Asynchronous operations in Symbian and Qt

Overview

This article provides a comparison between asynchronous request handling in Symbian and Qt.

Description

The following table compares the Active Objects/Active Scheduler mechanism in Symbian and signals/slots in Qt.

Solution

Operations	Symbian	Qt
Asynchronous event handling for applications using asynchronous service providers	Active Object	Signal/Slot
Class hierarchy	User-defined classes that need to use asynchronous services should be derived from CActive.	User-defined classes should be derived from QObject or any classes derived from it.
Wait loop once asynchronous request is made	CActiveScheduler class emulates the wait loop if an asynchronous event is waiting to be completed. Once the event completes, it calls the respective handler, for example, RunL.	No separate wait loop is required. Signal/slot mechanism combined with application main loop takes care of this.
Asynchronous event handling in UI	Active scheduler is built-in with UI applications which makes the application wait in an infinite loop until the next event is received. Events are generated when the user interacts with the UI components.	QApplication manages the event loop for an application. Signals are built-in for standard UI components and are emitted (generated) when the user interacts with the UI components.
Function invoked when asynchronous event is completed	For user-defined classes, the standard function RunL is invoked.	For user-defined classes, you can define the signals, emit those signals, and decide which function (slot) should handle those signals.
Functions invoked when events/signals from standard UI components need to be handled	Standard functions such as OfferKeyEventL, HandlePointerEventL, and HandleCommandL are invoked.	There are standard signals defined but you can decide the slot where it needs to be called when the signal is emitted.
Input parameters for the standard function	The function parameters for standard functions (RunL, OfferKeyEventL) are predefined and cannot be changed.	Signal parameter(s) and slot parameter(s) can be decided by the developer. The advantage of this is that the class level variables can be reduced as they can be created in a function and passed in signals/slots as parameters when the signals and slots are being connected. Even standard signals with predefined parameters can be altered using QSignalMapper classes.

(C) Copyright Nokia 2014. All rights reserved.

Number of asynchronous requests in user-defined class	There can be only one outstanding request in user-defined classes which will be serviced by RunL.	There can be many signals and slots connected in every class. Hence many requests can be serviced/processed.
User-defined events	Not possible.	User-defined classes can subclass standard classes and define their own signals. User-defined classes can also add their own signals.
Inter-object communication mechanism	No separate mechanism. The object whose information is required is usually kept as a member variable and standard C++ getter/setter functions are used. There is no direct mechanism of automatically getting intimated of the state information of an object.	Signals and slots can be used to exchange information between objects eliminating the need of an object being made a member variable of the class. The state of the object can also be tracked by making the object emit a signal and connecting the signal to a slot in the class.
Making asynchronous calls work like synchronous calls	Depending on whether the asynchronous function is executed in the same thread or different thread, one of the APIs in CActiveSchedulerWait or the API User::WaitForRequest() can be used.	QEventLoop can be used to achieve this: `QEventLoop eventLoop; connect(this, SIGNAL(TwmsAsyncFunctioncompleted()), &eventLoop, SLOT(quit()));` `eventLoop.exec(QEventLoop::ExcludeUserInputEvents);` This will also block the UI events in case the user does not want to process them until the asynchronous operation completes.

Other Links

Asynchronous Programming For Windows Phone 8

(C) Copyright Nokia 2014. All rights reserved.