QtDbus quick tutorial

Introduction

D-Bus has become the de-facto standard way of doing RPC on Linux desktop (and Maemo). Luckily, through QtDbus, it's also easy to use. This is a brief and "to-the-point" crash course on using QtDbus "the right way", i.e. through adapters and stubs. Simpler ways exist if you just need to call a random method on a random existing service somewhere, but for implementing "servers" on dbus the way presented here is the most robust one.

The approach should be familiar to everyone that has played with CORBA (IDL files).

Stubs

First, acquire or create the D-Bus *interface xml file* that describes the API.

You can see lots of sample interface files in */usr/share/dbus-1/interfaces* directory on your Linux desktop.

For purposes of this discussion, I'll be using a short xml file like this:

com.nokia.Demo.xml

```
<node>
  <interface name="com.nokia.Demo">
    <method name="SayHello">
      <annotation name="com.trolltech.QtDBus.QtTypeName.In1" value="QVariantMap"/>
      <arg name="name" type="s" direction="in" />
      <arg name="customdata" type="a{sv}" direction="in" />
    </method>
    <method name="SayBye"/>
    <signal name="LateEvent">
        <arg name="eventkind" type="s" direction="out"/>
    </signal>
 </interface>
</node>
```

It's not the simplest possible one, since it demonstrates a nice feature of being able to pass QVariantMap in the arguments (type "*a{sv}*"). Sooner or later you'll want to use it, since it allows extending the functionality of a method without augmenting the interface (and as such supporting "loose typing").

From this file, we'll create both client and server stubs:

Client **proxy**:

```
$ qdbusxml2cpp -v -c DemoIf -p demoif.h:demoif.cpp com.nokia.Demo.xml
```

Server stub (**adaptor**):

```
$ qdbusxml2cpp -c DemoIfAdaptor -a demoifadaptor.h:demoifadaptor.cpp com.nokia.Demo.xml
```

You may want to store these commands in a script - it's not meant to be invoked automatically, just add the generated files to version control and invoke the script when you edit the interface.

Implementing the server

The only nontrivial part is implementing the server. Create a project, add *demoifadaptor.cpp* to it, and take a look at *demoifadaptor.h*. We'll create a new class *MyDemo* to implement the functionality (I'm intentionally using a name as tacky as MyDemo - normally I would call it DemoService, but now I'm trying to convey there is nothing magical about this class, apart from the fact that it inherits from QObject).

Now the important part - you need to copy-paste the methods from DemoIfAdaptor class to MyDemo, **exactly** as they appear in demoifadaptor.h. After this, mydemo.h looks like this:

```
class MyDemo : public QObject
{
Q_OBJECT
public:
    explicit MyDemo(QObject *parent = 0);

public Q_SLOTS:
    void SayBye();
    void SayHello(const QString &name, const QVariantMap &customdata);
Q_SIGNALS:
    void LateEvent(const QString &eventkind);

};
```

The QObject magic used by the adaptor requires that the signatures in MyDemo match the ones in the adaptor.

(C) Copyright Nokia 2013. All rights reserved.

Note how Qt camelCase convention is not followed in the adaptor. This can help to serve as a clue about whether we are dealing with vanilla Qt or the slightly "magical" dbus entry points.

Now, in our `main()`, we hook the adaptor and our MyDemo class together:

```
int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);

    MyDemo* demo = new MyDemo;
    new DemoIfAdaptor(demo);

    QDBusConnection connection = QDBusConnection::sessionBus();
    bool ret = connection.registerService("com.nokia.Demo");
    ret = connection.registerObject("/", demo);
    return a.exec();
}
```

Note how we pass the MyDemo object as an argument to the constructor of the adaptor:

```
new DemoIfAdaptor(demo);
```

This invocation sets *demo* object as `QObject` *parent* of the adaptor. This is very important, as it is the route adaptor uses to relay method calls to our `MyDemo` object! Also, when a signal *LateEvent* is emitted by `MyDemo`, the signal is magically broadcast to dbus by the adaptor (i.e. a *Qt signal* is converted to *dbus signal*).

Here, we use "/" as the **object path** visible to dbus. Prevalent convention in case there is only one object is to use "/com/nokia/Demo", but this usually confuses newcomers (creating the illusion that the object path has to be a translated version of service / interface name).

Obviously, the program needs to be started before the object is visible on the D-Bus. For now, you can do this manually from the command line. In practice, D-Bus services should have an entry in */usr/share/dbus-1/services*, where dbus-daemon will find it and be able to launch the service when a call is done (so called "**dbus activation**"). See Qt application for Maemo with DBus support for an example.

Let's see that our object is available after launch by using "qdbus" inspection tool:

```
$ qdbus com.nokia.Demo
/

$ qdbus com.nokia.Demo /
signal void com.nokia.Demo.LateEvent(QString eventkind)
method void com.nokia.Demo.SayBye()
method void com.nokia.Demo.SayHello(QString name, QVariantMap customdata)
method QDBusVariant org.freedesktop.DBus.Properties.Get(QString interface_name, QString property_name)
method QVariantMap org.freedesktop.DBus.Properties.GetAll(QString interface_name)
method void org.freedesktop.DBus.Properties.Set(QString interface_name, QString property_name, QDBusVariant value)
method QString org.freedesktop.DBus.Introspectable.Introspect()
```

or

```
$ dbus-send --type=method_call --print-reply \
    --dest=com.nokia.Demo / org.freedesktop.DBus.Introspectable.Introspect
```

As you can see, `QtDbus` adds lots of extra functionality for free.

We can make it say "bye" by invoking:

```
$ qdbus com.nokia.Demo / com.nokia.Demo.SayBye
```

or

```
$ dbus-send --type=method_call --print-reply \
    --dest=com.nokia.Demo / com.nokia.Demo.SayBye
```

(qdbus is also available on the N900 device if you install "qqdbus", currently packaged in fremantle extras-devel. If you can tolerate a slightly more verbose syntax, you can use dbus-send tool, installed on the device by default).

Using the service

Using the client proxy is trivial - just include the generated "demoif.h", instantiate DemoIf (specifying the service and object path), connect to signals you are interested in and call the methods:

```
DemoIf* client = new DemoIf("com.nokia.Demo", "/", QDBusConnection::sessionBus(), 0);
QObject::connect(client, SIGNAL(LateEvent(QString)), this, SLOT(mySlot(QString)));
client->SayBye();
```

Optionally, you may use the fully qualified name *com::nokia::DemoIf* to refer to the `DemoIf` class.

Project Configuration

Add

(C) Copyright Nokia 2013. All rights reserved.

```
QT += dbus
```

to your .pro file

more info: http://doc-snapshot.qt-project.org/4.8/qtdbus.html

Recommended links

- QtDbus official documentation (includes introduction to D-Bus in general)
- Tutorials at KDE TechBase
- The official D-Bus page af freedesktop.org

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.