this interface, you may use it to access the plottable's data without needing to know the exact type of the plottable or its data point type.

6.64.2 Member Function Documentation

6.64.2.1 dataCount()

```
int QCPPlottableInterface1D::dataCount ( ) const [pure virtual]
```

Returns the number of data points of the plottable.

Implemented in [QCPErrorBars](#), [QCPAbstractPlottable1D< DataType >](#), [QCPAbstractPlottable1D< QCPFinancialData >](#), [QCPAbstractPlottable1D< QCPStatisticalBoxData >](#), [QCPAbstractPlottable1D< QCPGraphData >](#), [QCPAbstractPlottable1D< QCPBarsData >](#), and [QCPAbstractPlottable1D< QCPCurveData >](#).

6.64.2.2 dataMainKey()

```
double QCPPlottableInterface1D::dataMainKey (
    int index ) const [pure virtual]
```

Returns the main key of the data point at the given *index*.

What the main key is, is defined by the plottable's data type. See the [QCPDataContainer DataType](#) documentation for details about this naming convention.

Implemented in [QCPErrorBars](#), [QCPAbstractPlottable1D< DataType >](#), [QCPAbstractPlottable1D< QCPFinancialData >](#), [QCPAbstractPlottable1D< QCPStatisticalBoxData >](#), [QCPAbstractPlottable1D< QCPGraphData >](#), [QCPAbstractPlottable1D< QCPBarsData >](#), and [QCPAbstractPlottable1D< QCPCurveData >](#).

6.64.2.3 dataMainValue()

```
double QCPPlottableInterface1D::dataMainValue (
    int index ) const [pure virtual]
```

Returns the main value of the data point at the given *index*.

What the main value is, is defined by the plottable's data type. See the [QCPDataContainer DataType](#) documentation for details about this naming convention.

Implemented in [QCPErrorBars](#), [QCPAbstractPlottable1D< DataType >](#), [QCPAbstractPlottable1D< QCPFinancialData >](#), [QCPAbstractPlottable1D< QCPStatisticalBoxData >](#), [QCPAbstractPlottable1D< QCPGraphData >](#), [QCPAbstractPlottable1D< QCPBarsData >](#), and [QCPAbstractPlottable1D< QCPCurveData >](#).

6.64.2.4 dataPixelPosition()

```
QPointF QCPPlottableInterface1D::dataPixelPosition (
    int index ) const [pure virtual]
```

Returns the pixel position on the widget surface at which the data point at the given *index* appears.

Usually this corresponds to the point of [dataMainKey/dataMainValue](#), in pixel coordinates. However, depending on the plottable, this might be a different apparent position than just a coord-to-pixel transform of those values. For example, [QCPBars](#) apparent data values can be shifted depending on their stacking, bar grouping or configured base value.

Implemented in [QCPErrorBars](#), [QCPBars](#), [QCPAbstractPlottable1D< DataType >](#), [QCPAbstractPlottable1D< QCPFinancialData >](#), [QCPAbstractPlottable1D< QCPStatisticalBoxData >](#), [QCPAbstractPlottable1D< QCPGraphData >](#), [QCPAbstractPlottable1D< QCPBarsData >](#), and [QCPAbstractPlottable1D< QCPCurveData >](#).

6.64.2.5 dataSortKey()

```
double QCPPlottableInterface1D::dataSortKey (
    int index ) const [pure virtual]
```

Returns the sort key of the data point at the given *index*.

What the sort key is, is defined by the plottable's data type. See the [QCPDataContainer DataType](#) documentation for details about this naming convention.

Implemented in [QCPErrorBars](#), [QCPAbstractPlottable1D< DataType >](#), [QCPAbstractPlottable1D< QCPFinancialData >](#), [QCPAbstractPlottable1D< QCPStatisticalBoxData >](#), [QCPAbstractPlottable1D< QCPGraphData >](#), [QCPAbstractPlottable1D< QCPBarsData >](#), and [QCPAbstractPlottable1D< QCPCurveData >](#).

6.64.2.6 dataValueRange()

```
QCPRange QCPPlottableInterface1D::dataValueRange (
    int index ) const [pure virtual]
```

Returns the value range of the data point at the given *index*.

What the value range is, is defined by the plottable's data type. See the [QCPDataContainer DataType](#) documentation for details about this naming convention.

Implemented in [QCPErrorBars](#), [QCPAbstractPlottable1D< DataType >](#), [QCPAbstractPlottable1D< QCPFinancialData >](#), [QCPAbstractPlottable1D< QCPStatisticalBoxData >](#), [QCPAbstractPlottable1D< QCPGraphData >](#), [QCPAbstractPlottable1D< QCPBarsData >](#), and [QCPAbstractPlottable1D< QCPCurveData >](#).

6.64.2.7 findBegin()

```
int QCPPlottableInterface1D::findBegin (
    double sortKey,
    bool expandedRange = true ) const [pure virtual]
```

Returns the index of the data point with a (sort-)key that is equal to, just below, or just above *sortKey*. If *expandedRange* is true, the data point just below *sortKey* will be considered, otherwise the one just above.

This can be used in conjunction with [findEnd](#) to iterate over data points within a given key range, including or excluding the bounding data points that are just beyond the specified range.

If *expandedRange* is true but there are no data points below *sortKey*, 0 is returned.

If the container is empty, returns 0 (in that case, [findEnd](#) will also return 0, so a loop using these methods will not iterate over the index 0).

See also

[findEnd](#), [QCPDataContainer::findBegin](#)

Implemented in [QCPErrorBars](#), [QCPAbstractPlottable1D< DataType >](#), [QCPAbstractPlottable1D< QCPFinancialData >](#), [QCPAbstractPlottable1D< QCPStatisticalBoxData >](#), [QCPAbstractPlottable1D< QCPGraphData >](#), [QCPAbstractPlottable1D< QCPBarsData >](#), and [QCPAbstractPlottable1D< QCPCurveData >](#).

6.64.2.8 findEnd()

```
int QCPPlottableInterface1D::findEnd (
    double sortKey,
    bool expandedRange = true ) const [pure virtual]
```

Returns the index one after the data point with a (sort-)key that is equal to, just above, or just below *sortKey*. If *expandedRange* is true, the data point just above *sortKey* will be considered, otherwise the one just below.

This can be used in conjunction with [findBegin](#) to iterate over data points within a given key range, including the bounding data points that are just below and above the specified range.

If *expandedRange* is true but there are no data points above *sortKey*, the index just above the highest data point is returned.

If the container is empty, returns 0.

See also

[findBegin](#), [QCPDataContainer::findEnd](#)

Implemented in [QCPErrorBars](#), [QCPAbstractPlottable1D< DataType >](#), [QCPAbstractPlottable1D< QCPFinancialData >](#), [QCPAbstractPlottable1D< QCPStatisticalBoxData >](#), [QCPAbstractPlottable1D< QCPGraphData >](#), [QCPAbstractPlottable1D< QCPBarsData >](#), and [QCPAbstractPlottable1D< QCPCurveData >](#).

6.64.2.9 selectTestRect()

```
QCPDataSelection QCPPlottableInterface1D::selectTestRect (
    const QRectF & rect,
    bool onlySelectable ) const [pure virtual]
```

Returns a data selection containing all the data points of this plottable which are contained (or hit by) *rect*. This is used mainly in the selection rect interaction for data selection (data selection mechanism).

If *onlySelectable* is true, an empty [QCPDataSelection](#) is returned if this plottable is not selectable (i.e. if [QCPAbstractPlottable::setSelectable](#) is [QCP::stNone](#)).

Note

rect must be a normalized rect (positive or zero width and height). This is especially important when using the rect of [QCPSelectionRect::accepted](#), which is not necessarily normalized. Use `QRect::normalized()` when passing a rect which might not be normalized.

Implemented in [QCPErrorBars](#), [QCPFinancial](#), [QCPStatisticalBox](#), [QCPBars](#), [QCPAbstractPlottable1D< DataType >](#), [QCPAbstractPlottable1D< QCPFinancialData >](#), [QCPAbstractPlottable1D< QCPStatisticalBoxData >](#), [QCPAbstractPlottable1D< QCPGraphData >](#), [QCPAbstractPlottable1D< QCPBarsData >](#), and [QCPAbstractPlottable1D< QCPCurveData >](#).

6.64.2.10 sortKeyIsMainKey()

```
bool QCPPlottableInterface1D::sortKeyIsMainKey ( ) const [pure virtual]
```

Returns whether the sort key ([dataSortKey](#)) is identical to the main key ([dataMainKey](#)).

What the sort and main keys are, is defined by the plottable's data type. See the [QCPDataContainer DataType](#) documentation for details about this naming convention.

Implemented in [QCPErrorBars](#), [QCPAbstractPlottable1D< DataType >](#), [QCPAbstractPlottable1D< QCPFinancialData >](#), [QCPAbstractPlottable1D< QCPStatisticalBoxData >](#), [QCPAbstractPlottable1D< QCPGraphData >](#), [QCPAbstractPlottable1D< QCPBarsData >](#), and [QCPAbstractPlottable1D< QCPCurveData >](#).

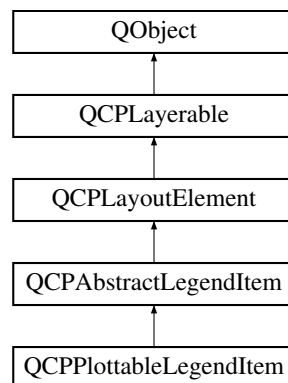
The documentation for this class was generated from the following file:

- `/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h`

6.65 QCPPlottableLegendItem Class Reference

A legend item representing a plottable with an icon and the plottable name.

Inheritance diagram for QCPPlottableLegendItem:



Public Member Functions

- [QCPPlottableLegendItem](#) ([QCPLegend](#) *parent, [QCPAbstractPlottable](#) *plottable)
- [QCPAbstractPlottable](#) * **plottable** ()

Protected Member Functions

- virtual void **draw** ([QCPPainter](#) *painter) Q_DECL_OVERRIDE
- virtual QSize **minimumSizeHint** () const Q_DECL_OVERRIDE
- QPen **getIconBorderPen** () const
- QColor **getTextColor** () const
- QFont **getFont** () const

Protected Attributes

- [QCPAbstractPlottable](#) * **mPlottable**

Additional Inherited Members

6.65.1 Detailed Description

A legend item representing a plottable with an icon and the plottable name.

This is the standard legend item for plottables. It displays an icon of the plottable next to the plottable name. The icon is drawn by the respective plottable itself ([QCPAbstractPlottable::drawLegendIcon](#)), and tries to give an intuitive symbol for the plottable. For example, the [QCPGraph](#) draws a centered horizontal line and/or a single scatter point in the middle.

Legend items of this type are always associated with one plottable (retrievable via the `plottable()` function and settable with the constructor). You may change the font of the plottable name with [setFont](#). Icon padding and border pen is taken from the parent [QCPLegend](#), see [QCPLegend::setIconBorderPen](#) and [QCPLegend::setIcon↵TextPadding](#).

The function [QCPAbstractPlottable::addToLegend/QCPAbstractPlottable::removeFromLegend](#) creates/removes legend items of this type in the default implementation. However, these functions may be reimplemented such that a different kind of legend item (e.g. a direct subclass of [QCPAbstractLegendItem](#)) is used for that plottable.

Since [QCPLegend](#) is based on [QCPLayoutGrid](#), a legend item itself is just a subclass of [QCPLayoutElement](#). While it could be added to a legend (or any other layout) via the normal layout interface, [QCPLegend](#) has specialized functions for handling legend items conveniently, see the documentation of [QCPLegend](#).

6.65.2 Constructor & Destructor Documentation

6.65.2.1 QCPPlottableLegendItem()

```
QCPPlottableLegendItem::QCPPlottableLegendItem (
    QCPLegend * parent,
    QCPAbstractPlottable * plottable )
```

Creates a new legend item associated with *plottable*.

Once it's created, it can be added to the legend via [QCPLegend::addItem](#).

A more convenient way of adding/removing a plottable to/from the legend is via the functions [QCPAbstract↵Plottable::addToLegend](#) and [QCPAbstractPlottable::removeFromLegend](#).

6.65.3 Member Function Documentation

6.65.3.1 `minimumSizeHint()`

```
QSize QCPlottableLegendItem::minimumSizeHint ( ) const [protected], [virtual]
```

Returns the minimum size this layout element (the inner [rect](#)) may be compressed to.

if a minimum size ([setMinimumSize](#)) was not set manually, parent layouts consult this function to determine the minimum allowed size of this layout element. (A manual minimum size is considered set if it is non-zero.)

Reimplemented from [QCPLayoutElement](#).

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

- [/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h](#)↵
- [/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp](#)↵

6.66 QCPRange Class Reference

Represents the range an axis is encompassing.

Public Member Functions

- [QCPRange](#) ()
- [QCPRange](#) (double lower, double upper)
- bool **operator==** (const [QCPRange](#) &other) const
- bool **operator!=** (const [QCPRange](#) &other) const
- [QCPRange](#) & **operator+=** (const double &value)
- [QCPRange](#) & **operator-=** (const double &value)
- [QCPRange](#) & **operator*=** (const double &value)
- [QCPRange](#) & **operator/=** (const double &value)
- double **size** () const
- double **center** () const
- void **normalize** ()
- void **expand** (const [QCPRange](#) &otherRange)
- void **expand** (double includeCoord)
- [QCPRange](#) **expanded** (const [QCPRange](#) &otherRange) const
- [QCPRange](#) **expanded** (double includeCoord) const
- [QCPRange](#) **bounded** (double lowerBound, double upperBound) const
- [QCPRange](#) **sanitizedForLogScale** () const
- [QCPRange](#) **sanitizedForLinScale** () const
- bool **contains** (double value) const

Static Public Member Functions

- static bool **validRange** (double lower, double upper)
- static bool **validRange** (const [QCPRange](#) &range)

Public Attributes

- double **lower**
- double **upper**

Static Public Attributes

- static const double **minRange** = 1e-280
- static const double **maxRange** = 1e250

Friends

- const [QCPRange](#) operator+ (const [QCPRange](#) &, double)
- const [QCPRange](#) operator+ (double, const [QCPRange](#) &)
- const [QCPRange](#) operator- (const [QCPRange](#) &range, double value)
- const [QCPRange](#) operator* (const [QCPRange](#) &range, double value)
- const [QCPRange](#) operator* (double value, const [QCPRange](#) &range)
- const [QCPRange](#) operator/ (const [QCPRange](#) &range, double value)

Related Functions

(Note that these are not member functions.)

- QDebug [operator<<](#) (QDebug d, const [QCPRange](#) &range)

6.66.1 Detailed Description

Represents the range an axis is encompassing.

contains a *lower* and *upper* double value and provides convenience input, output and modification functions.

See also

[QCPAxis::setRange](#)

6.66.2 Constructor & Destructor Documentation

6.66.2.1 [QCPRange\(\)](#) [1/2]

```
QCPRange::QCPRange ( )
```

Constructs a range with *lower* and *upper* set to zero.

6.66.2.2 QCPRange() [2/2]

```
QCPRange::QCPRange (
    double lower,
    double upper )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Constructs a range with the specified *lower* and *upper* values.

The resulting range will be normalized (see [normalize](#)), so if *lower* is not numerically smaller than *upper*, they will be swapped.

6.66.3 Member Function Documentation

6.66.3.1 bounded()

```
QCPRange QCPRange::bounded (
    double lowerBound,
    double upperBound ) const
```

Returns this range, possibly modified to not exceed the bounds provided as *lowerBound* and *upperBound*. If possible, the size of the current range is preserved in the process.

If the range shall only be bounded at the lower side, you can set *upperBound* to [QCPRange::maxRange](#). If it shall only be bounded at the upper side, set *lowerBound* to [-QCPRange::maxRange](#).

6.66.3.2 center()

```
double QCPRange::center ( ) const [inline]
```

Returns the center of the range, i.e. $(upper+lower)*0.5$

6.66.3.3 contains()

```
bool QCPRange::contains (
    double value ) const [inline]
```

Returns true when *value* lies within or exactly on the borders of the range.

6.66.3.4 `expand()` [1/2]

```
void QCPRange::expand (
    const QCPRange & otherRange )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Expands this range such that *otherRange* is contained in the new range. It is assumed that both this range and *otherRange* are normalized (see [normalize](#)).

If this range contains NaN as lower or upper bound, it will be replaced by the respective bound of *otherRange*.

If *otherRange* is already inside the current range, this function does nothing.

See also

[expanded](#)

6.66.3.5 `expand()` [2/2]

```
void QCPRange::expand (
    double includeCoord )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Expands this range such that *includeCoord* is contained in the new range. It is assumed that this range is normalized (see [normalize](#)).

If this range contains NaN as lower or upper bound, the respective bound will be set to *includeCoord*.

If *includeCoord* is already inside the current range, this function does nothing.

See also

[expand](#)

6.66.3.6 `expanded()` [1/2]

```
QCPRange QCPRange::expanded (
    const QCPRange & otherRange ) const
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Returns an expanded range that contains this and *otherRange*. It is assumed that both this range and *otherRange* are normalized (see [normalize](#)).

If this range contains NaN as lower or upper bound, the returned range's bound will be taken from *otherRange*.

See also

[expand](#)

6.66.3.7 expanded() [2/2]

```
QCPRange QCPRange::expanded (
    double includeCoord ) const
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Returns an expanded range that includes the specified *includeCoord*. It is assumed that this range is normalized (see [normalize](#)).

If this range contains NaN as lower or upper bound, the returned range's bound will be set to *includeCoord*.

See also

[expand](#)

6.66.3.8 normalize()

```
void QCPRange::normalize ( ) [inline]
```

Makes sure *lower* is numerically smaller than *upper*. If this is not the case, the values are swapped.

6.66.3.9 operator*=()

```
QCPRange & QCPRange::operator*= (
    const double & value ) [inline]
```

Multiplies both boundaries of the range by *value*.

6.66.3.10 operator+=()

```
QCPRange & QCPRange::operator+= (
    const double & value ) [inline]
```

Adds *value* to both boundaries of the range.

6.66.3.11 operator-=()

```
QCPRange & QCPRange::operator-= (
    const double & value ) [inline]
```

Subtracts *value* from both boundaries of the range.

6.66.3.12 operator/=()

```
QCPRange & QCPRange::operator/= (
    const double & value ) [inline]
```

Divides both boundaries of the range by *value*.

6.66.3.13 sanitizedForLinScale()

```
QCPRange QCPRange::sanitizedForLinScale ( ) const
```

Returns a sanitized version of the range. Sanitized means for linear scales, that *lower* will always be numerically smaller (or equal) to *upper*.

6.66.3.14 sanitizedForLogScale()

```
QCPRange QCPRange::sanitizedForLogScale ( ) const
```

Returns a sanitized version of the range. Sanitized means for logarithmic scales, that the range won't span the positive and negative sign domain, i.e. contain zero. Further *lower* will always be numerically smaller (or equal) to *upper*.

If the original range does span positive and negative sign domains or contains zero, the returned range will try to approximate the original range as good as possible. If the positive interval of the original range is wider than the negative interval, the returned range will only contain the positive interval, with lower bound set to *rangeFac* or *rangeFac * upper*, whichever is closer to zero. Same procedure is used if the negative interval is wider than the positive interval, this time by changing the *upper* bound.

6.66.3.15 size()

```
double QCPRange::size ( ) const [inline]
```

Returns the size of the range, i.e. *upper-lower*

6.66.3.16 validRange() [1/2]

```
bool QCPRange::validRange (
    double lower,
    double upper ) [static]
```

Checks, whether the specified range is within valid bounds, which are defined as `QCPRange::maxRange` and `QCPRange::minRange`. A valid range means:

- range bounds within `-maxRange` and `maxRange`
- range size above `minRange`
- range size below `maxRange`

6.66.3.17 validRange() [2/2]

```
bool QCPRange::validRange (
    const QCPRange & range ) [static]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts. Checks, whether the specified range is within valid bounds, which are defined as [QCPRange::maxRange](#) and [QCPRange::minRange](#). A valid range means:

- range bounds within -maxRange and maxRange
- range size above minRange
- range size below maxRange

6.66.4 Friends And Related Function Documentation

6.66.4.1 operator* [1/2]

```
const QCPRange operator* (
    const QCPRange & range,
    double value ) [friend]
```

Multiplies both boundaries of the range by *value*.

6.66.4.2 operator* [2/2]

```
const QCPRange operator* (
    double value,
    const QCPRange & range ) [friend]
```

Multiplies both boundaries of the range by *value*.

6.66.4.3 operator+ [1/2]

```
const QCPRange operator+ (
    const QCPRange & range,
    double value ) [friend]
```

Adds *value* to both boundaries of the range.

6.66.4.4 operator+ [2/2]

```
const QCPRange operator+ (
    double value,
    const QCPRange & range ) [friend]
```

Adds *value* to both boundaries of the range.

6.66.4.5 operator-

```
const QCPRange operator- (
    const QCPRange & range,
    double value ) [friend]
```

Subtracts *value* from both boundaries of the range.

6.66.4.6 operator/

```
const QCPRange operator/ (
    const QCPRange & range,
    double value ) [friend]
```

Divides both boundaries of the range by *value*.

6.66.4.7 operator<<()

```
QDebug operator<< (
    QDebug d,
    const QCPRange & range ) [related]
```

Prints *range* in a human readable format to the qDebug output.

6.66.5 Member Data Documentation

6.66.5.1 maxRange

```
const double QCPRange::maxRange = 1e250 [static]
```

Maximum values (negative and positive) the range will accept in range-changing functions. Larger absolute values would cause errors due to the 11-bit exponent of double precision numbers, corresponding to a maximum magnitude of roughly 1e308.

Warning

Do not use this constant to indicate "arbitrarily large" values in plotting logic (as values that will appear in the plot)! It is intended only as a bound to compare against, e.g. to prevent axis ranges from obtaining overflowing ranges.

See also

[validRange](#), [minRange](#)

6.66.5.2 minRange

```
const double QCPRange::minRange = 1e-280 [static]
```

Minimum range size (*upper* - *lower*) the range changing functions will accept. Smaller intervals would cause errors due to the 11-bit exponent of double precision numbers, corresponding to a minimum magnitude of roughly 1e-308.

Warning

Do not use this constant to indicate "arbitrarily small" values in plotting logic (as values that will appear in the plot)! It is intended only as a bound to compare against, e.g. to prevent axis ranges from obtaining underflowing ranges.

See also

[validRange](#), [maxRange](#)

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

- [/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h](#)↔
- [/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp](#)↔

6.67 QCPScatterStyle Class Reference

Represents the visual appearance of scatter points.

Public Types

- enum [ScatterProperty](#) {
 [spNone](#) = 0x00, [spPen](#) = 0x01, [spBrush](#) = 0x02, [spSize](#) = 0x04,
 [spShape](#) = 0x08, [spAll](#) = 0xFF }
- enum [ScatterShape](#) {
 [ssNone](#), [ssDot](#), [ssCross](#), [ssPlus](#),
 [ssCircle](#), [ssDisc](#), [ssSquare](#), [ssDiamond](#),
 [ssStar](#), [ssTriangle](#), [ssTriangleInverted](#), [ssCrossSquare](#),
 [ssPlusSquare](#), [ssCrossCircle](#), [ssPlusCircle](#), [ssPeace](#),
 [ssPixmap](#), [ssCustom](#) }

Public Member Functions

- [QCPScatterStyle](#) ()
- [QCPScatterStyle](#) ([ScatterShape](#) shape, double size=6)
- [QCPScatterStyle](#) ([ScatterShape](#) shape, const QColor &color, double size)
- [QCPScatterStyle](#) ([ScatterShape](#) shape, const QColor &color, const QColor &fill, double size)
- [QCPScatterStyle](#) ([ScatterShape](#) shape, const QPen &pen, const QBrush &brush, double size)
- [QCPScatterStyle](#) (const QPixmap &pixmap)
- [QCPScatterStyle](#) (const QPainterPath &customPath, const QPen &pen, const QBrush &brush=Qt::NoBrush, double size=6)
- double **size** () const
- [ScatterShape](#) **shape** () const
- QPen **pen** () const
- QBrush **brush** () const
- QPixmap **pixmap** () const
- QPainterPath **customPath** () const
- void [setFromOther](#) (const [QCPScatterStyle](#) &other, ScatterProperties properties)
- void [setSize](#) (double size)
- void [setShape](#) ([ScatterShape](#) shape)
- void [setPen](#) (const QPen &pen)
- void [setBrush](#) (const QBrush &brush)
- void [setPixmap](#) (const QPixmap &pixmap)
- void [setCustomPath](#) (const QPainterPath &customPath)
- bool [isNone](#) () const
- bool [isPenDefined](#) () const
- void [undefinePen](#) ()
- void [applyTo](#) ([QCPPainter](#) *painter, const QPen &defaultPen) const
- void [drawShape](#) ([QCPPainter](#) *painter, const QPointF &pos) const
- void [drawShape](#) ([QCPPainter](#) *painter, double x, double y) const

Protected Attributes

- double **mSize**
- [ScatterShape](#) **mShape**
- QPen **mPen**
- QBrush **mBrush**
- QPixmap **mPixmap**
- QPainterPath **mCustomPath**
- bool **mPenDefined**

6.67.1 Detailed Description

Represents the visual appearance of scatter points.

This class holds information about shape, color and size of scatter points. In plottables like [QCPGraph](#) it is used to store how scatter points shall be drawn. For example, [QCPGraph::setScatterStyle](#) takes a [QCPScatterStyle](#) instance.

A scatter style consists of a shape ([setShape](#)), a line color ([setPen](#)) and possibly a fill ([setBrush](#)), if the shape provides a fillable area. Further, the size of the shape can be controlled with [setSize](#).

6.67.2 Specifying a scatter style

You can set all these configurations either by calling the respective functions on an instance:

Or you can use one of the various constructors that take different parameter combinations, making it easy to specify a scatter style in a single call, like so:

6.67.3 Leaving the color/pen up to the plottable

There are two constructors which leave the pen undefined: [QCPScatterStyle\(\)](#) and [QCPScatterStyle\(ScatterShape shape, double size\)](#). If those constructors are used, a call to [isPenDefined](#) will return false. It leads to scatter points that inherit the pen from the plottable that uses the scatter style. Thus, if such a scatter style is passed to [QCPGraph](#), the line color of the graph ([QCPGraph::setPen](#)) will be used by the scatter points. This makes it very convenient to set up typical scatter settings:

Notice that it wasn't even necessary to explicitly call a [QCPScatterStyle](#) constructor. This works because [QCPScatterStyle](#) provides a constructor that can transform a [ScatterShape](#) directly into a [QCPScatterStyle](#) instance (that's the [QCPScatterStyle\(ScatterShape shape, double size\)](#) constructor with a default for *size*). In those cases, C++ allows directly supplying a [ScatterShape](#), where actually a [QCPScatterStyle](#) is expected.

6.67.4 Custom shapes and pixmaps

[QCPScatterStyle](#) supports drawing custom shapes and arbitrary pixmaps as scatter points.

For custom shapes, you can provide a [QPainterPath](#) with the desired shape to the [setCustomPath](#) function or call the constructor that takes a painter path. The scatter shape will automatically be set to [ssCustom](#).

For pixmaps, you call [setPixmap](#) with the desired [QPixmap](#). Alternatively you can use the constructor that takes a [QPixmap](#). The scatter shape will automatically be set to [ssPixmap](#). Note that [setSize](#) does not influence the appearance of the pixmap.

6.67.5 Member Enumeration Documentation

6.67.5.1 ScatterProperty

```
enum QCPScatterStyle::ScatterProperty
```

Represents the various properties of a scatter style instance. For example, this enum is used to specify which properties of [QCPSelectionDecorator::setScatterStyle](#) will be used when highlighting selected data points.

Specific scatter properties can be transferred between [QCPScatterStyle](#) instances via [setFromOther](#).

Enumerator

spNone	0x00 None
spPen	0x01 The pen property, see setPen
spBrush	0x02 The brush property, see setBrush
spSize	0x04 The size property, see setSize
spShape	0x08 The shape property, see setShape
spAll	0xFF All properties

6.67.5.2 ScatterShape

```
enum QCPScatterStyle::ScatterShape
```

Defines the shape used for scatter points.

On plottables/items that draw scatters, the sizes of these visualizations (with exception of [ssDot](#) and [ssPixmap](#)) can be controlled with the [setSize](#) function. Scatters are drawn with the pen and brush specified with [setPen](#) and [setBrush](#).

Enumerator

ssNone	no scatter symbols are drawn (e.g. in QCPGraph , data only represented with lines)
ssDot	{ssDot.png} a single pixel (use ssDisc or ssCircle if you want a round shape with a certain radius)
ssCross	{ssCross.png} a cross
ssPlus	{ssPlus.png} a plus
ssCircle	{ssCircle.png} a circle
ssDisc	{ssDisc.png} a circle which is filled with the pen's color (not the brush as with ssCircle)
ssSquare	{ssSquare.png} a square
ssDiamond	{ssDiamond.png} a diamond
ssStar	{ssStar.png} a star with eight arms, i.e. a combination of cross and plus
ssTriangle	{ssTriangle.png} an equilateral triangle, standing on baseline
ssTriangleInverted	{ssTriangleInverted.png} an equilateral triangle, standing on corner
ssCrossSquare	{ssCrossSquare.png} a square with a cross inside
ssPlusSquare	{ssPlusSquare.png} a square with a plus inside
ssCrossCircle	{ssCrossCircle.png} a circle with a cross inside
ssPlusCircle	{ssPlusCircle.png} a circle with a plus inside
ssPeace	{ssPeace.png} a circle, with one vertical and two downward diagonal lines
ssPixmap	a custom pixmap specified by setPixmap , centered on the data point coordinates
ssCustom	custom painter operations are performed per scatter (As QPainterPath, see setCustomPath)

6.67.6 Constructor & Destructor Documentation

6.67.6.1 QCPScatterStyle() [1/7]

```
QCPScatterStyle::QCPScatterStyle ( )
```

Creates a new [QCPScatterStyle](#) instance with size set to 6. No shape, pen or brush is defined.

Since the pen is undefined ([isPenDefined](#) returns false), the scatter color will be inherited from the plottable that uses this scatter style.

6.67.6.2 QCPScatterStyle() [2/7]

```
QCPScatterStyle::QCPScatterStyle (
    ScatterShape shape,
    double size = 6 )
```

Creates a new [QCPScatterStyle](#) instance with shape set to *shape* and size to *size*. No pen or brush is defined.

Since the pen is undefined ([isPenDefined](#) returns false), the scatter color will be inherited from the plottable that uses this scatter style.

6.67.6.3 QCPScatterStyle() [3/7]

```
QCPScatterStyle::QCPScatterStyle (
    ScatterShape shape,
    const QColor & color,
    double size )
```

Creates a new [QCPScatterStyle](#) instance with shape set to *shape*, the pen color set to *color*, and size to *size*. No brush is defined, i.e. the scatter point will not be filled.

6.67.6.4 QCPScatterStyle() [4/7]

```
QCPScatterStyle::QCPScatterStyle (
    ScatterShape shape,
    const QColor & color,
    const QColor & fill,
    double size )
```

Creates a new [QCPScatterStyle](#) instance with shape set to *shape*, the pen color set to *color*, the brush color to *fill* (with a solid pattern), and size to *size*.

6.67.6.5 QCPScatterStyle() [5/7]

```
QCPScatterStyle::QCPScatterStyle (
    ScatterShape shape,
    const QPen & pen,
    const QBrush & brush,
    double size )
```

Creates a new [QCPScatterStyle](#) instance with shape set to *shape*, the pen set to *pen*, the brush to *brush*, and size to *size*.

Warning

In some cases it might be tempting to directly use a pen style like `Qt::NoPen` as *pen* and a color like `Qt::blue` as *brush*. Notice however, that the corresponding call `QCPScatterStyle(QCPScatterShape::ssCircle, Qt::NoPen, Qt::blue, 5)` doesn't necessarily lead C++ to use this constructor in some cases, but might mistake `Qt::NoPen` for a `QColor` and use the `QCPScatterStyle(ScatterShape shape, const QColor &color, const QColor &fill, double size)` constructor instead (which will lead to an unexpected look of the scatter points). To prevent this, be more explicit with the parameter types. For example, use `QBrush(Qt::blue)` instead of just `Qt::blue`, to clearly point out to the compiler that this constructor is wanted.

6.67.6.6 QCPScatterStyle() [6/7]

```
QCPScatterStyle::QCPScatterStyle (
    const QPixmap & pixmap )
```

Creates a new [QCPScatterStyle](#) instance which will show the specified *pixmap*. The scatter shape is set to [ssPixmap](#).

6.67.6.7 QCPScatterStyle() [7/7]

```
QCPScatterStyle::QCPScatterStyle (
    const QPainterPath & customPath,
    const QPen & pen,
    const QBrush & brush = Qt::NoBrush,
    double size = 6 )
```

Creates a new [QCPScatterStyle](#) instance with a custom shape that is defined via *customPath*. The scatter shape is set to [ssCustom](#).

The custom shape line will be drawn with *pen* and filled with *brush*. The size has a slightly different meaning than for built-in scatter points: The custom path will be drawn scaled by a factor of *size/6.0*. Since the default *size* is 6, the custom path will appear at its natural size by default. To double the size of the path for example, set *size* to 12.

6.67.7 Member Function Documentation

6.67.7.1 `applyTo()`

```
void QCPScatterStyle::applyTo (
    QCPPainter * painter,
    const QPen & defaultPen ) const
```

Applies the pen and the brush of this scatter style to *painter*. If this scatter style has an undefined pen ([isPenDefined](#)), sets the pen of *painter* to *defaultPen* instead.

This function is used by plottables (or any class that wants to draw scatters) just before a number of scatters with this style shall be drawn with the *painter*.

See also

[drawShape](#)

6.67.7.2 `drawShape()` [1/2]

```
void QCPScatterStyle::drawShape (
    QCPPainter * painter,
    const QPointF & pos ) const
```

Draws the scatter shape with *painter* at position *pos*.

This function does not modify the pen or the brush on the painter, as [applyTo](#) is meant to be called before scatter points are drawn with [drawShape](#).

See also

[applyTo](#)

6.67.7.3 `drawShape()` [2/2]

```
void QCPScatterStyle::drawShape (
    QCPPainter * painter,
    double x,
    double y ) const
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts. Draws the scatter shape with *painter* at position *x* and *y*.

6.67.7.4 `isNone()`

```
bool QCPScatterStyle::isNone ( ) const [inline]
```

Returns whether the scatter shape is [ssNone](#).

See also

[setShape](#)

6.67.7.5 isPenDefined()

```
bool QCPScatterStyle::isPenDefined ( ) const [inline]
```

Returns whether a pen has been defined for this scatter style.

The pen is undefined if a constructor is called that does not carry *pen* as parameter. Those are [QCPScatterStyle\(\)](#) and [QCPScatterStyle\(ScatterShape shape, double size\)](#). If the pen is undefined, the pen of the respective plottable will be used for drawing scatters.

If a pen was defined for this scatter style instance, and you now wish to undefine the pen, call [undefinePen](#).

See also

[setPen](#)

6.67.7.6 setBrush()

```
void QCPScatterStyle::setBrush (
    const QBrush & brush )
```

Sets the brush that will be used to fill scatter points to *brush*. Note that not all scatter shapes have fillable areas. For example, [ssPlus](#) does not while [ssCircle](#) does.

See also

[setPen](#)

6.67.7.7 setCustomPath()

```
void QCPScatterStyle::setCustomPath (
    const QPainterPath & customPath )
```

Sets the custom shape that will be drawn as scatter point to *customPath*.

The scatter shape is automatically set to [ssCustom](#).

6.67.7.8 setFromOther()

```
void QCPScatterStyle::setFromOther (
    const QCPScatterStyle & other,
    ScatterProperties properties )
```

Copies the specified *properties* from the *other* scatter style to this scatter style.

6.67.7.9 setPen()

```
void QCPScatterStyle::setPen (
    const QPen & pen )
```

Sets the pen that will be used to draw scatter points to *pen*.

If the pen was previously undefined (see [isPenDefined](#)), the pen is considered defined after a call to this function, even if *pen* is `Qt::NoPen`. If you have defined a pen previously by calling this function and now wish to undefine the pen, call [undefinePen](#).

See also

[setBrush](#)

6.67.7.10 setPixmap()

```
void QCPScatterStyle::setPixmap (
    const QPixmap & pixmap )
```

Sets the pixmap that will be drawn as scatter point to *pixmap*.

Note that [setSize](#) does not influence the appearance of the pixmap.

The scatter shape is automatically set to [ssPixmap](#).

6.67.7.11 setShape()

```
void QCPScatterStyle::setShape (
    QCPScatterStyle::ScatterShape shape )
```

Sets the shape to *shape*.

Note that the calls [setPixmap](#) and [setCustomPath](#) automatically set the shape to [ssPixmap](#) and [ssCustom](#), respectively.

See also

[setSize](#)

6.67.7.12 setSize()

```
void QCPScatterStyle::setSize (
    double size )
```

Sets the size (pixel diameter) of the drawn scatter points to *size*.

See also

[setShape](#)

6.67.7.13 `undefinePen()`

```
void QCPScatterStyle::undefinePen ( )
```

Sets this scatter style to have an undefined pen (see [isPenDefined](#) for what an undefined pen implies).

A call to [setPen](#) will define a pen.

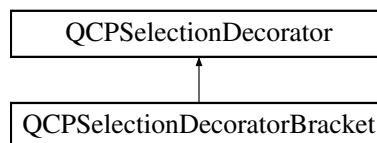
The documentation for this class was generated from the following files:

- `/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h`↔
- `/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp`↔

6.68 QCPSelectionDecorator Class Reference

Controls how a plottable's data selection is drawn.

Inheritance diagram for QCPSelectionDecorator:



Public Member Functions

- [QCPSelectionDecorator](#) ()
- `QPen` **pen** () const
- `QBrush` **brush** () const
- [QCPScatterStyle](#) **scatterStyle** () const
- `QCPScatterStyle::ScatterProperties` **usedScatterProperties** () const
- void [setPen](#) (const `QPen` &pen)
- void [setBrush](#) (const `QBrush` &brush)
- void [setScatterStyle](#) (const [QCPScatterStyle](#) &scatterStyle, `QCPScatterStyle::ScatterProperties` usedProperties=`QCPScatterStyle::spPen`)↔
- void [setUsedScatterProperties](#) (const `QCPScatterStyle::ScatterProperties` &properties)
- void [applyPen](#) (`QCPPainter` *painter) const
- void [applyBrush](#) (`QCPPainter` *painter) const
- [QCPScatterStyle](#) [getFinalScatterStyle](#) (const [QCPScatterStyle](#) &unselectedStyle) const
- virtual void [copyFrom](#) (const [QCPSelectionDecorator](#) *other)
- virtual void [drawDecoration](#) (`QCPPainter` *painter, [QCPDataSelection](#) selection)

Protected Member Functions

- virtual bool **registerWithPlottable** ([QCPAbstractPlottable](#) *plottable)

Protected Attributes

- QPen **mPen**
- QBrush **mBrush**
- [QCPScatterStyle](#) **mScatterStyle**
- QCPScatterStyle::ScatterProperties **mUsedScatterProperties**
- [QCPAbstractPlottable](#) * **mPlottable**

Friends

- class **QCPAbstractPlottable**

6.68.1 Detailed Description

Controls how a plottable's data selection is drawn.

Each [QCPAbstractPlottable](#) instance has one [QCPSelectionDecorator](#) (accessible via [QCPAbstractPlottable::selectionDecorator](#)) and uses it when drawing selected segments of its data.

The selection decorator controls both pen ([setPen](#)) and brush ([setBrush](#)), as well as the scatter style ([setScatterStyle](#)) if the plottable draws scatters. Since a [QCPScatterStyle](#) is itself composed of different properties such as color shape and size, the decorator allows specifying exactly which of those properties shall be used for the selected data point, via [setUsedScatterProperties](#).

A [QCPSelectionDecorator](#) subclass instance can be passed to a plottable via [QCPAbstractPlottable::setSelectionDecorator](#), allowing greater customizability of the appearance of selected segments.

Use [copyFrom](#) to easily transfer the settings of one decorator to another one. This is especially useful since plottables take ownership of the passed selection decorator, and thus the same decorator instance can not be passed to multiple plottables.

Selection decorators can also themselves perform drawing operations by reimplementing [drawDecoration](#), which is called by the plottable's draw method. The base class [QCPSelectionDecorator](#) does not make use of this however. For example, [QCPSelectionDecoratorBracket](#) draws brackets around selected data segments.

6.68.2 Constructor & Destructor Documentation

6.68.2.1 QCPSelectionDecorator()

```
QCPSelectionDecorator::QCPSelectionDecorator ( )
```

Creates a new [QCPSelectionDecorator](#) instance with default values

6.68.3 Member Function Documentation

6.68.3.1 `applyBrush()`

```
void QCPSelectionDecorator::applyBrush (
    QCPPainter * painter ) const
```

Sets the brush of *painter* to the brush of this selection decorator.

See also

[applyPen](#), [getFinalScatterStyle](#)

6.68.3.2 `applyPen()`

```
void QCPSelectionDecorator::applyPen (
    QCPPainter * painter ) const
```

Sets the pen of *painter* to the pen of this selection decorator.

See also

[applyBrush](#), [getFinalScatterStyle](#)

6.68.3.3 `copyFrom()`

```
void QCPSelectionDecorator::copyFrom (
    const QCPSelectionDecorator * other ) [virtual]
```

Copies all properties (e.g. color, fill, scatter style) of the *other* selection decorator to this selection decorator.

6.68.3.4 `drawDecoration()`

```
void QCPSelectionDecorator::drawDecoration (
    QCPPainter * painter,
    QCPDataSelection selection ) [virtual]
```

This method is called by all plottables' draw methods to allow custom selection decorations to be drawn. Use the passed *painter* to perform the drawing operations. *selection* carries the data selection for which the decoration shall be drawn.

The default base class implementation of [QCPSelectionDecorator](#) has no special decoration, so this method does nothing.

Reimplemented in [QCPSelectionDecoratorBracket](#).

6.68.3.5 getFinalScatterStyle()

```
QCPScatterStyle QCPSelectionDecorator::getFinalScatterStyle (
    const QCPScatterStyle & unselectedStyle ) const
```

Returns the scatter style that the parent plottable shall use for selected scatter points. The plottable's original (unselected) scatter style must be passed as *unselectedStyle*. Depending on the setting of [setUsedScatterProperties](#), the returned scatter style is a mixture of this selection decorator's scatter style ([setScatterStyle](#)), and *unselectedStyle*.

See also

[applyPen](#), [applyBrush](#), [setScatterStyle](#)

6.68.3.6 setBrush()

```
void QCPSelectionDecorator::setBrush (
    const QBrush & brush )
```

Sets the brush that will be used by the parent plottable to draw selected data segments.

6.68.3.7 setPen()

```
void QCPSelectionDecorator::setPen (
    const QPen & pen )
```

Sets the pen that will be used by the parent plottable to draw selected data segments.

6.68.3.8 setScatterStyle()

```
void QCPSelectionDecorator::setScatterStyle (
    const QCPScatterStyle & scatterStyle,
    QCPScatterStyle::ScatterProperties usedProperties = QCPScatterStyle::spPen )
```

Sets the scatter style that will be used by the parent plottable to draw scatters in selected data segments.

usedProperties specifies which parts of the passed *scatterStyle* will be used by the plottable. The used properties can also be changed via [setUsedScatterProperties](#).

6.68.3.9 setUsedScatterProperties()

```
void QCPSelectionDecorator::setUsedScatterProperties (
    const QCPScatterStyle::ScatterProperties & properties )
```

Use this method to define which properties of the scatter style (set via [setScatterStyle](#)) will be used for selected data segments. All properties of the scatter style that are not specified in *properties* will remain as specified in the plottable's original scatter style.

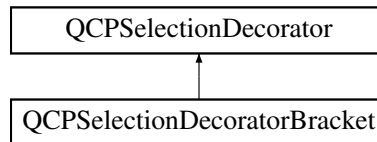
The documentation for this class was generated from the following files:

- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h↵
- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp↵

6.69 QCPSelectionDecoratorBracket Class Reference

A selection decorator which draws brackets around each selected data segment.

Inheritance diagram for QCPSelectionDecoratorBracket:



Public Types

- enum [BracketStyle](#) {
[bsSquareBracket](#), [bsHalfEllipse](#), [bsEllipse](#), [bsPlus](#),
[bsUserStyle](#) }

Public Member Functions

- [QCPSelectionDecoratorBracket](#) ()
- QPen **bracketPen** () const
- QBrush **bracketBrush** () const
- int **bracketWidth** () const
- int **bracketHeight** () const
- [BracketStyle](#) **bracketStyle** () const
- bool **tangentToData** () const
- int **tangentAverage** () const
- void [setBracketPen](#) (const QPen &pen)
- void [setBracketBrush](#) (const QBrush &brush)
- void [setBracketWidth](#) (int width)
- void [setBracketHeight](#) (int height)
- void [setBracketStyle](#) ([BracketStyle](#) style)
- void [setTangentToData](#) (bool enabled)
- void [setTangentAverage](#) (int pointCount)
- virtual void [drawBracket](#) ([QCPPainter](#) *painter, int direction) const
- virtual void [drawDecoration](#) ([QCPPainter](#) *painter, [QCPDataSelection](#) selection)

Protected Member Functions

- double **getTangentAngle** (const [QCPPlottableInterface1D](#) *interface1d, int dataIndex, int direction) const
- QPointF **getPixelCoordinates** (const [QCPPlottableInterface1D](#) *interface1d, int dataIndex) const

Protected Attributes

- QPen **mBracketPen**
- QBrush **mBracketBrush**
- int **mBracketWidth**
- int **mBracketHeight**
- [BracketStyle](#) **mBracketStyle**
- bool **mTangentToData**
- int **mTangentAverage**

6.69.1 Detailed Description

A selection decorator which draws brackets around each selected data segment.

Additionally to the regular highlighting of selected segments via color, fill and scatter style, this [QCPSelectionDecorator](#) subclass draws markers at the begin and end of each selected data segment of the plottable.

The shape of the markers can be controlled with [setBracketStyle](#), [setBracketWidth](#) and [setBracketHeight](#). The color/fill can be controlled with [setBracketPen](#) and [setBracketBrush](#).

To introduce custom bracket styles, it is only necessary to subclass [QCPSelectionDecoratorBracket](#) and reimplement [drawBracket](#). The rest will be managed by the base class.

6.69.2 Member Enumeration Documentation

6.69.2.1 BracketStyle

```
enum QCPSelectionDecoratorBracket::BracketStyle
```

Defines which shape is drawn at the boundaries of selected data ranges.

Some of the bracket styles further allow specifying a height and/or width, see [setBracketHeight](#) and [setBracketWidth](#).

Enumerator

bsSquareBracket	A square bracket is drawn.
bsHalfEllipse	A half ellipse is drawn. The size of the ellipse is given by the bracket width/height properties.
bsEllipse	An ellipse is drawn. The size of the ellipse is given by the bracket width/height properties.
bsPlus	A plus is drawn.
bsUserStyle	Start custom bracket styles at this index when subclassing and reimplementing drawBracket .

6.69.3 Constructor & Destructor Documentation

6.69.3.1 QCPSelectionDecoratorBracket()

```
QCPSelectionDecoratorBracket::QCPSelectionDecoratorBracket ( )
```

Creates a new [QCPSelectionDecoratorBracket](#) instance with default values.

6.69.4 Member Function Documentation

6.69.4.1 drawBracket()

```
void QCPSelectionDecoratorBracket::drawBracket (
    QCPPainter * painter,
    int direction ) const [virtual]
```

Draws the bracket shape with *painter*. The parameter *direction* is either -1 or 1 and indicates whether the bracket shall point to the left or the right (i.e. is a closing or opening bracket, respectively).

The passed *painter* already contains all transformations that are necessary to position and rotate the bracket appropriately. Painting operations can be performed as if drawing upright brackets on flat data with horizontal key axis, with (0, 0) being the center of the bracket.

If you wish to subclass [QCPSelectionDecoratorBracket](#) in order to provide custom bracket shapes (see [QCPSelectionDecoratorBracket::bsUserStyle](#)), this is the method you should reimplement.

6.69.4.2 drawDecoration()

```
void QCPSelectionDecoratorBracket::drawDecoration (
    QCPPainter * painter,
    QCPDataSelection selection ) [virtual]
```

Draws the bracket decoration on the data points at the begin and end of each selected data segment given in *selection*.

It uses the method [drawBracket](#) to actually draw the shapes.

Reimplemented from [QCPSelectionDecorator](#).

6.69.4.3 setBracketBrush()

```
void QCPSelectionDecoratorBracket::setBracketBrush (
    const QBrush & brush )
```

Sets the brush that will be used to draw the brackets at the beginning and end of each selected data segment.

6.69.4.4 setBracketHeight()

```
void QCPSelectionDecoratorBracket::setBracketHeight (
    int height )
```

Sets the height of the drawn bracket. The height dimension is always perpendicular to the key axis of the data, or the tangent direction of the current data slope, if [setTangentToData](#) is enabled.

6.69.4.5 setBracketPen()

```
void QCPSelectionDecoratorBracket::setBracketPen (
    const QPen & pen )
```

Sets the pen that will be used to draw the brackets at the beginning and end of each selected data segment.

6.69.4.6 setBracketStyle()

```
void QCPSelectionDecoratorBracket::setBracketStyle (
    QCPSelectionDecoratorBracket::BracketStyle style )
```

Sets the shape that the bracket/marker will have.

See also

[setBracketWidth](#), [setBracketHeight](#)

6.69.4.7 setBracketWidth()

```
void QCPSelectionDecoratorBracket::setBracketWidth (
    int width )
```

Sets the width of the drawn bracket. The width dimension is always parallel to the key axis of the data, or the tangent direction of the current data slope, if [setTangentToData](#) is enabled.

6.69.4.8 setTangentAverage()

```
void QCPSelectionDecoratorBracket::setTangentAverage (
    int pointCount )
```

Controls over how many data points the slope shall be averaged, when brackets shall be aligned with the data (if [setTangentToData](#) is true).

From the position of the bracket, *pointCount* points towards the selected data range will be taken into account. The smallest value of *pointCount* is 1, which is effectively equivalent to disabling [setTangentToData](#).

6.69.4.9 setTangentToData()

```
void QCPSelectionDecoratorBracket::setTangentToData (
    bool enabled )
```

Sets whether the brackets will be rotated such that they align with the slope of the data at the position that they appear in.

For noisy data, it might be more visually appealing to average the slope over multiple data points. This can be configured via [setTangentAverage](#).

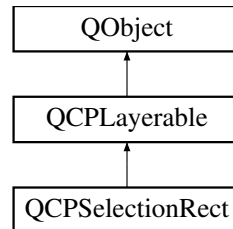
The documentation for this class was generated from the following files:

- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h↵
- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp↵

6.70 QCPSelectionRect Class Reference

Provides rect/rubber-band data selection and range zoom interaction.

Inheritance diagram for QCPSelectionRect:



Signals

- void **started** (QMouseEvent *event)
- void **changed** (const QRect &rect, QMouseEvent *event)
- void **canceled** (const QRect &rect, QInputEvent *event)
- void **accepted** (const QRect &rect, QMouseEvent *event)

Public Member Functions

- **QCPSelectionRect** (QCustomPlot *parentPlot)
- QRect **rect** () const
- **QCPRange range** (const **QCPAxis** *axis) const
- QPen **pen** () const
- QBrush **brush** () const
- bool **isActive** () const
- void **setPen** (const QPen &pen)
- void **setBrush** (const QBrush &brush)
- Q_SLOT void **cancel** ()

Protected Member Functions

- virtual void **startSelection** (QMouseEvent *event)
- virtual void **moveSelection** (QMouseEvent *event)
- virtual void **endSelection** (QMouseEvent *event)
- virtual void **keyPressEvent** (QKeyEvent *event)
- virtual void **applyDefaultAntialiasingHint** (**QCPPainter** *painter) const Q_DECL_OVERRIDE
- virtual void **draw** (**QCPPainter** *painter) Q_DECL_OVERRIDE

Protected Attributes

- QRect **mRect**
- QPen **mPen**
- QBrush **mBrush**
- bool **mActive**

Friends

- class **QCustomPlot**

6.70.1 Detailed Description

Provides rect/rubber-band data selection and range zoom interaction.

QCPSelectionRect is used by **QCustomPlot** when the **QCustomPlot::setSelectionRectMode** is not **QCP::srmNone**. When the user drags the mouse across the plot, the current selection rect instance (**QCustomPlot::setSelectionRect**) is forwarded these events and makes sure an according rect shape is drawn. At the begin, during, and after completion of the interaction, it emits the corresponding signals **started**, **changed**, **canceled**, and **accepted**.

The **QCustomPlot** instance connects own slots to the current selection rect instance, in order to react to an accepted selection rect interaction accordingly.

isActive can be used to check whether the selection rect is currently active. An ongoing selection interaction can be cancelled programmatically via calling **cancel** at any time.

The appearance of the selection rect can be controlled via **setPen** and **setBrush**.

If you wish to provide custom behaviour, e.g. a different visual representation of the selection rect (**QCPSelectionRect::draw**), you can subclass **QCPSelectionRect** and pass an instance of your subclass to **QCustomPlot::setSelectionRect**.

6.70.2 Constructor & Destructor Documentation

6.70.2.1 QCPSelectionRect()

```
QCPSelectionRect::QCPSelectionRect (
    QCustomPlot * parentPlot ) [explicit]
```

Creates a new **QCPSelectionRect** instance. To make **QCustomPlot** use the selection rect instance, pass it to **QCustomPlot::setSelectionRect**. *parentPlot* should be set to the same **QCustomPlot** widget.

6.70.3 Member Function Documentation

6.70.3.1 accepted

```
void QCPSelectionRect::accepted (
    const QRect & rect,
    QMouseEvent * event ) [signal]
```

This signal is emitted when the selection interaction was completed by the user releasing the mouse button.

Note that *rect* may have a negative width or height, if the selection is being dragged to the upper or left side of the selection rect origin.

6.70.3.2 cancel()

```
void QCPSelectionRect::cancel ( )
```

If there is currently a selection interaction going on ([isActive](#)), the interaction is canceled. The selection rect will emit the [canceled](#) signal.

6.70.3.3 canceled

```
void QCPSelectionRect::canceled (
    const QRect & rect,
    QInputEvent * event ) [signal]
```

This signal is emitted when the selection interaction was cancelled. Note that *event* is 0 if the selection interaction was cancelled programmatically, by a call to [cancel](#).

The user may cancel the selection interaction by pressing the escape key. In this case, *event* holds the respective input event.

Note that *rect* may have a negative width or height, if the selection is being dragged to the upper or left side of the selection rect origin.

6.70.3.4 changed

```
void QCPSelectionRect::changed (
    const QRect & rect,
    QMouseEvent * event ) [signal]
```

This signal is emitted while the selection rect interaction is ongoing and the *rect* has changed its size due to the user moving the mouse.

Note that *rect* may have a negative width or height, if the selection is being dragged to the upper or left side of the selection rect origin.

6.70.3.5 isActive()

```
bool QCPSelectionRect::isActive ( ) const [inline]
```

Returns true if there is currently a selection going on, i.e. the user has started dragging a selection rect, but hasn't released the mouse button yet.

See also

[cancel](#)

6.70.3.6 range()

```
QCPRange QCPSelectionRect::range (
    const QCPAxis * axis ) const
```

A convenience function which returns the coordinate range of the provided *axis*, that this selection rect currently encompasses.

6.70.3.7 setBrush()

```
void QCPSelectionRect::setBrush (
    const QBrush & brush )
```

Sets the brush that will be used to fill the selection rect. By default the selection rect is not filled, i.e. *brush* is `Qt::NoBrush`.

See also

[setPen](#)

6.70.3.8 setPen()

```
void QCPSelectionRect::setPen (
    const QPen & pen )
```

Sets the pen that will be used to draw the selection rect outline.

See also

[setBrush](#)

6.70.3.9 started

```
void QCPSelectionRect::started (
    QMouseEvent * event ) [signal]
```

This signal is emitted when a selection rect interaction was initiated, i.e. the user just started dragging the selection rect with the mouse.

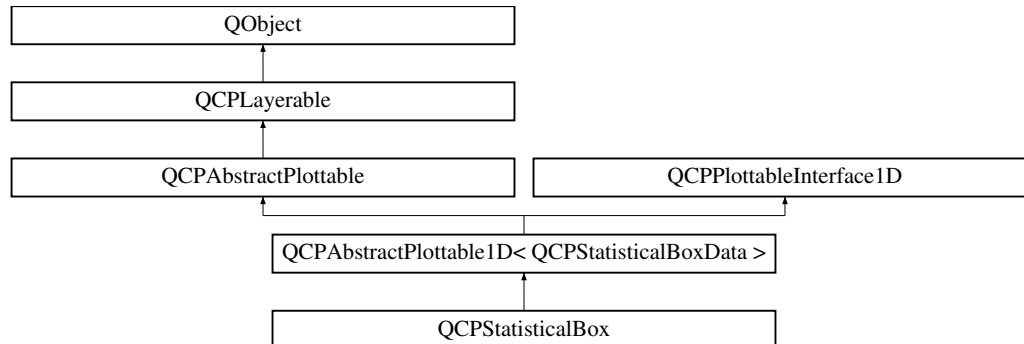
The documentation for this class was generated from the following files:

- `/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h`
- `/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp`

6.71 QCPStatisticalBox Class Reference

A plottable representing a single statistical box in a plot.

Inheritance diagram for QCPStatisticalBox:



Public Member Functions

- [QCPStatisticalBox](#) ([QCPAxis](#) *keyAxis, [QCPAxis](#) *valueAxis)
- [QSharedPointer](#)< [QCPStatisticalBoxDataContainer](#) > data () const
- double **width** () const
- double **whiskerWidth** () const
- [QPen](#) **whiskerPen** () const
- [QPen](#) **whiskerBarPen** () const
- bool **whiskerAntialiased** () const
- [QPen](#) **medianPen** () const
- [QCPScatterStyle](#) **outlierStyle** () const
- void [setData](#) ([QSharedPointer](#)< [QCPStatisticalBoxDataContainer](#) > data)
- void [setData](#) (const [QVector](#)< double > &keys, const [QVector](#)< double > &minimum, const [QVector](#)< double > &lowerQuartile, const [QVector](#)< double > &median, const [QVector](#)< double > &upperQuartile, const [QVector](#)< double > &maximum, bool alreadySorted=false)
- void [setWidth](#) (double width)
- void [setWhiskerWidth](#) (double width)
- void [setWhiskerPen](#) (const [QPen](#) &pen)
- void [setWhiskerBarPen](#) (const [QPen](#) &pen)
- void [setWhiskerAntialiased](#) (bool enabled)
- void [setMedianPen](#) (const [QPen](#) &pen)
- void [setOutlierStyle](#) (const [QCPScatterStyle](#) &style)
- void [addData](#) (const [QVector](#)< double > &keys, const [QVector](#)< double > &minimum, const [QVector](#)< double > &lowerQuartile, const [QVector](#)< double > &median, const [QVector](#)< double > &upperQuartile, const [QVector](#)< double > &maximum, bool alreadySorted=false)
- void [addData](#) (double key, double minimum, double lowerQuartile, double median, double upperQuartile, double maximum, const [QVector](#)< double > &outliers=[QVector](#)< double >())
- virtual [QCPDataSelection](#) [selectTestRect](#) (const [QRectF](#) &rect, bool onlySelectable) const [Q_DECL_OVERRIDE](#)
- virtual double [selectTest](#) (const [QPointF](#) &pos, bool onlySelectable, [QVariant](#) *details=0) const [Q_DECL_OVERRIDE](#)
- virtual [QCPRange](#) [getKeyRange](#) (bool &foundRange, [QCP::SignDomain](#) inSignDomain=[QCP::sdBoth](#)) const [Q_DECL_OVERRIDE](#)
- virtual [QCPRange](#) [getValueRange](#) (bool &foundRange, [QCP::SignDomain](#) inSignDomain=[QCP::sdBoth](#), const [QCPRange](#) &inKeyRange=[QCPRange](#)()) const [Q_DECL_OVERRIDE](#)

Protected Member Functions

- virtual void **draw** (QCPPainter *painter) Q_DECL_OVERRIDE
- virtual void **drawLegendIcon** (QCPPainter *painter, const QRectF &rect) const Q_DECL_OVERRIDE
- virtual void **drawStatisticalBox** (QCPPainter *painter, QCPStatisticalBoxDataContainer::const_iterator it, const QCPScatterStyle &outlierStyle) const
- void **getVisibleDataBounds** (QCPStatisticalBoxDataContainer::const_iterator &begin, QCPStatisticalBoxDataContainer::const_iterator &end) const
- QRectF **getQuartileBox** (QCPStatisticalBoxDataContainer::const_iterator it) const
- QVector< QLineF > **getWhiskerBackboneLines** (QCPStatisticalBoxDataContainer::const_iterator it) const
- QVector< QLineF > **getWhiskerBarLines** (QCPStatisticalBoxDataContainer::const_iterator it) const

Protected Attributes

- double **mWidth**
- double **mWhiskerWidth**
- QPen **mWhiskerPen**
- QPen **mWhiskerBarPen**
- bool **mWhiskerAntialiased**
- QPen **mMedianPen**
- QCPScatterStyle **mOutlierStyle**

Friends

- class **QCustomPlot**
- class **QCPLegend**

Additional Inherited Members

6.71.1 Detailed Description

A plottable representing a single statistical box in a plot.

To plot data, assign it with the [setData](#) or [addData](#) functions. Alternatively, you can also access and modify the data via the [data](#) method, which returns a pointer to the internal QCPStatisticalBoxDataContainer.

Additionally each data point can itself have a list of outliers, drawn as scatter points at the key coordinate of the respective statistical box data point. They can either be set by using the respective [addData](#) method or accessing the individual data points through [data](#), and setting the `QVector<double> outliers` of the data points directly.

6.71.2 Changing the appearance

The appearance of each data point box, ranging from the lower to the upper quartile, is controlled via [setPen](#) and [setBrush](#). You may change the width of the boxes with [setWidth](#) in plot coordinates.

Each data point's visual representation also consists of two whiskers. Whiskers are the lines which reach from the upper quartile to the maximum, and from the lower quartile to the minimum. The appearance of the whiskers can be modified with: [setWhiskerPen](#), [setWhiskerBarPen](#), [setWhiskerWidth](#). The whisker width is the width of the bar perpendicular to the whisker at the top (for maximum) and bottom (for minimum). If the whisker pen is changed, make sure to set the `capStyle` to `Qt::FlatCap`. Otherwise the backbone line might exceed the whisker bars by a few pixels due to the pen cap being not perfectly flat.

The median indicator line inside the box has its own pen, [setMedianPen](#).

The outlier data points are drawn as normal scatter points. Their look can be controlled with [setOutlierStyle](#)

6.71.3 Usage

Like all data representing objects in `QCustomPlot`, the `QCPStatisticalBox` is a plottable (`QCPAbstractPlottable`). So the plottable-interface of `QCustomPlot` applies (`QCustomPlot::plottable`, `QCustomPlot::removePlottable`, etc.)

Usually, you first create an instance:

which registers it with the `QCustomPlot` instance of the passed axes. Note that this `QCustomPlot` instance takes ownership of the plottable, so do not delete it manually but use `QCustomPlot::removePlottable()` instead. The newly created plottable can be modified, e.g.:

6.71.4 Constructor & Destructor Documentation

6.71.4.1 QCPStatisticalBox()

```
QCPStatisticalBox::QCPStatisticalBox (
    QCPAxis * keyAxis,
    QCPAxis * valueAxis ) [explicit]
```

Constructs a statistical box which uses *keyAxis* as its key axis ("x") and *valueAxis* as its value axis ("y"). *keyAxis* and *valueAxis* must reside in the same `QCustomPlot` instance and not have the same orientation. If either of these restrictions is violated, a corresponding message is printed to the debug output (qDebug), the construction is not aborted, though.

The created `QCPStatisticalBox` is automatically registered with the `QCustomPlot` instance inferred from *keyAxis*. This `QCustomPlot` instance takes ownership of the `QCPStatisticalBox`, so do not delete it manually but use `QCustomPlot::removePlottable()` instead.

6.71.5 Member Function Documentation

6.71.5.1 addData() [1/2]

```
void QCPStatisticalBox::addData (
    const QVector< double > & keys,
    const QVector< double > & minimum,
    const QVector< double > & lowerQuartile,
    const QVector< double > & median,
    const QVector< double > & upperQuartile,
    const QVector< double > & maximum,
    bool alreadySorted = false )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Adds the provided points in *keys*, *minimum*, *lowerQuartile*, *median*, *upperQuartile* and *maximum* to the current data. The provided vectors should have equal length. Else, the number of added points will be the size of the smallest vector.

If you can guarantee that the passed data points are sorted by *keys* in ascending order, you can set *alreadySorted* to true, to improve performance by saving a sorting run.

Alternatively, you can also access and modify the data directly via the `data` method, which returns a pointer to the internal data container.

6.71.5.2 `addData()` [2/2]

```
void QCPStatisticalBox::addData (
    double key,
    double minimum,
    double lowerQuartile,
    double median,
    double upperQuartile,
    double maximum,
    const QVector< double > & outliers = QVector<double>() )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Adds the provided data point as *key*, *minimum*, *lowerQuartile*, *median*, *upperQuartile* and *maximum* to the current data.

Alternatively, you can also access and modify the data directly via the [data](#) method, which returns a pointer to the internal data container.

6.71.5.3 `data()`

```
QSharedPointer< QCPStatisticalBoxDataContainer > QCPStatisticalBox::data ( ) const [inline]
```

Returns a shared pointer to the internal data storage of type `QCPStatisticalBoxDataContainer`. You may use it to directly manipulate the data, which may be more convenient and faster than using the regular [setData](#) or [addData](#) methods.

6.71.5.4 `drawStatisticalBox()`

```
void QCPStatisticalBox::drawStatisticalBox (
    QCPPainter * painter,
    QCPStatisticalBoxDataContainer::const_iterator it,
    const QCPScatterStyle & outlierStyle ) const [protected], [virtual]
```

Draws the graphical representation of a single statistical box with the data given by the iterator *it* with the provided *painter*.

If the statistical box has a set of outlier data points, they are drawn with *outlierStyle*.

See also

`getQuartileBox`, `getWhiskerBackboneLines`, `getWhiskerBarLines`

6.71.5.5 getKeyRange()

```
QCPRange QCPStatisticalBox::getKeyRange (
    bool & foundRange,
    QCP::SignDomain inSignDomain = QCP::sdBoth ) const [virtual]
```

Returns the coordinate range that all data in this plottable span in the key axis dimension. For logarithmic plots, one can set *inSignDomain* to either [QCP::sdNegative](#) or [QCP::sdPositive](#) in order to restrict the returned range to that sign domain. E.g. when only negative range is wanted, set *inSignDomain* to [QCP::sdNegative](#) and all positive points will be ignored for range calculation. For no restriction, just set *inSignDomain* to [QCP::sdBoth](#) (default). *foundRange* is an output parameter that indicates whether a range could be found or not. If this is false, you shouldn't use the returned range (e.g. no points in data).

Note that *foundRange* is not the same as [QCPRange::validRange](#), since the range returned by this function may have size zero (e.g. when there is only one data point). In this case *foundRange* would return true, but the returned range is not a valid range in terms of [QCPRange::validRange](#).

See also

[rescaleAxes](#), [getValueRange](#)

Implements [QCPAbstractPlottable](#).

6.71.5.6 getValueRange()

```
QCPRange QCPStatisticalBox::getValueRange (
    bool & foundRange,
    QCP::SignDomain inSignDomain = QCP::sdBoth,
    const QCPRange & inKeyRange = QCPRange() ) const [virtual]
```

Returns the coordinate range that the data points in the specified key range (*inKeyRange*) span in the value axis dimension. For logarithmic plots, one can set *inSignDomain* to either [QCP::sdNegative](#) or [QCP::sdPositive](#) in order to restrict the returned range to that sign domain. E.g. when only negative range is wanted, set *inSignDomain* to [QCP::sdNegative](#) and all positive points will be ignored for range calculation. For no restriction, just set *inSignDomain* to [QCP::sdBoth](#) (default). *foundRange* is an output parameter that indicates whether a range could be found or not. If this is false, you shouldn't use the returned range (e.g. no points in data).

If *inKeyRange* has both lower and upper bound set to zero (is equal to [QCPRange\(\)](#)), all data points are considered, without any restriction on the keys.

Note that *foundRange* is not the same as [QCPRange::validRange](#), since the range returned by this function may have size zero (e.g. when there is only one data point). In this case *foundRange* would return true, but the returned range is not a valid range in terms of [QCPRange::validRange](#).

See also

[rescaleAxes](#), [getKeyRange](#)

Implements [QCPAbstractPlottable](#).

6.71.5.7 selectTest()

```
double QCPStatisticalBox::selectTest (
    const QPointF & pos,
    bool onlySelectable,
    QVariant * details = 0 ) const [virtual]
```

Implements a point-selection algorithm assuming the data (accessed via the 1D data interface) is point-like. Most subclasses will want to reimplement this method again, to provide a more accurate hit test based on the true data visualization geometry.

Reimplemented from [QCPAbstractPlottable1D< QCPStatisticalBoxData >](#).

6.71.5.8 selectTestRect()

```
QCPDataSelection QCPStatisticalBox::selectTestRect (
    const QRectF & rect,
    bool onlySelectable ) const [virtual]
```

Returns a data selection containing all the data points of this plottable which are contained (or hit by) *rect*. This is used mainly in the selection rect interaction for data selection (data selection mechanism).

If *onlySelectable* is true, an empty [QCPDataSelection](#) is returned if this plottable is not selectable (i.e. if [QCPAbstractPlottable::setSelectable](#) is [QCP::stNone](#)).

Note

rect must be a normalized rect (positive or zero width and height). This is especially important when using the rect of [QCPSelectionRect::accepted](#), which is not necessarily normalized. Use [QRect::normalized\(\)](#) when passing a rect which might not be normalized.

Reimplemented from [QCPAbstractPlottable1D< QCPStatisticalBoxData >](#).

6.71.5.9 setData() [1/2]

```
void QCPStatisticalBox::setData (
    QSharedPointer< QCPStatisticalBoxDataContainer > data )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Replaces the current data container with the provided *data* container.

Since a [QSharedPointer](#) is used, multiple [QCPStatisticalBoxes](#) may share the same data container safely. Modifying the data in the container will then affect all statistical boxes that share the container. Sharing can be achieved by simply exchanging the data containers wrapped in shared pointers:

If you do not wish to share containers, but create a copy from an existing container, rather use the [QCPDataContainer<DataType>::set](#) method on the statistical box data container directly:

See also

[addData](#)

6.71.5.10 setData() [2/2]

```
void QCPStatisticalBox::setData (
    const QVector< double > & keys,
    const QVector< double > & minimum,
    const QVector< double > & lowerQuartile,
    const QVector< double > & median,
    const QVector< double > & upperQuartile,
    const QVector< double > & maximum,
    bool alreadySorted = false )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Replaces the current data with the provided points in *keys*, *minimum*, *lowerQuartile*, *median*, *upperQuartile* and *maximum*. The provided vectors should have equal length. Else, the number of added points will be the size of the smallest vector.

If you can guarantee that the passed data points are sorted by *keys* in ascending order, you can set *alreadySorted* to true, to improve performance by saving a sorting run.

See also

[addData](#)

6.71.5.11 setMedianPen()

```
void QCPStatisticalBox::setMedianPen (
    const QPen & pen )
```

Sets the pen used for drawing the median indicator line inside the statistical boxes.

6.71.5.12 setOutlierStyle()

```
void QCPStatisticalBox::setOutlierStyle (
    const QCPScatterStyle & style )
```

Sets the appearance of the outlier data points.

Outliers can be specified with the method [addData\(double key, double minimum, double lowerQuartile, double median, double upperQuartile, double maximum, const QVector<double> &outliers\)](#)

6.71.5.13 setWhiskerAntialiased()

```
void QCPStatisticalBox::setWhiskerAntialiased (
    bool enabled )
```

Sets whether the statistical boxes whiskers are drawn with antialiasing or not.

Note that antialiasing settings may be overridden by [QCustomPlot::setAntialiasedElements](#) and [QCustomPlot::setNotAntialiasedElements](#).

6.71.5.14 setWhiskerBarPen()

```
void QCPStatisticalBox::setWhiskerBarPen (
    const QPen & pen )
```

Sets the pen used for drawing the whisker bars. Those are the lines parallel to the key axis at each end of the whisker backbone.

Whiskers are the lines which reach from the upper quartile to the maximum, and from the lower quartile to the minimum.

See also

[setWhiskerPen](#)

6.71.5.15 setWhiskerPen()

```
void QCPStatisticalBox::setWhiskerPen (
    const QPen & pen )
```

Sets the pen used for drawing the whisker backbone.

Whiskers are the lines which reach from the upper quartile to the maximum, and from the lower quartile to the minimum.

Make sure to set the `capStyle` of the passed *pen* to `Qt::FlatCap`. Otherwise the backbone line might exceed the whisker bars by a few pixels due to the pen cap being not perfectly flat.

See also

[setWhiskerBarPen](#)

6.71.5.16 setWhiskerWidth()

```
void QCPStatisticalBox::setWhiskerWidth (
    double width )
```

Sets the width of the whiskers in key coordinates.

Whiskers are the lines which reach from the upper quartile to the maximum, and from the lower quartile to the minimum.

See also

[setWidth](#)

6.71.5.17 `setWidth()`

```
void QCPStatisticalBox::setWidth (
    double width )
```

Sets the width of the boxes in key coordinates.

See also

[setWidth](#)

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

- `/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h`
- `/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp`

6.72 QCPStatisticalBoxData Class Reference

Holds the data of one single data point for [QCPStatisticalBox](#).

Public Member Functions

- [QCPStatisticalBoxData](#) ()
- [QCPStatisticalBoxData](#) (double key, double minimum, double lowerQuartile, double median, double upperQuartile, double maximum, const QVector< double > &outliers=QVector< double >())
- double [sortKey](#) () const
- double [mainKey](#) () const
- double [mainValue](#) () const
- [QCPRange](#) [valueRange](#) () const

Static Public Member Functions

- static [QCPStatisticalBoxData](#) [fromSortKey](#) (double [sortKey](#))
- static bool [sortKeysIsMainKey](#) ()

Public Attributes

- double **key**
- double **minimum**
- double **lowerQuartile**
- double **median**
- double **upperQuartile**
- double **maximum**
- QVector< double > **outliers**

6.72.1 Detailed Description

Holds the data of one single data point for [QCPStatisticalBox](#).

The stored data is:

- *key*: coordinate on the key axis of this data point (this is the *mainKey* and the *sortKey*)
- *minimum*: the position of the lower whisker, typically the minimum measurement of the sample that's not considered an outlier.
- *lowerQuartile*: the lower end of the box. The lower and the upper quartiles are the two statistical quartiles around the median of the sample, they should contain 50% of the sample data.
- *median*: the value of the median mark inside the quartile box. The median separates the sample data in half (50% of the sample data is below/above the median). (This is the *mainValue*)
- *upperQuartile*: the upper end of the box. The lower and the upper quartiles are the two statistical quartiles around the median of the sample, they should contain 50% of the sample data.
- *maximum*: the position of the upper whisker, typically the maximum measurement of the sample that's not considered an outlier.
- *outliers*: a QVector of outlier values that will be drawn as scatter points at the *key* coordinate of this data point (see [QCPStatisticalBox::setOutlierStyle](#))

The container for storing multiple data points is QCPStatisticalBoxDataContainer. It is a typedef for [QCPDataContainer](#) with [QCPStatisticalBoxData](#) as the DataType template parameter. See the documentation there for an explanation regarding the data type's generic methods.

See also

QCPStatisticalBoxDataContainer

6.72.2 Constructor & Destructor Documentation

6.72.2.1 QCPStatisticalBoxData() [1/2]

```
QCPStatisticalBoxData::QCPStatisticalBoxData ( )
```

Constructs a data point with key and all values set to zero.

6.72.2.2 QCPStatisticalBoxData() [2/2]

```
QCPStatisticalBoxData::QCPStatisticalBoxData (
    double key,
    double minimum,
    double lowerQuartile,
    double median,
    double upperQuartile,
    double maximum,
    const QVector< double > & outliers = QVector<double>() )
```

Constructs a data point with the specified *key*, *minimum*, *lowerQuartile*, *median*, *upperQuartile*, *maximum* and optionally a number of *outliers*.

6.72.3 Member Function Documentation

6.72.3.1 fromSortKey()

```
static QCPStatisticalBoxData QCPStatisticalBoxData::fromSortKey (
    double sortKey ) [inline], [static]
```

Returns a data point with the specified *sortKey*. All other members are set to zero.

For a general explanation of what this method is good for in the context of the data container, see the documentation of [QCPDataContainer](#).

6.72.3.2 mainKey()

```
double QCPStatisticalBoxData::mainKey ( ) const [inline]
```

Returns the *key* member of this data point.

For a general explanation of what this method is good for in the context of the data container, see the documentation of [QCPDataContainer](#).

6.72.3.3 mainValue()

```
double QCPStatisticalBoxData::mainValue ( ) const [inline]
```

Returns the *median* member of this data point.

For a general explanation of what this method is good for in the context of the data container, see the documentation of [QCPDataContainer](#).

6.72.3.4 sortKey()

```
double QCPStatisticalBoxData::sortKey ( ) const [inline]
```

Returns the *key* member of this data point.

For a general explanation of what this method is good for in the context of the data container, see the documentation of [QCPDataContainer](#).

6.72.3.5 sortKeyIsMainKey()

```
static static bool QCPStatisticalBoxData::sortKeyIsMainKey ( ) [inline], [static]
```

Since the member *key* is both the data point key coordinate and the data ordering parameter, this method returns true.

For a general explanation of what this method is good for in the context of the data container, see the documentation of [QCPDataContainer](#).

6.72.3.6 valueRange()

```
QCPRange QCPStatisticalBoxData::valueRange ( ) const [inline]
```

Returns a [QCPRange](#) spanning from the *minimum* to the *maximum* member of this statistical box data point, possibly further expanded by outliers.

For a general explanation of what this method is good for in the context of the data container, see the documentation of [QCPDataContainer](#).

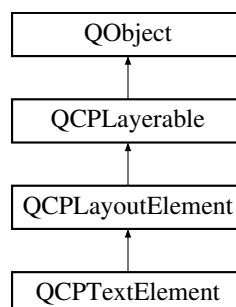
The documentation for this class was generated from the following files:

- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h
- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp

6.73 QCPETextElement Class Reference

A layout element displaying a text.

Inheritance diagram for QCPETextElement:



Signals

- void [selectionChanged](#) (bool selected)
- void **selectableChanged** (bool selectable)
- void [clicked](#) (QMouseEvent *event)
- void [doubleClicked](#) (QMouseEvent *event)

Public Member Functions

- [QCPTextElement](#) (QCustomPlot *parentPlot)
- [QCPTextElement](#) (QCustomPlot *parentPlot, const QString &text)
- [QCPTextElement](#) (QCustomPlot *parentPlot, const QString &text, double pointSize)
- [QCPTextElement](#) (QCustomPlot *parentPlot, const QString &text, const QString &fontFamily, double pointSize)
- [QCPTextElement](#) (QCustomPlot *parentPlot, const QString &text, const QFont &font)
- QString **text** () const
- int **textFlags** () const
- QFont **font** () const
- QColor **textColor** () const
- QFont **selectedFont** () const
- QColor **selectedTextColor** () const
- bool **selectable** () const
- bool **selected** () const
- void [setText](#) (const QString &text)
- void [setTextFlags](#) (int flags)
- void [setFont](#) (const QFont &font)
- void [setTextColor](#) (const QColor &color)
- void [setSelectedFont](#) (const QFont &font)
- void [setSelectedTextColor](#) (const QColor &color)
- Q_SLOT void [setSelectable](#) (bool selectable)
- Q_SLOT void [setSelected](#) (bool selected)
- virtual double [selectTest](#) (const QPointF &pos, bool onlySelectable, QVariant *details=0) const Q_DECL_OVERRIDE
- virtual void [mousePressEvent](#) (QMouseEvent *event, const QVariant &details) Q_DECL_OVERRIDE
- virtual void [mouseReleaseEvent](#) (QMouseEvent *event, const QPointF &startPos) Q_DECL_OVERRIDE
- virtual void [mouseDoubleClickEvent](#) (QMouseEvent *event, const QVariant &details) Q_DECL_OVERRIDE

Protected Member Functions

- virtual void **applyDefaultAntialiasingHint** (QCPPainter *painter) const Q_DECL_OVERRIDE
- virtual void **draw** (QCPPainter *painter) Q_DECL_OVERRIDE
- virtual QSize **minimumSizeHint** () const Q_DECL_OVERRIDE
- virtual QSize **maximumSizeHint** () const Q_DECL_OVERRIDE
- virtual void **selectEvent** (QMouseEvent *event, bool additive, const QVariant &details, bool *selectionStateChanged) Q_DECL_OVERRIDE
- virtual void **deselectEvent** (bool *selectionStateChanged) Q_DECL_OVERRIDE
- QFont **mainFont** () const
- QColor **mainTextColor** () const

Protected Attributes

- QString **mText**
- int **mTextFlags**
- QFont **mFont**
- QColor **mTextColor**
- QFont **mSelectedFont**
- QColor **mSelectedTextColor**
- QRect **mTextBoundingRect**
- bool **mSelectable**
- bool **mSelected**

Additional Inherited Members

6.73.1 Detailed Description

A layout element displaying a text.

The text may be specified with [setText](#), the formatting can be controlled with [setFont](#), [setTextColor](#), and [setTextFlags](#).

A text element can be added as follows:

6.73.2 Constructor & Destructor Documentation

6.73.2.1 QCPElement() [1/5]

```
QCPElement::QCPElement (
    QCustomPlot * parentPlot ) [explicit]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Creates a new [QCPElement](#) instance and sets default values. The initial text is empty ([setText](#)).

6.73.2.2 QCPElement() [2/5]

```
QCPElement::QCPElement (
    QCustomPlot * parentPlot,
    const QString & text )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Creates a new [QCPElement](#) instance and sets default values.

The initial text is set to *text*.

6.73.2.3 QCPTextElement() [3/5]

```
QCPTextElement::QCPTextElement (
    QCustomPlot * parentPlot,
    const QString & text,
    double pointSize )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Creates a new [QCPTextElement](#) instance and sets default values.

The initial text is set to *text* with *pointSize*.

6.73.2.4 QCPTextElement() [4/5]

```
QCPTextElement::QCPTextElement (
    QCustomPlot * parentPlot,
    const QString & text,
    const QString & fontFamily,
    double pointSize )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Creates a new [QCPTextElement](#) instance and sets default values.

The initial text is set to *text* with *pointSize* and the specified *fontFamily*.

6.73.2.5 QCPTextElement() [5/5]

```
QCPTextElement::QCPTextElement (
    QCustomPlot * parentPlot,
    const QString & text,
    const QFont & font )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Creates a new [QCPTextElement](#) instance and sets default values.

The initial text is set to *text* with the specified *font*.

6.73.3 Member Function Documentation

6.73.3.1 clicked

```
void QCPTextElement::clicked (
    QMouseEvent * event ) [signal]
```

This signal is emitted when the text element is clicked.

See also

[doubleClicked](#), [selectTest](#)

6.73.3.2 doubleClicked

```
void QCPTextElement::doubleClicked (
    QMouseEvent * event ) [signal]
```

This signal is emitted when the text element is double clicked.

See also

[clicked](#), [selectTest](#)

6.73.3.3 maximumSizeHint()

```
QSize QCPTextElement::maximumSizeHint ( ) const [protected], [virtual]
```

Returns the maximum size this layout element (the inner [rect](#)) may be expanded to.

if a maximum size ([setMaximumSize](#)) was not set manually, parent layouts consult this function to determine the maximum allowed size of this layout element. (A manual maximum size is considered set if it is smaller than Qt's QWIDGETSIZE_MAX.)

Reimplemented from [QCPLayoutElement](#).

6.73.3.4 minimumSizeHint()

```
QSize QCPTextElement::minimumSizeHint ( ) const [protected], [virtual]
```

Returns the minimum size this layout element (the inner [rect](#)) may be compressed to.

if a minimum size ([setMinimumSize](#)) was not set manually, parent layouts consult this function to determine the minimum allowed size of this layout element. (A manual minimum size is considered set if it is non-zero.)

Reimplemented from [QCPLayoutElement](#).

6.73.3.5 mouseDoubleClickEvent()

```
void QCPTextElement::mouseDoubleClickEvent (
    QMouseEvent * event,
    const QVariant & details ) [virtual]
```

Emits the [doubleClicked](#) signal.

Reimplemented from [QCPLayerable](#).

6.73.3.6 mousePressEvent()

```
void QCPTextElement::mousePressEvent (
    QMouseEvent * event,
    const QVariant & details ) [virtual]
```

Accepts the mouse event in order to emit the according click signal in the [mouseReleaseEvent](#).

Reimplemented from [QCPLayerable](#).

6.73.3.7 mouseReleaseEvent()

```
void QCPTextElement::mouseReleaseEvent (
    QMouseEvent * event,
    const QPointF & startPos ) [virtual]
```

Emits the [clicked](#) signal if the cursor hasn't moved by more than a few pixels since the [mousePressEvent](#).

Reimplemented from [QCPLayerable](#).

6.73.3.8 selectionChanged

```
void QCPTextElement::selectionChanged (
    bool selected ) [signal]
```

This signal is emitted when the selection state has changed to *selected*, either by user interaction or by a direct call to [setSelected](#).

See also

[setSelected](#), [setSelectable](#)

6.73.3.9 selectTest()

```
double QCPElement::selectTest (
    const QPointF & pos,
    bool onlySelectable,
    QVariant * details = 0 ) const [virtual]
```

Returns $0.99 * \text{selectionTolerance}$ (see [QCustomPlot::setSelectionTolerance](#)) when *pos* is within the bounding box of the text element's text. Note that this bounding box is updated in the draw call.

If *pos* is outside the text's bounding box or if *onlySelectable* is true and this text element is not selectable ([setSelectable](#)), returns -1.

Reimplemented from [QCPLayoutElement](#).

6.73.3.10 setFont()

```
void QCPElement::setFont (
    const QFont & font )
```

Sets the *font* of the text.

See also

[setTextColor](#), [setSelectedFont](#)

6.73.3.11 setSelectable()

```
void QCPElement::setSelectable (
    bool selectable )
```

Sets whether the user may select this text element.

Note that even when *selectable* is set to *false*, the selection state may be changed programmatically via [setSelected](#).

6.73.3.12 setSelected()

```
void QCPElement::setSelected (
    bool selected )
```

Sets the selection state of this text element to *selected*. If the selection has changed, [selectionChanged](#) is emitted.

Note that this function can change the selection state independently of the current [setSelectable](#) state.

6.73.3.13 setSelectedFont()

```
void QCPTextElement::setSelectedFont (
    const QFont & font )
```

Sets the *font* of the text that will be used if the text element is selected ([setSelected](#)).

See also

[setFont](#)

6.73.3.14 setSelectedTextColor()

```
void QCPTextElement::setSelectedTextColor (
    const QColor & color )
```

Sets the *color* of the text that will be used if the text element is selected ([setSelected](#)).

See also

[setTextColor](#)

6.73.3.15 setText()

```
void QCPTextElement::setText (
    const QString & text )
```

Sets the text that will be displayed to *text*. Multiple lines can be created by insertion of "\n".

See also

[setFont](#), [setTextColor](#), [setTextFlags](#)

6.73.3.16 setTextColor()

```
void QCPTextElement::setTextColor (
    const QColor & color )
```

Sets the *color* of the text.

See also

[setFont](#), [setSelectedTextColor](#)

6.73.3.17 setTextFlags()

```
void QCPTextElement::setTextFlags (
    int flags )
```

Sets options for text alignment and wrapping behaviour. *flags* is a bitwise OR-combination of `Qt::Alignment`↔`Flag` and `Qt::TextFlag` enums.

Possible enums are:

- `Qt::AlignLeft`
- `Qt::AlignRight`
- `Qt::AlignHCenter`
- `Qt::AlignJustify`
- `Qt::AlignTop`
- `Qt::AlignBottom`
- `Qt::AlignVCenter`
- `Qt::AlignCenter`
- `Qt::TextDontClip`
- `Qt::TextSingleLine`
- `Qt::TextExpandTabs`
- `Qt::TextShowMnemonic`
- `Qt::TextWordWrap`
- `Qt::TextIncludeTrailingSpaces`

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

- `/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.`↔`h`
- `/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.`↔`cpp`

6.74 QCPVector2D Class Reference

Represents two doubles as a mathematical 2D vector.

Public Member Functions

- [QCPVector2D](#) ()
- [QCPVector2D](#) (double x, double y)
- [QCPVector2D](#) (const QPoint &point)
- [QCPVector2D](#) (const QPointF &point)
- double **x** () const
- double **y** () const
- double & **rx** ()
- double & **ry** ()
- void [setX](#) (double x)
- void [setY](#) (double y)
- double [length](#) () const
- double [lengthSquared](#) () const
- QPoint [toPoint](#) () const
- QPointF [toPointF](#) () const
- bool [isNull](#) () const
- void [normalize](#) ()
- [QCPVector2D](#) [normalized](#) () const
- [QCPVector2D](#) [perpendicular](#) () const
- double [dot](#) (const [QCPVector2D](#) &vec) const
- double [distanceSquaredToLine](#) (const [QCPVector2D](#) &start, const [QCPVector2D](#) &end) const
- double [distanceSquaredToLine](#) (const QLineF &line) const
- double [distanceToStraightLine](#) (const [QCPVector2D](#) &base, const [QCPVector2D](#) &direction) const
- [QCPVector2D](#) & [operator*=](#) (double factor)
- [QCPVector2D](#) & [operator/=](#) (double divisor)
- [QCPVector2D](#) & [operator+=](#) (const [QCPVector2D](#) &vector)
- [QCPVector2D](#) & [operator-=](#) (const [QCPVector2D](#) &vector)

Friends

- const [QCPVector2D](#) [operator*](#) (double factor, const [QCPVector2D](#) &vec)
- const [QCPVector2D](#) [operator*](#) (const [QCPVector2D](#) &vec, double factor)
- const [QCPVector2D](#) [operator/](#) (const [QCPVector2D](#) &vec, double divisor)
- const [QCPVector2D](#) [operator+](#) (const [QCPVector2D](#) &vec1, const [QCPVector2D](#) &vec2)
- const [QCPVector2D](#) [operator-](#) (const [QCPVector2D](#) &vec1, const [QCPVector2D](#) &vec2)
- const [QCPVector2D](#) [operator-](#) (const [QCPVector2D](#) &vec)

Related Functions

(Note that these are not member functions.)

- QDebug [operator<<](#) (QDebug d, const [QCPVector2D](#) &vec)

6.74.1 Detailed Description

Represents two doubles as a mathematical 2D vector.

This class acts as a replacement for QVector2D with the advantage of double precision instead of single, and some convenience methods tailored for the [QCustomPlot](#) library.

6.74.2 Constructor & Destructor Documentation

6.74.2.1 QCPVector2D() [1/4]

```
QCPVector2D::QCPVector2D ( )
```

Creates a [QCPVector2D](#) object and initializes the x and y coordinates to 0.

6.74.2.2 QCPVector2D() [2/4]

```
QCPVector2D::QCPVector2D (
    double x,
    double y )
```

Creates a [QCPVector2D](#) object and initializes the x and y coordinates with the specified values.

6.74.2.3 QCPVector2D() [3/4]

```
QCPVector2D::QCPVector2D (
    const QPoint & point )
```

Creates a [QCPVector2D](#) object and initializes the x and y coordinates respective coordinates of the specified *point*.

6.74.2.4 QCPVector2D() [4/4]

```
QCPVector2D::QCPVector2D (
    const QPointF & point )
```

Creates a [QCPVector2D](#) object and initializes the x and y coordinates respective coordinates of the specified *point*.

6.74.3 Member Function Documentation

6.74.3.1 distanceSquaredToLine() [1/2]

```
double QCPVector2D::distanceSquaredToLine (
    const QCPVector2D & start,
    const QCPVector2D & end ) const
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Returns the squared shortest distance of this vector (interpreted as a point) to the finite line segment given by *start* and *end*.

See also

[distanceToStraightLine](#)

6.74.3.2 distanceSquaredToLine() [2/2]

```
double QCPVector2D::distanceSquaredToLine (  
    const QLineF & line ) const
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Returns the squared shortest distance of this vector (interpreted as a point) to the finite line segment given by *line*.

See also

[distanceToStraightLine](#)

6.74.3.3 distanceToStraightLine()

```
double QCPVector2D::distanceToStraightLine (  
    const QCPVector2D & base,  
    const QCPVector2D & direction ) const
```

Returns the shortest distance of this vector (interpreted as a point) to the infinite straight line given by a *base* point and a *direction* vector.

See also

[distanceSquaredToLine](#)

6.74.3.4 dot()

```
double QCPVector2D::dot (  
    const QCPVector2D & vec ) const [inline]
```

Returns the dot/scalar product of this vector with the specified vector *vec*.

6.74.3.5 isNull()

```
bool QCPVector2D::isNull ( ) const [inline]
```

Returns whether this vector is null. A vector is null if `isNull` returns true for both x and y coordinates, i.e. if both are binary equal to 0.

6.74.3.6 `length()`

```
double QCPVector2D::length ( ) const [inline]
```

Returns the length of this vector.

See also

[lengthSquared](#)

6.74.3.7 `lengthSquared()`

```
double QCPVector2D::lengthSquared ( ) const [inline]
```

Returns the squared length of this vector. In some situations, e.g. when just trying to find the shortest vector of a group, this is faster than calculating [length](#), because it avoids calculation of a square root.

See also

[length](#)

6.74.3.8 `normalize()`

```
void QCPVector2D::normalize ( )
```

Normalizes this vector. After this operation, the length of the vector is equal to 1.

See also

[normalized](#), [length](#), [lengthSquared](#)

6.74.3.9 `normalized()`

```
QCPVector2D QCPVector2D::normalized ( ) const
```

Returns a normalized version of this vector. The length of the returned vector is equal to 1.

See also

[normalize](#), [length](#), [lengthSquared](#)

6.74.3.10 operator*=()

```
QCPVector2D & QCPVector2D::operator*= (
    double factor )
```

Scales this vector by the given *factor*, i.e. the x and y components are multiplied by *factor*.

6.74.3.11 operator+=()

```
QCPVector2D & QCPVector2D::operator+= (
    const QCPVector2D & vector )
```

Adds the given *vector* to this vector component-wise.

6.74.3.12 operator-= ()

```
QCPVector2D & QCPVector2D::operator-= (
    const QCPVector2D & vector )
```

subtracts the given *vector* from this vector component-wise.

6.74.3.13 operator/= ()

```
QCPVector2D & QCPVector2D::operator/= (
    double divisor )
```

Scales this vector by the given *divisor*, i.e. the x and y components are divided by *divisor*.

6.74.3.14 perpendicular ()

```
QCPVector2D QCPVector2D::perpendicular ( ) const [inline]
```

Returns a vector perpendicular to this vector, with the same length.

6.74.3.15 setX ()

```
void QCPVector2D::setX (
    double x ) [inline]
```

Sets the x coordinate of this vector to *x*.

See also

[setY](#)

6.74.3.16 setY()

```
void QCPVector2D::setY (
    double y ) [inline]
```

Sets the y coordinate of this vector to *y*.

See also

[setX](#)

6.74.3.17 toPoint()

```
QPoint QCPVector2D::toPoint ( ) const [inline]
```

Returns a QPoint which has the x and y coordinates of this vector, truncating any floating point information.

See also

[toPointF](#)

6.74.3.18 toPointF()

```
QPointF QCPVector2D::toPointF ( ) const [inline]
```

Returns a QPointF which has the x and y coordinates of this vector.

See also

[toPoint](#)

6.74.4 Friends And Related Function Documentation

6.74.4.1 operator<<()

```
QDebug operator<< (
    QDebug d,
    const QCPVector2D & vec ) [related]
```

Prints *vec* in a human readable format to the qDebug output.

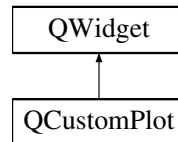
The documentation for this class was generated from the following files:

- [/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h](#)↔
- [/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp](#)↔

6.75 QCustomPlot Class Reference

The central class of the library. This is the QWidget which displays the plot and interacts with the user.

Inheritance diagram for QCustomPlot:



Public Types

- enum [LayerInsertMode](#) { [limBelow](#), [limAbove](#) }
- enum [RefreshPriority](#) { [rplImmediateRefresh](#), [rpQueuedRefresh](#), [rpRefreshHint](#), [rpQueuedReplot](#) }

Signals

- void [mouseDoubleClick](#) (QMouseEvent *event)
- void [mousePress](#) (QMouseEvent *event)
- void [mouseMove](#) (QMouseEvent *event)
- void [mouseRelease](#) (QMouseEvent *event)
- void [mouseWheel](#) (QWheelEvent *event)
- void [plottableClick](#) (QCPAbstractPlottable *plottable, int dataIndex, QMouseEvent *event)
- void [plottableDoubleClick](#) (QCPAbstractPlottable *plottable, int dataIndex, QMouseEvent *event)
- void [itemClick](#) (QCPAbstractItem *item, QMouseEvent *event)
- void [itemDoubleClick](#) (QCPAbstractItem *item, QMouseEvent *event)
- void [axisClick](#) (QCPAxis *axis, QCPAxis::SelectablePart part, QMouseEvent *event)
- void [axisDoubleClick](#) (QCPAxis *axis, QCPAxis::SelectablePart part, QMouseEvent *event)
- void [legendClick](#) (QCPLegend *legend, QCPAbstractLegendItem *item, QMouseEvent *event)
- void [legendDoubleClick](#) (QCPLegend *legend, QCPAbstractLegendItem *item, QMouseEvent *event)
- void [selectionChangedByUser](#) ()
- void [beforeReplot](#) ()
- void [afterReplot](#) ()

Public Member Functions

- [QCustomPlot](#) (QWidget *parent=0)
- QRect [viewport](#) () const
- double [bufferDevicePixelRatio](#) () const
- QPixmap [background](#) () const
- bool [backgroundScaled](#) () const
- Qt::AspectRatioMode [backgroundScaledMode](#) () const
- QCPLayoutGrid * [plotLayout](#) () const
- QCP::AntialiasedElements [antialiasedElements](#) () const
- QCP::AntialiasedElements [notAntialiasedElements](#) () const
- bool [autoAddPlottableToLegend](#) () const
- const QCP::Interactions [interactions](#) () const
- int [selectionTolerance](#) () const
- bool [noAntialiasingOnDrag](#) () const

- QCP::PlottingHints **plottingHints** () const
- Qt::KeyboardModifier **multiSelectModifier** () const
- QCP::SelectionMode **selectionRectMode** () const
- QCPSelectionRect * **selectionRect** () const
- bool **openGl** () const
- void **setViewport** (const QRect &rect)
- void **setBufferDevicePixelRatio** (double ratio)
- void **setBackground** (const QPixmap &pm)
- void **setBackground** (const QPixmap &pm, bool scaled, Qt::AspectRatioMode mode=Qt::KeepAspectRatio, ByExpanding)
- void **setBackground** (const QBrush &brush)
- void **setBackgroundScaled** (bool scaled)
- void **setBackgroundScaledMode** (Qt::AspectRatioMode mode)
- void **setAntialiasedElements** (const QCP::AntialiasedElements &antialiasedElements)
- void **setAntialiasedElement** (QCP::AntialiasedElement antialiasedElement, bool enabled=true)
- void **setNotAntialiasedElements** (const QCP::AntialiasedElements ¬AntialiasedElements)
- void **setNotAntialiasedElement** (QCP::AntialiasedElement notAntialiasedElement, bool enabled=true)
- void **setAutoAddPlottableToLegend** (bool on)
- void **setInteractions** (const QCP::Interactions &interactions)
- void **setInteraction** (const QCP::Interaction &interaction, bool enabled=true)
- void **setSelectionTolerance** (int pixels)
- void **setNoAntialiasingOnDrag** (bool enabled)
- void **setPlottingHints** (const QCP::PlottingHints &hints)
- void **setPlottingHint** (QCP::PlottingHint hint, bool enabled=true)
- void **setMultiSelectModifier** (Qt::KeyboardModifier modifier)
- void **setSelectionMode** (QCP::SelectionMode mode)
- void **setSelectionRect** (QCPSelectionRect *selectionRect)
- void **setOpenGl** (bool enabled, int multisampling=16)
- QCPAbstractPlottable * **plottable** (int index)
- QCPAbstractPlottable * **plottable** ()
- bool **removePlottable** (QCPAbstractPlottable *plottable)
- bool **removePlottable** (int index)
- int **clearPlottables** ()
- int **plottableCount** () const
- QList< QCPAbstractPlottable * > **selectedPlottables** () const
- QCPAbstractPlottable * **plottableAt** (const QPointF &pos, bool onlySelectable=false) const
- bool **hasPlottable** (QCPAbstractPlottable *plottable) const
- QCPGraph * **graph** (int index) const
- QCPGraph * **graph** () const
- QCPGraph * **addGraph** (QCPAxis *keyAxis=0, QCPAxis *valueAxis=0)
- bool **removeGraph** (QCPGraph *graph)
- bool **removeGraph** (int index)
- int **clearGraphs** ()
- int **graphCount** () const
- QList< QCPGraph * > **selectedGraphs** () const
- QCPAbstractItem * **item** (int index) const
- QCPAbstractItem * **item** () const
- bool **removeItem** (QCPAbstractItem *item)
- bool **removeItem** (int index)
- int **clearItems** ()
- int **itemCount** () const
- QList< QCPAbstractItem * > **selectedItems** () const
- QCPAbstractItem * **itemAt** (const QPointF &pos, bool onlySelectable=false) const
- bool **hasItem** (QCPAbstractItem *item) const
- QCPLayer * **layer** (const QString &name) const

- [QCPLayer](#) * [layer](#) (int index) const
- [QCPLayer](#) * [currentLayer](#) () const
- bool [setCurrentLayer](#) (const QString &name)
- bool [setCurrentLayer](#) ([QCPLayer](#) *[layer](#))
- int [layerCount](#) () const
- bool [addLayer](#) (const QString &name, [QCPLayer](#) *otherLayer=0, [LayerInsertMode](#) insertMode=[limAbove](#))
- bool [removeLayer](#) ([QCPLayer](#) *[layer](#))
- bool [moveLayer](#) ([QCPLayer](#) *[layer](#), [QCPLayer](#) *otherLayer, [LayerInsertMode](#) insertMode=[limAbove](#))
- int [axisRectCount](#) () const
- [QCPAxisRect](#) * [axisRect](#) (int index=0) const
- QList< [QCPAxisRect](#) * > [axisRects](#) () const
- [QCPLayoutElement](#) * [layoutElementAt](#) (const QPointF &pos) const
- [QCPAxisRect](#) * [axisRectAt](#) (const QPointF &pos) const
- Q_SLOT void [rescaleAxes](#) (bool onlyVisiblePlottables=false)
- QList< [QCPAxis](#) * > [selectedAxes](#) () const
- QList< [QCPLegend](#) * > [selectedLegends](#) () const
- Q_SLOT void [deselectAll](#) ()
- bool [savePdf](#) (const QString &fileName, int width=0, int height=0, [QCP::ExportPen](#) exportPen=[QCP::epCosmetic](#), const QString &pdfCreator=QString(), const QString &pdfTitle=QString())
- bool [savePng](#) (const QString &fileName, int width=0, int height=0, double scale=1.0, int quality=-1, int resolution=96, [QCP::ResolutionUnit](#) resolutionUnit=[QCP::ruDotsPerInch](#))
- bool [saveJpg](#) (const QString &fileName, int width=0, int height=0, double scale=1.0, int quality=-1, int resolution=96, [QCP::ResolutionUnit](#) resolutionUnit=[QCP::ruDotsPerInch](#))
- bool [saveBmp](#) (const QString &fileName, int width=0, int height=0, double scale=1.0, int resolution=96, [QCP::ResolutionUnit](#) resolutionUnit=[QCP::ruDotsPerInch](#))
- bool [saveRastered](#) (const QString &fileName, int width, int height, double scale, const char *format, int quality=-1, int resolution=96, [QCP::ResolutionUnit](#) resolutionUnit=[QCP::ruDotsPerInch](#))
- QPixmap [toPixmap](#) (int width=0, int height=0, double scale=1.0)
- void [toPainter](#) ([QCPPainter](#) *painter, int width=0, int height=0)
- Q_SLOT void [replot](#) ([QCustomPlot::RefreshPriority](#) refreshPriority=[QCustomPlot::rpRefreshHint](#))

Public Attributes

- [QCPAxis](#) * [xAxis](#)
- [QCPAxis](#) * [yAxis](#)
- [QCPAxis](#) * [xAxis2](#)
- [QCPAxis](#) * [yAxis2](#)
- [QCPLegend](#) * [legend](#)

Protected Member Functions

- virtual QSize [minimumSizeHint](#) () const Q_DECL_OVERRIDE
- virtual QSize [sizeHint](#) () const Q_DECL_OVERRIDE
- virtual void [paintEvent](#) (QPaintEvent *event) Q_DECL_OVERRIDE
- virtual void [resizeEvent](#) (QResizeEvent *event) Q_DECL_OVERRIDE
- virtual void [mouseDoubleClickEvent](#) (QMouseEvent *event) Q_DECL_OVERRIDE
- virtual void [mousePressEvent](#) (QMouseEvent *event) Q_DECL_OVERRIDE
- virtual void [mouseMoveEvent](#) (QMouseEvent *event) Q_DECL_OVERRIDE
- virtual void [mouseReleaseEvent](#) (QMouseEvent *event) Q_DECL_OVERRIDE
- virtual void [wheelEvent](#) (QWheelEvent *event) Q_DECL_OVERRIDE
- virtual void [draw](#) ([QCPPainter](#) *painter)
- virtual void [updateLayout](#) ()
- virtual void [axisRemoved](#) ([QCPAxis](#) *axis)

- virtual void **legendRemoved** ([QCPLegend](#) *legend)
- virtual Q_SLOT void **processRectSelection** (QRect rect, QMouseEvent *event)
- virtual Q_SLOT void **processRectZoom** (QRect rect, QMouseEvent *event)
- virtual Q_SLOT void **processPointSelection** (QMouseEvent *event)
- bool **registerPlottable** ([QCPAbstractPlottable](#) *plottable)
- bool **registerGraph** ([QCPGraph](#) *graph)
- bool **registerItem** ([QCPAbstractItem](#) *item)
- void **updateLayerIndices** () const
- [QCPLayerable](#) * **layerableAt** (const QPointF &pos, bool onlySelectable, QVariant *selectionDetails=0) const
- QList< [QCPLayerable](#) * > **layerableListAt** (const QPointF &pos, bool onlySelectable, QList< QVariant > *selectionDetails=0) const
- void **drawBackground** ([QCPPainter](#) *painter)
- void **setupPaintBuffers** ()
- [QCPAbstractPaintBuffer](#) * **createPaintBuffer** ()
- bool **hasInvalidatedPaintBuffers** ()
- bool **setupOpenGL** ()
- void **freeOpenGL** ()

Protected Attributes

- QRect **mViewport**
- double **mBufferDevicePixelRatio**
- [QCPLayoutGrid](#) * **mPlotLayout**
- bool **mAutoAddPlottableToLegend**
- QList< [QCPAbstractPlottable](#) * > **mPlottables**
- QList< [QCPGraph](#) * > **mGraphs**
- QList< [QCPAbstractItem](#) * > **mItems**
- QList< [QCPLayer](#) * > **mLayers**
- QCP::AntialiasedElements **mAntialiasedElements**
- QCP::AntialiasedElements **mNotAntialiasedElements**
- QCP::Interactions **mInteractions**
- int **mSelectionTolerance**
- bool **mNoAntialiasingOnDrag**
- QBrush **mBackgroundBrush**
- QPixmap **mBackgroundPixmap**
- QPixmap **mScaledBackgroundPixmap**
- bool **mBackgroundScaled**
- Qt::AspectRatioMode **mBackgroundScaledMode**
- [QCPLayer](#) * **mCurrentLayer**
- QCP::PlottingHints **mPlottingHints**
- Qt::KeyboardModifier **mMultiSelectModifier**
- [QCP::SelectionRectMode](#) **mSelectionRectMode**
- [QCPSelectionRect](#) * **mSelectionRect**
- bool **mOpenGL**
- QList< QSharedPointer< [QCPAbstractPaintBuffer](#) > > **mPaintBuffers**
- QPoint **mMousePressPos**
- bool **mMouseHasMoved**
- QPointer< [QCPLayerable](#) > **mMouseEventLayerable**
- QVariant **mMouseEventLayerableDetails**
- bool **mReplotting**
- bool **mReplotQueued**
- int **mOpenGLMultisamples**
- QCP::AntialiasedElements **mOpenGLAntialiasedElementsBackup**
- bool **mOpenGLCacheLabelsBackup**

Friends

- class **QCPLegend**
- class **QCPAxis**
- class **QCPLayer**
- class **QCPAxisRect**
- class **QCPAbstractPlottable**
- class **QCPGraph**
- class **QCPAbstractItem**

6.75.1 Detailed Description

The central class of the library. This is the QWidget which displays the plot and interacts with the user.

For tutorials on how to use [QCustomPlot](http://www.qcustomplot.com/), see the website <http://www.qcustomplot.com/>

6.75.2 Member Enumeration Documentation

6.75.2.1 LayerInsertMode

enum [QCustomPlot::LayerInsertMode](#)

Defines how a layer should be inserted relative to an other layer.

See also

[addLayer](#), [moveLayer](#)

Enumerator

limBelow	Layer is inserted below other layer.
limAbove	Layer is inserted above other layer.

6.75.2.2 RefreshPriority

enum [QCustomPlot::RefreshPriority](#)

Defines with what timing the [QCustomPlot](#) surface is refreshed after a replot.

See also

[replot](#)

Enumerator

rpImmediateRefresh	Replots immediately and repaints the widget immediately by calling QWidget::repaint() after the replot.
rpQueuedRefresh	Replots immediately, but queues the widget repaint, by calling QWidget::update() after the replot. This way multiple redundant widget repaints can be avoided.
rpRefreshHint	Whether to use immediate or queued refresh depends on whether the plotting hint QCP::phImmediateRefresh is set, see setPlottingHints .
rpQueuedReplot	Queues the entire replot for the next event loop iteration. This way multiple redundant replots can be avoided. The actual replot is then done with rpRefreshHint priority.

6.75.3 Constructor & Destructor Documentation

6.75.3.1 QCustomPlot()

```
QCustomPlot::QCustomPlot (
    QWidget * parent = 0 ) [explicit]
```

Constructs a [QCustomPlot](#) and sets reasonable default values.

6.75.4 Member Function Documentation

6.75.4.1 addGraph()

```
QCPGraph * QCustomPlot::addGraph (
    QCPAxis * keyAxis = 0,
    QCPAxis * valueAxis = 0 )
```

Creates a new graph inside the plot. If *keyAxis* and *valueAxis* are left unspecified (0), the bottom (xAxis) is used as key and the left (yAxis) is used as value axis. If specified, *keyAxis* and *valueAxis* must reside in this [QCustomPlot](#).

keyAxis will be used as key axis (typically "x") and *valueAxis* as value axis (typically "y") for the graph.

Returns a pointer to the newly created graph, or 0 if adding the graph failed.

See also

[graph](#), [graphCount](#), [removeGraph](#), [clearGraphs](#)

6.75.4.2 addLayer()

```
bool QCustomPlot::addLayer (
    const QString & name,
    QCPLayer * otherLayer = 0,
    QCustomPlot::LayerInsertMode insertMode = limAbove )
```

Adds a new layer to this [QCustomPlot](#) instance. The new layer will have the name *name*, which must be unique. Depending on *insertMode*, it is positioned either below or above *otherLayer*.

Returns true on success, i.e. if there is no other layer named *name* and *otherLayer* is a valid layer inside this [QCustomPlot](#).

If *otherLayer* is 0, the highest layer in the [QCustomPlot](#) will be used.

For an explanation of what layers are in [QCustomPlot](#), see the documentation of [QCPLayer](#).

See also

[layer](#), [moveLayer](#), [removeLayer](#)

6.75.4.3 afterReplot

```
void QCustomPlot::afterReplot ( ) [signal]
```

This signal is emitted immediately after a replot has taken place (caused by a call to the slot [replot](#)).

It is safe to mutually connect the replot slot with this signal on two [QCustomPlots](#) to make them replot synchronously, it won't cause an infinite recursion.

See also

[replot](#), [beforeReplot](#)

6.75.4.4 axisClick

```
void QCustomPlot::axisClick (
    QCPLAxis * axis,
    QCPLAxis::SelectablePart part,
    QMouseEvent * event ) [signal]
```

This signal is emitted when an axis is clicked.

event is the mouse event that caused the click, *axis* is the axis that received the click and *part* indicates the part of the axis that was clicked.

See also

[axisDoubleClick](#)

6.75.4.5 axisDoubleClick

```
void QCustomPlot::axisDoubleClick (
    QCPAxis * axis,
    QCPAxis::SelectablePart part,
    QMouseEvent * event ) [signal]
```

This signal is emitted when an axis is double clicked.

event is the mouse event that caused the click, *axis* is the axis that received the click and *part* indicates the part of the axis that was clicked.

See also

[axisClick](#)

6.75.4.6 axisRect()

```
QCPAxisRect * QCustomPlot::axisRect (
    int index = 0 ) const
```

Returns the axis rect with *index*.

Initially, only one axis rect (with index 0) exists in the plot. If multiple axis rects were added, all of them may be accessed with this function in a linear fashion (even when they are nested in a layout hierarchy or inside other axis rects via [QCPAxisRect::insetLayout](#)).

See also

[axisRectCount](#), [axisRects](#)

6.75.4.7 axisRectAt()

```
QCPAxisRect * QCustomPlot::axisRectAt (
    const QPointF & pos ) const
```

Returns the layout element of type [QCPAxisRect](#) at pixel position *pos*. This method ignores other layout elements even if they are visually in front of the axis rect (e.g. a [QCPLegend](#)). If there is no axis rect at that position, returns 0.

Only visible axis rects are used. If [QCPLayoutElement::setVisible](#) on the axis rect itself or on any of its parent elements is set to false, it will not be considered.

See also

[layoutElementAt](#)

6.75.4.8 axisRectCount()

```
int QCustomPlot::axisRectCount ( ) const
```

Returns the number of axis rects in the plot.

All axis rects can be accessed via [QCustomPlot::axisRect\(\)](#).

Initially, only one axis rect exists in the plot.

See also

[axisRect](#), [axisRects](#)

6.75.4.9 axisRects()

```
QList< QCPAxisRect * > QCustomPlot::axisRects ( ) const
```

Returns all axis rects in the plot.

See also

[axisRectCount](#), [axisRect](#)

6.75.4.10 beforeReplot

```
void QCustomPlot::beforeReplot ( ) [signal]
```

This signal is emitted immediately before a replot takes place (caused by a call to the slot [replot](#)).

It is safe to mutually connect the replot slot with this signal on two QCustomPlots to make them replot synchronously, it won't cause an infinite recursion.

See also

[replot](#), [afterReplot](#)

6.75.4.11 clearGraphs()

```
int QCustomPlot::clearGraphs ( )
```

Removes all graphs from the plot and deletes them. Corresponding legend items are also removed from the default legend ([QCustomPlot::legend](#)).

Returns the number of graphs removed.

See also

[removeGraph](#)

6.75.4.12 clearItems()

```
int QCustomPlot::clearItems ( )
```

Removes all items from the plot and deletes them.

Returns the number of items removed.

See also

[removeItem](#)

6.75.4.13 clearPlottables()

```
int QCustomPlot::clearPlottables ( )
```

Removes all plottables from the plot and deletes them. Corresponding legend items are also removed from the default legend ([QCustomPlot::legend](#)).

Returns the number of plottables removed.

See also

[removePlottable](#)

6.75.4.14 currentLayer()

```
QCPLayer * QCustomPlot::currentLayer ( ) const
```

Returns the layer that is set as current layer (see [setCurrentLayer](#)).

6.75.4.15 deselectAll()

```
void QCustomPlot::deselectAll ( )
```

Deselects all layerables (plottables, items, axes, legends,...) of the [QCustomPlot](#).

Since calling this function is not a user interaction, this does not emit the [selectionChangedByUser](#) signal. The individual selectionChanged signals are emitted though, if the objects were previously selected.

See also

[setInteractions](#), [selectedPlottables](#), [selectedItems](#), [selectedAxes](#), [selectedLegends](#)

6.75.4.16 `graph()` [1/2]

```
QCPGraph * QCustomPlot::graph (
    int index ) const
```

Returns the graph with *index*. If the index is invalid, returns 0.

There is an overloaded version of this function with no parameter which returns the last created graph, see [QCustomPlot::graph\(\)](#)

See also

[graphCount](#), [addGraph](#)

6.75.4.17 `graph()` [2/2]

```
QCPGraph * QCustomPlot::graph ( ) const
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Returns the last graph, that was created with [addGraph](#). If there are no graphs in the plot, returns 0.

See also

[graphCount](#), [addGraph](#)

6.75.4.18 `graphCount()`

```
int QCustomPlot::graphCount ( ) const
```

Returns the number of currently existing graphs in the plot

See also

[graph](#), [addGraph](#)

6.75.4.19 `hasInvalidatedPaintBuffers()`

```
bool QCustomPlot::hasInvalidatedPaintBuffers ( ) [protected]
```

This method returns whether any of the paint buffers held by this [QCustomPlot](#) instance are invalidated.

If any buffer is invalidated, a partial replot ([QCPLayer::replot](#)) is not allowed and always causes a full replot ([QCustomPlot::replot](#)) of all layers. This is the case when for example the layer order has changed, new layers were added, layers were removed, or layer modes were changed ([QCPLayer::setMode](#)).

See also

[QCPAbstractPaintBuffer::setInvalidated](#)

6.75.4.20 hasItem()

```
bool QCustomPlot::hasItem (
    QCPAbstractItem * item ) const
```

Returns whether this [QCustomPlot](#) contains the *item*.

See also

[item](#)

6.75.4.21 hasPlottable()

```
bool QCustomPlot::hasPlottable (
    QCPAbstractPlottable * plottable ) const
```

Returns whether this [QCustomPlot](#) instance contains the *plottable*.

6.75.4.22 item() [1/2]

```
QCPAbstractItem * QCustomPlot::item (
    int index ) const
```

Returns the item with *index*. If the index is invalid, returns 0.

There is an overloaded version of this function with no parameter which returns the last added item, see [QCustomPlot::item\(\)](#)

See also

[itemCount](#)

6.75.4.23 item() [2/2]

```
QCPAbstractItem * QCustomPlot::item ( ) const
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Returns the last item that was added to this plot. If there are no items in the plot, returns 0.

See also

[itemCount](#)

6.75.4.24 itemAt()

```
QCPAbstractItem * QCustomPlot::itemAt (
    const QPointF & pos,
    bool onlySelectable = false ) const
```

Returns the item at the pixel position *pos*. Items that only consist of single lines (e.g. [QCPItemLine](#) or [QCPItemCurve](#)) have a tolerance band around them, see [setSelectionTolerance](#). If multiple items come into consideration, the one closest to *pos* is returned.

If *onlySelectable* is true, only items that are selectable ([QCPAbstractItem::setSelectable](#)) are considered.

If there is no item at *pos*, the return value is 0.

See also

[plottableAt](#), [layoutElementAt](#)

6.75.4.25 itemClick

```
void QCustomPlot::itemClick (
    QCPAbstractItem * item,
    QMouseEvent * event ) [signal]
```

This signal is emitted when an item is clicked.

event is the mouse event that caused the click and *item* is the item that received the click.

See also

[itemDoubleClick](#)

6.75.4.26 itemCount()

```
int QCustomPlot::itemCount ( ) const
```

Returns the number of currently existing items in the plot

See also

[item](#)

6.75.4.27 itemDoubleClick

```
void QCustomPlot::itemDoubleClick (
    QCPAbstractItem * item,
    QMouseEvent * event ) [signal]
```

This signal is emitted when an item is double clicked.

event is the mouse event that caused the click and *item* is the item that received the click.

See also

[itemClick](#)

6.75.4.28 layer() [1/2]

```
QCPLayer * QCustomPlot::layer (
    const QString & name ) const
```

Returns the layer with the specified *name*. If there is no layer with the specified name, 0 is returned.

Layer names are case-sensitive.

See also

[addLayer](#), [moveLayer](#), [removeLayer](#)

6.75.4.29 layer() [2/2]

```
QCPLayer * QCustomPlot::layer (
    int index ) const
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Returns the layer by *index*. If the index is invalid, 0 is returned.

See also

[addLayer](#), [moveLayer](#), [removeLayer](#)

6.75.4.30 layerCount()

```
int QCustomPlot::layerCount ( ) const
```

Returns the number of currently existing layers in the plot

See also

[layer](#), [addLayer](#)

6.75.4.31 layoutElementAt()

```
QCPLayoutElement * QCustomPlot::layoutElementAt (
    const QPointF & pos ) const
```

Returns the layout element at pixel position *pos*. If there is no element at that position, returns 0.

Only visible elements are used. If [QCPLayoutElement::setVisible](#) on the element itself or on any of its parent elements is set to false, it will not be considered.

See also

[itemAt](#), [plottableAt](#)

6.75.4.32 legendClick

```
void QCustomPlot::legendClick (
    QCPLegend * legend,
    QCPAbstractLegendItem * item,
    QMouseEvent * event ) [signal]
```

This signal is emitted when a legend (item) is clicked.

event is the mouse event that caused the click, *legend* is the legend that received the click and *item* is the legend item that received the click. If only the legend and no item is clicked, *item* is 0. This happens for a click inside the legend padding or the space between two items.

See also

[legendDoubleClick](#)

6.75.4.33 legendDoubleClick

```
void QCustomPlot::legendDoubleClick (
    QCPLegend * legend,
    QCPAbstractLegendItem * item,
    QMouseEvent * event ) [signal]
```

This signal is emitted when a legend (item) is double clicked.

event is the mouse event that caused the click, *legend* is the legend that received the click and *item* is the legend item that received the click. If only the legend and no item is clicked, *item* is 0. This happens for a click inside the legend padding or the space between two items.

See also

[legendClick](#)

6.75.4.34 mouseDoubleClick

```
void QCustomPlot::mouseDoubleClick (
    QMouseEvent * event ) [signal]
```

This signal is emitted when the [QCustomPlot](#) receives a mouse double click event.

6.75.4.35 mouseMove

```
void QCustomPlot::mouseMove (
    QMouseEvent * event ) [signal]
```

This signal is emitted when the [QCustomPlot](#) receives a mouse move event.

It is emitted before [QCustomPlot](#) handles any other mechanism like range dragging. So a slot connected to this signal can still influence the behaviour e.g. with [QCPAxisRect::setRangeDrag](#) or [QCPAxisRect::setRangeDrag↔Axes](#).

Warning

It is discouraged to change the drag-axes with [QCPAxisRect::setRangeDragAxes](#) here, because the dragging starting point was saved the moment the mouse was pressed. Thus it only has a meaning for the range drag axes that were set at that moment. If you want to change the drag axes, consider doing this in the [mousePress](#) signal instead.

6.75.4.36 mousePress

```
void QCustomPlot::mousePress (
    QMouseEvent * event ) [signal]
```

This signal is emitted when the [QCustomPlot](#) receives a mouse press event.

It is emitted before [QCustomPlot](#) handles any other mechanism like range dragging. So a slot connected to this signal can still influence the behaviour e.g. with [QCPAxisRect::setRangeDrag](#) or [QCPAxisRect::setRangeDrag↔Axes](#).

6.75.4.37 mouseRelease

```
void QCustomPlot::mouseRelease (
    QMouseEvent * event ) [signal]
```

This signal is emitted when the [QCustomPlot](#) receives a mouse release event.

It is emitted before [QCustomPlot](#) handles any other mechanisms like object selection. So a slot connected to this signal can still influence the behaviour e.g. with [setInteractions](#) or [QCPAbstractPlottable::setSelectable](#).

6.75.4.38 mouseWheel

```
void QCustomPlot::mouseWheel (
    QWheelEvent * event ) [signal]
```

This signal is emitted when the [QCustomPlot](#) receives a mouse wheel event.

It is emitted before [QCustomPlot](#) handles any other mechanisms like range zooming. So a slot connected to this signal can still influence the behaviour e.g. with [QCPAxisRect::setRangeZoom](#), [QCPAxisRect::setRangeZoomAxes](#) or [QCPAxisRect::setRangeZoomFactor](#).

6.75.4.39 moveLayer()

```
bool QCustomPlot::moveLayer (
    QCPLayer * layer,
    QCPLayer * otherLayer,
    QCustomPlot::LayerInsertMode insertMode = limAbove )
```

Moves the specified *layer* either above or below *otherLayer*. Whether it's placed above or below is controlled with *insertMode*.

Returns true on success, i.e. when both *layer* and *otherLayer* are valid layers in the [QCustomPlot](#).

See also

[layer](#), [addLayer](#), [moveLayer](#)

6.75.4.40 plotLayout()

```
QCPLayoutGrid * QCustomPlot::plotLayout ( ) const [inline]
```

Returns the top level layout of this [QCustomPlot](#) instance. It is a [QCPLayoutGrid](#), initially containing just one cell with the main [QCPAxisRect](#) inside.

6.75.4.41 `plottable()` [1/2]

```
QCPAbstractPlottable * QCustomPlot::plottable (
    int index )
```

Returns the plottable with *index*. If the index is invalid, returns 0.

There is an overloaded version of this function with no parameter which returns the last added plottable, see [QCustomPlot::plottable\(\)](#)

See also

[plottableCount](#)

6.75.4.42 `plottable()` [2/2]

```
QCPAbstractPlottable * QCustomPlot::plottable ( )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Returns the last plottable that was added to the plot. If there are no plottables in the plot, returns 0.

See also

[plottableCount](#)

6.75.4.43 `plottableAt()`

```
QCPAbstractPlottable * QCustomPlot::plottableAt (
    const QPointF & pos,
    bool onlySelectable = false ) const
```

Returns the plottable at the pixel position *pos*. Plottables that only consist of single lines (like graphs) have a tolerance band around them, see [setSelectionTolerance](#). If multiple plottables come into consideration, the one closest to *pos* is returned.

If *onlySelectable* is true, only plottables that are selectable ([QCPAbstractPlottable::setSelectable](#)) are considered.

If there is no plottable at *pos*, the return value is 0.

See also

[itemAt](#), [layoutElementAt](#)

6.75.4.44 `plottableClick`

```
void QCustomPlot::plottableClick (
    QCPAbstractPlottable * plottable,
    int dataIndex,
    QMouseEvent * event ) [signal]
```

This signal is emitted when a plottable is clicked.

event is the mouse event that caused the click and *plottable* is the plottable that received the click. The parameter *dataIndex* indicates the data point that was closest to the click position.

See also

[plottableDoubleClick](#)

6.75.4.45 `plottableCount()`

```
int QCustomPlot::plottableCount ( ) const
```

Returns the number of currently existing plottables in the plot

See also

[plottable](#)

6.75.4.46 `plottableDoubleClick`

```
void QCustomPlot::plottableDoubleClick (
    QCPAbstractPlottable * plottable,
    int dataIndex,
    QMouseEvent * event ) [signal]
```

This signal is emitted when a plottable is double clicked.

event is the mouse event that caused the click and *plottable* is the plottable that received the click. The parameter *dataIndex* indicates the data point that was closest to the click position.

See also

[plottableClick](#)

6.75.4.47 removeGraph() [1/2]

```
bool QCustomPlot::removeGraph (
    QCPGraph * graph )
```

Removes the specified *graph* from the plot and deletes it. If necessary, the corresponding legend item is also removed from the default legend ([QCustomPlot::legend](#)). If any other graphs in the plot have a channel fill set towards the removed graph, the channel fill property of those graphs is reset to zero (no channel fill).

Returns true on success.

See also

[clearGraphs](#)

6.75.4.48 removeGraph() [2/2]

```
bool QCustomPlot::removeGraph (
    int index )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Removes and deletes the graph by its *index*.

6.75.4.49 removeItem() [1/2]

```
bool QCustomPlot::removeItem (
    QCPAbstractItem * item )
```

Removes the specified item from the plot and deletes it.

Returns true on success.

See also

[clearItems](#)

6.75.4.50 removeItem() [2/2]

```
bool QCustomPlot::removeItem (
    int index )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Removes and deletes the item by its *index*.

6.75.4.51 `removeLayer()`

```
bool QCustomPlot::removeLayer (
    QCPLayer * layer )
```

Removes the specified *layer* and returns true on success.

All layerables (e.g. plottables and items) on the removed layer will be moved to the layer below *layer*. If *layer* is the bottom layer, the layerables are moved to the layer above. In both cases, the total rendering order of all layerables in the `QCustomPlot` is preserved.

If *layer* is the current layer (`setCurrentLayer`), the layer below (or above, if bottom layer) becomes the new current layer.

It is not possible to remove the last layer of the plot.

See also

[layer](#), [addLayer](#), [moveLayer](#)

6.75.4.52 `removePlottable()` [1/2]

```
bool QCustomPlot::removePlottable (
    QCPLAbstractPlottable * plottable )
```

Removes the specified plottable from the plot and deletes it. If necessary, the corresponding legend item is also removed from the default legend (`QCustomPlot::legend`).

Returns true on success.

See also

[clearPlottables](#)

6.75.4.53 `removePlottable()` [2/2]

```
bool QCustomPlot::removePlottable (
    int index )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Removes and deletes the plottable by its *index*.

6.75.4.54 `replot()`

```
void QCustomPlot::replot (
    QCustomPlot::RefreshPriority refreshPriority = QCustomPlot::rpRefreshHint )
```

Causes a complete replot into the internal paint buffer(s). Finally, the widget surface is refreshed with the new buffer contents. This is the method that must be called to make changes to the plot, e.g. on the axis ranges or data points of graphs, visible.

The parameter *refreshPriority* can be used to fine-tune the timing of the replot. For example if your application calls `replot` very quickly in succession (e.g. multiple independent functions change some aspects of the plot and each wants to make sure the change gets replotted), it is advisable to set *refreshPriority* to `QCustomPlot::rpQueued↔Replot`. This way, the actual replotting is deferred to the next event loop iteration. Multiple successive calls of `replot` with this priority will only cause a single replot, avoiding redundant replots and improving performance.

Under a few circumstances, `QCustomPlot` causes a replot by itself. Those are resize events of the `QCustomPlot` widget and user interactions (object selection and range dragging/zooming).

Before the replot happens, the signal `beforeReplot` is emitted. After the replot, `afterReplot` is emitted. It is safe to mutually connect the replot slot with any of those two signals on two `QCustomPlots` to make them replot synchronously, it won't cause an infinite recursion.

If a layer is in mode `QCPLayer::lmBuffered` (`QCPLayer::setMode`), it is also possible to replot only that specific layer via `QCPLayer::replot`. See the documentation there for details.

6.75.4.55 `rescaleAxes()`

```
void QCustomPlot::rescaleAxes (
    bool onlyVisiblePlottables = false )
```

Rescales the axes such that all plottables (like graphs) in the plot are fully visible.

if *onlyVisiblePlottables* is set to true, only the plottables that have their visibility set to true (`QCPLayerable::set↔Visible`), will be used to rescale the axes.

See also

`QCPAbstractPlottable::rescaleAxes`, `QCPAxis::rescale`

6.75.4.56 `saveBmp()`

```
bool QCustomPlot::saveBmp (
    const QString & fileName,
    int width = 0,
    int height = 0,
    double scale = 1.0,
    int resolution = 96,
    QCP::ResolutionUnit resolutionUnit = QCP::ruDotsPerInch )
```

Saves a BMP image file to *fileName* on disc. The output plot will have the dimensions *width* and *height* in pixels, multiplied by *scale*. If either *width* or *height* is zero, the current width and height of the `QCustomPlot` widget is used

instead. Line widths and texts etc. are not scaled up when larger widths/heights are used. If you want that effect, use the *scale* parameter.

For example, if you set both *width* and *height* to 100 and *scale* to 2, you will end up with an image file of size 200*200 in which all graphical elements are scaled up by factor 2 (line widths, texts, etc.). This scaling is not done by stretching a 100*100 image, the result will have full 200*200 pixel resolution.

If you use a high scaling factor, it is recommended to enable antialiasing for all elements by temporarily setting [QCustomPlot::setAntialiasedElements](#) to [QCP::aeAll](#) as this allows [QCustomPlot](#) to place objects with sub-pixel accuracy.

The *resolution* will be written to the image file header and has no direct consequence for the quality or the pixel size. However, if opening the image with a tool which respects the metadata, it will be able to scale the image to match either a given size in real units of length (inch, centimeters, etc.), or the target display DPI. You can specify in which units *resolution* is given, by setting *resolutionUnit*. The *resolution* is converted to the format's expected resolution unit internally.

Returns true on success. If this function fails, most likely the BMP format isn't supported by the system, see Qt docs about [QImageWriter::supportedImageFormats\(\)](#).

The objects of the plot will appear in the current selection state. If you don't want any selected objects to be painted in their selected look, deselect everything with [deselectAll](#) before calling this function.

Warning

If calling this function inside the constructor of the parent of the [QCustomPlot](#) widget (i.e. the [MainWindow](#) constructor, if [QCustomPlot](#) is inside the [MainWindow](#)), always provide explicit non-zero widths and heights. If you leave *width* or *height* as 0 (default), this function uses the current width and height of the [QCustomPlot](#) widget. However, in Qt, these aren't defined yet inside the constructor, so you would get an image that has strange widths/heights.

See also

[savePdf](#), [savePng](#), [saveJpg](#), [saveRastered](#)

6.75.4.57 saveJpg()

```
bool QCustomPlot::saveJpg (
    const QString & fileName,
    int width = 0,
    int height = 0,
    double scale = 1.0,
    int quality = -1,
    int resolution = 96,
    QCP::ResolutionUnit resolutionUnit = QCP::ruDotsPerInch )
```

Saves a JPEG image file to *fileName* on disc. The output plot will have the dimensions *width* and *height* in pixels, multiplied by *scale*. If either *width* or *height* is zero, the current width and height of the [QCustomPlot](#) widget is used instead. Line widths and texts etc. are not scaled up when larger widths/heights are used. If you want that effect, use the *scale* parameter.

For example, if you set both *width* and *height* to 100 and *scale* to 2, you will end up with an image file of size 200*200 in which all graphical elements are scaled up by factor 2 (line widths, texts, etc.). This scaling is not done by stretching a 100*100 image, the result will have full 200*200 pixel resolution.

If you use a high scaling factor, it is recommended to enable antialiasing for all elements by temporarily setting `QCustomPlot::setAntialiasedElements` to `QCP::aeAll` as this allows `QCustomPlot` to place objects with sub-pixel accuracy.

image compression can be controlled with the *quality* parameter which must be between 0 and 100 or -1 to use the default setting.

The *resolution* will be written to the image file header and has no direct consequence for the quality or the pixel size. However, if opening the image with a tool which respects the metadata, it will be able to scale the image to match either a given size in real units of length (inch, centimeters, etc.), or the target display DPI. You can specify in which units *resolution* is given, by setting *resolutionUnit*. The *resolution* is converted to the format's expected resolution unit internally.

Returns true on success. If this function fails, most likely the JPEG format isn't supported by the system, see Qt docs about `QImageWriter::supportedImageFormats()`.

The objects of the plot will appear in the current selection state. If you don't want any selected objects to be painted in their selected look, deselect everything with `deselectAll` before calling this function.

Warning

If calling this function inside the constructor of the parent of the `QCustomPlot` widget (i.e. the `MainWindow` constructor, if `QCustomPlot` is inside the `MainWindow`), always provide explicit non-zero widths and heights. If you leave *width* or *height* as 0 (default), this function uses the current width and height of the `QCustomPlot` widget. However, in Qt, these aren't defined yet inside the constructor, so you would get an image that has strange widths/heights.

See also

[savePdf](#), [savePng](#), [saveBmp](#), [saveRastered](#)

6.75.4.58 savePdf()

```
bool QCustomPlot::savePdf (
    const QString & fileName,
    int width = 0,
    int height = 0,
    QCP::ExportPen exportPen = QCP::epAllowCosmetic,
    const QString & pdfCreator = QString(),
    const QString & pdfTitle = QString() )
```

Saves a PDF with the vectorized plot to the file *fileName*. The axis ratio as well as the scale of texts and lines will be derived from the specified *width* and *height*. This means, the output will look like the normal on-screen output of a `QCustomPlot` widget with the corresponding pixel width and height. If either *width* or *height* is zero, the exported image will have the same dimensions as the `QCustomPlot` widget currently has.

Setting *exportPen* to `QCP::epNoCosmetic` allows to disable the use of cosmetic pens when drawing to the PDF file. Cosmetic pens are pens with numerical width 0, which are always drawn as a one pixel wide line, no matter what zoom factor is set in the PDF-Viewer. For more information about cosmetic pens, see the `QPainter` and `QPen` documentation.

The objects of the plot will appear in the current selection state. If you don't want any selected objects to be painted in their selected look, deselect everything with `deselectAll` before calling this function.

Returns true on success.

Warning

- If you plan on editing the exported PDF file with a vector graphics editor like Inkscape, it is advised to set *exportPen* to [QCP::epNoCosmetic](#) to avoid losing those cosmetic lines (which might be quite many, because cosmetic pens are the default for e.g. axes and tick marks).
- If calling this function inside the constructor of the parent of the [QCustomPlot](#) widget (i.e. the [MainWindow](#) constructor, if [QCustomPlot](#) is inside the [MainWindow](#)), always provide explicit non-zero widths and heights. If you leave *width* or *height* as 0 (default), this function uses the current width and height of the [QCustomPlot](#) widget. However, in Qt, these aren't defined yet inside the constructor, so you would get an image that has strange widths/heights.

pdfCreator and *pdfTitle* may be used to set the according metadata fields in the resulting PDF file.

Note

On Android systems, this method does nothing and issues an according qDebug warning message. This is also the case if for other reasons the define flag `QT_NO_PRINTER` is set.

See also

[savePng](#), [saveBmp](#), [saveJpg](#), [saveRastered](#)

6.75.4.59 `savePng()`

```
bool QCustomPlot::savePng (
    const QString & fileName,
    int width = 0,
    int height = 0,
    double scale = 1.0,
    int quality = -1,
    int resolution = 96,
    QCP::ResolutionUnit resolutionUnit = QCP::ruDotsPerInch )
```

Saves a PNG image file to *fileName* on disc. The output plot will have the dimensions *width* and *height* in pixels, multiplied by *scale*. If either *width* or *height* is zero, the current width and height of the [QCustomPlot](#) widget is used instead. Line widths and texts etc. are not scaled up when larger widths/heights are used. If you want that effect, use the *scale* parameter.

For example, if you set both *width* and *height* to 100 and *scale* to 2, you will end up with an image file of size 200*200 in which all graphical elements are scaled up by factor 2 (line widths, texts, etc.). This scaling is not done by stretching a 100*100 image, the result will have full 200*200 pixel resolution.

If you use a high scaling factor, it is recommended to enable antialiasing for all elements by temporarily setting [QCustomPlot::setAntialiasedElements](#) to [QCP::aeAll](#) as this allows [QCustomPlot](#) to place objects with sub-pixel accuracy.

image compression can be controlled with the *quality* parameter which must be between 0 and 100 or -1 to use the default setting.

The *resolution* will be written to the image file header and has no direct consequence for the quality or the pixel size. However, if opening the image with a tool which respects the metadata, it will be able to scale the image to match either a given size in real units of length (inch, centimeters, etc.), or the target display DPI. You can specify in which units *resolution* is given, by setting *resolutionUnit*. The *resolution* is converted to the format's expected resolution unit internally.

Returns true on success. If this function fails, most likely the PNG format isn't supported by the system, see Qt docs about `QImageWriter::supportedImageFormats()`.

The objects of the plot will appear in the current selection state. If you don't want any selected objects to be painted in their selected look, deselect everything with [deselectAll](#) before calling this function.

If you want the PNG to have a transparent background, call [setBackground\(const QBrush &brush\)](#) with no brush (`Qt::NoBrush`) or a transparent color (`Qt::transparent`), before saving.

Warning

If calling this function inside the constructor of the parent of the [QCustomPlot](#) widget (i.e. the [MainWindow](#) constructor, if [QCustomPlot](#) is inside the [MainWindow](#)), always provide explicit non-zero widths and heights. If you leave *width* or *height* as 0 (default), this function uses the current width and height of the [QCustomPlot](#) widget. However, in Qt, these aren't defined yet inside the constructor, so you would get an image that has strange widths/heights.

See also

[savePdf](#), [saveBmp](#), [saveJpg](#), [saveRastered](#)

6.75.4.60 saveRastered()

```
bool QCustomPlot::saveRastered (
    const QString & fileName,
    int width,
    int height,
    double scale,
    const char * format,
    int quality = -1,
    int resolution = 96,
    QCP::ResolutionUnit resolutionUnit = QCP::ruDotsPerInch )
```

Saves the plot to a rastered image file *fileName* in the image format *format*. The plot is sized to *width* and *height* in pixels and scaled with *scale*. (width 100 and scale 2.0 lead to a full resolution file with width 200.) If the *format* supports compression, *quality* may be between 0 and 100 to control it.

Returns true on success. If this function fails, most likely the given *format* isn't supported by the system, see Qt docs about `QImageWriter::supportedImageFormats()`.

The *resolution* will be written to the image file header (if the file format supports this) and has no direct consequence for the quality or the pixel size. However, if opening the image with a tool which respects the metadata, it will be able to scale the image to match either a given size in real units of length (inch, centimeters, etc.), or the target display DPI. You can specify in which units *resolution* is given, by setting *resolutionUnit*. The *resolution* is converted to the format's expected resolution unit internally.

See also

[saveBmp](#), [saveJpg](#), [savePng](#), [savePdf](#)

6.75.4.61 selectedAxes()

```
QList< QCPAxis * > QCustomPlot::selectedAxes ( ) const
```

Returns the axes that currently have selected parts, i.e. whose selection state is not [QCPAxis::spNone](#).

See also

[selectedPlottables](#), [selectedLegends](#), [setInteractions](#), [QCPAxis::setSelectedParts](#), [QCPAxis::setSelectableParts](#)

6.75.4.62 selectedGraphs()

```
QList< QCPGraph * > QCustomPlot::selectedGraphs ( ) const
```

Returns a list of the selected graphs. If no graphs are currently selected, the list is empty.

If you are not only interested in selected graphs but other plottables like [QCPCurve](#), [QCPBars](#), etc., use [selectedPlottables](#).

See also

[setInteractions](#), [selectedPlottables](#), [QCPAbstractPlottable::setSelectable](#), [QCPAbstractPlottable::setSelection](#)

6.75.4.63 selectedItems()

```
QList< QCPAbstractItem * > QCustomPlot::selectedItems ( ) const
```

Returns a list of the selected items. If no items are currently selected, the list is empty.

See also

[setInteractions](#), [QCPAbstractItem::setSelectable](#), [QCPAbstractItem::setSelected](#)

6.75.4.64 selectedLegends()

```
QList< QCPLegend * > QCustomPlot::selectedLegends ( ) const
```

Returns the legends that currently have selected parts, i.e. whose selection state is not [QCPLegend::spNone](#).

See also

[selectedPlottables](#), [selectedAxes](#), [setInteractions](#), [QCPLegend::setSelectedParts](#), [QCPLegend::setSelectableParts](#), [QCPLegend::selectedItems](#)

6.75.4.65 selectedPlottables()

```
QList< QCPAbstractPlottable * > QCustomPlot::selectedPlottables ( ) const
```

Returns a list of the selected plottables. If no plottables are currently selected, the list is empty.

There is a convenience function if you're only interested in selected graphs, see [selectedGraphs](#).

See also

[setInteractions](#), [QCPAbstractPlottable::setSelectable](#), [QCPAbstractPlottable::setSelection](#)

6.75.4.66 selectionChangedByUser

```
void QCustomPlot::selectionChangedByUser ( ) [signal]
```

This signal is emitted after the user has changed the selection in the [QCustomPlot](#), e.g. by clicking. It is not emitted when the selection state of an object has changed programmatically by a direct call to `setSelected()/setSelection()` on an object or by calling [deselectAll](#).

In addition to this signal, selectable objects also provide individual signals, for example [QCPAxis::selectionChanged](#) or [QCPAbstractPlottable::selectionChanged](#). Note that those signals are emitted even if the selection state is changed programmatically.

See the documentation of [setInteractions](#) for details about the selection mechanism.

See also

[selectedPlottables](#), [selectedGraphs](#), [selectedItems](#), [selectedAxes](#), [selectedLegends](#)

6.75.4.67 selectionRect()

```
QCPSelectionRect * QCustomPlot::selectionRect ( ) const [inline]
```

Allows access to the currently used [QCPSelectionRect](#) instance (or subclass thereof), that is used to handle and draw selection rect interactions (see [setSelectionRectMode](#)).

See also

[setSelectionRect](#)

6.75.4.68 setAntialiasedElement()

```
void QCustomPlot::setAntialiasedElement (
    QCP::AntialiasedElement antialiasedElement,
    bool enabled = true )
```

Sets whether the specified *antialiasedElement* is forcibly drawn antialiased.

See [setAntialiasedElements](#) for details.

See also

[setNotAntialiasedElement](#)

6.75.4.69 `setAntialiasedElements()`

```
void QCustomPlot::setAntialiasedElements (
    const QCP::AntialiasedElements & antialiasedElements )
```

Sets which elements are forcibly drawn antialiased as an *or* combination of [QCP::AntialiasedElement](#).

This overrides the antialiasing settings for whole element groups, normally controlled with the *setAntialiasing* function on the individual elements. If an element is neither specified in [setAntialiasedElements](#) nor in [setNotAntialiasedElements](#), the antialiasing setting on each individual element instance is used.

For example, if *antialiasedElements* contains [QCP::aePlottables](#), all plottables will be drawn antialiased, no matter what the specific [QCPAbstractPlottable::setAntialiased](#) value was set to.

if an element in *antialiasedElements* is already set in [setNotAntialiasedElements](#), it is removed from there.

See also

[setNotAntialiasedElements](#)

6.75.4.70 `setAutoAddPlottableToLegend()`

```
void QCustomPlot::setAutoAddPlottableToLegend (
    bool on )
```

If set to true, adding a plottable (e.g. a graph) to the [QCustomPlot](#) automatically also adds the plottable to the legend ([QCustomPlot::legend](#)).

See also

[addGraph](#), [QCPLegend::addItem](#)

6.75.4.71 `setBackground()` [1/3]

```
void QCustomPlot::setBackground (
    const QPixmap & pm )
```

Sets *pm* as the viewport background pixmap (see [setViewport](#)). The pixmap is always drawn below all other objects in the plot.

For cases where the provided pixmap doesn't have the same size as the viewport, scaling can be enabled with [setBackgroundScaled](#) and the scaling mode (whether and how the aspect ratio is preserved) can be set with [setBackgroundScaledMode](#). To set all these options in one call, consider using the overloaded version of this function.

If a background brush was set with [setBackground\(const QBrush &brush\)](#), the viewport will first be filled with that brush, before drawing the background pixmap. This can be useful for background pixmaps with translucent areas.

See also

[setBackgroundScaled](#), [setBackgroundScaledMode](#)

6.75.4.72 setBackground() [2/3]

```
void QCustomPlot::setBackground (
    const QPixmap & pm,
    bool scaled,
    Qt::AspectRatioMode mode = Qt::KeepAspectRatioByExpanding )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Allows setting the background pixmap of the viewport, whether it shall be scaled and how it shall be scaled in one call.

See also

[setBackground\(const QPixmap &pm\)](#), [setBackgroundScaled](#), [setBackgroundScaledMode](#)

6.75.4.73 setBackground() [3/3]

```
void QCustomPlot::setBackground (
    const QBrush & brush )
```

Sets the background brush of the viewport (see [setViewport](#)).

Before drawing everything else, the background is filled with *brush*. If a background pixmap was set with [setBackground\(const QPixmap &pm\)](#), this brush will be used to fill the viewport before the background pixmap is drawn. This can be useful for background pixmaps with translucent areas.

Set *brush* to Qt::NoBrush or Qt::Transparent to leave background transparent. This can be useful for exporting to image formats which support transparency, e.g. [savePng](#).

See also

[setBackgroundScaled](#), [setBackgroundScaledMode](#)

6.75.4.74 setBackgroundScaled()

```
void QCustomPlot::setBackgroundScaled (
    bool scaled )
```

Sets whether the viewport background pixmap shall be scaled to fit the viewport. If *scaled* is set to true, control whether and how the aspect ratio of the original pixmap is preserved with [setBackgroundScaledMode](#).

Note that the scaled version of the original pixmap is buffered, so there is no performance penalty on replots. (Except when the viewport dimensions are changed continuously.)

See also

[setBackground](#), [setBackgroundScaledMode](#)

6.75.4.75 setBackgroundScaledMode()

```
void QCustomPlot::setBackgroundScaledMode (
    Qt::AspectRatioMode mode )
```

If scaling of the viewport background pixmap is enabled ([setBackgroundScaled](#)), use this function to define whether and how the aspect ratio of the original pixmap is preserved.

See also

[setBackground](#), [setBackgroundScaled](#)

6.75.4.76 setBufferDevicePixelRatio()

```
void QCustomPlot::setBufferDevicePixelRatio (
    double ratio )
```

Sets the device pixel ratio used by the paint buffers of this [QCustomPlot](#) instance.

Normally, this doesn't need to be set manually, because it is initialized with the regular *QWidget::devicePixelRatio* which is configured by Qt to fit the display device (e.g. 1 for normal displays, 2 for High-DPI displays).

Device pixel ratios are supported by Qt only for Qt versions since 5.4. If this method is called when [QCustomPlot](#) is being used with older Qt versions, outputs an according qDebug message and leaves the internal buffer device pixel ratio at 1.0.

6.75.4.77 setCurrentLayer() [1/2]

```
bool QCustomPlot::setCurrentLayer (
    const QString & name )
```

Sets the layer with the specified *name* to be the current layer. All layerables ([QCPLayerable](#)), e.g. plottables and items, are created on the current layer.

Returns true on success, i.e. if there is a layer with the specified *name* in the [QCustomPlot](#).

Layer names are case-sensitive.

See also

[addLayer](#), [moveLayer](#), [removeLayer](#), [QCPLayerable::setLayer](#)

6.75.4.78 setCurrentLayer() [2/2]

```
bool QCustomPlot::setCurrentLayer (
    QCPLayer * layer )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Sets the provided *layer* to be the current layer.

Returns true on success, i.e. when *layer* is a valid layer in the [QCustomPlot](#).

See also

[addLayer](#), [moveLayer](#), [removeLayer](#)

6.75.4.79 setInteraction()

```
void QCustomPlot::setInteraction (
    const QCP::Interaction & interaction,
    bool enabled = true )
```

Sets the single *interaction* of this [QCustomPlot](#) to *enabled*.

For details about the interaction system, see [setInteractions](#).

See also

[setInteractions](#)

6.75.4.80 setInteractions()

```
void QCustomPlot::setInteractions (
    const QCP::Interactions & interactions )
```

Sets the possible interactions of this [QCustomPlot](#) as an or-combination of [QCP::Interaction](#) enums. There are the following types of interactions:

Axis range manipulation is controlled via [QCP::iRangeDrag](#) and [QCP::iRangeZoom](#). When the respective interaction is enabled, the user may drag axes ranges and zoom with the mouse wheel. For details how to control which axes the user may drag/zoom and in what orientations, see [QCPAxisRect::setRangeDrag](#), [QCPAxisRect::setRangeZoom](#), [QCPAxisRect::setRangeDragAxes](#), [QCPAxisRect::setRangeZoomAxes](#).

Plottable data selection is controlled by [QCP::iSelectPlottables](#). If [QCP::iSelectPlottables](#) is set, the user may select plottables (graphs, curves, bars,...) and their data by clicking on them or in their vicinity ([setSelectionTolerance](#)). Whether the user can actually select a plottable and its data can further be restricted with the [QCPAbstractPlottable::setSelectable](#) method on the specific plottable. For details, see the special page about the data selection mechanism. To retrieve a list of all currently selected plottables, call [selectedPlottables](#). If you're only interested in [QCPGraphs](#), you may use the convenience function [selectedGraphs](#).

Item selection is controlled by [QCP::iSelectItems](#). If [QCP::iSelectItems](#) is set, the user may select items ([QCPItemLine](#), [QCPItemText](#),...) by clicking on them or in their vicinity. To find out whether a specific item is selected, call [QCPAbstractItem::selected\(\)](#). To retrieve a list of all currently selected items, call [selectedItems](#).

Axis selection is controlled with [QCP::iSelectAxes](#). If [QCP::iSelectAxes](#) is set, the user may select parts of the axes by clicking on them. What parts exactly (e.g. Axis base line, tick labels, axis label) are selectable can be controlled via [QCPAxis::setSelectableParts](#) for each axis. To retrieve a list of all axes that currently contain selected parts, call [selectedAxes](#). Which parts of an axis are selected, can be retrieved with [QCPAxis::selectedParts\(\)](#).

Legend selection is controlled with [QCP::iSelectLegend](#). If this is set, the user may select the legend itself or individual items by clicking on them. What parts exactly are selectable can be controlled via [QCPLegend::setSelectableParts](#). To find out whether the legend or any of its child items are selected, check the value of [QCPLegend::selectedParts](#). To find out which child items are selected, call [QCPLegend::selectedItems](#).

All other selectable elements The selection of all other selectable objects (e.g. [QCPTextElement](#), or your own layerable subclasses) is controlled with [QCP::iSelectOther](#). If set, the user may select those objects by clicking on them. To find out which are currently selected, you need to check their selected state explicitly.

If the selection state has changed by user interaction, the [selectionChangedByUser](#) signal is emitted. Each selectable object additionally emits an individual [selectionChanged](#) signal whenever their selection state has changed, i.e. not only by user interaction.

To allow multiple objects to be selected by holding the selection modifier ([setMultiSelectModifier](#)), set the flag [QCP::iMultiSelect](#).

Note

In addition to the selection mechanism presented here, [QCustomPlot](#) always emits corresponding signals, when an object is clicked or double clicked. see [plottableClick](#) and [plottableDoubleClick](#) for example.

See also

[setInteraction](#), [setSelectionTolerance](#)

6.75.4.81 [setMultiSelectModifier\(\)](#)

```
void QCustomPlot::setMultiSelectModifier (
    Qt::KeyboardModifier modifier )
```

Sets the keyboard modifier that will be recognized as multi-select-modifier.

If [QCP::iMultiSelect](#) is specified in [setInteractions](#), the user may select multiple objects (or data points) by clicking on them one after the other while holding down *modifier*.

By default the multi-select-modifier is set to [Qt::ControlModifier](#).

See also

[setInteractions](#)

6.75.4.82 setNoAntialiasingOnDrag()

```
void QCustomPlot::setNoAntialiasingOnDrag (
    bool enabled )
```

Sets whether antialiasing is disabled for this [QCustomPlot](#) while the user is dragging axes ranges. If many objects, especially plottables, are drawn antialiased, this greatly improves performance during dragging. Thus it creates a more responsive user experience. As soon as the user stops dragging, the last replot is done with normal antialiasing, to restore high image quality.

See also

[setAntialiasedElements](#), [setNotAntialiasedElements](#)

6.75.4.83 setNotAntialiasedElement()

```
void QCustomPlot::setNotAntialiasedElement (
    QCP::AntialiasedElement notAntialiasedElement,
    bool enabled = true )
```

Sets whether the specified *notAntialiasedElement* is forcibly drawn not antialiased.

See [setNotAntialiasedElements](#) for details.

See also

[setAntialiasedElement](#)

6.75.4.84 setNotAntialiasedElements()

```
void QCustomPlot::setNotAntialiasedElements (
    const QCP::AntialiasedElements & notAntialiasedElements )
```

Sets which elements are forcibly drawn not antialiased as an *or* combination of [QCP::AntialiasedElement](#).

This overrides the antialiasing settings for whole element groups, normally controlled with the *setAntialiasing* function on the individual elements. If an element is neither specified in [setAntialiasedElements](#) nor in [setNotAntialiasedElements](#), the antialiasing setting on each individual element instance is used.

For example, if *notAntialiasedElements* contains [QCP::aePlottables](#), no plottables will be drawn antialiased, no matter what the specific [QCPAbstractPlottable::setAntialiased](#) value was set to.

if an element in *notAntialiasedElements* is already set in [setAntialiasedElements](#), it is removed from there.

See also

[setAntialiasedElements](#)

6.75.4.85 setOpenGL()

```
void QCustomPlot::setOpenGL (
    bool enabled,
    int multisampling = 16 )
```

This method allows to enable OpenGL plot rendering, for increased plotting performance of graphically demanding plots (thick lines, translucent fills, etc.).

If *enabled* is set to true, [QCustomPlot](#) will try to initialize OpenGL and, if successful, continue plotting with hardware acceleration. The parameter *multisampling* controls how many samples will be used per pixel, it essentially controls the antialiasing quality. If *multisampling* is set too high for the current graphics hardware, the maximum allowed value will be used.

You can test whether switching to OpenGL rendering was successful by checking whether the according getter [QCustomPlot::openGl\(\)](#) returns true. If the OpenGL initialization fails, rendering continues with the regular software rasterizer, and an according qDebug output is generated.

If switching to OpenGL was successful, this method disables label caching ([setPlottingHint\(QCP::phCacheLabels, false\)](#)) and turns on [QCustomPlot](#)'s antialiasing override for all elements ([setAntialiasedElements\(QCP::aeAll\)](#)), leading to a higher quality output. The antialiasing override allows for pixel-grid aligned drawing in the OpenGL paint device. As stated before, in OpenGL rendering the actual antialiasing of the plot is controlled with *multisampling*. If *enabled* is set to false, the antialiasing/label caching settings are restored to what they were before OpenGL was enabled, if they weren't altered in the meantime.

Note

OpenGL support is only enabled if [QCustomPlot](#) is compiled with the macro `QCUSTOMPLOT_USE_OPENGL` defined. This define must be set before including the [QCustomPlot](#) header both during compilation of the [QCustomPlot](#) library as well as when compiling your application. It is best to just include the line `DEFINES += QCUSTOMPLOT_USE_OPENGL` in the respective qmake project files.

If you are using a Qt version before 5.0, you must also add the module "opengl" to your `QT` variable in the qmake project files. For Qt versions 5.0 and higher, [QCustomPlot](#) switches to a newer OpenGL interface which is already in the "gui" module.

6.75.4.86 setPlottingHint()

```
void QCustomPlot::setPlottingHint (
    QCP::PlottingHint hint,
    bool enabled = true )
```

Sets the specified plotting *hint* to *enabled*.

See also

[setPlottingHints](#)

6.75.4.87 setPlottingHints()

```
void QCustomPlot::setPlottingHints (
    const QCP::PlottingHints & hints )
```

Sets the plotting hints for this [QCustomPlot](#) instance as an *or* combination of [QCP::PlottingHint](#).

See also

[setPlottingHint](#)

6.75.4.88 setSelectionRect()

```
void QCustomPlot::setSelectionRect (
    QCPSelectionRect * selectionRect )
```

Sets the [QCPSelectionRect](#) instance that [QCustomPlot](#) will use if *mode* is not [QCP::srmNone](#) and the user performs a click-and-drag interaction. [QCustomPlot](#) takes ownership of the passed *selectionRect*. It can be accessed later via [selectionRect](#).

This method is useful if you wish to replace the default [QCPSelectionRect](#) instance with an instance of a [QCP↔SelectionRect](#) subclass, to introduce custom behaviour of the selection rect.

See also

[setSelectionRectMode](#)

6.75.4.89 setSelectionRectMode()

```
void QCustomPlot::setSelectionRectMode (
    QCP::SelectionRectMode mode )
```

Sets how [QCustomPlot](#) processes mouse click-and-drag interactions by the user.

If *mode* is [QCP::srmNone](#), the mouse drag is forwarded to the underlying objects. For example, [QCPAxisRect](#) may process a mouse drag by dragging axis ranges, see [QCPAxisRect::setRangeDrag](#). If *mode* is not [QCP::srmNone](#), the current selection rect ([selectionRect](#)) becomes activated and allows e.g. rect zooming and data point selection.

If you wish to provide your user both with axis range dragging and data selection/range zooming, use this method to switch between the modes just before the interaction is processed, e.g. in reaction to the [mousePress](#) or [mouse↔Move](#) signals. For example you could check whether the user is holding a certain keyboard modifier, and then decide which *mode* shall be set.

If a selection rect interaction is currently active, and *mode* is set to [QCP::srmNone](#), the interaction is canceled ([QCPSelectionRect::cancel](#)). Switching between any of the other modes will keep the selection rect active. Upon completion of the interaction, the behaviour is as defined by the currently set *mode*, not the mode that was set when the interaction started.

See also

[setInteractions](#), [setSelectionRect](#), [QCPSelectionRect](#)

6.75.4.90 `setSelectionTolerance()`

```
void QCustomPlot::setSelectionTolerance (
    int pixels )
```

Sets the tolerance that is used to decide whether a click selects an object (e.g. a plottable) or not.

If the user clicks in the vicinity of the line of e.g. a [QCPGraph](#), it's only regarded as a potential selection when the minimum distance between the click position and the graph line is smaller than *pixels*. Objects that are defined by an area (e.g. [QCPBars](#)) only react to clicks directly inside the area and ignore this selection tolerance. In other words, it only has meaning for parts of objects that are too thin to exactly hit with a click and thus need such a tolerance.

See also

[setInteractions](#), [QCPLayerable::selectTest](#)

6.75.4.91 `setViewport()`

```
void QCustomPlot::setViewport (
    const QRect & rect )
```

Sets the viewport of this [QCustomPlot](#). Usually users of [QCustomPlot](#) don't need to change the viewport manually.

The viewport is the area in which the plot is drawn. All mechanisms, e.g. margin calculation take the viewport to be the outer border of the plot. The viewport normally is the `rect()` of the [QCustomPlot](#) widget, i.e. a rect with top left (0, 0) and size of the [QCustomPlot](#) widget.

Don't confuse the viewport with the axis rect ([QCustomPlot::axisRect](#)). An axis rect is typically an area enclosed by four axes, where the graphs/plottables are drawn in. The viewport is larger and contains also the axes themselves, their tick numbers, their labels, or even additional axis rects, color scales and other layout elements.

This function is used to allow arbitrary size exports with [toPixmap](#), [savePng](#), [savePdf](#), etc. by temporarily changing the viewport size.

6.75.4.92 `toPainter()`

```
void QCustomPlot::toPainter (
    QCPPainter * painter,
    int width = 0,
    int height = 0 )
```

Renders the plot using the passed *painter*.

The plot is sized to *width* and *height* in pixels. If the *painter's* scale is not 1.0, the resulting plot will appear scaled accordingly.

Note

If you are restricted to using a `QPainter` (instead of [QCPPainter](#)), create a temporary `QPicture` and open a [QCPPainter](#) on it. Then call [toPainter](#) with this [QCPPainter](#). After ending the paint operation on the picture, draw it with the `QPainter`. This will reproduce the painter actions the [QCPPainter](#) took, with a `QPainter`.

See also

[toPixmap](#)

6.75.4.93 toPixmap()

```
QPixmap QCustomPlot::toPixmap (
    int width = 0,
    int height = 0,
    double scale = 1.0 )
```

Renders the plot to a pixmap and returns it.

The plot is sized to *width* and *height* in pixels and scaled with *scale*. (width 100 and scale 2.0 lead to a full resolution pixmap with width 200.)

See also

[toPainter](#), [saveRastered](#), [saveBmp](#), [savePng](#), [saveJpg](#), [savePdf](#)

6.75.5 Member Data Documentation

6.75.5.1 legend

```
QCPLegend * QCustomPlot::legend
```

A pointer to the default legend of the main axis rect. The legend is invisible by default. Use [QCPLegend::setVisible](#) to change this.

[QCustomPlot](#) offers convenient pointers to the axes ([xAxis](#), [yAxis](#), [xAxis2](#), [yAxis2](#)) and the [legend](#). They make it very easy working with plots that only have a single axis rect and at most one axis at each axis rect side. If you use the layout system to add multiple legends to the plot, use the layout system interface to access the new legend. For example, legends can be placed inside an axis rect's [inset layout](#), and must then also be accessed via the inset layout. If the default legend is removed due to manipulation of the layout system (e.g. by removing the main axis rect), the corresponding pointer becomes 0.

If an axis convenience pointer is currently zero and a new axis rect or a corresponding axis is added in the place of the main axis rect, [QCustomPlot](#) resets the convenience pointers to the according new axes. Similarly the [legend](#) convenience pointer will be reset if a legend is added after the main legend was removed before.

6.75.5.2 xAxis

```
QCPAxis * QCustomPlot::xAxis
```

A pointer to the primary x Axis (bottom) of the main axis rect of the plot.

[QCustomPlot](#) offers convenient pointers to the axes ([xAxis](#), [yAxis](#), [xAxis2](#), [yAxis2](#)) and the [legend](#). They make it very easy working with plots that only have a single axis rect and at most one axis at each axis rect side. If you use the layout system to add multiple axis rects or multiple axes to one side, use the [QCPAxisRect::axis](#) interface to access the new axes. If one of the four default axes or the default legend is removed due to manipulation of the layout system (e.g. by removing the main axis rect), the corresponding pointers become 0.

If an axis convenience pointer is currently zero and a new axis rect or a corresponding axis is added in the place of the main axis rect, [QCustomPlot](#) resets the convenience pointers to the according new axes. Similarly the [legend](#) convenience pointer will be reset if a legend is added after the main legend was removed before.

6.75.5.3 xAxis2

`QCPAxis * QCustomPlot::xAxis2`

A pointer to the secondary x Axis (top) of the main axis rect of the plot. Secondary axes are invisible by default. Use [QCPAxis::setVisible](#) to change this (or use [QCPAxisRect::setupFullAxesBox](#)).

[QCustomPlot](#) offers convenient pointers to the axes ([xAxis](#), [yAxis](#), [xAxis2](#), [yAxis2](#)) and the [legend](#). They make it very easy working with plots that only have a single axis rect and at most one axis at each axis rect side. If you use the layout system to add multiple axis rects or multiple axes to one side, use the [QCPAxisRect::axis](#) interface to access the new axes. If one of the four default axes or the default legend is removed due to manipulation of the layout system (e.g. by removing the main axis rect), the corresponding pointers become 0.

If an axis convenience pointer is currently zero and a new axis rect or a corresponding axis is added in the place of the main axis rect, [QCustomPlot](#) resets the convenience pointers to the according new axes. Similarly the [legend](#) convenience pointer will be reset if a legend is added after the main legend was removed before.

6.75.5.4 yAxis

`QCPAxis * QCustomPlot::yAxis`

A pointer to the primary y Axis (left) of the main axis rect of the plot.

[QCustomPlot](#) offers convenient pointers to the axes ([xAxis](#), [yAxis](#), [xAxis2](#), [yAxis2](#)) and the [legend](#). They make it very easy working with plots that only have a single axis rect and at most one axis at each axis rect side. If you use the layout system to add multiple axis rects or multiple axes to one side, use the [QCPAxisRect::axis](#) interface to access the new axes. If one of the four default axes or the default legend is removed due to manipulation of the layout system (e.g. by removing the main axis rect), the corresponding pointers become 0.

If an axis convenience pointer is currently zero and a new axis rect or a corresponding axis is added in the place of the main axis rect, [QCustomPlot](#) resets the convenience pointers to the according new axes. Similarly the [legend](#) convenience pointer will be reset if a legend is added after the main legend was removed before.

6.75.5.5 yAxis2

`QCPAxis * QCustomPlot::yAxis2`

A pointer to the secondary y Axis (right) of the main axis rect of the plot. Secondary axes are invisible by default. Use [QCPAxis::setVisible](#) to change this (or use [QCPAxisRect::setupFullAxesBox](#)).

[QCustomPlot](#) offers convenient pointers to the axes ([xAxis](#), [yAxis](#), [xAxis2](#), [yAxis2](#)) and the [legend](#). They make it very easy working with plots that only have a single axis rect and at most one axis at each axis rect side. If you use the layout system to add multiple axis rects or multiple axes to one side, use the [QCPAxisRect::axis](#) interface to access the new axes. If one of the four default axes or the default legend is removed due to manipulation of the layout system (e.g. by removing the main axis rect), the corresponding pointers become 0.

If an axis convenience pointer is currently zero and a new axis rect or a corresponding axis is added in the place of the main axis rect, [QCustomPlot](#) resets the convenience pointers to the according new axes. Similarly the [legend](#) convenience pointer will be reset if a legend is added after the main legend was removed before.

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

- [/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h](#)
- [/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.cpp](#)

6.76 Rohr Class Reference

Stellt ein Rohrbauteil zur Verfügung.

```
#include <rohr.h>
```

Public Member Functions

- **Rohr** (double l, double r)
- double [get_querschnitt](#) ()
Gibt Querschnittsfläche.
- double [get_querschnitt](#) (double x)
Gibt den Querschnitt an der Stelle x zurück.
- double [get_radius](#) ()
Gibt Radius zurück.
- double [get_radius](#) (double x)
Gibt Radius an der Stelle x zurück.
- double [get_laenge](#) ()
Gibt Länge zurück.
- double [get_alpha_innen](#) ()
Gibt Konvektionswiderstand auf der Innenseite zurück.
- double [get_alpha_aussen](#) ()
Gibt Konvektionswiderstand auf der Außenseite zurück.
- double [get_kA](#) ()
Berechnet Wärmedurchgangskoeffizient.
- double [get_kA](#) (double x)
Berechnet Wärmedurchgangskoeffizient an der Stelle x und speichert ihn auch unter kA.
- double [get_t_aussen](#) ()
Gibt Aussentemperatur zurück.
- void [set_alpha_aussen](#) (double alpha_aussen)
Setzt Wert für Konvektionswiderstand auf der Außenseite.
- void [set_alpha_innen](#) (double alpha_innen)
Setzt Wert für Konvektionswiderstand auf der Innenseite.
- void [set_t_aussen](#) (double t_aussen)
Setzt Wert für Aussentemperatur.

6.76.1 Detailed Description

Stellt ein Rohrbauteil zur Verfügung.

6.76.2 Member Function Documentation

6.76.2.1 `get_kA()` [1/2]

```
double Rohr::get_kA ( )
```

Berechnet Wärmedurchgangskoeffizient.

Berechnet den Wärmedurchgangskoeffizient.

Warning

Anderer Ansatz: Es werden keine Werte der Funktion übergeben sondern in der Funktion get-Funktionen aufgerufen

See also

`Fluid::set_cp_strom`

6.76.2.2 `get_kA()` [2/2]

```
double Rohr::get_kA (
    double x )
```

Berechnet Wärmedurchgangskoeffizient an der Stelle x und speichert ihn auch unter kA.

Warning

Diese Funktion überschreiben, um $kA=f(x)$ zu realisieren.

berechnet den Wert von kA an einer bestimmten Stelle. Kann überschrieben werden, um den Radius als $kA=f(x)$ auszudrücken.

6.76.2.3 `get_querschnitt()`

```
double Rohr::get_querschnitt (
    double x )
```

Gibt den Querschnitt an der Stelle x zurück.

Warning

Basiert auf `get_radius`, daher wird $r=f(x)$ hier automatisch berücksichtigt. Nicht doppelt überschreiben!

See also

[Rohr::get_radius\(double x\)](#)

6.76.2.4 get_radius()

```
double Rohr::get_radius (
    double x )
```

Gibt Radius an der Stelle x zurück.

Diese Funktion überschreiben, um $r=f(x)$ zu realisieren gibt den Wert des Radius an einer bestimmten Stelle zurück. Kann überschrieben werden, um den Radius als $r=f(x)$ auszudrücken.

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

- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/rohr.h
- /home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/rohr.cpp

6.77 Rohrstroemung Class Reference

Beschreibt eine Rohrströmung.

```
#include <stroemung.h>
```

Public Member Functions

- **Rohrstroemung** (Rohr rohr, Fluid fluid)
- double [get_Re](#) ()
Reynoldszahl des Member-Fluids berechnen.
- double [get_lambda](#) ()
Rohrreibungszahl Lambda berechnen, in Abhängigkeit von Re.
- double [get_speed](#) ()
Berechnung der Fluidgeschwindigkeit durch den Massenstrom.
- double [get_bauart](#) ()
Berechnet Bauartszahl.
- double [get_epsilon](#) ()
Gibt Leistungsgröße zurück.
- double [get_temp](#) (double x)
Berechnet Austrittstemperatur.
- double [get_startpressure](#) ()
Gibt Startdruck zurück.
- double [get_pressure](#) (double x)
Berechnung des Drucks an der Stelle x.
- double [get_temp](#) ()
Berechnet Temperatur an der Stelle x.
- double [get_epsilon](#) (double x)
Berechnet Epsilon an der Stelle x.
- double [get_bauart](#) (double x)
Berechnet die Bauart an der Stelle x.

6.77.1 Detailed Description

Beschreibt eine Rohrströmung.

Benötigt dazu Objekte von [Rohr](#) und [Fluid](#).

Warning

Es werden keine kompressiblen Fluide oder Ablösungen und Einlaufstörungen am Ein- und Ausgang des Rohrs berücksichtigt.

6.77.2 Member Function Documentation

6.77.2.1 `get_lambda()`

```
double Rohrstroemung::get_lambda ( )
```

Rohrreibungszahl Lambda berechnen, in Abhängigkeit von Re.

See also

[get_Re\(\)](#)

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

- `/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/stroemung.h`↔
- `/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/stroemung.cpp`↔

6.78 QCPAxisPainterPrivate::TickLabelData Struct Reference

Public Attributes

- QString **basePart**
- QString **expPart**
- QString **suffixPart**
- QRect **baseBounds**
- QRect **expBounds**
- QRect **suffixBounds**
- QRect **totalBounds**
- QRect **rotatedTotalBounds**
- QFont **baseFont**
- QFont **expFont**

The documentation for this struct was generated from the following file:

- `/home/maximilian/Sync/Dokumente/2_Studium/2017_SS/Informatik/2_API/projektarbeit-berdoullies/src/qcustomplot.h`↔

Index

- APITest, [19](#)
 - printTestResult, [19](#)
- accepted
 - QCPSelectionRect, [395](#)
- add
 - QCPDataContainer, [189](#), [190](#)
- addAxes
 - QCPAxisRect, [88](#)
- addAxis
 - QCPAxisRect, [89](#)
- addData
 - QCPBars, [130](#)
 - QCPCurve, [179](#), [180](#)
 - QCPErrorBars, [213](#)
 - QCPFinancial, [226](#), [227](#)
 - QCPGraph, [240](#)
 - QCPStatisticalBox, [400](#)
- addDataRange
 - QCPDataSelection, [204](#)
- addElement
 - QCPLayoutGrid, [324](#)
 - QCPLayoutInset, [335](#)
- addGraph
 - QCustomPlot, [429](#)
- addItem
 - QCPLegend, [342](#)
- addLayer
 - QCustomPlot, [429](#)
- addTick
 - QCPAxisTickerText, [120](#)
- addTicks
 - QCPAxisTickerText, [120](#)
- addToLegend
 - QCPAbstractPlottable, [43](#), [44](#)
- adjusted
 - QCPDataRange, [199](#)
- afterReplot
 - QCustomPlot, [430](#)
- alpha
 - QCPColorMapData, [161](#)
- anchor
 - QCPAbstractItem, [27](#)
- anchors
 - QCPAbstractItem, [28](#)
- AntialiasedElement
 - QCP, [14](#)
- append
 - QCPBarsGroup, [141](#)
- applyBrush
 - QCPSelectionDecorator, [387](#)
- applyPen
 - QCPSelectionDecorator, [388](#)
- applyTo
 - QCPScatterStyle, [382](#)
- at
 - QCPDataContainer, [190](#)
- axes
 - QCPAxisRect, [89](#)
- axis
 - QCPAxisRect, [90](#)
 - QCPColorScale, [169](#)
- axisClick
 - QCustomPlot, [430](#)
- axisCount
 - QCPAxisRect, [90](#)
- axisDoubleClick
 - QCustomPlot, [430](#)
- axisRect
 - QCustomPlot, [431](#)
- axisRectAt
 - QCustomPlot, [431](#)
- axisRectCount
 - QCustomPlot, [431](#)
- axisRects
 - QCustomPlot, [432](#)
- AxisType
 - QCPAxis, [64](#)
- barAbove
 - QCPBars, [130](#)
- barBelow
 - QCPBars, [130](#)
- bars
 - QCPBarsGroup, [141](#)
- beforeReplot
 - QCustomPlot, [432](#)
- begin
 - QCPDataContainer, [191](#)
 - QCPPainter, [359](#)
- bottom
 - QCPAxisRect, [90](#)
- bottomLeft
 - QCPAxisRect, [90](#)
- bottomRight
 - QCPAxisRect, [90](#)
- bounded
 - QCPDataRange, [199](#)
 - QCPRange, [371](#)
- BracketStyle

- QCPItemBracket, 254
- QCPSelectionDecoratorBracket, 391
- cancel
 - QCPSelectionRect, 395
- canceled
 - QCPSelectionRect, 396
- cellToCoord
 - QCPColorMapData, 161
- center
 - QCPAxisRect, 91
 - QCPRange, 371
- changed
 - QCPSelectionRect, 396
- ChartStyle
 - QCPFinancial, 225
- children
 - QCPLayer, 300
- clear
 - QCPAbstractPaintBuffer, 37
 - QCPAxisTickerText, 121
 - QCPBarsGroup, 141
 - QCPColorMapData, 161
 - QCPDataContainer, 191
 - QCPDataSelection, 204
 - QCPLayout, 310
 - QCPMarginGroup, 355
 - QCPPaintBufferPixmap, 356
- clearAlpha
 - QCPColorMapData, 162
- clearColorStops
 - QCPColorGradient, 147
- clearGraphs
 - QCustomPlot, 432
- clearItems
 - QCPLegend, 342
 - QCustomPlot, 432
- clearPlottables
 - QCustomPlot, 433
- clicked
 - QCPTextElement, 412
- ColorInterpolation
 - QCPColorGradient, 145
- colorMaps
 - QCPColorScale, 169
- colorize
 - QCPColorGradient, 147
- columnCount
 - QCPLayoutGrid, 325
- constBegin
 - QCPDataContainer, 191
- constEnd
 - QCPDataContainer, 191
- contains
 - QCPBarsGroup, 142
 - QCPDataRange, 199
 - QCPDataSelection, 204
 - QCPRange, 371
- coordToCell
 - QCPColorMapData, 162
- coordToPixel
 - QCPAxis, 66
- coordsToPixels
 - QCPAbstractPlottable, 44
- copyFrom
 - QCPSelectionDecorator, 388
- createTickVector
 - QCPAxisTickerText, 121
- currentLayer
 - QCustomPlot, 433
- data
 - QCPBars, 131
 - QCPColorMap, 153
 - QCPCurve, 180
 - QCPErrorBars, 214
 - QCPFinancial, 227
 - QCPGraph, 240
 - QCPStatisticalBox, 401
- dataCount
 - QCPAbstractPlottable1D, 55
 - QCPErrorBars, 214
 - QCPPlottableInterface1D, 363
- dataMainKey
 - QCPAbstractPlottable1D, 55
 - QCPErrorBars, 214
 - QCPPlottableInterface1D, 364
- dataMainValue
 - QCPAbstractPlottable1D, 55
 - QCPErrorBars, 214
 - QCPPlottableInterface1D, 364
- dataPixelPosition
 - QCPAbstractPlottable1D, 56
 - QCPBars, 131
 - QCPErrorBars, 215
 - QCPPlottableInterface1D, 364
- dataPointCount
 - QCPDataSelection, 204
- dataRange
 - QCPDataContainer, 191
 - QCPDataSelection, 205
- dataRangeChanged
 - QCPColorMap, 153
 - QCPColorScale, 169
- dataRangeCount
 - QCPDataSelection, 205
- dataRanges
 - QCPDataSelection, 205
- dataScaleTypeChanged
 - QCPColorMap, 153
 - QCPColorScale, 169
- dataSortKey
 - QCPAbstractPlottable1D, 56
 - QCPErrorBars, 215
 - QCPPlottableInterface1D, 364
- dataValueRange
 - QCPAbstractPlottable1D, 56
 - QCPErrorBars, 215

- QCPPlottableInterface1D, 365
- dateTimeToKey
 - QCPAxisTickerDateTime, 109
- DatenEingabe, 20
- deselectAll
 - QCustomPlot, 433
- distanceSquaredToLine
 - QCPVector2D, 419
- distanceToStraightLine
 - QCPVector2D, 420
- donePainting
 - QCPAbstractPaintBuffer, 37
- dot
 - QCPVector2D, 420
- doubleClicked
 - QCPTextElement, 413
- draw
 - QCPAbstractPaintBuffer, 37
 - QCPPaintBufferPixmap, 356
- drawBracket
 - QCPSelectionDecoratorBracket, 391
- drawDecoration
 - QCPSelectionDecorator, 388
 - QCPSelectionDecoratorBracket, 392
- drawLine
 - QCPPainter, 360
- drawPolyline
 - QCPAbstractPlottable1D, 57
- drawShape
 - QCPScatterStyle, 383
- drawStatisticalBox
 - QCPStatisticalBox, 401
- element
 - QCPLayoutGrid, 325
- elementAt
 - QCPLayout, 310
 - QCPLayoutGrid, 325
 - QCPLayoutInset, 335
- elementCount
 - QCPLayout, 311
 - QCPLayoutGrid, 326
 - QCPLayoutInset, 336
- elements
 - QCPAxisRect, 91
 - QCPLayout, 311
 - QCPLayoutElement, 317
 - QCPLayoutGrid, 326
 - QCPMarginGroup, 355
- end
 - QCPDataContainer, 192
- EndingStyle
 - QCPLineEnding, 351
- enforceType
 - QCPDataSelection, 205
- ErrorType
 - QCPErrorBars, 212
- expand
 - QCPRange, 371, 372
- expandTo
 - QCPLayoutGrid, 326
- expanded
 - QCPDataRange, 199
 - QCPRange, 372
- ExportPen
 - QCP, 14
- fill
 - QCPColorMapData, 162
- fillAlpha
 - QCPColorMapData, 162
- FillOrder
 - QCPLayoutGrid, 323
- findBegin
 - QCPAbstractPlottable1D, 57
 - QCPDataContainer, 192
 - QCPErrorBars, 215
 - QCPPlottableInterface1D, 365
- findEnd
 - QCPAbstractPlottable1D, 57
 - QCPDataContainer, 192
 - QCPErrorBars, 216
 - QCPPlottableInterface1D, 365
- Fluid, 20
 - get_cp_strom, 21
 - get_massenstrom, 21
- FractionStyle
 - QCPAxisTickerPi, 117
- fromSortKey
 - QCPBarsData, 137
 - QCPCurveData, 185
 - QCPFinancialData, 234
 - QCPGraphData, 246
 - QCPStatisticalBoxData, 408
- generate
 - QCPAxisTicker, 106
- get_cp_strom
 - Fluid, 21
- get_kA
 - Rohr, 463, 464
- get_lambda
 - Rohrstromung, 466
- get_massenstrom
 - Fluid, 21
- get_querschnitt
 - Rohr, 464
- get_radius
 - Rohr, 464
- getDataSegments
 - QCPAbstractPlottable1D, 58
- getFinalScatterStyle
 - QCPSelectionDecorator, 388
- getKeyRange
 - QCPAbstractPlottable, 45
 - QCPBars, 131
 - QCPColorMap, 153
 - QCPCurve, 180

- QCPErrorBars, [216](#)
 - QCPFinancial, [228](#)
 - QCPGraph, [240](#)
 - QCPStatisticalBox, [401](#)
- getPartAt
 - QCPAxis, [66](#)
- getSubTickCount
 - QCPAxisTickerText, [121](#)
- getTickLabel
 - QCPAxisTickerText, [122](#)
- getTickStep
 - QCPAxisTickerText, [122](#)
- getValueRange
 - QCPAbstractPlottable, [45](#)
 - QCPBars, [131](#)
 - QCPColorMap, [154](#)
 - QCPCurve, [181](#)
 - QCPErrorBars, [217](#)
 - QCPFinancial, [228](#)
 - QCPGraph, [241](#)
 - QCPStatisticalBox, [402](#)
- getVisibleDataBounds
 - QCPGraph, [241](#)
- gradientChanged
 - QCPColorMap, [154](#)
 - QCPColorScale, [170](#)
- GradientPreset
 - QCPColorGradient, [145](#)
- graph
 - QCustomPlot, [433](#), [434](#)
- graphCount
 - QCustomPlot, [434](#)
- graphs
 - QCPAxis, [66](#)
 - QCPAxisRect, [91](#)
- grid
 - QCPAxis, [67](#)
- hasAnchor
 - QCPAbstractItem, [28](#)
- hasElement
 - QCPLayoutGrid, [327](#)
- hasInvalidatedPaintBuffers
 - QCustomPlot, [434](#)
- hasItem
 - QCPLegend, [343](#)
 - QCustomPlot, [434](#)
- hasItemWithPlottable
 - QCPLegend, [343](#)
- hasPlottable
 - QCustomPlot, [435](#)
- height
 - QCPAxisRect, [91](#)
- index
 - QCPLayer, [300](#)
- indexToRowCol
 - QCPLayoutGrid, [327](#)
- insert
 - QCPBarsGroup, [142](#)
 - insertColumn
 - QCPLayoutGrid, [327](#)
 - insertRow
 - QCPLayoutGrid, [328](#)
 - insetAlignment
 - QCPLayoutInset, [336](#)
 - insetLayout
 - QCPAxisRect, [91](#)
 - InsetPlacement
 - QCPLayoutInset, [334](#)
 - insetPlacement
 - QCPLayoutInset, [336](#)
 - insetRect
 - QCPLayoutInset, [336](#)
 - Interaction
 - QCP, [15](#)
 - interface1D
 - QCPAbstractPlottable, [46](#)
 - QCPAbstractPlottable1D, [58](#)
 - QCPErrorBars, [217](#)
 - intersection
 - QCPDataRange, [199](#)
 - QCPDataSelection, [206](#)
 - intersects
 - QCPDataRange, [200](#)
 - inverse
 - QCPDataSelection, [206](#)
 - inverted
 - QCPColorGradient, [147](#)
 - isActive
 - QCPSelectionRect, [396](#)
 - isEmpty
 - QCPBarsGroup, [142](#)
 - QCPColorMapData, [163](#)
 - QCPDataContainer, [192](#)
 - QCPDataRange, [200](#)
 - QCPDataSelection, [206](#)
 - QCPMarginGroup, [355](#)
 - isNone
 - QCPScatterStyle, [383](#)
 - isNull
 - QCPVector2D, [420](#)
 - isPenDefined
 - QCPScatterStyle, [383](#)
 - isValid
 - QCPDataRange, [200](#)
 - item
 - QCPLegend, [343](#)
 - QCustomPlot, [435](#)
 - itemAt
 - QCustomPlot, [435](#)
 - itemClick
 - QCustomPlot, [436](#)
 - itemCount
 - QCPLegend, [343](#)
 - QCustomPlot, [436](#)
 - itemDoubleClick

- QCustomPlot, [436](#)
- itemWithPlottable
 - QCPLegend, [344](#)
- items
 - QCPAxis, [67](#)
 - QCPAxisRect, [92](#)
- keyRange
 - QCPDataContainer, [193](#)
- keyToDateTime
 - QCPAxisTickerDateTime, [109](#)
- LabelSide
 - QCPAxis, [64](#)
- layer
 - QCustomPlot, [437](#)
- layerChanged
 - QCPLayerable, [304](#)
- layerCount
 - QCustomPlot, [437](#)
- LayerInsertMode
 - QCustomPlot, [428](#)
- LayerMode
 - QCPLayer, [299](#)
- layout
 - QCPLayoutElement, [317](#)
- layoutElementAt
 - QCustomPlot, [438](#)
- left
 - QCPAxisRect, [92](#)
- legend
 - QCustomPlot, [461](#)
- legendClick
 - QCustomPlot, [438](#)
- legendDoubleClick
 - QCustomPlot, [438](#)
- length
 - QCPDataRange, [200](#)
 - QCPVector2D, [420](#)
- lengthSquared
 - QCPVector2D, [421](#)
- limitIteratorsToDataRange
 - QCPDataContainer, [193](#)
- LineStyle
 - QCPCurve, [178](#)
 - QCPGraph, [239](#)
- loadPreset
 - QCPColorGradient, [148](#)
- mainKey
 - QCPBarsData, [137](#)
 - QCPCurveData, [186](#)
 - QCPFinancialData, [235](#)
 - QCPGraphData, [246](#)
 - QCPStatisticalBoxData, [408](#)
- mainValue
 - QCPBarsData, [137](#)
 - QCPCurveData, [186](#)
 - QCPFinancialData, [235](#)
- QCPGraphData, [247](#)
- QCPStatisticalBoxData, [408](#)
- MainWindow, [22](#)
- makeNonCosmetic
 - QCPPainter, [360](#)
- MarginSide
 - QCP, [15](#)
- marginSideToAxisType
 - QCPAxis, [67](#)
- maxRange
 - QCPRange, [376](#)
- maximumSizeHint
 - QCPLayoutElement, [317](#)
 - QCPLayoutGrid, [328](#)
 - QCPTextElement, [413](#)
- minRange
 - QCPRange, [376](#)
- minimumSizeHint
 - QCPLayoutElement, [317](#)
 - QCPLayoutGrid, [328](#)
 - QCPPlottableLegendItem, [368](#)
 - QCPTextElement, [413](#)
- mouseDoubleClick
 - QCustomPlot, [439](#)
- mouseDoubleClickEvent
 - QCPLayerable, [304](#)
 - QCPTextElement, [413](#)
- mouseMove
 - QCustomPlot, [439](#)
- mouseMoveEvent
 - QCPAxisRect, [92](#)
 - QCPColorScale, [170](#)
 - QCPLayerable, [304](#)
- mousePress
 - QCustomPlot, [439](#)
- mousePressEvent
 - QCPAxisRect, [92](#)
 - QCPColorScale, [170](#)
 - QCPLayerable, [305](#)
 - QCPTextElement, [414](#)
- mouseRelease
 - QCustomPlot, [439](#)
- mouseReleaseEvent
 - QCPAxisRect, [93](#)
 - QCPColorScale, [171](#)
 - QCPLayerable, [305](#)
 - QCPTextElement, [414](#)
- mouseWheel
 - QCustomPlot, [440](#)
- moveAbove
 - QCPBars, [132](#)
- moveBelow
 - QCPBars, [132](#)
- moveLayer
 - QCustomPlot, [440](#)
- moveRange
 - QCPAxis, [67](#)
- normalize

- QCPRange, [373](#)
- QCPVector2D, [421](#)
- normalized
 - QCPVector2D, [421](#)
- operator<<
 - QCPDataRange, [202](#)
 - QCPDataSelection, [209](#)
 - QCPRange, [376](#)
 - QCPVector2D, [423](#)
- operator*
 - QCPRange, [375](#)
- operator*=
 - QCPRange, [373](#)
 - QCPVector2D, [421](#)
- operator+
 - QCPDataSelection, [208](#)
 - QCPRange, [375](#)
- operator+=
 - QCPDataSelection, [207](#)
 - QCPRange, [373](#)
 - QCPVector2D, [422](#)
- operator-
 - QCPDataSelection, [209](#)
 - QCPRange, [375](#)
- operator-=
 - QCPDataSelection, [207](#)
 - QCPRange, [373](#)
 - QCPVector2D, [422](#)
- operator/
 - QCPRange, [376](#)
- operator/=
 - QCPRange, [373](#)
 - QCPVector2D, [422](#)
- operator=
 - QCPColorMapData, [163](#)
- operator==
 - QCPDataSelection, [207](#)
- opposite
 - QCPAxis, [67](#)
- orientation
 - QCPAxis, [68](#)
- PainterMode
 - QCPPainter, [359](#)
- parentAnchor
 - QCPItemPosition, [274](#)
- parentLayerable
 - QCPLayerable, [306](#)
- perpendicular
 - QCPVector2D, [422](#)
- pixelOrientation
 - QCPAxis, [68](#)
- pixelPosition
 - QCPItemAnchor, [252](#)
 - QCPItemPosition, [274](#)
- pixelToCoord
 - QCPAxis, [68](#)
- pixelsToCoords
 - QCPAbstractPlottable, [46](#)
- plotLayout
 - QCustomPlot, [440](#)
- plottable
 - QCustomPlot, [440](#), [441](#)
- plottableAt
 - QCustomPlot, [441](#)
- plottableClick
 - QCustomPlot, [441](#)
- plottableCount
 - QCustomPlot, [442](#)
- plottableDoubleClick
 - QCustomPlot, [442](#)
- plottables
 - QCPAxis, [69](#)
 - QCPAxisRect, [93](#)
- Plotter, [22](#)
- PlottingHint
 - QCP, [16](#)
- position
 - QCPAbstractItem, [28](#)
- PositionType
 - QCPItemPosition, [273](#)
- positions
 - QCPAbstractItem, [29](#)
- printTestResult
 - APITest, [19](#)
- QCPAbstractItem, [23](#)
 - anchor, [27](#)
 - anchors, [28](#)
 - hasAnchor, [28](#)
 - position, [28](#)
 - positions, [29](#)
 - QCPAbstractItem, [27](#)
 - selectTest, [29](#)
 - selectionChanged, [29](#)
 - setClipAxisRect, [30](#)
 - setClipToAxisRect, [30](#)
 - setSelectable, [30](#)
 - setSelected, [31](#)
- QCPAbstractLegendItem, [31](#)
 - QCPAbstractLegendItem, [33](#)
 - selectTest, [33](#)
 - selectionChanged, [33](#)
 - setFont, [34](#)
 - setSelectable, [34](#)
 - setSelected, [34](#)
 - setSelectedFont, [35](#)
 - setSelectedTextColor, [35](#)
 - setTextColor, [35](#)
- QCPAbstractPaintBuffer, [36](#)
 - clear, [37](#)
 - donePainting, [37](#)
 - draw, [37](#)
 - QCPAbstractPaintBuffer, [37](#)
 - reallocateBuffer, [37](#)
 - setDevicePixelRatio, [38](#)
 - setInvalidated, [38](#)

- setSize, 38
- startPainting, 39
- QCPAbstractPlottable, 39
 - addToLegend, 43, 44
 - coordsToPixels, 44
 - getKeyRange, 45
 - getValueRange, 45
 - interface1D, 46
 - pixelsToCoords, 46
 - QCPAbstractPlottable, 43
 - removeFromLegend, 46, 47
 - rescaleAxes, 47
 - rescaleKeyAxis, 47
 - rescaleValueAxis, 48
 - selectTest, 49
 - selectableChanged, 48
 - selected, 48
 - selection, 48
 - selectionChanged, 48, 49
 - selectionDecorator, 49
 - setAntialiasedFill, 50
 - setAntialiasedScatters, 50
 - setBrush, 51
 - setKeyAxis, 51
 - setName, 51
 - setPen, 51
 - setSelectable, 52
 - setSelection, 52
 - setSelectionDecorator, 52
 - setValueAxis, 53
- QCPAbstractPlottable1D< DataType >, 53
- QCPAbstractPlottable1D
 - dataCount, 55
 - dataMainKey, 55
 - dataMainValue, 55
 - dataPixelPosition, 56
 - dataSortKey, 56
 - dataValueRange, 56
 - drawPolyline, 57
 - findBegin, 57
 - findEnd, 57
 - getDataSegments, 58
 - interface1D, 58
 - QCPAbstractPlottable1D, 55
 - selectTest, 58
 - selectTestRect, 59
 - sortKeysIsMainKey, 59
- QCPAxis, 60
 - AxisType, 64
 - coordToPixel, 66
 - getPartAt, 66
 - graphs, 66
 - grid, 67
 - items, 67
 - LabelSide, 64
 - marginSideToAxisType, 67
 - moveRange, 67
 - opposite, 67
 - orientation, 68
 - pixelOrientation, 68
 - pixelToCoord, 68
 - plottables, 69
 - QCPAxis, 66
 - rangeChanged, 69
 - rescale, 69
 - scaleRange, 70
 - ScaleType, 65
 - scaleTypeChanged, 70
 - selectTest, 71
 - selectableChanged, 70
 - SelectablePart, 65
 - selectionChanged, 71
 - setBasePen, 71
 - setLabel, 72
 - setLabelColor, 72
 - setLabelFont, 72
 - setLabelPadding, 72
 - setLowerEnding, 73
 - setNumberFormat, 73
 - setNumberPrecision, 73
 - setOffset, 74
 - setPadding, 74
 - setRange, 74, 75
 - setRangeLower, 75
 - setRangeReversed, 75
 - setRangeUpper, 75
 - setScaleRatio, 76
 - setScaleType, 76
 - setSelectableParts, 76
 - setSelectedBasePen, 76
 - setSelectedLabelColor, 77
 - setSelectedLabelFont, 77
 - setSelectedParts, 77
 - setSelectedSubTickPen, 77
 - setSelectedTickLabelColor, 78
 - setSelectedTickLabelFont, 78
 - setSelectedTickPen, 78
 - setSubTickLength, 78
 - setSubTickLengthIn, 79
 - setSubTickLengthOut, 79
 - setSubTickPen, 79
 - setSubTicks, 79
 - setTickLabelColor, 80
 - setTickLabelFont, 80
 - setTickLabelPadding, 81
 - setTickLabelRotation, 81
 - setTickLabelSide, 81
 - setTickLabels, 81
 - setTickLength, 81
 - setTickLengthIn, 82
 - setTickLengthOut, 82
 - setTickPen, 82
 - setTicker, 80
 - setTicks, 82
 - setUpperEnding, 83
 - ticker, 83

- QCPAxisPainterPrivate, 84
 - QCPAxisPainterPrivate, 85
- QCPAxisPainterPrivate::CachedLabel, 20
- QCPAxisPainterPrivate::TickLabelData, 466
- QCPAxisRect, 85
 - addAxes, 88
 - addAxis, 89
 - axes, 89
 - axis, 90
 - axisCount, 90
 - bottom, 90
 - bottomLeft, 90
 - bottomRight, 90
 - center, 91
 - elements, 91
 - graphs, 91
 - height, 91
 - insetLayout, 91
 - items, 92
 - left, 92
 - mouseMoveEvent, 92
 - mousePressEvent, 92
 - mouseReleaseEvent, 93
 - plottables, 93
 - QCPAxisRect, 88
 - rangeDragAxes, 94
 - rangeDragAxis, 94
 - rangeZoomAxes, 94
 - rangeZoomAxis, 94
 - rangeZoomFactor, 95
 - removeAxis, 95
 - right, 95
 - setBackground, 95, 96
 - setBackgroundScaled, 96
 - setBackgroundScaledMode, 97
 - setRangeDrag, 97
 - setRangeDragAxes, 97, 98
 - setRangeZoom, 98
 - setRangeZoomAxes, 99
 - setRangeZoomFactor, 100
 - setupFullAxesBox, 100
 - size, 100
 - top, 101
 - topLeft, 101
 - topRight, 101
 - update, 101
 - wheelEvent, 101
 - width, 102
 - zoom, 102
- QCPAxisTicker, 103
 - generate, 106
 - QCPAxisTicker, 105
 - setTickCount, 106
 - setTickOrigin, 106
 - setTickStepStrategy, 106
 - TickStepStrategy, 105
- QCPAxisTickerDateTime, 107
 - dateTimeToKey, 109
 - keyToDateTime, 109
 - QCPAxisTickerDateTime, 109
 - setDateTimeFormat, 110
 - setDateTimeSpec, 110
 - setTickOrigin, 110, 111
- QCPAxisTickerFixed, 111
 - QCPAxisTickerFixed, 113
 - ScaleStrategy, 112
 - setScaleStrategy, 113
 - setTickStep, 113
- QCPAxisTickerLog, 114
 - QCPAxisTickerLog, 115
 - setLogBase, 115
 - setSubTickCount, 115
- QCPAxisTickerPi, 115
 - FractionStyle, 117
 - QCPAxisTickerPi, 117
 - setFractionStyle, 117
 - setPeriodicity, 117
 - setPiSymbol, 118
 - setPiValue, 118
- QCPAxisTickerText, 118
 - addTick, 120
 - addTicks, 120
 - clear, 121
 - createTickVector, 121
 - getSubTickCount, 121
 - getTickLabel, 122
 - getTickStep, 122
 - QCPAxisTickerText, 119
 - setSubTickCount, 122
 - setTicks, 122, 123
 - ticks, 123
- QCPAxisTickerTime, 124
 - QCPAxisTickerTime, 125
 - setFieldWidth, 126
 - setTimeFormat, 126
 - TimeUnit, 125
- QCPBars, 127
 - addData, 130
 - barAbove, 130
 - barBelow, 130
 - data, 131
 - dataPixelPosition, 131
 - getKeyRange, 131
 - getValueRange, 131
 - moveAbove, 132
 - moveBelow, 132
 - QCPBars, 129
 - selectTest, 133
 - selectTestRect, 133
 - setBarsGroup, 133
 - setBaseValue, 134
 - setData, 134
 - setStackingGap, 135
 - setWidth, 135
 - setWidthType, 135
 - WidthType, 129

- QCPBarsData, 136
 - fromSortKey, 137
 - mainKey, 137
 - mainValue, 137
 - QCPBarsData, 136, 137
 - sortKey, 137
 - sortKeysMainKey, 138
 - valueRange, 138
- QCPBarsGroup, 138
 - append, 141
 - bars, 141
 - clear, 141
 - contains, 142
 - insert, 142
 - isEmpty, 142
 - QCPBarsGroup, 141
 - remove, 142
 - setSpacing, 143
 - setSpacingType, 143
 - size, 143
 - SpacingType, 140
- QCPColorGradient, 144
 - clearColorStops, 147
 - ColorInterpolation, 145
 - colorize, 147
 - GradientPreset, 145
 - inverted, 147
 - loadPreset, 148
 - QCPColorGradient, 146
 - setColorInterpolation, 148
 - setColorStopAt, 148
 - setColorStops, 148
 - setLevelCount, 149
 - setPeriodic, 149
- QCPColorMap, 150
 - data, 153
 - dataRangeChanged, 153
 - dataScaleTypeChanged, 153
 - getKeyRange, 153
 - getValueRange, 154
 - gradientChanged, 154
 - QCPColorMap, 152
 - rescaleDataRange, 155
 - selectTest, 155
 - setColorScale, 156
 - setData, 156
 - setDataRange, 156
 - setDataScaleType, 157
 - setGradient, 157
 - setInterpolate, 157
 - setTightBoundary, 158
 - updateLegendIcon, 158
- QCPColorMapData, 159
 - alpha, 161
 - cellToCoord, 161
 - clear, 161
 - clearAlpha, 162
 - coordToCell, 162
 - fill, 162
 - fillAlpha, 162
 - isEmpty, 163
 - operator=, 163
 - QCPColorMapData, 160
 - recalculateDataBounds, 163
 - setAlpha, 163
 - setCell, 164
 - setData, 164
 - setKeyRange, 164
 - setKeySize, 165
 - setRange, 165
 - setSize, 165
 - setValueRange, 166
 - setValueSize, 166
- QCPColorScale, 167
 - axis, 169
 - colorMaps, 169
 - dataRangeChanged, 169
 - dataScaleTypeChanged, 169
 - gradientChanged, 170
 - mouseMoveEvent, 170
 - mousePressEvent, 170
 - mouseReleaseEvent, 171
 - QCPColorScale, 169
 - rescaleDataRange, 171
 - setBarWidth, 172
 - setDataRange, 172
 - setDataScaleType, 172
 - setGradient, 172
 - setLabel, 173
 - setRangeDrag, 173
 - setRangeZoom, 173
 - setType, 173
 - update, 173
 - wheelEvent, 174
- QCPColorScaleAxisRectPrivate, 175
 - QCPColorScaleAxisRectPrivate, 175
- QCPCurve, 176
 - addData, 179, 180
 - data, 180
 - getKeyRange, 180
 - getValueRange, 181
 - LineStyle, 178
 - QCPCurve, 179
 - selectTest, 181
 - setData, 182
 - setLineStyle, 183
 - setScatterSkip, 183
 - setScatterStyle, 183
- QCPCurveData, 184
 - fromSortKey, 185
 - mainKey, 186
 - mainValue, 186
 - QCPCurveData, 185
 - sortKey, 186
 - sortKeysMainKey, 186
 - valueRange, 186

- QCPDataContainer
 - add, [189](#), [190](#)
 - at, [190](#)
 - begin, [191](#)
 - clear, [191](#)
 - constBegin, [191](#)
 - constEnd, [191](#)
 - dataRange, [191](#)
 - end, [192](#)
 - findBegin, [192](#)
 - findEnd, [192](#)
 - isEmpty, [192](#)
 - keyRange, [193](#)
 - limitIteratorsToDataRange, [193](#)
 - QCPDataContainer, [189](#)
 - qcpLessThanSortKey, [197](#)
 - remove, [193](#)
 - removeAfter, [194](#)
 - removeBefore, [194](#)
 - set, [194](#), [195](#)
 - setAutoSqueeze, [195](#)
 - size, [195](#)
 - sort, [196](#)
 - squeeze, [196](#)
 - valueRange, [196](#)
- QCPDataContainer< DataType >, [187](#)
- QCPDataRange, [197](#)
 - adjusted, [199](#)
 - bounded, [199](#)
 - contains, [199](#)
 - expanded, [199](#)
 - intersection, [199](#)
 - intersects, [200](#)
 - isEmpty, [200](#)
 - isValid, [200](#)
 - length, [200](#)
 - operator<<, [202](#)
 - QCPDataRange, [198](#)
 - setBegin, [201](#)
 - setEnd, [201](#)
 - size, [201](#)
- QCPDataSelection, [202](#)
 - addDataRange, [204](#)
 - clear, [204](#)
 - contains, [204](#)
 - dataPointCount, [204](#)
 - dataRange, [205](#)
 - dataRangeCount, [205](#)
 - dataRanges, [205](#)
 - enforceType, [205](#)
 - intersection, [206](#)
 - inverse, [206](#)
 - isEmpty, [206](#)
 - operator<<, [209](#)
 - operator+, [208](#)
 - operator+=", [207](#)
 - operator-, [209](#)
 - operator-=, [207](#)
 - operator==, [207](#)
 - QCPDataSelection, [203](#), [204](#)
 - simplify, [207](#)
 - span, [208](#)
- QCPErrorBars, [210](#)
 - addData, [213](#)
 - data, [214](#)
 - dataCount, [214](#)
 - dataMainKey, [214](#)
 - dataMainValue, [214](#)
 - dataPixelPosition, [215](#)
 - dataSortKey, [215](#)
 - dataValueRange, [215](#)
 - ErrorType, [212](#)
 - findBegin, [215](#)
 - findEnd, [216](#)
 - getKeyRange, [216](#)
 - getValueRange, [217](#)
 - interface1D, [217](#)
 - QCPErrorBars, [212](#)
 - selectTest, [218](#)
 - selectTestRect, [218](#)
 - setData, [219](#), [220](#)
 - setDataPlottable, [220](#)
 - setErrorType, [220](#)
 - setSymbolGap, [221](#)
 - setWhiskerWidth, [221](#)
 - sortKeyIsMainKey, [221](#)
- QCPErrorBarsData, [221](#)
 - QCPErrorBarsData, [222](#)
- QCPFinancial, [223](#)
 - addData, [226](#), [227](#)
 - ChartStyle, [225](#)
 - data, [227](#)
 - getKeyRange, [228](#)
 - getValueRange, [228](#)
 - QCPFinancial, [226](#)
 - selectTest, [228](#)
 - selectTestRect, [229](#)
 - setBrushNegative, [229](#)
 - setBrushPositive, [229](#)
 - setChartStyle, [230](#)
 - setData, [230](#)
 - setPenNegative, [231](#)
 - setPenPositive, [231](#)
 - setTwoColored, [231](#)
 - setWidth, [232](#)
 - setWidthType, [232](#)
 - timeSeriesToOhlc, [232](#)
 - WidthType, [226](#)
- QCPFinancialData, [233](#)
 - fromSortKey, [234](#)
 - mainKey, [235](#)
 - mainValue, [235](#)
 - QCPFinancialData, [234](#)
 - sortKey, [235](#)
 - sortKeyIsMainKey, [235](#)
 - valueRange, [235](#)

- QCPGraph, 236
 - addData, 240
 - data, 240
 - getKeyRange, 240
 - getValueRange, 241
 - getVisibleDataBounds, 241
 - LineStyle, 239
 - QCPGraph, 239
 - selectTest, 242
 - setAdaptiveSampling, 242
 - setChannelFillGraph, 242
 - setData, 243
 - setLineStyle, 244
 - setScatterSkip, 244
 - setScatterStyle, 244
- QCPGraphData, 245
 - fromSortKey, 246
 - mainKey, 246
 - mainValue, 247
 - QCPGraphData, 246
 - sortKey, 247
 - sortKeysIsMainKey, 247
 - valueRange, 247
- QCPGrid, 248
 - QCPGrid, 249
 - setAntialiasedSubGrid, 249
 - setAntialiasedZeroLine, 249
 - setPen, 249
 - setSubGridPen, 250
 - setSubGridVisible, 250
 - setZeroLinePen, 250
- QCPIItemAnchor, 251
 - pixelPosition, 252
 - QCPIItemAnchor, 252
 - toQCPIItemPosition, 252
- QCPIItemBracket, 253
 - BracketStyle, 254
 - QCPIItemBracket, 255
 - selectTest, 255
 - setLength, 255
 - setPen, 256
 - setSelectedPen, 256
 - setStyle, 256
- QCPIItemCurve, 257
 - QCPIItemCurve, 258
 - selectTest, 259
 - setHead, 259
 - setPen, 259
 - setSelectedPen, 260
 - setTail, 260
- QCPIItemEllipse, 261
 - QCPIItemEllipse, 262
 - selectTest, 262
 - setBrush, 263
 - setPen, 263
 - setSelectedBrush, 264
 - setSelectedPen, 264
- QCPIItemLine, 265
 - QCPIItemLine, 266
 - selectTest, 266
 - setHead, 267
 - setPen, 267
 - setSelectedPen, 267
 - setTail, 267
- QCPIItemPixmap, 268
 - QCPIItemPixmap, 270
 - selectTest, 270
 - setPen, 270
 - setPixmap, 271
 - setScaled, 271
 - setSelectedPen, 271
- QCPIItemPosition, 272
 - parentAnchor, 274
 - pixelPosition, 274
 - PositionType, 273
 - QCPIItemPosition, 274
 - setAxes, 275
 - setAxisRect, 275
 - setCoords, 275
 - setParentAnchor, 276
 - setParentAnchorX, 276
 - setParentAnchorY, 276
 - setPixelPosition, 277
 - setType, 277
 - setTypeX, 277
 - setTypeY, 278
 - toQCPIItemPosition, 278
 - type, 278
- QCPIItemRect, 279
 - QCPIItemRect, 280
 - selectTest, 281
 - setBrush, 281
 - setPen, 282
 - setSelectedBrush, 282
 - setSelectedPen, 282
- QCPIItemStraightLine, 283
 - QCPIItemStraightLine, 284
 - selectTest, 284
 - setPen, 284
 - setSelectedPen, 285
- QCPIItemText, 285
 - QCPIItemText, 287
 - selectTest, 288
 - setBrush, 288
 - setColor, 288
 - setFont, 289
 - setPadding, 289
 - setPen, 289
 - setPositionAlignment, 289
 - setRotation, 289
 - setSelectedBrush, 290
 - setSelectedColor, 290
 - setSelectedFont, 290
 - setSelectedPen, 290
 - setText, 290
 - setTextAlignment, 291

- QCPIItemTracer, 291
 - QCPIItemTracer, 294
 - selectTest, 294
 - setBrush, 294
 - setGraph, 295
 - setGraphKey, 295
 - setInterpolating, 295
 - setPen, 296
 - setSelectedBrush, 296
 - setSelectedPen, 296
 - setSize, 296
 - setStyle, 297
 - TracerStyle, 293
 - updatePosition, 297
- QCPLayer, 297
 - children, 300
 - index, 300
 - LayerMode, 299
 - QCPLayer, 300
 - replot, 300
 - setMode, 301
 - setVisible, 301
- QCPLayerable, 302
 - layerChanged, 304
 - mouseDoubleClickEvent, 304
 - mouseMoveEvent, 304
 - mousePressEvent, 305
 - mouseReleaseEvent, 305
 - parentLayerable, 306
 - QCPLayerable, 303
 - realVisibility, 306
 - selectTest, 306
 - setAntialiased, 307
 - setLayer, 307, 308
 - setVisible, 308
 - wheelEvent, 308
- QCPLayout, 309
 - clear, 310
 - elementAt, 310
 - elementCount, 311
 - elements, 311
 - QCPLayout, 310
 - remove, 311
 - removeAt, 312
 - simplify, 312
 - sizeConstraintsChanged, 312
 - take, 313
 - takeAt, 313
 - update, 313
- QCPLayoutElement, 314
 - elements, 317
 - layout, 317
 - maximumSizeHint, 317
 - minimumSizeHint, 317
 - QCPLayoutElement, 316
 - rect, 318
 - selectTest, 318
 - setAutoMargins, 318
 - setMarginGroup, 319
 - setMargins, 319
 - setMaximumSize, 319, 320
 - setMinimumMargins, 320
 - setMinimumSize, 320
 - setOuterRect, 321
 - update, 321
 - UpdatePhase, 316
- QCPLayoutGrid, 322
 - addElement, 324
 - columnCount, 325
 - element, 325
 - elementAt, 325
 - elementCount, 326
 - elements, 326
 - expandTo, 326
 - FillOrder, 323
 - hasElement, 327
 - indexToRowCol, 327
 - insertColumn, 327
 - insertRow, 328
 - maximumSizeHint, 328
 - minimumSizeHint, 328
 - QCPLayoutGrid, 324
 - rowColToIndex, 328
 - rowCount, 329
 - setColumnSpacing, 329
 - setColumnStretchFactor, 329
 - setColumnStretchFactors, 330
 - setFillOrder, 330
 - setRowSpacing, 330
 - setRowStretchFactor, 331
 - setRowStretchFactors, 331
 - setWrap, 331
 - simplify, 332
 - take, 332
 - takeAt, 332
- QCPLayoutInset, 333
 - addElement, 335
 - elementAt, 335
 - elementCount, 336
 - insetAlignment, 336
 - InsetPlacement, 334
 - insetPlacement, 336
 - insetRect, 336
 - QCPLayoutInset, 335
 - selectTest, 337
 - setInsetAlignment, 337
 - setInsetPlacement, 337
 - setInsetRect, 337
 - simplify, 338
 - take, 338
 - takeAt, 338
- QCPLegend, 339
 - addItem, 342
 - clearItems, 342
 - hasItem, 343
 - hasItemWithPlottable, 343

- item, 343
- itemCount, 343
- itemWithPlottable, 344
- QCPLegend, 342
- removeItem, 344
- selectTest, 345
- SelectablePart, 342
- selectedItems, 345
- selectionChanged, 345
- setBorderPen, 346
- setBrush, 346
- setFont, 346
- setIconBorderPen, 346
- setIconSize, 347
- setIconTextPadding, 347
- setSelectableParts, 347
- setSelectedBorderPen, 347
- setSelectedBrush, 348
- setSelectedFont, 348
- setSelectedIconBorderPen, 348
- setSelectedParts, 348
- setSelectedTextColor, 349
- setTextColor, 349
- QCPLineEnding, 350
 - EndingStyle, 351
 - QCPLineEnding, 351
 - realLength, 352
 - setInverted, 352
 - setLength, 352
 - setStyle, 352
 - setWidth, 352
- QCPMarginGroup, 353
 - clear, 355
 - elements, 355
 - isEmpty, 355
 - QCPMarginGroup, 354
- QCPSaveBufferPixmap, 355
 - clear, 356
 - draw, 356
 - QCPSaveBufferPixmap, 356
 - reallocateBuffer, 357
 - startPainting, 357
- QCPSavePainter, 358
 - begin, 359
 - drawLine, 360
 - makeNonCosmetic, 360
 - PainterMode, 359
 - QCPSavePainter, 359
 - restore, 360
 - save, 360
 - setAntialiasing, 361
 - setMode, 361
 - setModes, 361
 - setPen, 361, 362
- QCPSavePlottableInterface1D, 363
 - dataCount, 363
 - dataMainKey, 364
 - dataMainValue, 364
 - dataPixelPosition, 364
 - dataSortKey, 364
 - dataValueRange, 365
 - findBegin, 365
 - findEnd, 365
 - selectTestRect, 366
 - sortKeysMainKey, 366
- QCPSavePlottableLegendItem, 367
 - minimumSizeHint, 368
 - QCPSavePlottableLegendItem, 368
- QCPSaveRange, 369
 - bounded, 371
 - center, 371
 - contains, 371
 - expand, 371, 372
 - expanded, 372
 - maxRange, 376
 - minRange, 376
 - normalize, 373
 - operator<<, 376
 - operator*, 375
 - operator*&, 373
 - operator+, 375
 - operator+&, 373
 - operator-, 375
 - operator-&, 373
 - operator/, 376
 - operator/&, 373
 - QCPSaveRange, 370
 - sanitizedForLinScale, 374
 - sanitizedForLogScale, 374
 - size, 374
 - validRange, 374
- QCPSaveScatterStyle, 377
 - applyTo, 382
 - drawShape, 383
 - isNone, 383
 - isPenDefined, 383
 - QCPSaveScatterStyle, 380–382
 - ScatterProperty, 379
 - ScatterShape, 380
 - setBrush, 384
 - setCustomPath, 384
 - setFromOther, 384
 - setPen, 384
 - setPixmap, 385
 - setShape, 385
 - setSize, 385
 - undefinePen, 385
- QCPSaveSelectionDecorator, 386
 - applyBrush, 387
 - applyPen, 388
 - copyFrom, 388
 - drawDecoration, 388
 - getFinalScatterStyle, 388
 - QCPSaveSelectionDecorator, 387
 - setBrush, 389
 - setPen, 389

- setScatterStyle, 389
 - setUsedScatterProperties, 389
- QCPSelectionDecoratorBracket, 390
 - BracketStyle, 391
 - drawBracket, 391
 - drawDecoration, 392
 - QCPSelectionDecoratorBracket, 391
 - setBracketBrush, 392
 - setBracketHeight, 392
 - setBracketPen, 392
 - setBracketStyle, 392
 - setBracketWidth, 393
 - setTangentAverage, 393
 - setTangentToData, 393
- QCPSelectionRect, 394
 - accepted, 395
 - cancel, 395
 - canceled, 396
 - changed, 396
 - isActive, 396
 - QCPSelectionRect, 395
 - range, 396
 - setBrush, 397
 - setPen, 397
 - started, 397
- QCPStatisticalBox, 398
 - addData, 400
 - data, 401
 - drawStatisticalBox, 401
 - getKeyRange, 401
 - getValueRange, 402
 - QCPStatisticalBox, 400
 - selectTest, 402
 - selectTestRect, 403
 - setData, 403
 - setMedianPen, 404
 - setOutlierStyle, 404
 - setWhiskerAntialiased, 404
 - setWhiskerBarPen, 404
 - setWhiskerPen, 405
 - setWhiskerWidth, 405
 - setWidth, 405
- QCPStatisticalBoxData, 406
 - fromSortKey, 408
 - mainKey, 408
 - mainValue, 408
 - QCPStatisticalBoxData, 407
 - sortKey, 408
 - sortKeysMainKey, 409
 - valueRange, 409
- QCPTextElement, 409
 - clicked, 412
 - doubleClicked, 413
 - maximumSizeHint, 413
 - minimumSizeHint, 413
 - mouseDoubleClickEvent, 413
 - mousePressEvent, 414
 - mouseReleaseEvent, 414
 - QCPTextElement, 411, 412
 - selectTest, 414
 - selectionChanged, 414
 - setFont, 415
 - setSelectable, 415
 - setSelected, 415
 - setSelectedFont, 415
 - setSelectedTextColor, 416
 - setText, 416
 - setTextColor, 416
 - setTextFlags, 416
- QCPVector2D, 417
 - distanceSquaredToLine, 419
 - distanceToStraightLine, 420
 - dot, 420
 - isNull, 420
 - length, 420
 - lengthSquared, 421
 - normalize, 421
 - normalized, 421
 - operator<<, 423
 - operator*=:, 421
 - operator+=, 422
 - operator-=, 422
 - operator/=:, 422
 - perpendicular, 422
 - QCPVector2D, 419
 - setX, 422
 - setY, 422
 - toPoint, 423
 - toPointF, 423
- QCP, 13
 - AntialiasedElement, 14
 - ExportPen, 14
 - Interaction, 15
 - MarginSide, 15
 - PlottingHint, 16
 - ResolutionUnit, 16
 - SelectionRectMode, 17
 - SelectionType, 17
 - SignDomain, 18
- QCustomPlot, 424
 - addGraph, 429
 - addLayer, 429
 - afterReplot, 430
 - axisClick, 430
 - axisDoubleClick, 430
 - axisRect, 431
 - axisRectAt, 431
 - axisRectCount, 431
 - axisRects, 432
 - beforeReplot, 432
 - clearGraphs, 432
 - clearItems, 432
 - clearPlottables, 433
 - currentLayer, 433
 - deselectAll, 433
 - graph, 433, 434

- graphCount, [434](#)
- hasInvalidatedPaintBuffers, [434](#)
- hasItem, [434](#)
- hasPlottable, [435](#)
- item, [435](#)
- itemAt, [435](#)
- itemClick, [436](#)
- itemCount, [436](#)
- itemDoubleClick, [436](#)
- layer, [437](#)
- layerCount, [437](#)
- LayerInsertMode, [428](#)
- layoutElementAt, [438](#)
- legend, [461](#)
- legendClick, [438](#)
- legendDoubleClick, [438](#)
- mouseDoubleClick, [439](#)
- mouseMove, [439](#)
- mousePress, [439](#)
- mouseRelease, [439](#)
- mouseWheel, [440](#)
- moveLayer, [440](#)
- plotLayout, [440](#)
- plottable, [440](#), [441](#)
- plottableAt, [441](#)
- plottableClick, [441](#)
- plottableCount, [442](#)
- plottableDoubleClick, [442](#)
- QCustomPlot, [429](#)
- RefreshPriority, [428](#)
- removeGraph, [442](#), [443](#)
- removeItem, [443](#)
- removeLayer, [443](#)
- removePlottable, [444](#)
- replot, [444](#)
- rescaleAxes, [445](#)
- saveBmp, [445](#)
- saveJpg, [446](#)
- savePdf, [447](#)
- savePng, [448](#)
- saveRastered, [449](#)
- selectedAxes, [449](#)
- selectedGraphs, [449](#)
- selectedItems, [450](#)
- selectedLegends, [450](#)
- selectedPlottables, [450](#)
- selectionChangedByUser, [450](#)
- selectionRect, [451](#)
- setAntialiasedElement, [451](#)
- setAntialiasedElements, [451](#)
- setAutoAddPlottableToLegend, [452](#)
- setBackground, [452](#), [453](#)
- setBackgroundScaled, [453](#)
- setBackgroundScaledMode, [453](#)
- setBufferDevicePixelRatio, [454](#)
- setCurrentLayer, [454](#)
- setInteraction, [455](#)
- setInteractions, [455](#)
- setMultiSelectModifier, [456](#)
- setNoAntialiasingOnDrag, [456](#)
- setNotAntialiasedElement, [457](#)
- setNotAntialiasedElements, [457](#)
- setOpenGL, [457](#)
- setPlottingHint, [458](#)
- setPlottingHints, [458](#)
- setSelectionRect, [459](#)
- setSelectionRectMode, [459](#)
- setSelectionTolerance, [459](#)
- setViewport, [460](#)
- toPainter, [460](#)
- toPixmap, [460](#)
- xAxis, [461](#)
- xAxis2, [461](#)
- yAxis, [462](#)
- yAxis2, [462](#)
- qcpLessThanSortKey
 - QCPDataContainer, [197](#)
- range
 - QCPSelectionRect, [396](#)
- rangeChanged
 - QCPAxis, [69](#)
- rangeDragAxes
 - QCPAxisRect, [94](#)
- rangeDragAxis
 - QCPAxisRect, [94](#)
- rangeZoomAxes
 - QCPAxisRect, [94](#)
- rangeZoomAxis
 - QCPAxisRect, [94](#)
- rangeZoomFactor
 - QCPAxisRect, [95](#)
- realLength
 - QCPLineEnding, [352](#)
- realVisibility
 - QCPLayerable, [306](#)
- reallocateBuffer
 - QCPAbstractPaintBuffer, [37](#)
 - QCPPaintBufferPixmap, [357](#)
- recalculateDataBounds
 - QCPColorMapData, [163](#)
- rect
 - QCPLayoutElement, [318](#)
- RefreshPriority
 - QCustomPlot, [428](#)
- remove
 - QCPBarsGroup, [142](#)
 - QCPDataContainer, [193](#)
 - QCPLayout, [311](#)
- removeAfter
 - QCPDataContainer, [194](#)
- removeAt
 - QCPLayout, [312](#)
- removeAxis
 - QCPAxisRect, [95](#)
- removeBefore
 - QCPDataContainer, [194](#)

- removeFromLegend
 - QCPAbstractPlottable, [46](#), [47](#)
- removeGraph
 - QCustomPlot, [442](#), [443](#)
- removeItem
 - QCPLegend, [344](#)
 - QCustomPlot, [443](#)
- removeLayer
 - QCustomPlot, [443](#)
- removePlottable
 - QCustomPlot, [444](#)
- replot
 - QCPLayer, [300](#)
 - QCustomPlot, [444](#)
- rescale
 - QCPAxis, [69](#)
- rescaleAxes
 - QCPAbstractPlottable, [47](#)
 - QCustomPlot, [445](#)
- rescaleDataRange
 - QCPColorMap, [155](#)
 - QCPColorScale, [171](#)
- rescaleKeyAxis
 - QCPAbstractPlottable, [47](#)
- rescaleValueAxis
 - QCPAbstractPlottable, [48](#)
- ResolutionUnit
 - QCP, [16](#)
- restore
 - QCPPainter, [360](#)
- right
 - QCPAxisRect, [95](#)
- Rohr, [463](#)
 - get_kA, [463](#), [464](#)
 - get_querschnitt, [464](#)
 - get_radius, [464](#)
- Rohrstroemung, [465](#)
 - get_lambda, [466](#)
- rowColToIndex
 - QCPLayoutGrid, [328](#)
- rowCount
 - QCPLayoutGrid, [329](#)
- sanitizedForLinScale
 - QCPRange, [374](#)
- sanitizedForLogScale
 - QCPRange, [374](#)
- save
 - QCPPainter, [360](#)
- saveBmp
 - QCustomPlot, [445](#)
- saveJpg
 - QCustomPlot, [446](#)
- savePdf
 - QCustomPlot, [447](#)
- savePng
 - QCustomPlot, [448](#)
- saveRastered
 - QCustomPlot, [449](#)
- scaleRange
 - QCPAxis, [70](#)
- ScaleStrategy
 - QCPAxisTickerFixed, [112](#)
- ScaleType
 - QCPAxis, [65](#)
- scaleTypeChanged
 - QCPAxis, [70](#)
- ScatterProperty
 - QCPScatterStyle, [379](#)
- ScatterShape
 - QCPScatterStyle, [380](#)
- selectTest
 - QCPAbstractItem, [29](#)
 - QCPAbstractLegendItem, [33](#)
 - QCPAbstractPlottable, [49](#)
 - QCPAbstractPlottable1D, [58](#)
 - QCPAxis, [71](#)
 - QCPBars, [133](#)
 - QCPColorMap, [155](#)
 - QCPCurve, [181](#)
 - QCPErrorBars, [218](#)
 - QCPFinancial, [228](#)
 - QCPGraph, [242](#)
 - QCPIItemBracket, [255](#)
 - QCPIItemCurve, [259](#)
 - QCPIItemEllipse, [262](#)
 - QCPIItemLine, [266](#)
 - QCPIItemPixmap, [270](#)
 - QCPIItemRect, [281](#)
 - QCPIItemStraightLine, [284](#)
 - QCPIItemText, [288](#)
 - QCPIItemTracer, [294](#)
 - QCPLayerable, [306](#)
 - QCPLayoutElement, [318](#)
 - QCPLayoutInset, [337](#)
 - QCPLegend, [345](#)
 - QCPStatisticalBox, [402](#)
 - QCPTextElement, [414](#)
- selectTestRect
 - QCPAbstractPlottable1D, [59](#)
 - QCPBars, [133](#)
 - QCPErrorBars, [218](#)
 - QCPFinancial, [229](#)
 - QCPPlottableInterface1D, [366](#)
 - QCPStatisticalBox, [403](#)
- selectableChanged
 - QCPAbstractPlottable, [48](#)
 - QCPAxis, [70](#)
- SelectablePart
 - QCPAxis, [65](#)
 - QCPLegend, [342](#)
- selected
 - QCPAbstractPlottable, [48](#)
- selectedAxes
 - QCustomPlot, [449](#)
- selectedGraphs
 - QCustomPlot, [449](#)

- selectedItems
 - QCPLegend, 345
 - QCustomPlot, 450
- selectedLegends
 - QCustomPlot, 450
- selectedPlottables
 - QCustomPlot, 450
- selection
 - QCPAbstractPlottable, 48
- selectionChanged
 - QCPAbstractItem, 29
 - QCPAbstractLegendItem, 33
 - QCPAbstractPlottable, 48, 49
 - QCPAxis, 71
 - QCPLegend, 345
 - QCPTextElement, 414
- selectionChangedByUser
 - QCustomPlot, 450
- selectionDecorator
 - QCPAbstractPlottable, 49
- selectionRect
 - QCustomPlot, 451
- SelectionRectMode
 - QCP, 17
- SelectionType
 - QCP, 17
- set
 - QCPDataContainer, 194, 195
- setAdaptiveSampling
 - QCPGraph, 242
- setAlpha
 - QCPColorMapData, 163
- setAntialiased
 - QCPLayerable, 307
- setAntialiasedElement
 - QCustomPlot, 451
- setAntialiasedElements
 - QCustomPlot, 451
- setAntialiasedFill
 - QCPAbstractPlottable, 50
- setAntialiasedScatters
 - QCPAbstractPlottable, 50
- setAntialiasedSubGrid
 - QCPGrid, 249
- setAntialiasedZeroLine
 - QCPGrid, 249
- setAntialiasing
 - QCPPainter, 361
- setAutoAddPlottableToLegend
 - QCustomPlot, 452
- setAutoMargins
 - QCPLayoutElement, 318
- setAutoSqueeze
 - QCPDataContainer, 195
- setAxes
 - QCPItemPosition, 275
- setAxisRect
 - QCPItemPosition, 275
- setBackground
 - QCPAxisRect, 95, 96
 - QCustomPlot, 452, 453
- setBackgroundScaled
 - QCPAxisRect, 96
 - QCustomPlot, 453
- setBackgroundScaledMode
 - QCPAxisRect, 97
 - QCustomPlot, 453
- setBarWidth
 - QCPColorScale, 172
- setBarsGroup
 - QCPBars, 133
- setBasePen
 - QCPAxis, 71
- setBaseValue
 - QCPBars, 134
- setBegin
 - QCPDataRange, 201
- setBorderPen
 - QCPLegend, 346
- setBracketBrush
 - QCPSelectionDecoratorBracket, 392
- setBracketHeight
 - QCPSelectionDecoratorBracket, 392
- setBracketPen
 - QCPSelectionDecoratorBracket, 392
- setBracketStyle
 - QCPSelectionDecoratorBracket, 392
- setBracketWidth
 - QCPSelectionDecoratorBracket, 393
- setBrush
 - QCPAbstractPlottable, 51
 - QCPItemEllipse, 263
 - QCPItemRect, 281
 - QCPItemText, 288
 - QCPItemTracer, 294
 - QCPLegend, 346
 - QCPScatterStyle, 384
 - QCPSelectionDecorator, 389
 - QCPSelectionRect, 397
- setBrushNegative
 - QCPFinancial, 229
- setBrushPositive
 - QCPFinancial, 229
- setBufferDevicePixelRatio
 - QCustomPlot, 454
- setCell
 - QCPColorMapData, 164
- setChannelFillGraph
 - QCPGraph, 242
- setChartStyle
 - QCPFinancial, 230
- setClipAxisRect
 - QCPAbstractItem, 30
- setClipToAxisRect
 - QCPAbstractItem, 30
- setColor

- QCPItemText, 288
- setColorInterpolation
 - QCPColorGradient, 148
- setColorScale
 - QCPColorMap, 156
- setColorStopAt
 - QCPColorGradient, 148
- setColorStops
 - QCPColorGradient, 148
- setColumnSpacing
 - QCPLayoutGrid, 329
- setColumnStretchFactor
 - QCPLayoutGrid, 329
- setColumnStretchFactors
 - QCPLayoutGrid, 330
- setCoords
 - QCPItemPosition, 275
- setCurrentLayer
 - QCustomPlot, 454
- setCustomPath
 - QCPScatterStyle, 384
- setData
 - QCPBars, 134
 - QCPColorMap, 156
 - QCPColorMapData, 164
 - QCPCurve, 182
 - QCPErrorBars, 219, 220
 - QCPFinancial, 230
 - QCPGraph, 243
 - QCPStatisticalBox, 403
- setDataPlottable
 - QCPErrorBars, 220
- setDataRange
 - QCPColorMap, 156
 - QCPColorScale, 172
- setDataScaleType
 - QCPColorMap, 157
 - QCPColorScale, 172
- setDateTimeFormat
 - QCPAxisTickerDateTime, 110
- setDateTimeSpec
 - QCPAxisTickerDateTime, 110
- setDevicePixelRatio
 - QCPAbstractPaintBuffer, 38
- setEnd
 - QCPDataRange, 201
- setErrorType
 - QCPErrorBars, 220
- setFieldWidth
 - QCPAxisTickerTime, 126
- setFillOrder
 - QCPLayoutGrid, 330
- setFont
 - QCPAbstractLegendItem, 34
 - QCPItemText, 289
 - QCPLegend, 346
 - QCPTextElement, 415
- setFractionStyle
 - QCPAxisTickerPi, 117
- setFromOther
 - QCPScatterStyle, 384
- setGradient
 - QCPColorMap, 157
 - QCPColorScale, 172
- setGraph
 - QCPItemTracer, 295
- setGraphKey
 - QCPItemTracer, 295
- setHead
 - QCPItemCurve, 259
 - QCPItemLine, 267
- setIconBorderPen
 - QCPLegend, 346
- setIconSize
 - QCPLegend, 347
- setIconTextPadding
 - QCPLegend, 347
- setInsetAlignment
 - QCPLayoutInset, 337
- setInsetPlacement
 - QCPLayoutInset, 337
- setInsetRect
 - QCPLayoutInset, 337
- setInteraction
 - QCustomPlot, 455
- setInteractions
 - QCustomPlot, 455
- setInterpolate
 - QCPColorMap, 157
- setInterpolating
 - QCPItemTracer, 295
- setInvalidated
 - QCPAbstractPaintBuffer, 38
- setInverted
 - QCPLineEnding, 352
- setKeyAxis
 - QCPAbstractPlottable, 51
- setKeyRange
 - QCPColorMapData, 164
- setKeySize
 - QCPColorMapData, 165
- setLabel
 - QCPAxis, 72
 - QCPColorScale, 173
- setLabelColor
 - QCPAxis, 72
- setLabelFont
 - QCPAxis, 72
- setLabelPadding
 - QCPAxis, 72
- setLayer
 - QCPLayerable, 307, 308
- setLength
 - QCPItemBracket, 255
 - QCPLineEnding, 352
- setLevelCount

- QCPColorGradient, [149](#)
- setLineStyle
 - QCPCurve, [183](#)
 - QCPGraph, [244](#)
- setLogBase
 - QCPAxisTickerLog, [115](#)
- setLowerEnding
 - QCPAxis, [73](#)
- setMarginGroup
 - QCPLayoutElement, [319](#)
- setMargins
 - QCPLayoutElement, [319](#)
- setMaximumSize
 - QCPLayoutElement, [319](#), [320](#)
- setMedianPen
 - QCPStatisticalBox, [404](#)
- setMinimumMargins
 - QCPLayoutElement, [320](#)
- setMinimumSize
 - QCPLayoutElement, [320](#)
- setMode
 - QCPLayer, [301](#)
 - QCPPainter, [361](#)
- setModes
 - QCPPainter, [361](#)
- setMultiSelectModifier
 - QCustomPlot, [456](#)
- setName
 - QCPAbstractPlottable, [51](#)
- setNoAntialiasingOnDrag
 - QCustomPlot, [456](#)
- setNotAntialiasedElement
 - QCustomPlot, [457](#)
- setNotAntialiasedElements
 - QCustomPlot, [457](#)
- setNumberFormat
 - QCPAxis, [73](#)
- setNumberPrecision
 - QCPAxis, [73](#)
- setOffset
 - QCPAxis, [74](#)
- setOpenGL
 - QCustomPlot, [457](#)
- setOuterRect
 - QCPLayoutElement, [321](#)
- setOutlierStyle
 - QCPStatisticalBox, [404](#)
- setPadding
 - QCPAxis, [74](#)
 - QCPItemText, [289](#)
- setParentAnchor
 - QCPItemPosition, [276](#)
- setParentAnchorX
 - QCPItemPosition, [276](#)
- setParentAnchorY
 - QCPItemPosition, [276](#)
- setPen
 - QCPAbstractPlottable, [51](#)
 - QCPGrid, [249](#)
 - QCPItemBracket, [256](#)
 - QCPItemCurve, [259](#)
 - QCPItemEllipse, [263](#)
 - QCPItemLine, [267](#)
 - QCPItemPixmap, [270](#)
 - QCPItemRect, [282](#)
 - QCPItemStraightLine, [284](#)
 - QCPItemText, [289](#)
 - QCPItemTracer, [296](#)
 - QCPPainter, [361](#), [362](#)
 - QCPScatterStyle, [384](#)
 - QCPSelectionDecorator, [389](#)
 - QCPSelectionRect, [397](#)
- setPenNegative
 - QCPFinancial, [231](#)
- setPenPositive
 - QCPFinancial, [231](#)
- setPeriodic
 - QCPColorGradient, [149](#)
- setPeriodicity
 - QCPAxisTickerPi, [117](#)
- setPiSymbol
 - QCPAxisTickerPi, [118](#)
- setPiValue
 - QCPAxisTickerPi, [118](#)
- setPixelPosition
 - QCPItemPosition, [277](#)
- setPixmap
 - QCPItemPixmap, [271](#)
 - QCPScatterStyle, [385](#)
- setPlottingHint
 - QCustomPlot, [458](#)
- setPlottingHints
 - QCustomPlot, [458](#)
- setPositionAlignment
 - QCPItemText, [289](#)
- setRange
 - QCPAxis, [74](#), [75](#)
 - QCPColorMapData, [165](#)
- setRangeDrag
 - QCPAxisRect, [97](#)
 - QCPColorScale, [173](#)
- setRangeDragAxes
 - QCPAxisRect, [97](#), [98](#)
- setRangeLower
 - QCPAxis, [75](#)
- setRangeReversed
 - QCPAxis, [75](#)
- setRangeUpper
 - QCPAxis, [75](#)
- setRangeZoom
 - QCPAxisRect, [98](#)
 - QCPColorScale, [173](#)
- setRangeZoomAxes
 - QCPAxisRect, [99](#)
- setRangeZoomFactor
 - QCPAxisRect, [100](#)

- setRotation
 - QCItemText, 289
- setRowSpacing
 - QCLayoutGrid, 330
- setRowStretchFactor
 - QCLayoutGrid, 331
- setRowStretchFactors
 - QCLayoutGrid, 331
- setScaleRatio
 - QCPAxis, 76
- setScaleStrategy
 - QCPAxisTickerFixed, 113
- setScaleType
 - QCPAxis, 76
- setScaled
 - QCItemPixmap, 271
- setScatterSkip
 - QCPCurve, 183
 - QCPCurve, 244
- setScatterStyle
 - QCPCurve, 183
 - QCPCurve, 244
 - QCPSelectionDecorator, 389
- setSelectable
 - QCAbstractItem, 30
 - QCAbstractLegendItem, 34
 - QCAbstractPlottable, 52
 - QCTextElement, 415
- setSelectableParts
 - QCPAxis, 76
 - QCPLegend, 347
- setSelected
 - QCAbstractItem, 31
 - QCAbstractLegendItem, 34
 - QCTextElement, 415
- setSelectedBasePen
 - QCPAxis, 76
- setSelectedBorderPen
 - QCPLegend, 347
- setSelectedBrush
 - QCItemEllipse, 264
 - QCItemRect, 282
 - QCItemText, 290
 - QCItemTracer, 296
 - QCPLegend, 348
- setSelectedColor
 - QCItemText, 290
- setSelectedFont
 - QCAbstractLegendItem, 35
 - QCItemText, 290
 - QCPLegend, 348
 - QCTextElement, 415
- setSelectedIconBorderPen
 - QCPLegend, 348
- setSelectedLabelColor
 - QCPAxis, 77
- setSelectedLabelFont
 - QCPAxis, 77
- setSelectedParts
 - QCPAxis, 77
 - QCPLegend, 348
- setSelectedPen
 - QCItemBracket, 256
 - QCItemCurve, 260
 - QCItemEllipse, 264
 - QCItemLine, 267
 - QCItemPixmap, 271
 - QCItemRect, 282
 - QCItemStraightLine, 285
 - QCItemText, 290
 - QCItemTracer, 296
- setSelectedSubTickPen
 - QCPAxis, 77
- setSelectedTextColor
 - QCAbstractLegendItem, 35
 - QCPLegend, 349
 - QCTextElement, 416
- setSelectedTickLabelColor
 - QCPAxis, 78
- setSelectedTickLabelFont
 - QCPAxis, 78
- setSelectedTickPen
 - QCPAxis, 78
- setSelection
 - QCAbstractPlottable, 52
- setSelectionDecorator
 - QCAbstractPlottable, 52
- setSelectionRect
 - QCustomPlot, 459
- setSelectionRectMode
 - QCustomPlot, 459
- setSelectionTolerance
 - QCustomPlot, 459
- setShape
 - QCPScatterStyle, 385
- setSize
 - QCAbstractPaintBuffer, 38
 - QCColorMapData, 165
 - QCItemTracer, 296
 - QCPScatterStyle, 385
- setSpacing
 - QCBarsGroup, 143
- setSpacingType
 - QCBarsGroup, 143
- setStackingGap
 - QCBars, 135
- setStyle
 - QCItemBracket, 256
 - QCItemTracer, 297
 - QCLineEnding, 352
- setSubGridPen
 - QCGrid, 250
- setSubGridVisible
 - QCGrid, 250
- setSubTickCount
 - QCPAxisTickerLog, 115

- QCPAxisTickerText, 122
- setSubTickLength
 - QCPAxis, 78
- setSubTickLengthIn
 - QCPAxis, 79
- setSubTickLengthOut
 - QCPAxis, 79
- setSubTickPen
 - QCPAxis, 79
- setSubTicks
 - QCPAxis, 79
- setSymbolGap
 - QCPErrorBars, 221
- setTail
 - QCPItemCurve, 260
 - QCPItemLine, 267
- setTangentAverage
 - QCPSelectionDecoratorBracket, 393
- setTangentToData
 - QCPSelectionDecoratorBracket, 393
- setText
 - QCPItemText, 290
 - QCPTextElement, 416
- setTextAlignment
 - QCPItemText, 291
- setTextColor
 - QCPAbstractLegendItem, 35
 - QCPLegend, 349
 - QCPTextElement, 416
- setTextFlags
 - QCPTextElement, 416
- setTickCount
 - QCPAxisTicker, 106
- setTickLabelColor
 - QCPAxis, 80
- setTickLabelFont
 - QCPAxis, 80
- setTickLabelPadding
 - QCPAxis, 81
- setTickLabelRotation
 - QCPAxis, 81
- setTickLabelSide
 - QCPAxis, 81
- setTickLabels
 - QCPAxis, 81
- setTickLength
 - QCPAxis, 81
- setTickLengthIn
 - QCPAxis, 82
- setTickLengthOut
 - QCPAxis, 82
- setTickOrigin
 - QCPAxisTicker, 106
 - QCPAxisTickerDateTime, 110, 111
- setTickPen
 - QCPAxis, 82
- setTickStep
 - QCPAxisTickerFixed, 113
- setTickStepStrategy
 - QCPAxisTicker, 106
- setTicker
 - QCPAxis, 80
- setTicks
 - QCPAxis, 82
 - QCPAxisTickerText, 122, 123
- setTightBoundary
 - QCPColorMap, 158
- setTimeFormat
 - QCPAxisTickerTime, 126
- setTwoColored
 - QCPFinancial, 231
- setType
 - QCPColorScale, 173
 - QCPItemPosition, 277
- setTypeX
 - QCPItemPosition, 277
- setTypeY
 - QCPItemPosition, 278
- setUpperEnding
 - QCPAxis, 83
- setUsedScatterProperties
 - QCPSelectionDecorator, 389
- setValueAxis
 - QCPAbstractPlottable, 53
- setValueRange
 - QCPColorMapData, 166
- setValueSize
 - QCPColorMapData, 166
- setViewport
 - QCustomPlot, 460
- setVisible
 - QCPLayer, 301
 - QCPLayerable, 308
- setWhiskerAntialiased
 - QCPStatisticalBox, 404
- setWhiskerBarPen
 - QCPStatisticalBox, 404
- setWhiskerPen
 - QCPStatisticalBox, 405
- setWhiskerWidth
 - QCPErrorBars, 221
 - QCPStatisticalBox, 405
- setWidth
 - QCPBars, 135
 - QCPFinancial, 232
 - QCPLineEnding, 352
 - QCPStatisticalBox, 405
- setWidthType
 - QCPBars, 135
 - QCPFinancial, 232
- setWrap
 - QCPLayoutGrid, 331
- setZeroLinePen
 - QCPGrid, 250
- setupFullAxesBox
 - QCPAxisRect, 100

- setX
 - QCPVector2D, [422](#)
- setY
 - QCPVector2D, [422](#)
- SignDomain
 - QCP, [18](#)
- simplify
 - QCPDataSelection, [207](#)
 - QCPLayout, [312](#)
 - QCPLayoutGrid, [332](#)
 - QCPLayoutInset, [338](#)
- size
 - QCPAxisRect, [100](#)
 - QCPBarsGroup, [143](#)
 - QCPDataContainer, [195](#)
 - QCPDataRange, [201](#)
 - QCPRange, [374](#)
- sizeConstraintsChanged
 - QCPLayout, [312](#)
- sort
 - QCPDataContainer, [196](#)
- sortKey
 - QCPBarsData, [137](#)
 - QCPCurveData, [186](#)
 - QCPFinancialData, [235](#)
 - QCPGraphData, [247](#)
 - QCPStatisticalBoxData, [408](#)
- sortKeysMainKey
 - QCPAbstractPlottable1D, [59](#)
 - QCPBarsData, [138](#)
 - QCPCurveData, [186](#)
 - QCPErrorBars, [221](#)
 - QCPFinancialData, [235](#)
 - QCPGraphData, [247](#)
 - QCPPlottableInterface1D, [366](#)
 - QCPStatisticalBoxData, [409](#)
- SpacingType
 - QCPBarsGroup, [140](#)
- span
 - QCPDataSelection, [208](#)
- squeeze
 - QCPDataContainer, [196](#)
- startPainting
 - QCPAbstractPaintBuffer, [39](#)
 - QCPPaintBufferPixmap, [357](#)
- started
 - QCPSelectionRect, [397](#)
- take
 - QCPLayout, [313](#)
 - QCPLayoutGrid, [332](#)
 - QCPLayoutInset, [338](#)
- takeAt
 - QCPLayout, [313](#)
 - QCPLayoutGrid, [332](#)
 - QCPLayoutInset, [338](#)
- TickStepStrategy
 - QCPAxisTicker, [105](#)
- ticker
 - QCPAxis, [83](#)
- ticks
 - QCPAxisTickerText, [123](#)
- timeSeriesToOhlc
 - QCPFinancial, [232](#)
- TimeUnit
 - QCPAxisTickerTime, [125](#)
- toPainter
 - QCustomPlot, [460](#)
- toPixmap
 - QCustomPlot, [460](#)
- toPoint
 - QCPVector2D, [423](#)
- toPointF
 - QCPVector2D, [423](#)
- toQCPItemPosition
 - QCPItemAnchor, [252](#)
 - QCPItemPosition, [278](#)
- top
 - QCPAxisRect, [101](#)
- topLeft
 - QCPAxisRect, [101](#)
- topRight
 - QCPAxisRect, [101](#)
- TracerStyle
 - QCPItemTracer, [293](#)
- type
 - QCPItemPosition, [278](#)
- undefinePen
 - QCPScatterStyle, [385](#)
- update
 - QCPAxisRect, [101](#)
 - QCPColorScale, [173](#)
 - QCPLayout, [313](#)
 - QCPLayoutElement, [321](#)
- updateLegendIcon
 - QCPColorMap, [158](#)
- UpdatePhase
 - QCPLayoutElement, [316](#)
- updatePosition
 - QCPItemTracer, [297](#)
- validRange
 - QCPRange, [374](#)
- valueRange
 - QCPBarsData, [138](#)
 - QCPCurveData, [186](#)
 - QCPDataContainer, [196](#)
 - QCPFinancialData, [235](#)
 - QCPGraphData, [247](#)
 - QCPStatisticalBoxData, [409](#)
- wheelEvent
 - QCPAxisRect, [101](#)
 - QCPColorScale, [174](#)
 - QCPLayerable, [308](#)
- width
 - QCPAxisRect, [102](#)

WidthType

- QCPBars, [129](#)
- QCPFinancial, [226](#)

xAxis

- QCustomPlot, [461](#)

xAxis2

- QCustomPlot, [461](#)

yAxis

- QCustomPlot, [462](#)

yAxis2

- QCustomPlot, [462](#)

zoom

- QCPAxisRect, [102](#)