Archived:Overview of standard QWidget library on Symbian

Archived: This article is **archived** because it is not considered relevant for third-party developers creating commercial solutions today. If you think this article is still relevant, let us know by adding the template {{ReviewForRemovalFromArchive|user=~~~~|*write your reason here*}}.

Qt Quick should be used for all UI development on mobile devices. The approach described in this article (based on QWidget) is deprecated.

Qt provides a rich set of standard widgets that can be used to create graphical user interfaces for applications. Qt's widgets are flexible and can easily be sub classed to suit specialized requirements.

Widgets are visual elements that are combined to create user interfaces. Buttons, menus, scroll bars, message boxes, and application windows are all examples of widgets. Qt's widgets are not arbitrarily divided between "controls" and "containers"; all widgets can be used both as controls and as containers. Custom widgets can easily be created by sub classing existing Qt widgets, or created from scratch if necessary.

Standard widgets are provided by the QWidget class and its subclasses, and custom widgets can be created by sub classing them and reimplementing virtual functions.

A widget may contain any number of child widgets. Child widgets are shown within the parent widget's area. A widget with no parent is a top-level widget (a "window"), and usually has its own entry in the desktop environment's task bar. Qt imposes no arbitrary limitations on widgets. Any widget can be a top-level widget; any widget can be a child of any other widget. The position of child widgets within the parent's area can be set automatically using layout managers, or manually if preferred. When a parent widget is disabled, hidden, or deleted, the same action is recursively applied to all its child widgets.

Labels, message boxes, tooltips, and other textual widgets are not confined to using a single color, font, and language. Qt's text-rendering widgets can display multi-language rich text using a subset of HTML and most widgets can be styled using a description language.

WIDGETS

Qt widgets used in various user interface components. The widgets were arranged using Qt Designer and rendered using the Plastique style, demonstrating Qt 4's standard look.

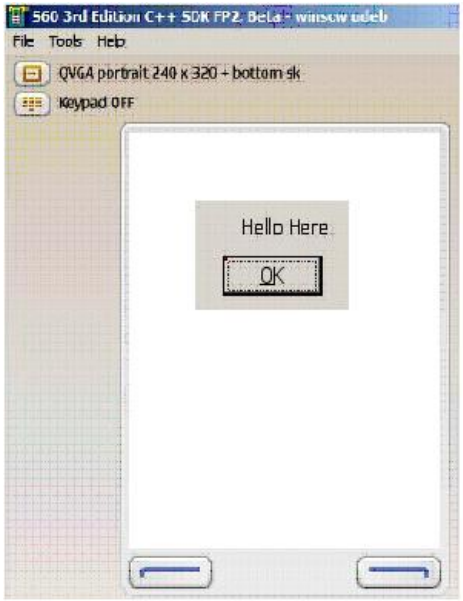
The widgets shown include standard input widgets like QLineEdit for one-line text entry, QCheckBox for enabling and disabling simple independent settings, QSpinBox and QSlider for specifying quantities, QRadioButton for enabling and disabling exclusive settings, and QComboBox, which opens to present a menu of choices when clicked. Clickable buttons are provided by QPushButton. Container widgets such as QTabWidget and QGroupBox are also shown. These widgets are managed specially in Qt Designer to allow designers to rapidly create new user interfaces while helping to keep them maintainable. More complex widgets such as QScrollArea, are often used more by developers than by user interface designers because they are often used to display specialized or dynamic content.

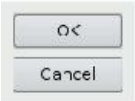
Why WIDGETS ?

Call them gadgets, mini-applications, or badges, but widgets are becoming more and more popular with both consumers and advertisers. They give users a way to interact with your brand and your content far beyond your web site. Keep reading to find out more about these useful bits of code, and why they may spawn a small revolution in the way we use the web.

The below graphical example shows us that how we can drag item and make QWidgets.

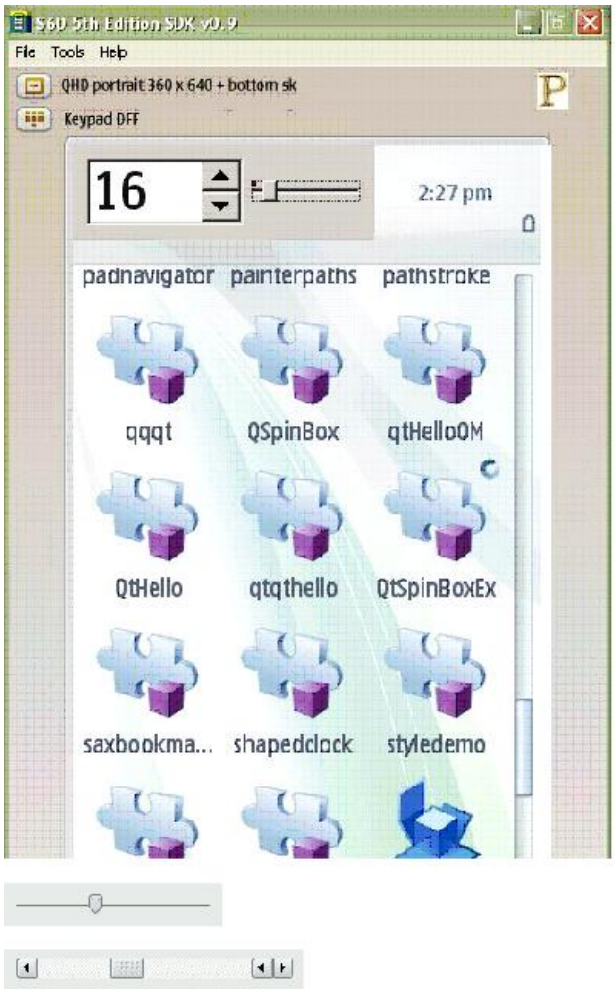
1.QPushButton Screen Shot





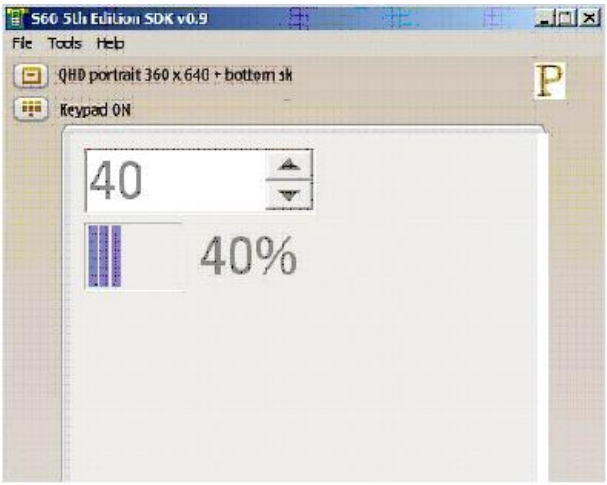
2. QSpinBox and QSlider

QSpinBox *spinBox = new QSpinBox; QSlider *slider = new QSlider(Qt::Horizontal);



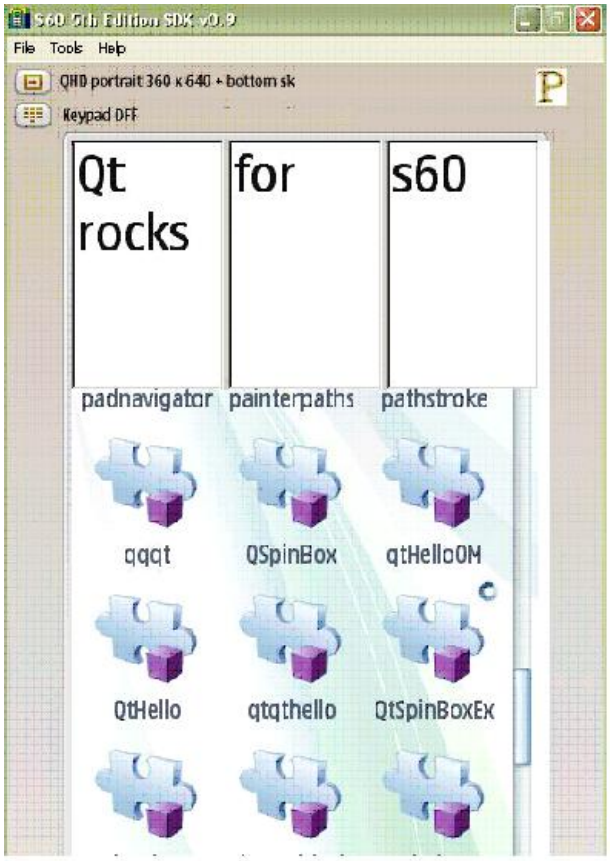
3. QProgressBar

Screen shot at 40% progress



4.

QSplitter and QTextEdit



5. QToolBar and QToolButton

6.QDial and QSpinBox