

B

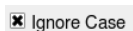
Selected PyQt Widgets

The screenshots shown here were taken on Linux using KDE to provide an eye-pleasing consistency. In the body of the book, screenshots are shown for Windows, Linux, and Mac OS X, generally varying from chapter to chapter.



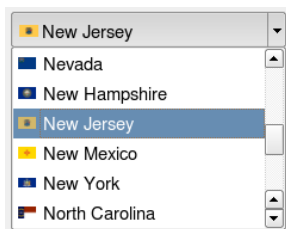
QCalendarWidget

This widget can be used as a display widget, although it was designed primarily as an input widget through which the user can choose a particular date. The widget's display is highly configurable; for example, week numbers can be displayed or not, day names can be represented by a single letter, or in short or full forms, the colors and fonts used can be set, and so can which day is treated as the first day of the week. Minimum and maximum allowable dates can also be set. Calling `setCalendarPopup(True)` on a `QDateEdit` or a `QDateTimeEdit` will cause their spin buttons to be replaced by an arrow button. If the user clicks the arrow button, a `QCalendarWidget` will pop up.



QCheckBox

A checkbox can be used to present users with a simple yes/no choice. If `QCheckBox.setTriState(True)` is called, the checkbox will have three states: The user checked it, the user unchecked it, or the user did not change it from whatever it was before. The tri-state approach may be useful for representing Boolean database fields where IS NULL is allowed.



QComboBox

The screenshot shows a QComboBox with its list popped down. A combobox is used to present the user with a list of items where too little vertical space is available for a QListView to be used. Calling `QComboBox.setEditable(True)` allows the user to either choose one of the items in the list, or to type in their own text instead. Each combobox item has text, an optional icon, and optional data. We can populate a combobox using `QComboBox.addItem()` or `QComboBox.addItems()`, or we can use a custom or built-in `QAbstractItemModel` subclass with `QComboBox.setModel()`.

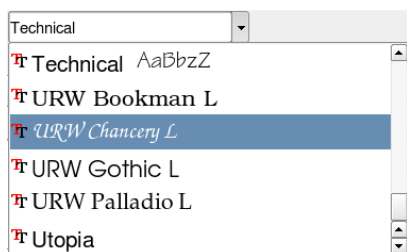
QDateEdit,
QDateTimeEdit, and
QTimeEdit

The QDateEdit is used for displaying and entering dates, the QDateTimeEdit is used for dates and times, and the QTimeEdit is used for times. By default, the widgets use locale-specific date and time formats—they are shown here using a U.S. locale. The formats can be changed, and minimum and maximum allowable dates and times can be set.



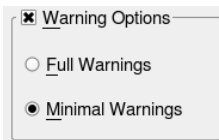
QDialogButtonBox

This widget can be used to create a row or column of buttons. The buttons can be standard buttons with predefined roles and text, or can be added with the roles and text of our choice. This widget automatically arranges the buttons according to their roles and the underlying windowing system's user interface guidelines.



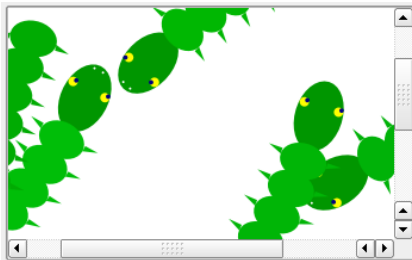
QFontComboBox

PyQt provides a pop-up font dialog, using the native font dialog where available. If we want to provide font choices ourselves—for example, in a toolbar—we can use the QFontComboBox, shown here popped down. For Qt 4.0 and Qt 4.1, the nearest equivalent (but without font previewing) is to use an ordinary QComboBox, populating it with the list returned by `QFontDatabase.families()`.



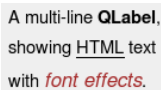
QGroupBox and
QRadioButton

A group box can be used purely as a visual grouping device, or it can be made checkable, as shown here. If checkable, the widgets contained in the group box can be interacted with only when the group box is checked. If a frame is required without a title, a QFrame can be used instead. When QRadioButtons are put in a group box they automatically behave correctly, that is, the user can choose only one of them. QComboBoxes and QListViews are often more convenient than QRadioButtons.



QGraphicsView

This widget is used to view the QGraphicsItems in a QGraphicsScene. Any number of QGraphicsViews can view the same scene, each with its own transformations (e.g., scaling and rotation), in effect. The scrollbars appear automatically if they are needed. Each QGraphicsView can provide its own background and foreground, overriding those provided by the scene.



QLabel

The QLabel widget is a display widget that can be used to show an image, a plain text string, a QTextDocument, or HTML. A label with an accelerator (a single underlined character) can be associated with a “buddy” widget, passing the keyboard focus to the buddy when the accelerator is pressed.



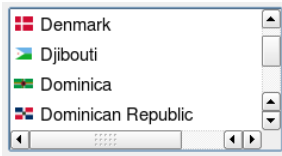
QLCDNumber

This is a display widget for showing numbers in the style of a seven-segment LCD.



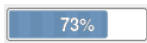
QLineEdit

This widget can accept one line of text from the user. The text can be constrained by using a validator (e.g., a QIntValidator or a QRegExpValidator), or by setting an input mask, or both. The echo mode can be set to show “*”s (or nothing) instead of the text entered.



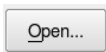
QListView and QListWidget

Through these widgets users can choose an item, or with a suitable selection mode, multiple items. The widgets can be in list mode (as shown), or icon mode, where the icons appear larger and the text is displayed under the icons. A QListView must be used in conjunction with a custom or built-in QAbstractItemModel subclass using QListView.setModel(). A QListWidget has a built-in model, so items can be added to it directly using QListWidget.addItem() and QListWidget.addItem(). Where vertical space is at a premium, a QComboBox can be used instead.



QProgressBar

This widget can be used to show users the progress of long-running operations. It is often put in a QMainWindow's status bar using QStatusBar.addWidget() or QStatusBar.addPermanentWidget(). An alternative is to pop up a QProgressDialog.



QPushButton

Buttons are used to invoke actions. If a button click will lead to a dialog being popped up, we normally add an ellipsis (...) to the end of the button's text. Buttons can also be set to have pop-up menus (in which case, PyQt will add a little triangle indicator), or they can be set as toggle buttons, staying down when clicked an odd number of times and coming back up when clicked an even number of times. Since Qt 4.2, most applications use QDialogButtonBoxes rather than individual QPushButtons.



QSlider

A slider is often used to show proportionality, and is commonly used in conjunction with a QLabel or QLCDNumber that shows an actual amount. Sliders can be aligned vertically or horizontally. A QScrollBar could be used for a similar purpose.

QDoubleSpinBox and
QSpinBox

These widgets are used to accept and display numbers. The number can be shown with a prefix or suffix and with a particular alignment. (The QDoubleSpinBox shown here has a "\$" prefix.) They can have minimum and maximum values set, and for the QDoubleSpinBox, the number of digits shown after the decimal point can be set. An alternative is to use a QLineEdit in conjunction with a QIntValidator or a QDoubleValidator.

	Category	Short Desc.	Long Desc.
1	Actions	Wait	Wait for Information
2	Actions	Progress	Progress to Next Stage
3	Actions	Escalate	Send to Supervisor

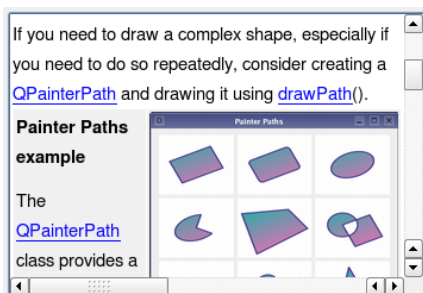
QTableView and QTableWidget

These widgets are used to present data in tabular form. A QTableView must be used in conjunction with a custom or built-in QAbstractItemModel subclass, such as QSqlTableModel, using QTableView.setModel(). A QTableWidget has a built-in model, so items can be added to it directly—for example, using QTableWidget.setItem(). Both widgets can show icons as well as text in every cell, including in the header cells.

Cash	Check	Credit Card
Number: <input type="text"/>		
Valid From:	Jan 2004	Expiry Date: Jan 2009

QTabWidget

This widget is used when space is at a premium, or simply as a means of logically grouping widgets. The tabs have two shape settings and can appear at the top, left, right, or bottom, with the text rotated when shown left or right.



QTextEdit and QTextBrowser

These widgets can display HTML, including lists, tables, and images. The QTextEdit can also be used as an editing widget, either for plain text or for PyQt “rich text” (essentially HTML, although a custom subclass would be needed to provide table and image editing). The QTextBrowser supports clickable links, so it can be used as a simple Web browser. Both widgets have support for CSS (Cascading Style Sheets).

Country/State (US)/City/Provider	Server	IP
Turkey		
(No State)		
Ankara		
National Academic Backbone		
UlakNet Ankara	Apache/1.3.4	193.140.83.20
Ukraine		
United Kingdom		
USA		
Arizona		
California		
Georgia		
Atlanta		
Courtesy of Lotus Productions	Apache/1.3.3	216.25.63.17
Lyceum Support Tools	NCSA/1.4.2	205.142.28.7
Southern CrossRoads	Apache/1.3.1	130.207.244.51

QTreeView and QTreeWidget

These widgets are used to present hierarchical data. A QTreeView must be used with a custom or built-in QAbstractItemModel subclass using QTreeView.setModel(). Like all widgets that use a model, only the data that is visible to the user is retrieved, so even large datasets are very fast. A QTreeWidget has a built-in model, so items can be added to it directly using QTreeWidget.insertTopLevelItem() and insertTopLevelItems(), or by creating QTreeWidgetItem as children of other items.

This page intentionally left blank

C

Selected PyQt Class Hierarchies

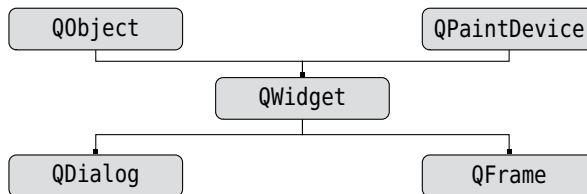


Figure C.1 *Selected base classes*

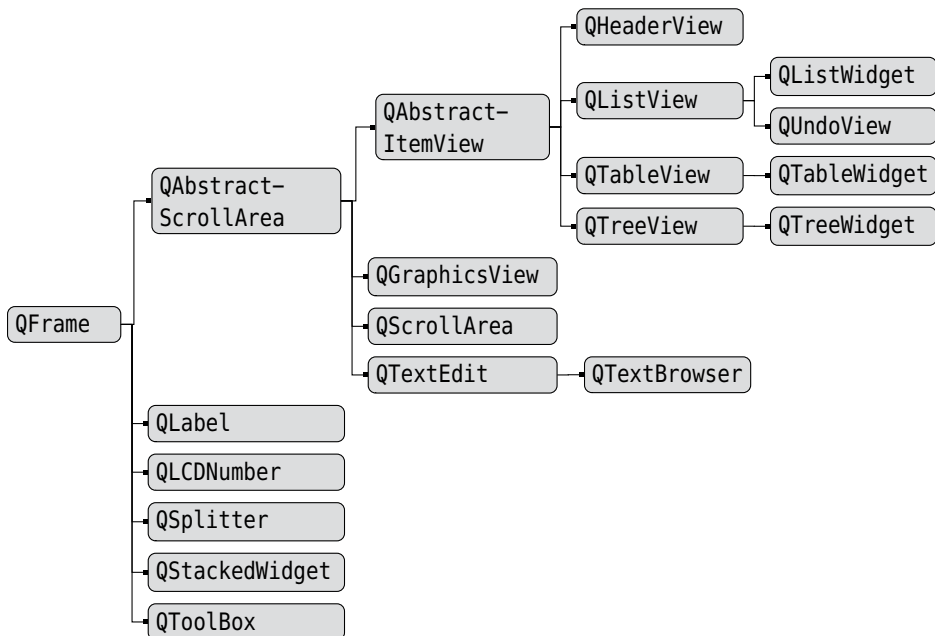
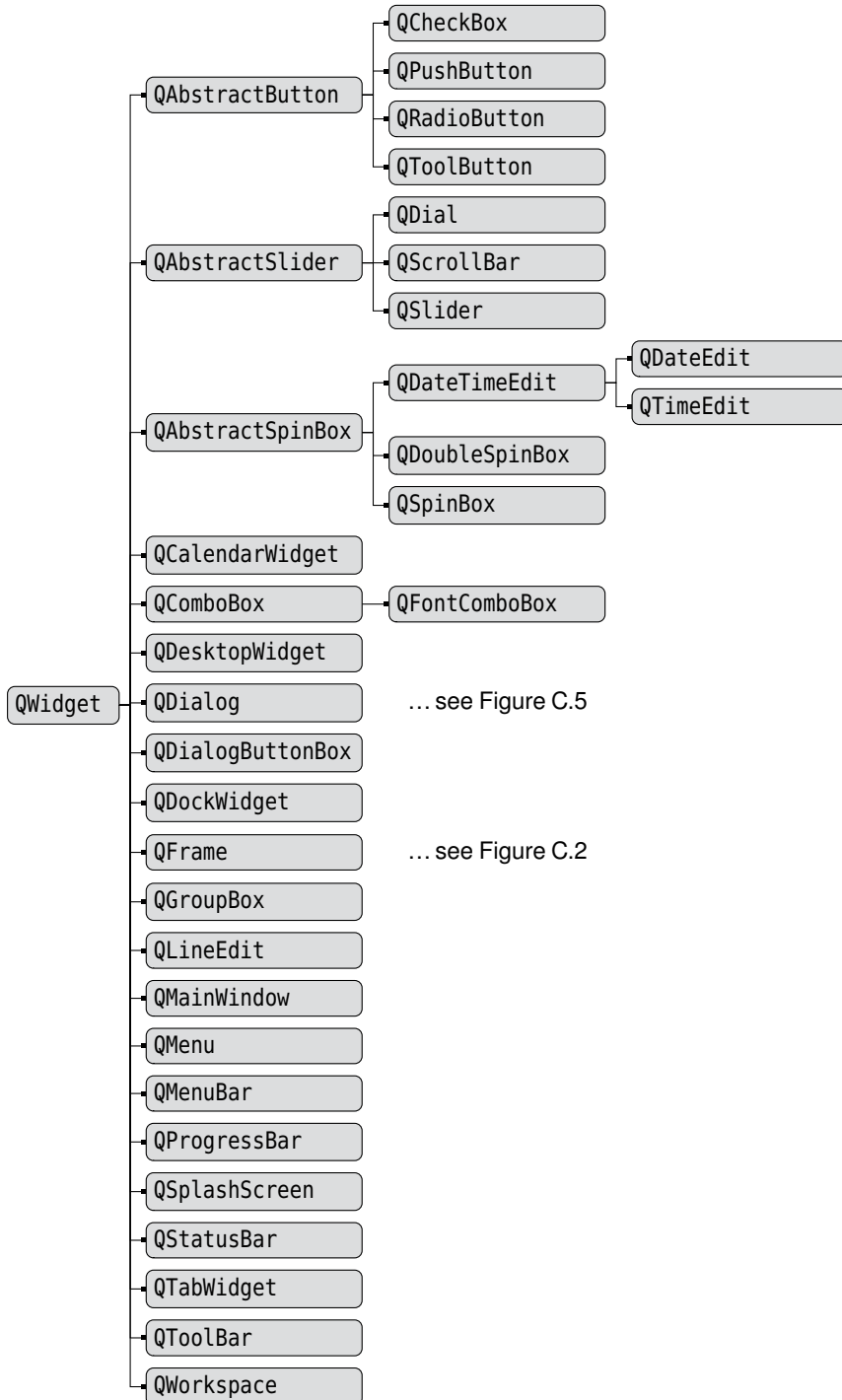


Figure C.2 *Selected classes from the QFrame hierarchy*

**Figure C.3** Selected classes from the *QWidget* hierarchy

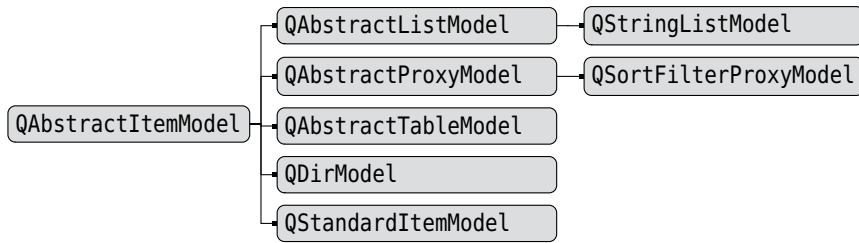


Figure C.4 Selected classes from the *QAbstractItemModel* hierarchy

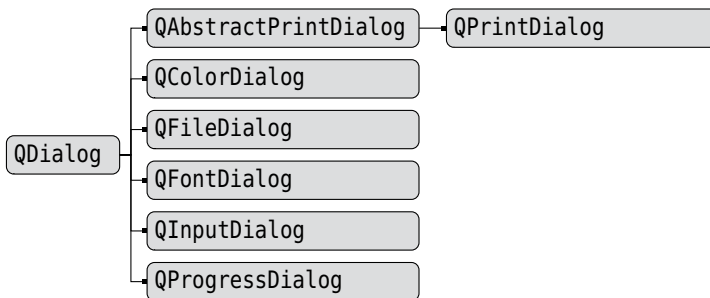


Figure C.5 Selected classes from the *QDialog* hierarchy

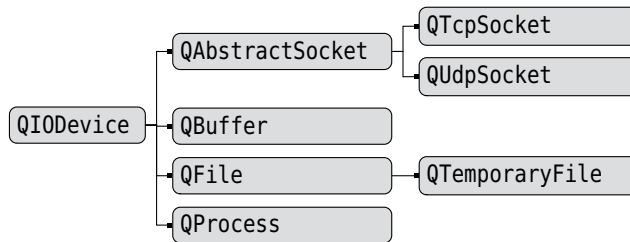


Figure C.6 Selected classes from the *QIODevice* hierarchy

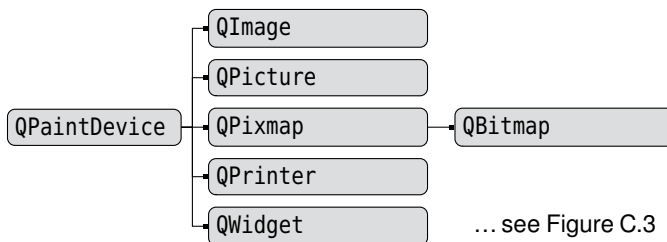


Figure C.7 Selected classes from the *QPaintDevice* hierarchy

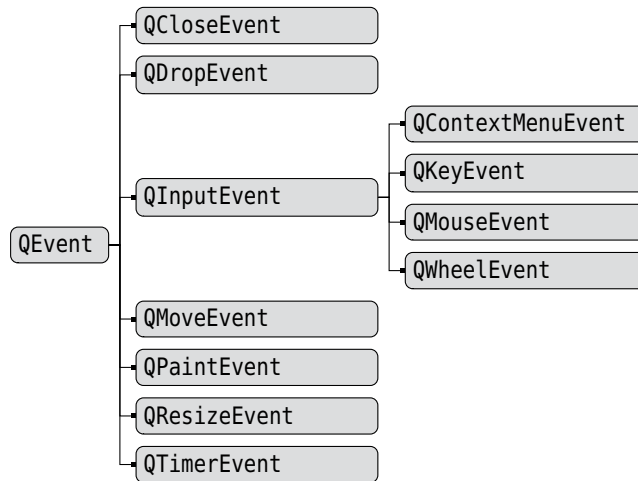


Figure C.8 Selected classes from the *QEvent* hierarchy

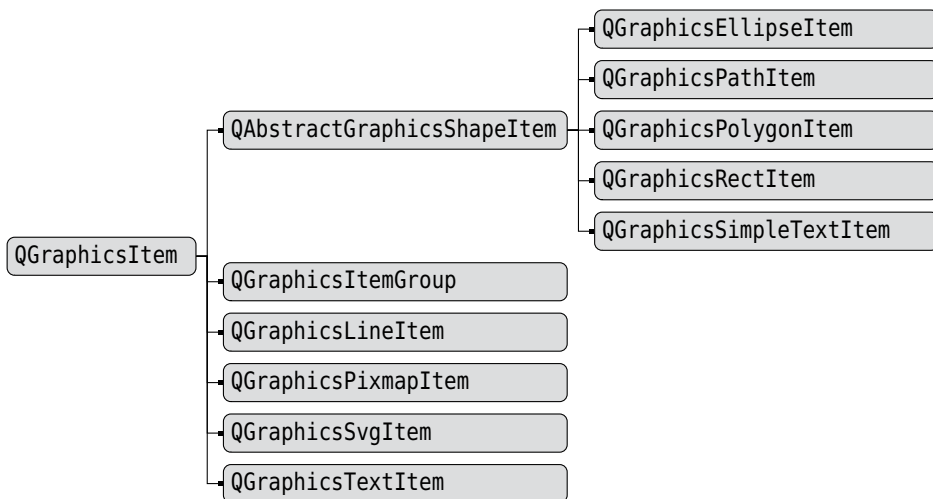


Figure C.9 Selected classes from the *QGraphicsItem* hierarchy

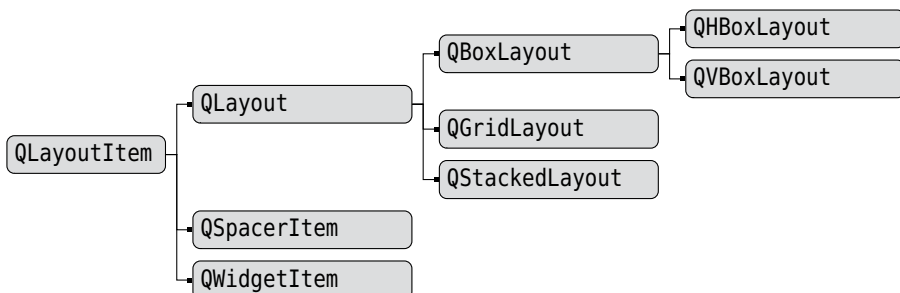


Figure C.10 Selected classes from the *QLayoutItem* hierarchy