Support for Signals and Slots

One of the key features of Qt is its use of signals and slots to communicate between objects. Their use encourages the development of reusable components.

A signal is emitted when something of potential interest happens. A slot is a Python callable. If a signal is connected to a slot then the slot is called when the signal is emitted. If a signal isn’t connected then nothing happens. The code (or component) that emits the signal does not know or care if the signal is being used.

The signal/slot mechanism has the following features.

* A signal may be connected to many slots.
* A signal may also be connected to another signal.
* Signal arguments may be any Python type.
* A slot may be connected to many signals.
* Connections may be direct (ie. synchronous) or queued (ie. [asynchronous](http://pyqt.sourceforge.net/Docs/PyQt5/signals_slots.html)).
* Connections may be made across threads.
* Signals may be disconnected.

Unbound and Bound Signals

A signal (specifically an unbound signal) is a class attribute. When a signal is referenced as an attribute of an instance of the class then PyQt5 automatically binds the instance to the signal in order to create a *bound signal*. This is the same mechanism that Python itself uses to create bound methods from class functions.

A bound signal has connect(), disconnect() and emit() methods that implement the associated functionality. It also has a signal attribute that is the signature of the signal that would be returned by Qt’s SIGNAL() macro.

A signal may be overloaded, ie. a signal with a particular name may support more than one signature. A signal may be indexed with a signature in order to select the one required. A signature is a sequence of types. A type is either a Python type object or a string that is the name of a C++ type. The name of a C++ type is automatically normalised so that, for example, QVariant can be used instead of the non-normalised const QVariant &.

If a signal is overloaded then it will have a default that will be used if no index is given.

When a signal is emitted then any arguments are converted to C++ types if possible. If an argument doesn’t have a corresponding C++ type then it is wrapped in a special C++ type that allows it to be passed around Qt’s meta-type system while ensuring that its reference count is properly maintained.

Defining New Signals with [**pyqtSignal()**](http://pyqt.sourceforge.net/Docs/PyQt5/signals_slots.html#PyQt5.QtCore.pyqtSignal)

PyQt5 automatically defines signals for all Qt’s built-in signals. New signals can be defined as class attributes using the [**pyqtSignal()**](http://pyqt.sourceforge.net/Docs/PyQt5/signals_slots.html#PyQt5.QtCore.pyqtSignal) factory.

PyQt5.QtCore.**pyqtSignal**(*types*[, *name*[, *revision=0*[, *arguments=[]*]]])

Create one or more overloaded unbound signals as a class attribute.

|  |  |
| --- | --- |
| **Parameters:** | * **types** – the types that define the C++ signature of the signal. Each type may be a Python type object or a string that is the name of a C++ type. Alternatively each may be a sequence of type arguments. In this case each sequence defines the signature of a different signal overload. The first overload will be the default. * **name** – the name of the signal. If it is omitted then the name of the class attribute is used. This may only be given as a keyword argument. * **revision** – the revision of the signal that is exported to QML. This may only be given as a keyword argument. * **arguments** – the sequence of the names of the signal’s arguments that is exported to QML. This may only be given as a keyword argument. |
| **Return type:** | an unbound signal |

The following example shows the [definition](http://pyqt.sourceforge.net/Docs/PyQt5/signals_slots.html) of a number of new signals:

**from** **PyQt5.QtCore** **import** QObject, pyqtSignal

**class** **Foo**(QObject):

*# This defines a signal called 'closed' that takes no arguments.*

closed = pyqtSignal()

*# This defines a signal called 'rangeChanged' that takes two*

*# integer arguments.*

range\_changed = pyqtSignal(int, int, name='rangeChanged')

*# This defines a signal called 'valueChanged' that has two overloads,*

*# one that takes an integer argument and one that takes a QString*

*# argument. Note that because we use a string to specify the type of*

*# the QString argument then this code will run under Python v2 and v3.*

valueChanged = pyqtSignal([int], ['QString'])

New signals should only be defined in sub-classes of [**QObject**](http://pyqt.sourceforge.net/Docs/PyQt5/api/QtCore/qobject.html#PyQt5-QtCore-QObject). They must be part of the class definition and cannot be dynamically added as class attributes after the class has been defined.

New signals defined in this way will be automatically added to the class’s [**QMetaObject**](http://pyqt.sourceforge.net/Docs/PyQt5/api/QtCore/qmetaobject.html#PyQt5-QtCore-QMetaObject). This means that they will appear in Qt Designer and can be introspected using the [**QMetaObject**](http://pyqt.sourceforge.net/Docs/PyQt5/api/QtCore/qmetaobject.html#PyQt5-QtCore-QMetaObject) API.

Overloaded signals should be used with care when an argument has a Python type that has no corresponding C++ type. PyQt5 uses the same internal C++ class to represent such objects and so it is possible to have overloaded signals with different Python signatures that are implemented with identical C++ signatures with unexpected results. The following is an example of this:

**class** **Foo**(QObject):

*# This will cause problems because each has the same C++ signature.*

valueChanged = pyqtSignal([dict], [list])

Connecting, Disconnecting and Emitting Signals

Signals are connected to slots using the [**connect()**](http://pyqt.sourceforge.net/Docs/PyQt5/signals_slots.html#connect) method of a bound signal.

**connect**(*slot*[, *type=PyQt5.QtCore.Qt.AutoConnection*[, *no\_receiver\_check=False*]])

Connect a signal to a slot. An exception will be raised if the connection failed.

|  |  |
| --- | --- |
| **Parameters:** | * **slot** – the slot to connect to, either a Python callable or another bound signal. * **type** – the type of the connection to make. * **no\_receiver\_check** – suppress the check that the underlying C++ receiver instance still exists and deliver the signal anyway. |

Signals are disconnected from slots using the [**disconnect()**](http://pyqt.sourceforge.net/Docs/PyQt5/signals_slots.html#disconnect) method of a bound signal.

**disconnect**([*slot*])

Disconnect one or more slots from a signal. An exception will be raised if the slot is not connected to the signal or if the signal has no connections at all.

|  |  |
| --- | --- |
| **Parameters:** | **slot** – the optional slot to disconnect from, either a Python callable or another bound signal. If it is omitted then all slots connected to the signal are disconnected. |

Signals are emitted from using the [**emit()**](http://pyqt.sourceforge.net/Docs/PyQt5/signals_slots.html#emit) method of a bound signal.

**emit**(*\*args*)

Emit a signal.

|  |  |
| --- | --- |
| **Parameters:** | [**args**](http://pyqt.sourceforge.net/Docs/PyQt5/signals_slots.html) – the optional sequence of arguments to pass to any connected slots. |

The following code demonstrates the definition, connection and emit of a signal without arguments:

**from** **PyQt5.QtCore** **import** QObject, pyqtSignal

**class** **Foo**(QObject):

*# Define a new signal called 'trigger' that has no arguments.*

trigger = pyqtSignal()

**def** connect\_and\_emit\_trigger(self):

*# Connect the trigger signal to a slot.*

self.trigger.connect(self.handle\_trigger)

*# Emit the signal.*

self.trigger.emit()

**def** handle\_trigger(self):

*# Show that the slot has been called.*

print "trigger signal received"

The following code demonstrates the connection of overloaded signals:

**from** **PyQt5.QtWidgets** **import** QComboBox

**class** **Bar**(QComboBox):

**def** connect\_activated(self):

*# The PyQt5 documentation will define what the default overload is.*

*# In this case it is the overload with the single integer argument.*

self.activated.connect(self.handle\_int)

*# For non-default overloads we have to specify which we want to*

*# connect. In this case the one with the single string argument.*

*# (Note that we could also explicitly specify the default if we*

*# wanted to.)*

self.activated[str].connect(self.handle\_string)

**def** handle\_int(self, index):

print "activated signal passed integer", index

**def** handle\_string(self, text):

print "activated signal passed QString", text

Connecting Signals Using Keyword Arguments

It is also possible to connect signals by [passing a](http://pyqt.sourceforge.net/Docs/PyQt5/signals_slots.html) slot as a keyword argument corresponding to the name of the signal when creating an object, or using the **pyqtConfigure()** method. For example the following three fragments are equivalent:

act = QAction("Action", self)

act.triggered.connect(self.on\_triggered)

act = QAction("Action", self, triggered=self.on\_triggered)

act = QAction("Action", self)

act.pyqtConfigure(triggered=self.on\_triggered)

The [**pyqtSlot()**](http://pyqt.sourceforge.net/Docs/PyQt5/signals_slots.html#PyQt5.QtCore.pyqtSlot) Decorator

Although PyQt5 allows any Python callable to be used as a slot when connecting signals, it is sometimes necessary to explicitly mark a Python method as being a Qt slot and to provide a C++ signature for it. PyQt5 provides the [**pyqtSlot()**](http://pyqt.sourceforge.net/Docs/PyQt5/signals_slots.html#PyQt5.QtCore.pyqtSlot) function decorator to do this.

PyQt5.QtCore.**pyqtSlot**(*types*[, *name*[, *result*[, *revision=0*]]])

Decorate a Python method to create a Qt slot.

|  |  |
| --- | --- |
| **Parameters:** | * **types** – the types that define the C++ signature of the slot. Each type may be a Python type object or a string that is the name of a C++ type. * **name** – the name of the slot that will be seen by C++. If omitted the name of the Python method being decorated will be used. This may only be given as a keyword argument. * **revision** – the revision of the slot that is exported to QML. This may only be given as a keyword argument. * **result** – the type of the result and may be a Python type object or a string that specifies a C++ type. This may only be given as a keyword argument. |

Connecting a signal to a decorated Python method also has the advantage of reducing the amount of memory used and is slightly faster.

For example:

**from** **PyQt5.QtCore** **import** QObject, pyqtSlot

**class** **Foo**(QObject):

**@pyqtSlot**()

**def** foo(self):

*""" C++: void foo() """*

**@pyqtSlot**(int, str)

**def** foo(self, arg1, arg2):

*""" C++: void foo(int, QString) """*

**@pyqtSlot**(int, name='bar')

**def** foo(self, arg1):

*""" C++: void bar(int) """*

**@pyqtSlot**(int, result=int)

**def** foo(self, arg1):

*""" C++: int foo(int) """*

**@pyqtSlot**(int, QObject)

**def** foo(self, arg1):

*""" C++: int foo(int, QObject \*) """*

It is also possible to chain the decorators in order to define a Python method several times with different signatures. For example:

**from** **PyQt5.QtCore** **import** QObject, pyqtSlot

**class** **Foo**(QObject):

**@pyqtSlot**(int)

**@pyqtSlot**('QString')

**def** valueChanged(self, value):

*""" Two slots will be defined in the QMetaObject. """*

The PyQt\_PyObject Signal Argument Type

It is possible to pass any Python object as a signal argument by specifying PyQt\_PyObject as the type of the argument in the signature. For example:

finished = pyqtSignal('PyQt\_PyObject')

This would normally be used for passing objects where the actual Python type isn’t known. It can also be used to pass an integer, for example, so that the normal conversions from a Python object to a C++ integer and back again are not required.

The reference count of the object being passed is maintained automatically. There is no need for the emitter of a signal to keep a reference to the object after the call to finished.emit(), even if a connection is queued.

Connecting Slots By Name

PyQt5 supports the **connectSlotsByName()** function that is most commonly used by **pyuic5** generated Python code to automatically connect signals to slots that conform to a simple naming convention. However, where a class has overloaded Qt signals (ie. with the same name but with different arguments) PyQt5 needs additional information in order to automatically connect the correct signal.

For example the [**QSpinBox**](http://pyqt.sourceforge.net/Docs/PyQt5/api/QtWidgets/qspinbox.html#PyQt5-QtWidgets-QSpinBox) class has the following signals:

void valueChanged(int i);

void valueChanged(const QString &text);

When the value of the spin box changes both of these signals will be emitted. If you have implemented a slot called on\_spinbox\_valueChanged (which assumes that you have given the **QSpinBox** instance the name spinbox) then it will be connected to both variations of the signal. Therefore, when the user changes the value, your slot will be called twice - once with an integer argument, and once with a string argument.

The [**pyqtSlot()**](http://pyqt.sourceforge.net/Docs/PyQt5/signals_slots.html#PyQt5.QtCore.pyqtSlot) decorator can be used to specify which of the signals should be connected to the slot.

For example, if you were only interested in the integer variant of the signal then your slot definition would look like the following:

**@pyqtSlot**(int)

**def** on\_spinbox\_valueChanged(self, i):

*# i will be an integer.*

**pass**

If you wanted to handle both variants of the signal, but with different Python methods, then your slot definitions might look like the following:

**@pyqtSlot**(int, name='on\_spinbox\_valueChanged')

**def** spinbox\_int\_value(self, i):

*# i will be an integer.*

**pass**

**@pyqtSlot**(str, name='on\_spinbox\_valueChanged')

**def** spinbox\_qstring\_value(self, s):

*# s will be a Python string object (or a QString if they are enabled).*

**pass**

[](http://pyqt.sourceforge.net/Docs/PyQt5/index.html)

[Table Of Contents](http://pyqt.sourceforge.net/Docs/PyQt5/index.html)

* [Support for Signals and Slots](http://pyqt.sourceforge.net/Docs/PyQt5/signals_slots.html)
  + [Unbound and Bound Signals](http://pyqt.sourceforge.net/Docs/PyQt5/signals_slots.html#unbound-and-bound-signals)
  + [Defining New Signals with **pyqtSignal()**](http://pyqt.sourceforge.net/Docs/PyQt5/signals_slots.html#defining-new-signals-with-pyqtsignal)
  + [Connecting, Disconnecting and Emitting Signals](http://pyqt.sourceforge.net/Docs/PyQt5/signals_slots.html#connecting-disconnecting-and-emitting-signals)
  + [Connecting Signals Using Keyword Arguments](http://pyqt.sourceforge.net/Docs/PyQt5/signals_slots.html#connecting-signals-using-keyword-arguments)
  + [The **pyqtSlot()** Decorator](http://pyqt.sourceforge.net/Docs/PyQt5/signals_slots.html#the-pyqtslot-decorator)
  + [The **PyQt\_PyObject** Signal Argument Type](http://pyqt.sourceforge.net/Docs/PyQt5/signals_slots.html#the-pyqt-pyobject-signal-argument-type)
  + [Connecting Slots By Name](http://pyqt.sourceforge.net/Docs/PyQt5/signals_slots.html#connecting-slots-by-name)

Previous topic

[Differences Between PyQt4 and PyQt5](http://pyqt.sourceforge.net/Docs/PyQt5/pyqt4_differences.html)

Next topic

[Support for Qt Properties](http://pyqt.sourceforge.net/Docs/PyQt5/qt_properties.html)

Quick search

Top of Form





Bottom of Form

* [Index](http://pyqt.sourceforge.net/Docs/PyQt5/genindex.html)
* [Globals](http://pyqt.sourceforge.net/Docs/PyQt5/sip-globals.html) |
* [Classes](http://pyqt.sourceforge.net/Docs/PyQt5/sip-classes.html) |
* [Modules](http://pyqt.sourceforge.net/Docs/PyQt5/modules.html) |
* [PyQt 5.9 Reference Guide](http://pyqt.sourceforge.net/Docs/PyQt5/index.html) »

© Copyright 2017 Riverbank Computing Limited. Created using [Sphinx](http://sphinx-doc.org/) 1.6.2.