 "Fossies" - the Fresh Open Source Software Archive 

Member "qtiplot-0.9.8.9/3rdparty/qwt/src/qwt_slider.cpp" (7 Oct 2010, 22256 Bytes) of archive /linux/misc/qtiplot-0.9.8.9.tar.gz:

As a special service "Fossies" has tried to format the requested source page into HTML format using (guessed) C and C++ source code syntax highlighting with prefixed line numbers. Alternatively you can here view or download the uninterpreted source code file. A member file download can also be achieved by clicking within a package contents listing on the according byte size field.

```
  1 /* -*- mode: C++ ; c-file-style: "stroustrup" -*- *****************************
  2  * Qwt Widget Library
  3  * Copyright (C) 1997   Josef Wilgen
  4  * Copyright (C) 2002   Uwe Rathmann
  5  *
  6  * This library is free software; you can redistribute it and/or
  7  * modify it under the terms of the Qwt License, Version 1.0
  8  *****************************************************************************/
  9
 10 // vim: expandtab
 11
 12 #include <math.h>
 13 #include <qevent.h>
 14 #include <qdrawutil.h>
 15 #include <qpainter.h>
 16 #include "qwt_painter.h"
 17 #include "qwt_paint_buffer.h"
 18 #include "qwt_scale_draw.h"
 19 #include "qwt_scale_map.h"
 20 #include "qwt_slider.h"
 21
 22 class QwtSlider::PrivateData
 23 {
 24 public:
 25     QRect sliderRect;
 26
 27     int thumbLength;
 28     int thumbWidth;
 29     int borderWidth;
 30     int scaleDist;
 31     int xMargin;
 32     int yMargin;
 33
 34     QwtSlider::ScalePos scalePos;
 35     QwtSlider::BGSTYLE bgStyle;
 36
 37     /*
 38       Scale and values might have different maps. This is
 39       confusing and I can't see strong arguments for such
 40       a feature. TODO ...
 41      */
 42     QwtScaleMap map; // linear map
 43     mutable QSize sizeHintCache;
 44 };
 45
 46 /*!
 47   \brief Constructor
 48   \param parent parent widget
 49   \param orientation Orientation of the slider. Can be Qt::Horizontal
 50          or Qt::Vertical. Defaults to Qt::Horizontal.
 51   \param scalePos Position of the scale.
```

```
52           Defaults to QwtSlider::NoScale.
53   \param bgStyle Background style. QwtSlider::BgTrough draws the
54           slider button in a trough, QwtSlider::BgSlot draws
55           a slot underneath the button. An or-combination of both
56           may also be used. The default is QwtSlider::BgTrough.
57
58   QwtSlider enforces valid combinations of its orientation and scale position.
59   If the combination is invalid, the scale position will be set to NoScale.
60   Valid combinations are:
61   - Qt::Horizonal with NoScale, TopScale, or BottomScale;
62   - Qt::Vertical with NoScale, LeftScale, or RightScale.
63 */
64 QwtSlider::QwtSlider(QWidget *parent,
65         Qt::Orientation orientation, ScalePos scalePos, BGSTYLE bgStyle):
66     QwtAbstractSlider(orientation, parent)
67 {
68     initSlider(orientation, scalePos, bgStyle);
69 }
70
71 #if QT_VERSION < 0x040000
72 /*!
73   \brief Constructor
74
75   Build a horizontal slider with no scale and BgTrough as
76   background style
77
78   \param parent parent widget
79   \param name Object name
80 */
81 QwtSlider::QwtSlider(QWidget *parent, const char* name):
82     QwtAbstractSlider(Qt::Horizontal, parent)
83 {
84     setName(name);
85     initSlider(Qt::Horizontal, NoScale, BgTrough);
86 }
87 #endif
88
89 void QwtSlider::initSlider(Qt::Orientation orientation,
90     ScalePos scalePos, BGSTYLE bgStyle)
91 {
92     if (orientation == Qt::Vertical)
93         setSizePolicy(QSizePolicy::Fixed, QSizePolicy::Expanding);
94     else
95         setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Fixed);
96
97 #if QT_VERSION >= 0x040000
98     setAttribute(Qt::WA_WState_OwnSizePolicy, false);
99 #else
100     clearWState( WState_OwnSizePolicy );
101 #endif
102
103
104 #if QT_VERSION < 0x040000
105     setWFlags(Qt::WNoAutoErase);
106 #endif
107
108     d_data = new QwtSlider::PrivateData;
109
110     d_data->borderWidth = 2;
111     d_data->scaleDist = 4;
112     d_data->scalePos = scalePos;
113     d_data->xMargin = 0;
114     d_data->yMargin = 0;
115     d_data->bgStyle = bgStyle;
116
117     if (bgStyle == BgSlot)
118     {
119         d_data->thumbLength = 16;
```

```
120         d_data->thumbWidth = 30;
121     }
122     else
123     {
124         d_data->thumbLength = 31;
125         d_data->thumbWidth = 16;
126     }
127 
128     d_data->sliderRect.setRect(0,0,8,8);
129 
130     QwtScaleDraw::Alignment align;
131     if ( orientation == Qt::Vertical )
132     {
133         // enforce a valid combination of scale position and orientation
134         if ((d_data->scalePos == BottomScale) || (d_data->scalePos == TopScale))
135             d_data->scalePos = NoScale;
136         // adopt the policy of layoutSlider (NoScale lays out like Left)
137         if (d_data->scalePos == RightScale)
138            align = QwtScaleDraw::RightScale;
139         else
140            align = QwtScaleDraw::LeftScale;
141     }
142     else
143     {
144         // enforce a valid combination of scale position and orientation
145         if ((d_data->scalePos == LeftScale) || (d_data->scalePos == RightScale))
146             d_data->scalePos = NoScale;
147         // adopt the policy of layoutSlider (NoScale lays out like Bottom)
148         if (d_data->scalePos == TopScale)
149            align = QwtScaleDraw::TopScale;
150         else
151            align = QwtScaleDraw::BottomScale;
152     }
153 
154     scaleDraw()->setAlignment(align);
155     scaleDraw()->setLength(100);
156 
157     setRange(0.0, 100.0, 1.0);
158     setValue(0.0);
159 }
160 
161 QwtSlider::~QwtSlider()
162 {
163     delete d_data;
164 }
165 
166 /*!
167   \brief Set the orientation.
168   \param o Orientation. Allowed values are Qt::Horizontal and Qt::Vertical.
169 
170   If the new orientation and the old scale position are an invalid combination,
171   the scale position will be set to QwtSlider::NoScale.
172   \sa QwtAbstractSlider::orientation()
173 */
174 void QwtSlider::setOrientation(Qt::Orientation o)
175 {
176     if ( o == orientation() )
177         return;
178 
179     if (o == Qt::Horizontal)
180     {
181         if ((d_data->scalePos == LeftScale) || (d_data->scalePos == RightScale))
182             d_data->scalePos = NoScale;
183     }
184     else // if (o == Qt::Vertical)
185     {
186         if ((d_data->scalePos == BottomScale) || (d_data->scalePos == TopScale))
187             d_data->scalePos = NoScale;
```

```
188     }
189
190 #if QT_VERSION >= 0x040000
191     if ( !testAttribute(Qt::WA_WState_OwnSizePolicy) )
192 #else
193     if ( !testWState( WState_OwnSizePolicy ) )
194 #endif
195     {
196         QSizePolicy sp = sizePolicy();
197         sp.transpose();
198         setSizePolicy(sp);
199
200 #if QT_VERSION >= 0x040000
201         setAttribute(Qt::WA_WState_OwnSizePolicy, false);
202 #else
203         clearWState( WState_OwnSizePolicy );
204 #endif
205     }
206
207     QwtAbstractSlider::setOrientation(o);
208     layoutSlider();
209 }
210
211 /*!
212   \brief Change the scale position (and slider orientation).
213
214   \param s Position of the scale.
215
216   A valid combination of scale position and orientation is enforced:
217   - if the new scale position is Left or Right, the scale orientation will
218     become Qt::Vertical;
219   - if the new scale position is Bottom or Top the scale orientation will
220     become Qt::Horizontal;
221   - if the new scale position is QwtSlider::NoScale, the scale
222     orientation will not change.
223 */
224 void QwtSlider::setScalePosition(ScalePos s)
225 {
226     if ( d_data->scalePos == s )
227         return;
228
229     d_data->scalePos = s;
230
231     switch(d_data->scalePos)
232     {
233         case BottomScale:
234         {
235             setOrientation(Qt::Horizontal);
236             scaleDraw()->setAlignment(QwtScaleDraw::BottomScale);
237             break;
238         }
239         case TopScale:
240         {
241             setOrientation(Qt::Horizontal);
242             scaleDraw()->setAlignment(QwtScaleDraw::TopScale);
243             break;
244         }
245         case LeftScale:
246         {
247             setOrientation(Qt::Vertical);
248             scaleDraw()->setAlignment(QwtScaleDraw::LeftScale);
249             break;
250         }
251         case RightScale:
252         {
253             setOrientation(Qt::Vertical);
254             scaleDraw()->setAlignment(QwtScaleDraw::RightScale);
255             break;
```

```
256         }
257         default:
258         {
259             // nothing
260         }
261     }
262 
263     layoutSlider();
264 }
265 
266 //! Return the scale position.
267 QwtSlider::ScalePos QwtSlider::scalePosition() const
268 {
269     return d_data->scalePos;
270 }
271 
272 /*!
273   \brief Change the slider's border width
274   \param bd border width
275 */
276 void QwtSlider::setBorderWidth(int bd)
277 {
278     if ( bd < 0 )
279         bd = 0;
280 
281     if ( bd != d_data->borderWidth )
282     {
283         d_data->borderWidth = bd;
284         layoutSlider();
285     }
286 }
287 
288 /*!
289   \brief Set the slider's thumb length
290   \param thumbLength new length
291 */
292 void QwtSlider::setThumbLength(int thumbLength)
293 {
294     if ( thumbLength < 8 )
295         thumbLength = 8;
296 
297     if ( thumbLength != d_data->thumbLength )
298     {
299         d_data->thumbLength = thumbLength;
300         layoutSlider();
301     }
302 }
303 
304 /*!
305   \brief Change the width of the thumb
306   \param w new width
307 */
308 void QwtSlider::setThumbWidth(int w)
309 {
310     if ( w < 4 )
311         w = 4;
312 
313     if ( d_data->thumbWidth != w )
314     {
315         d_data->thumbWidth = w;
316         layoutSlider();
317     }
318 }
319 
320 /*!
321   \brief Set a scale draw
322 
323   For changing the labels of the scales, it
```

```
324   is necessary to derive from QwtScaleDraw and
325   overload QwtScaleDraw::label().
326
327   \param scaleDraw ScaleDraw object, that has to be created with
328                    new and will be deleted in ~QwtSlider or the next
329                    call of setScaleDraw().
330 */
331 void QwtSlider::setScaleDraw(QwtScaleDraw *scaleDraw)
332 {
333     const QwtScaleDraw *previousScaleDraw = this->scaleDraw();
334     if ( scaleDraw == NULL || scaleDraw == previousScaleDraw )
335         return;
336
337     if ( previousScaleDraw )
338         scaleDraw->setAlignment(previousScaleDraw->alignment());
339
340     setAbstractScaleDraw(scaleDraw);
341     layoutSlider();
342 }
343
344 /*!
345   \return the scale draw of the slider
346   \sa setScaleDraw()
347 */
348 const QwtScaleDraw *QwtSlider::scaleDraw() const
349 {
350     return (QwtScaleDraw *)abstractScaleDraw();
351 }
352
353 /*!
354   \return the scale draw of the slider
355   \sa setScaleDraw()
356 */
357 QwtScaleDraw *QwtSlider::scaleDraw()
358 {
359     return (QwtScaleDraw *)abstractScaleDraw();
360 }
361
362 //! Notify changed scale
363 void QwtSlider::scaleChange()
364 {
365     layoutSlider();
366 }
367
368
369 //! Notify change in font
370 void QwtSlider::fontChange(const QFont &f)
371 {
372     QwtAbstractSlider::fontChange( f );
373     layoutSlider();
374 }
375
376 /*!
377    Draw the slider into the specified rectangle.
378
379    \param painter Painter
380    \param r Rectangle
381 */
382 void QwtSlider::drawSlider(QPainter *painter, const QRect &r)
383 {
384     QRect cr(r);
385
386     if (d_data->bgStyle & BgTrough)
387     {
388         qDrawShadePanel(painter, r.x(), r.y(),
389             r.width(), r.height(),
390 #if QT_VERSION < 0x040000
391             colorGroup(),
```

```
392 #else
393             palette(),
394 #endif
395             true, d_data->borderWidth,0);
396 
397         cr.setRect(r.x() + d_data->borderWidth,
398             r.y() + d_data->borderWidth,
399             r.width() - 2 * d_data->borderWidth,
400             r.height() - 2 * d_data->borderWidth);
401 
402         painter->fillRect(cr.x(), cr.y(), cr.width(), cr.height(),
403 #if QT_VERSION < 0x040000
404             colorGroup().brush(QColorGroup::Mid)
405 #else
406             palette().brush(QPalette::Mid)
407 #endif
408         );
409     }
410 
411     if ( d_data->bgStyle & BgSlot)
412     {
413         int ws = 4;
414         int ds = d_data->thumbLength / 2 - 4;
415         if ( ds < 1 )
416             ds = 1;
417 
418         QRect rSlot;
419         if (orientation() == Qt::Horizontal)
420         {
421             if ( cr.height() & 1 )
422                 ws++;
423             rSlot = QRect(cr.x() + ds,
424                     cr.y() + (cr.height() - ws) / 2,
425                     cr.width() - 2 * ds, ws);
426         }
427         else
428         {
429             if ( cr.width() & 1 )
430                 ws++;
431             rSlot = QRect(cr.x() + (cr.width() - ws) / 2,
432                     cr.y() + ds,
433                     ws, cr.height() - 2 * ds);
434         }
435         painter->fillRect(rSlot.x(), rSlot.y(), rSlot.width(), rSlot.height(),
436 #if QT_VERSION < 0x040000
437             colorGroup().brush(QColorGroup::Dark)
438 #else
439             palette().brush(QPalette::Dark)
440 #endif
441         );
442         qDrawShadePanel(painter, rSlot.x(), rSlot.y(),
443             rSlot.width(), rSlot.height(),
444 #if QT_VERSION < 0x040000
445             colorGroup(),
446 #else
447             palette(),
448 #endif
449             true, 1 ,0);
450 
451     }
452 
453     if ( isValid() )
454         drawThumb(painter, cr, xyPosition(value()));
455 }
456 
457 /*!
458   Draw the thumb at a position
459 
```

```
460   \param painter Painter
461   \param sliderRect Bounding rectangle of the slider
462   \param pos Position of the slider thumb
463 */
464 void QwtSlider::drawThumb(QPainter *painter, const QRect &sliderRect, int pos)
465 {
466     pos++; // shade line points one pixel below
467     if (orientation() == Qt::Horizontal)
468     {
469         qDrawShadePanel(painter, pos - d_data->thumbLength / 2,
470             sliderRect.y(), d_data->thumbLength, sliderRect.height(),
471 #if QT_VERSION < 0x040000
472             colorGroup(),
473 #else
474             palette(),
475 #endif
476             false, d_data->borderWidth,
477 #if QT_VERSION < 0x040000
478             &colorGroup().brush(QColorGroup::Button)
479 #else
480             &palette().brush(QPalette::Button)
481 #endif
482         );
483 
484         qDrawShadeLine(painter, pos, sliderRect.y(),
485             pos, sliderRect.y() + sliderRect.height() - 2,
486 #if QT_VERSION < 0x040000
487             colorGroup(),
488 #else
489             palette(),
490 #endif
491             true, 1);
492     }
493     else // Vertical
494     {
495         qDrawShadePanel(painter, sliderRect.x(), pos - d_data->thumbLength / 2,
496             sliderRect.width(), d_data->thumbLength,
497 #if QT_VERSION < 0x040000
498             colorGroup(),
499 #else
500             palette(),
501 #endif
502             false, d_data->borderWidth,
503 #if QT_VERSION < 0x040000
504             &colorGroup().brush(QColorGroup::Button)
505 #else
506             &palette().brush(QPalette::Button)
507 #endif
508         );
509 
510         qDrawShadeLine(painter, sliderRect.x(), pos,
511             sliderRect.x() + sliderRect.width() - 2, pos,
512 #if QT_VERSION < 0x040000
513             colorGroup(),
514 #else
515             palette(),
516 #endif
517             true, 1);
518     }
519 }
520 
521 /*!
522    Find the x/y position for a given value v
523    \param value Value
524 */
525 int QwtSlider::xyPosition(double value) const
526 {
527     return d_data->map.transform(value);
```

```
528 }
529
530 /*!
531    Determine the value corresponding to a specified mouse location.
532    \param pos Mouse position
533 */
534 double QwtSlider::getValue(const QPoint &pos)
535 {
536     return d_data->map.invTransform(
537         orientation() == Qt::Horizontal ? pos.x() : pos.y());
538 }
539
540 /*!
541   \brief Determine scrolling mode and direction
542   \param p point
543   \param scrollMode Scrolling mode
544   \param direction Direction
545 */
546 void QwtSlider::getScrollMode(const QPoint &p,
547     int &scrollMode, int &direction )
548 {
549     if (!d_data->sliderRect.contains(p))
550     {
551         scrollMode = ScrNone;
552         direction = 0;
553         return;
554     }
555
556     const int pos = ( orientation() == Qt::Horizontal ) ? p.x() : p.y();
557     const int markerPos = xyPosition(value());
558
559     if ((pos > markerPos - d_data->thumbLength / 2)
560         && (pos < markerPos + d_data->thumbLength / 2))
561     {
562         scrollMode = ScrMouse;
563         direction = 0;
564         return;
565     }
566
567     scrollMode = ScrPage;
568     direction = (pos > markerPos) ? 1 : -1;
569
570     if ( scaleDraw()->map().p1() > scaleDraw()->map().p2() )
571         direction = -direction;
572 }
573
574 /*!
575    Qt paint event
576    \param event Paint event
577 */
578 void QwtSlider::paintEvent(QPaintEvent *event)
579 {
580     const QRect &ur = event->rect();
581     if ( ur.isValid() )
582     {
583 #if QT_VERSION < 0x040000
584         QwtPaintBuffer paintBuffer(this, ur);
585         draw(paintBuffer.painter(), ur);
586 #else
587         QPainter painter(this);
588         draw(&painter, ur);
589 #endif
590     }
591 }
592
593 //! Draw the QwtSlider
594 void QwtSlider::draw(QPainter *painter, const QRect&)
595 {
```

```
596     if (d_data->scalePos != NoScale)
597     {
598 #if QT_VERSION < 0x040000
599         scaleDraw()->draw(painter, colorGroup());
600 #else
601         scaleDraw()->draw(painter, palette());
602 #endif
603     }
604 
605     drawSlider(painter, d_data->sliderRect);
606 
607     if ( hasFocus() )
608         QwtPainter::drawFocusRect(painter, this, d_data->sliderRect);
609 }
610 
611 //! Qt resize event
612 void QwtSlider::resizeEvent(QResizeEvent *)
613 {
614     layoutSlider( false );
615 }
616 
617 /*!
618   Recalculate the slider's geometry and layout based on
619   the current rect and fonts.
620   \param update_geometry  notify the layout system and call update
621          to redraw the scale
622 */
623 void QwtSlider::layoutSlider( bool update_geometry )
624 {
625     int sliderWidth = d_data->thumbWidth;
626     int sld1 = d_data->thumbLength / 2 - 1;
627     int sld2 = d_data->thumbLength / 2 + d_data->thumbLength % 2;
628     if ( d_data->bgStyle & BgTrough )
629     {
630         sliderWidth += 2 * d_data->borderWidth;
631         sld1 += d_data->borderWidth;
632         sld2 += d_data->borderWidth;
633     }
634 
635     int scd = 0;
636     if ( d_data->scalePos != NoScale )
637     {
638         int d1, d2;
639         scaleDraw()->getBorderDistHint(font(), d1, d2);
640         scd = qwtMax(d1, d2);
641     }
642 
643     int slo = scd - sld1;
644     if ( slo < 0 )
645         slo = 0;
646 
647     int x, y, length;
648 
649     const QRect r = rect();
650     if (orientation() == Qt::Horizontal)
651     {
652         switch (d_data->scalePos)
653         {
654             case TopScale:
655             {
656                 d_data->sliderRect.setRect(
657                     r.x() + d_data->xMargin + slo,
658                     r.y() + r.height() -
659                     d_data->yMargin - sliderWidth,
660                     r.width() - 2 * d_data->xMargin - 2 * slo,
661                     sliderWidth);
662 
663                 x = d_data->sliderRect.x() + sld1;
```

```
664                 y = d_data->sliderRect.y() - d_data->scaleDist;
665
666                 break;
667             }
668
669             case BottomScale:
670             {
671                 d_data->sliderRect.setRect(
672                     r.x() + d_data->xMargin + slo,
673                     r.y() + d_data->yMargin,
674                     r.width() - 2 * d_data->xMargin - 2 * slo,
675                     sliderWidth);
676
677                 x = d_data->sliderRect.x() + sld1;
678                 y = d_data->sliderRect.y() + d_data->sliderRect.height()
679                     + d_data->scaleDist;
680
681                 break;
682             }
683
684             case NoScale: // like Bottom, but no scale. See QwtSlider().
685             default:   // inconsistent orientation and scale position
686             {
687                 d_data->sliderRect.setRect(
688                     r.x() + d_data->xMargin + slo,
689                     r.y() + d_data->yMargin,
690                     r.width() - 2 * d_data->xMargin - 2 * slo,
691                     sliderWidth);
692
693                 x = d_data->sliderRect.x() + sld1;
694                 y = 0;
695
696                 break;
697             }
698         }
699         length = d_data->sliderRect.width() - (sld1 + sld2);
700     }
701     else // if (orientation() == Qt::Vertical
702     {
703         switch (d_data->scalePos)
704         {
705             case RightScale:
706                 d_data->sliderRect.setRect(
707                     r.x() + d_data->xMargin,
708                     r.y() + d_data->yMargin + slo,
709                     sliderWidth,
710                     r.height() - 2 * d_data->yMargin - 2 * slo);
711
712                 x = d_data->sliderRect.x() + d_data->sliderRect.width()
713                     + d_data->scaleDist;
714                 y = d_data->sliderRect.y() + sld1;
715
716                 break;
717
718             case LeftScale:
719                 d_data->sliderRect.setRect(
720                     r.x() + r.width() - sliderWidth - d_data->xMargin,
721                     r.y() + d_data->yMargin + slo,
722                     sliderWidth,
723                     r.height() - 2 * d_data->yMargin - 2 * slo);
724
725                 x = d_data->sliderRect.x() - d_data->scaleDist;
726                 y = d_data->sliderRect.y() + sld1;
727
728                 break;
729
730             case NoScale: // like Left, but no scale. See QwtSlider().
731             default:   // inconsistent orientation and scale position
```

```
732                 d_data->sliderRect.setRect(
733                     r.x() + r.width() - sliderWidth - d_data->xMargin,
734                     r.y() + d_data->yMargin + slo,
735                     sliderWidth,
736                     r.height() - 2 * d_data->yMargin - 2 * slo);
737 
738                 x = 0;
739                 y = d_data->sliderRect.y() + sld1;
740 
741                 break;
742             }
743             length = d_data->sliderRect.height() - (sld1 + sld2);
744         }
745 
746         scaleDraw()->move(x, y);
747         scaleDraw()->setLength(length);
748 
749         d_data->map.setPaintXInterval(scaleDraw()->map().p1(),
750             scaleDraw()->map().p2());
751 
752         if ( update_geometry )
753         {
754             d_data->sizeHintCache = QSize(); // invalidate
755             updateGeometry();
756             update();
757         }
758 }
759 
760 //! Notify change of value
761 void QwtSlider::valueChange()
762 {
763     QwtAbstractSlider::valueChange();
764     update();
765 }
766 
767 
768 //! Notify change of range
769 void QwtSlider::rangeChange()
770 {
771     d_data->map.setScaleInterval(minValue(), maxValue());
772 
773     if (autoScale())
774         rescale(minValue(), maxValue());
775 
776     QwtAbstractSlider::rangeChange();
777     layoutSlider();
778 }
779 
780 /*!
781   \brief Set distances between the widget's border and internals.
782   \param xMargin Horizontal margin
783   \param yMargin Vertical margin
784 */
785 void QwtSlider::setMargins(int xMargin, int yMargin)
786 {
787     if ( xMargin < 0 )
788         xMargin = 0;
789     if ( yMargin < 0 )
790         yMargin = 0;
791 
792     if ( xMargin != d_data->xMargin || yMargin != d_data->yMargin )
793     {
794         d_data->xMargin = xMargin;
795         d_data->yMargin = yMargin;
796         layoutSlider();
797     }
798 }
799 
```

```
800 /*!
801   Set the background style.
802 */
803 void QwtSlider::setBgStyle(BGSTYLE st)
804 {
805     d_data->bgStyle = st;
806     layoutSlider();
807 }
808
809 /*!
810   \return the background style.
811 */
812 QwtSlider::BGSTYLE QwtSlider::bgStyle() const
813 {
814     return d_data->bgStyle;
815 }
816
817 /*!
818   \return the thumb length.
819 */
820 int QwtSlider::thumbLength() const
821 {
822     return d_data->thumbLength;
823 }
824
825 /*!
826   \return the thumb width.
827 */
828 int QwtSlider::thumbWidth() const
829 {
830     return d_data->thumbWidth;
831 }
832
833 /*!
834   \return the border width.
835 */
836 int QwtSlider::borderWidth() const
837 {
838     return d_data->borderWidth;
839 }
840
841 /*!
842   \return QwtSlider::minimumSizeHint()
843 */
844 QSize QwtSlider::sizeHint() const
845 {
846     return minimumSizeHint();
847 }
848
849 /*!
850   \brief Return a minimum size hint
851   \warning The return value of QwtSlider::minimumSizeHint() depends on
852            the font and the scale.
853 */
854 QSize QwtSlider::minimumSizeHint() const
855 {
856     if (!d_data->sizeHintCache.isEmpty())
857         return d_data->sizeHintCache;
858
859     int sliderWidth = d_data->thumbWidth;
860     if (d_data->bgStyle & BgTrough)
861         sliderWidth += 2 * d_data->borderWidth;
862
863     int w = 0, h = 0;
864     if (d_data->scalePos != NoScale)
865     {
866         int d1, d2;
867         scaleDraw()->getBorderDistHint(font(), d1, d2);
```

```
868         int msMbd = qwtMax(d1, d2);
869
870         int mbd = d_data->thumbLength / 2;
871         if (d_data->bgStyle & BgTrough)
872             mbd += d_data->borderWidth;
873
874         if ( mbd < msMbd )
875             mbd = msMbd;
876
877         const int sdExtent = scaleDraw()->extent( QPen(), font() );
878         const int sdLength = scaleDraw()->minLength( QPen(), font() );
879
880         h = sliderWidth + sdExtent + d_data->scaleDist;
881         w = sdLength - 2 * msMbd + 2 * mbd;
882     }
883     else  // no scale
884     {
885         w = 200;
886         h = sliderWidth;
887     }
888
889     if ( orientation() == Qt::Vertical )
890         qSwap(w, h);
891
892     w += 2 * d_data->xMargin;
893     h += 2 * d_data->yMargin;
894
895     d_data->sizeHintCache = QSize(w, h);
896     return d_data->sizeHintCache;
897 }
```