Tips and Tricks - Using Qt Service Framework in Symbian

This article explains how to use Qt Service Framework in Symbian effectively.

Introduction

The Qt Service Framework defines a unified way of finding, implementing and accessing services across multiple platforms. Due to the service frameworks knowledge of service interfaces, their versions and `QObject`-based introspection it may even be used to unify and access multiple platform specific service implementations via the same Qt-based client application.

Overview

A service is an independent component that allows a client to perform a well-defined operation. Clients can find services based on their name and version as well as the interface that is implemented by the service object. Once the service has been identified the framework starts the service and returns a pointer to it. `QServiceManager` is the main interface through which clients can access the mentioned framework functionality. In addition services themselves may also act as clients to the service framework by requesting other services installed on the system.

Service provider are implemented via plug-ins. `QServicePluginInterface` defines the main interface for each plug-in. In order to avoid that clients have to link against service specific libraries each service object must be derived from QObject. Therefore the QMetaObject system can be used to dynamically discover and invoke the service's capabilities. To achieve the best level of access via the Qt meta object system services should be implemented in such a way that their entire functionality is accessible via the signals, slots, properties or invokable functions (see `Q_INVOKABLE` macro for more details).

Each service plug-in implements one service only but can provide multiple implementations for multiple interfaces. Therefore a service (plug-in) can retain a certain level of backwards compatibility even if the main service interface breaks in such a way that a new interface name has to be used. Existing clients can still use the previous version of service interface whereas new clients can utilize the new interface.

Services can also include remote processes. By registering with the service manager processes can communicate via signals, slots, invokable functions and properties as if they were local objects. Services can choose to be either shared between all clients, or unique to that client.

Tips and Tricks

Qt Service Framework (part of QtMobility, current version is 1.2) for Symbian has certain limitations and drawbacks. So the program using Service Framework for registering service for a particular interface and using it needs to be intelligent enough to avoid unforeseen risks. The Service Framework stores the services entries (which are added by respective service installer by calling `AddService()`) in a Service Database located at /private/2002AC7F. The default database is part of the system ROM and thus there is always a service db file available at Z:/private/2002AC7F. When a service is added, a new db file is generated at C:/private/2002AC7F. This contains the entries from the default db (that is still present in Z:/private/2002AC7F) and gets more preference than the default one. All new services gets added into the db in C:/private/2002AC7F.

The service db file name is based on the Qt version installed, for example: if Qt 4.8 is installed then "QtServiceFramework_4.7_system.db". If the device has Qt 4.7 + Qt Mobility 1.2 and is upgraded to Qt 4.8 then the Service Framework creates another database file at the same location with upgraded version file name, i.e. "QtServiceFramework_4.8_system.db". The old database file is not used anymore as the new version is available. However the services that were installed in the in the old version of db gets migrated to the new service db. So all the services still work.

There are uncommon cases where a service stops working because of missing servicedb entry. One of such cases is hard reset of the device. This can be achieved with the hard reset code (*#7370#), which in turn erases all user data from the System drive (C:) and Mass Memory drive (E:). In this case, the servicedb gets removed from C: and all the service entries that were in it are also lost. Though the actual service binary package might still be available in the device (if the binaries were installed in a MMC card or the package has NR flag which makes the package Non-Removable).

Solution: To handle this kind of problem, the first call to the service framework from the service-client needs to ensure that it tries to register the service if it is not found in the service database for the first time. For this Service Installer exe for the desired service has to be executed or write the code calling `AddService()` (that is actually available inside the service installer code).

```
QServiceManager serviceManager;
```

```
QServiceFilter serviceFilter("com.nokia.qt.ILocation");
serviceFilter.setServiceName("MyLocationService");
QList<QServiceInterfaceDescriptor> foundServices;
foundServices = serviceManager.findInterfaces(serviceFilter);
bool result = false;
if (foundServices.count() <= 0) //that means there is no service registered, may be
after factory reset
      {
      int success = QProcess::execute("MyServiceInstaller"); //wait till registration
completes, this is synchronous call
      if (success >= 0) //If the process cannot be started, -2 is returned. If the
process crashes, -1 is returned. Otherwise, the process' exit code is returned.
          foundServices = serviceManager.findInterfaces(serviceFilter); //refresh the
found services list
      }
if (foundServices.count() > 0)
     {
      ...
      ...
      }
```

While calling `RemoveService()`, it is important to note that though `RemoveService()` returns a success value, there is no change done inside the service db, the entry for that service remains same as it was.