QTextEdit Class Reference

[QtGui module]

The QTextEdit class provides a widget that is used to edit and display both plain and rich text. More...

Inherits QAbstractScrollArea.

Inherited by QTextBrowser.

Types

- class AutoFormatting
- enum AutoFormattingFlag { AutoNone, AutoBulletList, AutoAll }
- class ExtraSelection
- enum LineWrapMode { NoWrap, WidgetWidth, FixedPixelWidth, FixedColumnWidth }

Methods

- __init__ (*self*, QWidget *parent* = None)
- __init__ (*self*, QString *text*, QWidget *parent* = None)
- bool acceptRichText (*self*)
- Qt.Alignment alignment (*self*)
- QString anchorAt (*self*, QPoint *pos*)
- append (*self*, QString *text*)
- AutoFormatting autoFormatting (*self*)
- bool canInsertFromMimeData (*self*, QMimeData *source*)
- bool canPaste (*self*)
- changeEvent (*self*, QEvent *e*)
- clear (*self*)
- contextMenuEvent (*self*, QContextMenuEvent *e*)
- copy (*self*)
- QMimeData createMimeDataFromSelection (*self*)
- QMenu createStandardContextMenu (*self*)
- QMenu createStandardContextMenu (*self*, QPoint *position*)
- QTextCharFormat currentCharFormat (*self*)
- QFont currentFont (*self*)
- QTextCursor cursorForPosition (*self*, QPoint *pos*)
- QRect cursorRect (*self*, QTextCursor *cursor*)
- QRect cursorRect (*self*)
- int cursorWidth (*self*)
- cut (*self*)
- QTextDocument document (*self*)
- QString documentTitle (*self*)
- dragEnterEvent (*self*, QDragEnterEvent *e*)
- dragLeaveEvent (*self*, QDragLeaveEvent *e*)
- dragMoveEvent (*self*, QDragMoveEvent *e*)
- dropEvent (*self*, QDropEvent *e*)
- ensureCursorVisible (*self*)
- bool event (*self*, QEvent *e*)
- list-of-QTextEdit.ExtraSelection extraSelections (*self*)

- bool find (*self*, QString *exp*, QTextDocument.FindFlags *options* = 0)
- focusInEvent (*self*, QFocusEvent *e*)
- bool focusNextPrevChild (*self*, bool *next*)
- focusOutEvent (*self*, QFocusEvent *e*)
- QString fontFamily (*self*)
- bool fontItalic (*self*)
- float fontPointSize (*self*)
- bool fontUnderline (*self*)
- int fontWeight (*self*)
- inputMethodEvent (*self*, QInputMethodEvent)
- QVariant inputMethodQuery (*self*, Qt.InputMethodQuery *property*)
- insertFromMimeData (*self*, QMimeData *source*)
- insertHtml (*self*, QString *text*)
- insertPlainText (*self*, QString *text*)
- bool isReadOnly (*self*)
- bool isUndoRedoEnabled (*self*)
- keyPressEvent (*self*, QKeyEvent *e*)
- keyReleaseEvent (*self*, QKeyEvent *e*)
- int lineWrapColumnOrWidth (*self*)
- LineWrapMode lineWrapMode (*self*)
- QVariant loadResource (*self*, int *type*, QUrl *name*)
- mergeCurrentCharFormat (*self*, QTextCharFormat *modifier*)
- mouseDoubleClickEvent (*self*, QMouseEvent *e*)
- mouseMoveEvent (*self*, QMouseEvent *e*)
- mousePressEvent (*self*, QMouseEvent *e*)
- mouseReleaseEvent (*self*, QMouseEvent *e*)
- moveCursor (*self*, QTextCursor.MoveOperation *operation*, QTextCursor.MoveMode *mode* = QTextCursor.MoveAnchor)
- bool overwriteMode (*self*)
- paintEvent (*self*, QPaintEvent *e*)
- paste (*self*)
- print (*self*, QPrinter *printer*)
- print_ (*self*, QPrinter *printer*)
- redo (*self*)
- resizeEvent (*self*, QResizeEvent)
- scrollContentsBy (*self*, int *dx*, int *dy*)
- scrollToAnchor (*self*, QString *name*)
- selectAll (*self*)
- setAcceptRichText (*self*, bool *accept*)
- setAlignment (*self*, Qt.Alignment *a*)
- setAutoFormatting (*self*, AutoFormatting *features*)
- setCurrentCharFormat (*self*, QTextCharFormat *format*)
- setCurrentFont (*self*, QFont *f*)
- setCursorWidth (*self*, int *width*)
- setDocument (*self*, QTextDocument *document*)
- setDocumentTitle (*self*, QString *title*)
- setExtraSelections (*self*, list-of-QTextEdit.ExtraSelection *selections*)
- setFontFamily (*self*, QString *fontFamily*)
- setFontItalic (*self*, bool *b*)
- setFontPointSize (*self*, float *s*)
- setFontUnderline (*self*, bool *b*)
- setFontWeight (*self*, int *w*)
- setHtml (*self*, QString *text*)
- setLineWrapColumnOrWidth (*self*, int *w*)
- setLineWrapMode (*self*, LineWrapMode *mode*)
- setOverwriteMode (*self*, bool *overwrite*)
- setPlainText (*self*, QString *text*)
- setReadOnly (*self*, bool *ro*)

- setTabChangesFocus (*self*, bool *b*)
- setTabStopWidth (*self*, int *width*)
- setText (*self*, QString *text*)
- setTextBackgroundColor (*self*, QColor *c*)
- setTextColor (*self*, QColor *c*)
- setTextCursor (*self*, QTextCursor *cursor*)
- setTextInteractionFlags (*self*, Qt.TextInteractionFlags *flags*)
- setUndoRedoEnabled (*self*, bool *enable*)
- setWordWrapMode (*self*, QTextOption.WrapMode *policy*)
- showEvent (*self*, QShowEvent)
- bool tabChangesFocus (*self*)
- int tabStopWidth (*self*)
- QColor textBackgroundColor (*self*)
- QColor textColor (*self*)
- QTextCursor textCursor (*self*)
- Qt.TextInteractionFlags textInteractionFlags (*self*)
- timerEvent (*self*, QTimerEvent *e*)
- QString toHtml (*self*)
- QString toPlainText (*self*)
- undo (*self*)
- wheelEvent (*self*, QWheelEvent *e*)
- QTextOption.WrapMode wordWrapMode (*self*)
- zoomIn (*self*, int *range* = 1)
- zoomOut (*self*, int *range* = 1)

Qt Signals

- void copyAvailable (bool)
- void currentCharFormatChanged (const QTextCharFormat&)
- void cursorPositionChanged ()
- void redoAvailable (bool)
- void selectionChanged ()
- void textChanged ()
- void undoAvailable (bool)

Detailed Description

The QTextEdit class provides a widget that is used to edit and display both plain and rich text.

Introduction and Concepts

QTextEdit is an advanced WYSIWYG viewer/editor supporting rich text formatting using HTML-style tags. It is optimized to handle large documents and to respond quickly to user input.

QTextEdit works on paragraphs and characters. A paragraph is a formatted string which is word-wrapped to fit into the width of the widget. By default when reading plain text, one newline signifies a paragraph. A document consists of zero or more paragraphs. The words in the paragraph are aligned in accordance with the paragraph's alignment. Paragraphs are separated by hard line breaks. Each character within a paragraph has its own attributes, for example, font and color.

QTextEdit can display images, lists and tables. If the text is too large to view within the text edit's viewport, scroll bars will appear. The text edit can load both plain text and HTML files (a subset of HTML 3.2 and 4).

If you just need to display a small piece of rich text use QLabel.

The rich text support in Qt is designed to provide a fast, portable and efficient way to add reasonable online help facilities to applications, and to provide a basis for rich text editors. If you find the HTML support insufficient for your needs you may consider the use of QtWebKit, which provides a full-featured web browser widget.

The shape of the mouse cursor on a QTextEdit is Qt.IBeamCursor by default. It can be changed through the viewport()'s cursor property.

Using QTextEdit as a Display Widget

QTextEdit can display a large HTML subset, including tables and images.

The text is set or replaced using setHtml() which deletes any existing text and replaces it with the text passed in the setHtml() call. If you call setHtml() with legacy HTML, and then call toHtml(), the text that is returned may have different markup, but will render the same. The entire text can be deleted with clear().

Text itself can be inserted using the QTextCursor class or using the convenience functions insertHtml(), insertPlainText(), append() or paste(). QTextCursor is also able to insert complex objects like tables or lists into the document, and it deals with creating selections and applying changes to selected text.

By default the text edit wraps words at whitespace to fit within the text edit widget. The setLineWrapMode() function is used to specify the kind of line wrap you want, or NoWrap if you don't want any wrapping. Call setLineWrapMode() to set a fixed pixel width FixedPixelWidth, or character column (e.g. 80 column) FixedColumnWidth with the pixels or columns specified with setLineWrapColumnOrWidth(). If you use word wrap to the widget's width WidgetWidth, you can specify whether to break on whitespace or anywhere with setWordWrapMode().

The find() function can be used to find and select a given string within the text.

If you want to limit the total number of paragraphs in a QTextEdit, as for example it is often useful in a log viewer, then you can use QTextDocument's maximumBlockCount property for that.

Read-only Key Bindings

When QTextEdit is used read-only the key bindings are limited to navigation, and text may only be selected with the mouse:

Keypresses	Action
Up	Moves one line up.
Down	Moves one line down.
Left	Moves one character to the left.
Right	Moves one character to the right.
PageUp	Moves one (viewport) page up.
PageDown	Moves one (viewport) page down.
Home	Moves to the beginning of the text.
End	Moves to the end of the text.
Alt+Wheel	Scrolls the page horizontally (the Wheel is the mouse wheel).
Ctrl+Wheel	Zooms the text.
Ctrl+A	Selects all text.

The text edit may be able to provide some meta-information. For example, the documentTitle() function will return the text from within HTML <title> tags.

Using QTextEdit as an Editor

All the information about using QTextEdit as a display widget also applies here.

The current char format's attributes are set with setFontItalic(), setFontWeight(), setFontUnderline(), setFontFamily(), setFontPointSize(), setTextColor() and setCurrentFont(). The current paragraph's alignment is set with setAlignment().

Selection of text is handled by the QTextCursor class, which provides functionality for creating selections, retrieving the text contents or deleting selections. You can retrieve the object that corresponds with the user-visible cursor using the textCursor() method. If you want to set a selection in QTextEdit just create one on a QTextCursor object and then make that cursor the visible cursor using setTextCursor(). The selection can be copied to the clipboard with copy(), or cut to the clipboard with cut(). The entire text can be selected using selectAll().

When the cursor is moved and the underlying formatting attributes change, the currentCharFormatChanged() signal is emitted to reflect the new attributes at the new cursor position.

QTextEdit holds a QTextDocument object which can be retrieved using the document() method. You can also set your own document object using setDocument(). QTextDocument emits a textChanged() signal if the text changes and it also provides a isModified() function which will return true if the text has been modified since it was either loaded or since the last call to setModified with false as argument. In addition it provides methods for undo and redo.

Drag and Drop

QTextEdit also supports custom drag and drop behavior. By default, QTextEdit will insert plain text, HTML and rich text when the user drops data of these MIME types onto a document. Reimplement canInsertFromMimeData() and insertFromMimeData() to add support for additional MIME types.

For example, to allow the user to drag and drop an image onto a QTextEdit, you could the implement these functions in the following way:

```
bool TextEdit.canInsertFromMimeData( const QMimeData *source ) const
{
    if (source->hasImage())
        return true;
    else
        return QTextEdit.canInsertFromMimeData(source);
}
```

We add support for image MIME types by returning true. For all other MIME types, we use the default implementation.

```
void TextEdit.insertFromMimeData( const QMimeData *source )
{
    if (source->hasImage())
    {
        QImage image = qvariant_cast<QImage>(source->imageData());
        QTextCursor cursor = this->textCursor();
        QTextDocument *document = this->document();
        document->addResource(QTextDocument.ImageResource, QUrl("image"), image);
        cursor.insertImage("image");
```

```
    }
}
```

We unpack the image from the QVariant held by the MIME source and insert it into the document as a resource.

Editing Key Bindings

The list of key bindings which are implemented for editing:

Keypresses	Action
Backspace	Deletes the character to the left of the cursor.
Delete	Deletes the character to the right of the cursor.
Ctrl+C	Copy the selected text to the clipboard.
Ctrl+Insert	Copy the selected text to the clipboard.
Ctrl+K	Deletes to the end of the line.
Ctrl+V	Pastes the clipboard text into text edit.
Shift+Insert	Pastes the clipboard text into text edit.
Ctrl+X	Deletes the selected text and copies it to the clipboard.
Shift+Delete	Deletes the selected text and copies it to the clipboard.
Ctrl+Z	Undoes the last operation.
Ctrl+Y	Redoes the last operation.
Left	Moves the cursor one character to the left.
Ctrl+Left	Moves the cursor one word to the left.
Right	Moves the cursor one character to the right.
Ctrl+Right	Moves the cursor one word to the right.
Up	Moves the cursor one line up.
Down	Moves the cursor one line down.
PageUp	Moves the cursor one page up.
PageDown	Moves the cursor one page down.
Home	Moves the cursor to the beginning of the line.
Ctrl+Home	Moves the cursor to the beginning of the text.
End	Moves the cursor to the end of the line.
Ctrl+End	Moves the cursor to the end of the text.
Alt+Wheel	Scrolls the page horizontally (the Wheel is the mouse wheel).

To select (mark) text hold down the Shift key whilst pressing one of the movement keystrokes, for example, *Shift+Right* will select the character to the right, and *Shift+Ctrl+Right* will select the word to the right, etc.

Type Documentation

QTextEdit.AutoFormattingFlag

Constant	Value	Description
QTextEdit.AutoNone	0	Don't do any automatic formatting.
QTextEdit.AutoBulletList	0x00000001	Automatically create bullet lists (e.g. when the user enters an asterisk ('*') in the left most column, or presses Enter in an existing list item.

QTextEdit.AutoAll	0xffffffff	Apply all automatic formatting. Currently only automatic bullet lists are supported.

The AutoFormatting type is a typedef for QFlags<AutoFormattingFlag>. It stores an OR combination of AutoFormattingFlag values.

QTextEdit.LineWrapMode

Constant	Value
QTextEdit.NoWrap	0
QTextEdit.WidgetWidth	1
QTextEdit.FixedPixelWidth	2
QTextEdit.FixedColumnWidth	3

Method Documentation

QTextEdit.__init__ (*self*, QWidget *parent* = None)

The *parent* argument, if not None, causes *self* to be owned by Qt instead of PyQt.

Constructs an empty QTextEdit with parent *parent*.

QTextEdit.__init__ (*self*, QString *text*, QWidget *parent* = None)

The *parent* argument, if not None, causes *self* to be owned by Qt instead of PyQt.

Constructs a QTextEdit with parent *parent*. The text edit will display the text *text*. The text is interpreted as html.

bool QTextEdit.acceptRichText (*self*)

Qt.Alignment QTextEdit.alignment (*self*)

Returns the alignment of the current paragraph.

See also setAlignment().

QString QTextEdit.anchorAt (*self*, QPoint *pos*)

Returns the reference of the anchor at position *pos*, or an empty string if no anchor exists at that point.

QTextEdit.append (*self*, QString *text*)

Appends a new paragraph with *text* to the end of the text edit.

Note: The new paragraph appended will have the same character format and block format as the current paragraph, determined by the position of the cursor.

See also currentCharFormat() and QTextCursor.blockFormat().

AutoFormatting QTextEdit.autoFormatting (*self*)

bool QTextEdit.canInsertFromMimeData (*self*, QMimeData *source*)

This function returns true if the contents of the MIME data object, specified by *source*, can be decoded and inserted into the document. It is called for example when during a drag operation the mouse enters this widget and it is necessary to determine whether it is possible to accept the drag and drop operation.

Reimplement this function to enable drag and drop support for additional MIME types.

bool QTextEdit.canPaste (*self*)

Returns whether text can be pasted from the clipboard into the textedit.

This function was introduced in Qt 4.2.

QTextEdit.changeEvent (*self*, QEvent *e*)

Reimplemented from QWidget.changeEvent().

QTextEdit.clear (*self*)

This method is also a Qt slot with the C++ signature `void clear()`.

Deletes all the text in the text edit.

Note that the undo/redo history is cleared by this function.

See also cut(), setPlainText(), and setHtml().

QTextEdit.contextMenuEvent (*self*, QContextMenuEvent *e*)

Reimplemented from QWidget.contextMenuEvent().

Shows the standard context menu created with createStandardContextMenu().

If you do not want the text edit to have a context menu, you can set its contextMenuPolicy to Qt.NoContextMenu. If you want to customize the context menu, reimplement this function. If you want to extend the standard context menu, reimplement this function, call createStandardContextMenu() and extend the menu returned.

Information about the event is passed in the *event* object.

```
void MyTextEdit.contextMenuEvent(QContextMenuEvent *event)
{
    QMenu *menu = createStandardContextMenu();
    menu->addAction(tr("My Menu Item"));
    //...
    menu->exec(event->globalPos());
    delete menu;
}
```

QTextEdit.copy (*self*)

This method is also a Qt slot with the C++ signature `void copy()`.

Copies any selected text to the clipboard.

See also copyAvailable().

QMimeData QTextEdit.createMimeDataFromSelection (*self*)

This function returns a new MIME data object to represent the contents of the text edit's current selection. It is called when the selection needs to be encapsulated into a new QMimeData object; for example, when a drag and drop operation is started, or when data is copyied to the clipboard.

If you reimplement this function, note that the ownership of the returned QMimeData object is passed to the caller. The selection can be retrieved by using the textCursor() function.

QMenu QTextEdit.createStandardContextMenu (*self*)

This function creates the standard context menu which is shown when the user clicks on the text edit with the right mouse button. It is called from the default contextMenuEvent() handler. The popup menu's ownership is transferred to the caller.

We recommend that you use the createStandardContextMenu(QPoint) version instead which will enable the actions that are sensitive to where the user clicked.

QMenu QTextEdit.createStandardContextMenu (*self*, QPoint *position*)

This function creates the standard context menu which is shown when the user clicks on the text edit with the right mouse button. It is called from the default contextMenuEvent() handler and it takes the *position* of where the mouse click was. This can enable actions that are sensitive to the position where the user clicked. The popup menu's ownership is transferred to the caller.

This function was introduced in Qt 4.4.

QTextCharFormat QTextEdit.currentCharFormat (*self*)

Returns the char format that is used when inserting new text.

See also setCurrentCharFormat().

QFont QTextEdit.currentFont (*self*)

Returns the font of the current format.

See also setCurrentFont(), setFontFamily(), and setFontPointSize().

QTextCursor QTextEdit.cursorForPosition (*self*, QPoint *pos*)

returns a QTextCursor at position *pos* (in viewport coordinates).

QRect QTextEdit.cursorRect (*self*, QTextCursor *cursor*)

returns a rectangle (in viewport coordinates) that includes the *cursor*.

QRect QTextEdit.cursorRect (*self*)

returns a rectangle (in viewport coordinates) that includes the cursor of the text edit.

int QTextEdit.cursorWidth (*self*)

QTextEdit.cut (*self*)

This method is also a Qt slot with the C++ signature void cut().

Copies the selected text to the clipboard and deletes it from the text edit.

If there is no selected text nothing happens.

See also copy() and paste().

QTextDocument QTextEdit.document (*self*)

Returns a pointer to the underlying document.

See also setDocument().

QString QTextEdit.documentTitle (*self*)

QTextEdit.dragEnterEvent (*self*, QDragEnterEvent *e*)

Reimplemented from QWidget.dragEnterEvent().

QTextEdit.dragLeaveEvent (*self*, QDragLeaveEvent *e*)

Reimplemented from QWidget.dragLeaveEvent().

QTextEdit.dragMoveEvent (*self*, QDragMoveEvent *e*)

Reimplemented from QWidget.dragMoveEvent().

QTextEdit.dropEvent (*self*, QDropEvent *e*)

Reimplemented from QWidget.dropEvent().

QTextEdit.ensureCursorVisible (*self*)

Ensures that the cursor is visible by scrolling the text edit if necessary.

bool QTextEdit.event (*self*, QEvent *e*)

list-of-QTextEdit.ExtraSelection QTextEdit.extraSelections (*self*)

Returns previously set extra selections.

This function was introduced in Qt 4.2.

See also setExtraSelections().

bool QTextEdit.find (*self*, QString *exp*,
QTextDocument.FindFlags *options* = 0)

Finds the next occurrence of the string, *exp*, using the given *options*. Returns true if *exp* was found and changes the cursor to select the match; otherwise returns false.

QTextEdit.focusInEvent (*self*, QFocusEvent *e*)

Reimplemented from QWidget.focusInEvent().

bool QTextEdit.focusNextPrevChild (*self*, bool *next*)

Reimplemented from QWidget.focusNextPrevChild().

QTextEdit.focusOutEvent (*self*, QFocusEvent *e*)

Reimplemented from QWidget.focusOutEvent().

QString QTextEdit.fontFamily (*self*)

Returns the font family of the current format.

See also setFontFamily(), setCurrentFont(), and setFontPointSize().

bool QTextEdit.fontItalic (*self*)

Returns true if the font of the current format is italic; otherwise returns false.

See also setFontItalic().

float QTextEdit.fontPointSize (*self*)

Returns the point size of the font of the current format.

See also setFontFamily(), setCurrentFont(), and setFontPointSize().

bool QTextEdit.fontUnderline (*self*)

Returns true if the font of the current format is underlined; otherwise returns false.

See also setFontUnderline().

int QTextEdit.fontWeight (*self*)

Returns the font weight of the current format.

See also setFontWeight(), setCurrentFont(), setFontPointSize(), and QFont.Weight.

QTextEdit.inputMethodEvent (*self*, QInputMethodEvent)

Reimplemented from QWidget.inputMethodEvent().

QVariant QTextEdit.inputMethodQuery (*self*, Qt.InputMethodQuery *property*)

Reimplemented from QWidget.inputMethodQuery().

QTextEdit.insertFromMimeData (*self*, QMimeData *source*)

This function inserts the contents of the MIME data object, specified by *source*, into the text edit at the current cursor position. It is called whenever text is inserted as the result of a clipboard paste operation, or when the text edit accepts data from a drag and drop operation.

Reimplement this function to enable drag and drop support for additional MIME types.

QTextEdit.insertHtml (*self*, QString *text*)

This method is also a Qt slot with the C++ signature `void insertHtml(const QString&)`.

Convenience slot that inserts *text* which is assumed to be of html formatting at the current cursor position.

It is equivalent to:

```
edit->textCursor().insertHtml(fragment);
```

Note: When using this function with a style sheet, the style sheet will only apply to the current block in the document. In order to apply a style sheet throughout a document, use QTextDocument.setDefaultStyleSheet() instead.

QTextEdit.insertPlainText (*self*, QString *text*)

This method is also a Qt slot with the C++ signature `void insertPlainText(const QString&)`.

Convenience slot that inserts *text* at the current cursor position.

It is equivalent to

```
edit->textCursor().insertText(text);
```

bool QTextEdit.isReadOnly (*self*)

bool QTextEdit.isUndoRedoEnabled (*self*)

QTextEdit.keyPressEvent (*self*, QKeyEvent *e*)

Reimplemented from QWidget.keyPressEvent().

QTextEdit.keyReleaseEvent (*self*, QKeyEvent *e*)

Reimplemented from QWidget.keyReleaseEvent().

int QTextEdit.lineWrapColumnOrWidth (*self*)

LineWrapMode QTextEdit.lineWrapMode (*self*)

QVariant QTextEdit.loadResource (*self*, int *type*, QUrl *name*)

Loads the resource specified by the given *type* and *name*.

This function is an extension of QTextDocument.loadResource().

See also QTextDocument.loadResource().

QTextEdit.mergeCurrentCharFormat (*self*, QTextCharFormat *modifier*)

Merges the properties specified in *modifier* into the current character format by calling QTextCursor.mergeCharFormat on the editor's cursor. If the editor has a selection then the properties of *modifier* are directly applied to the selection.

See also QTextCursor.mergeCharFormat().

QTextEdit.mouseDoubleClickEvent (*self*, QMouseEvent *e*)

Reimplemented from QWidget.mouseDoubleClickEvent().

QTextEdit.mouseMoveEvent (*self*, QMouseEvent *e*)

Reimplemented from QWidget.mouseMoveEvent().

QTextEdit.mousePressEvent (*self*, QMouseEvent *e*)

Reimplemented from QWidget.mousePressEvent().

QTextEdit.mouseReleaseEvent (*self*, QMouseEvent *e*)

Reimplemented from QWidget.mouseReleaseEvent().

QTextEdit.moveCursor (*self*, QTextCursor.MoveOperation *operation*, QTextCursor.MoveMode *mode* = QTextCursor.MoveAnchor)

Moves the cursor by performing the given *operation*.

If *mode* is QTextCursor.KeepAnchor, the cursor selects the text it moves over. This is the same effect that the user achieves when they hold down the Shift key and move the cursor with the cursor keys.

This function was introduced in Qt 4.2.

See also QTextCursor.movePosition().

bool QTextEdit.overwriteMode (*self*)

QTextEdit.paintEvent (*self*, QPaintEvent *e*)

Reimplemented from QWidget.paintEvent().

This event handler can be reimplemented in a subclass to receive paint events passed in *event*. It is usually unnecessary to reimplement this function in a subclass of QTextEdit.

Warning: The underlying text document must not be modified from within a reimplementation of this function.

QTextEdit.paste (*self*)

This method is also a Qt slot with the C++ signature `void paste()`.

Pastes the text from the clipboard into the text edit at the current cursor position.

If there is no text in the clipboard nothing happens.

To change the behavior of this function, i.e. to modify what QTextEdit can paste and how it is being pasted, reimplement the virtual canInsertFromMimeData() and insertFromMimeData() functions.

See also cut() and copy().

QTextEdit.print (*self*, QPrinter *printer*)

Convenience function to print the text edit's document to the given *printer*. This is equivalent to calling the print method on the document directly except that this function also supports QPrinter.Selection as print range.

This function was introduced in Qt 4.3.

See also QTextDocument.print().

QTextEdit.print_ (*self*, QPrinter *printer*)

Convenience function to print the text edit's document to the given *printer*. This is equivalent to calling the print method on the document directly except that this function also supports QPrinter.Selection as print range.

This function was introduced in Qt 4.3.

See also QTextDocument.print().

QTextEdit.redo (*self*)

This method is also a Qt slot with the C++ signature `void redo()`.

Redoes the last operation.

If there is no operation to redo, i.e. there is no redo step in the undo/redo history, nothing happens.

This function was introduced in Qt 4.2.

See also undo().

QTextEdit.resizeEvent (*self*, QResizeEvent)

Reimplemented from QWidget.resizeEvent().

QTextEdit.scrollContentsBy (*self*, int *dx*, int *dy*)

Reimplemented from QAbstractScrollArea.scrollContentsBy().

QTextEdit.scrollToAnchor (*self*, QString *name*)

This method is also a Qt slot with the C++ signature `void scrollToAnchor(const QString&)`.

Scrolls the text edit so that the anchor with the given *name* is visible; does nothing if the *name* is empty, or is already visible, or isn't found.

QTextEdit.selectAll (*self*)

This method is also a Qt slot with the C++ signature `void selectAll()`.

Selects all text.

See also copy(), cut(), and textCursor().

QTextEdit.setAcceptRichText (*self*, bool *accept*)

QTextEdit.setAlignment (*self*, Qt.Alignment *a*)

This method is also a Qt slot with the C++ signature `void setAlignment(Qt::Alignment)`.

Sets the alignment of the current paragraph to *a*. Valid alignments are Qt.AlignLeft, Qt.AlignRight, Qt.AlignJustify and Qt.AlignCenter (which centers horizontally).

See also alignment().

QTextEdit.setAutoFormatting (*self*, AutoFormatting *features*)

QTextEdit.setCurrentCharFormat (*self*, QTextCharFormat *format*)

Sets the char format that is be used when inserting new text to *format* by calling QTextCursor.setCharFormat() on the editor's cursor. If the editor has a selection then the char format is directly applied to the selection.

See also currentCharFormat().

QTextEdit.setCurrentFont (*self*, QFont *f*)

This method is also a Qt slot with the C++ signature `void setCurrentFont(const QFont&)`.

Sets the font of the current format to *f*.

See also currentFont(), setFontPointSize(), and setFontFamily().

QTextEdit.setCursorWidth (*self*, int *width*)

QTextEdit.setDocument (*self*, QTextDocument *document*)

Makes *document* the new document of the text editor.

Note: The editor *does not take ownership of the document* unless it is the document's parent object. The parent object of the provided document remains the owner of the object.

The editor does not delete the current document, even if it is a child of the editor.

See also document().

QTextEdit.setDocumentTitle (*self*, QString *title*)

QTextEdit.setExtraSelections (*self*, list-of-QTextEdit.ExtraSelection *selections*)

This function allows temporarily marking certain regions in the document with a given color, specified as *selections*. This can be useful for example in a programming editor to mark a whole line of text with a given background color to indicate the existence of a breakpoint.

This function was introduced in Qt 4.2.

See also QTextEdit.ExtraSelection and extraSelections().

QTextEdit.setFontFamily (*self*, QString *fontFamily*)

This method is also a Qt slot with the C++ signature `void setFontFamily(const QString&)`.

Sets the font family of the current format to *fontFamily*.

See also fontFamily() and setCurrentFont().

QTextEdit.setFontItalic (*self*, bool *b*)

This method is also a Qt slot with the C++ signature `void setFontItalic(bool)`.

If *italic* is true, sets the current format to italic; otherwise sets the current format to non-italic.

See also fontItalic().

QTextEdit.setFontPointSize (*self*, float *s*)

This method is also a Qt slot with the C++ signature `void setFontPointSize(qreal)`.

Sets the point size of the current format to *s*.

Note that if *s* is zero or negative, the behavior of this function is not defined.

See also fontPointSize(), setCurrentFont(), and setFontFamily().

QTextEdit.setFontUnderline (*self*, bool *b*)

This method is also a Qt slot with the C++ signature `void setFontUnderline(bool)`.

If *underline* is true, sets the current format to underline; otherwise sets the current format to non-underline.

See also fontUnderline().

QTextEdit.setFontWeight (*self*, int *w*)

This method is also a Qt slot with the C++ signature `void setFontWeight(int)`.

Sets the font weight of the current format to the given *weight*, where the value used is in the range defined by the QFont.Weight enum.

See also fontWeight(), setCurrentFont(), and setFontFamily().

QTextEdit.setHtml (*self*, QString *text*)

This method is also a Qt slot with the C++ signature `void setHtml(const QString&)`.

QTextEdit.setLineWrapColumnOrWidth (*self*, int *w*)

QTextEdit.setLineWrapMode (*self*, LineWrapMode *mode*)

QTextEdit.setOverwriteMode (*self*, bool *overwrite*)

QTextEdit.setPlainText (*self*, QString *text*)

This method is also a Qt slot with the C++ signature void setPlainText(const QString&).

QTextEdit.setReadOnly (*self*, bool *ro*)

QTextEdit.setTabChangesFocus (*self*, bool *b*)

QTextEdit.setTabStopWidth (*self*, int *width*)

QTextEdit.setText (*self*, QString *text*)

This method is also a Qt slot with the C++ signature void setText(const QString&).

Sets the text edit's *text*. The text can be plain text or HTML and the text edit will try to guess the right format.

Use setHtml() or setPlainText() directly to avoid text edit's guessing.

This function was introduced in Qt 4.2.

See also text(), toPlainText(), and toHtml().

QTextEdit.setTextBackgroundColor (*self*, QColor *c*)

This method is also a Qt slot with the C++ signature void setTextBackgroundColor(const QColor&).

Sets the text background color of the current format to *c*.

This function was introduced in Qt 4.4.

See also textBackgroundColor().

QTextEdit.setTextColor (*self*, QColor *c*)

This method is also a Qt slot with the C++ signature void setTextColor(const QColor&).

Sets the text color of the current format to *c*.

See also textColor().

QTextEdit.setTextCursor (*self*, QTextCursor *cursor*)

Sets the visible *cursor*.

See also textCursor().

QTextEdit.setTextInteractionFlags (*self*, Qt.TextInteractionFlags *flags*)

QTextEdit.setUndoRedoEnabled (*self*, bool *enable*)

QTextEdit.setWordWrapMode (*self*, QTextOption.WrapMode *policy*)

QTextEdit.showEvent (*self*, QShowEvent)

Reimplemented from QWidget.showEvent().

bool QTextEdit.tabChangesFocus (*self*)

int QTextEdit.tabStopWidth (*self*)

QColor QTextEdit.textBackgroundColor (*self*)

Returns the text background color of the current format.

This function was introduced in Qt 4.4.

See also setTextBackgroundColor().

QColor QTextEdit.textColor (*self*)

Returns the text color of the current format.

See also setTextColor().

QTextCursor QTextEdit.textCursor (*self*)

Returns a copy of the QTextCursor that represents the currently visible cursor. Note that changes on the returned cursor do not affect QTextEdit's cursor; use setTextCursor() to update the visible cursor.

See also setTextCursor().

Qt.TextInteractionFlags QTextEdit.textInteractionFlags (*self*)

QTextEdit.timerEvent (*self*, QTimerEvent *e*)

QString QTextEdit.toHtml (*self*)

QString QTextEdit.toPlainText (*self*)

QTextEdit.undo (*self*)

This method is also a Qt slot with the C++ signature void undo().

Undoes the last operation.

If there is no operation to undo, i.e. there is no undo step in the undo/redo history, nothing happens.

This function was introduced in Qt 4.2.

See also redo().

QTextEdit.wheelEvent (*self*, QWheelEvent *e*)

Reimplemented from QWidget.wheelEvent().

QTextOption.WrapMode QTextEdit.wordWrapMode (*self*)

QTextEdit.zoomIn (*self*, int *range* = 1)

This method is also a Qt slot with the C++ signature `void zoomIn(int = 1)`.

Zooms in on the text by making the base font size *range* points larger and recalculating all font sizes to be the new size. This does not change the size of any images.

See also zoomOut().

QTextEdit.zoomOut (*self*, int *range* = 1)

This method is also a Qt slot with the C++ signature `void zoomOut(int = 1)`.

This is an overloaded function.

Zooms out on the text by making the base font size *range* points smaller and recalculating all font sizes to be the new size. This does not change the size of any images.

See also zoomIn().

Qt Signal Documentation

void copyAvailable (bool)

This is the default overload of this signal.

This signal is emitted when text is selected or de-selected in the text edit.

When text is selected this signal will be emitted with *yes* set to true. If no text has been selected or if the selected text is de-selected this signal is emitted with *yes* set to false.

If *yes* is true then copy() can be used to copy the selection to the clipboard. If *yes* is false then copy() does nothing.

See also selectionChanged().

void currentCharFormatChanged (const QTextCharFormat&)

This is the default overload of this signal.

This signal is emitted if the current character format has changed, for example caused by a change of the cursor position.

The new format is *f*.

See also setCurrentCharFormat().

void cursorPositionChanged ()

This is the default overload of this signal.

This signal is emitted whenever the position of the cursor changed.

void redoAvailable (bool)

This is the default overload of this signal.

This signal is emitted whenever redo operations become available (*available* is true) or unavailable (*available* is false).

void selectionChanged ()

This is the default overload of this signal.

This signal is emitted whenever the selection changes.

See also copyAvailable().

void textChanged ()

This is the default overload of this signal.

This signal is emitted whenever the document's content changes; for example, when text is inserted or deleted, or when formatting is applied.

void undoAvailable (bool)

This is the default overload of this signal.

This signal is emitted whenever undo operations become available (*available* is true) or unavailable (*available* is false).
