Use of MNotification class with a minimal D-Bus Adaptor

This article covers two interrelated topics: the use of `MNotification class` to show system-wide notifications in different possible forms; and implementing a minimal D-Bus adaptor in order to perform an action when user taps on such notification.

Introduction

MNotification class provided by Meego Touch API is responsible for different system-wide notifications like banners shown when a device is connected to a phone, grouped notifications on Event Feed screen indicating file transfers, lockscreen notification about new text messages, or short status bar notifications about IM presence.

This article will show how to use such notifications if your applicaton performs some actions in the background, and you want to notify a user about the result of such action.

Also you may want your application to do something when a user interacts with a created notification (taps a banner or an Event Feed item, or slides the lockscreen notification). Since the described notifications are a part of the system process, they cannot invoke your application methods directly. For this case you must implement a D-Bus adaptor which will allow you to make your methods invokable externally.

Implementation: Notification

To use UI elements from MeegoTouch API you first need to add the following line into the application's `.pro` file:

```
CONFIG += meegotouch
```

Then, if you are writing a Qt-based application (not a `MApplication` based one), you must instanciate a `MComponentData` object in your `main.cpp`:

```
// main.cpp
...
#include <MComponentData>
...
Q_DECL_EXPORT int main(int argc, char *argv[])
{
    QScopedPointer<QApplication> app(createApplication(argc, argv));
    MComponentData::createInstance(argc, argv);
...
}
```

Now you can move on to displaying a system-wide notifications in your methods:

```
// myappclass.cpp
...
MNotification notification(MNotification::DeviceEvent, "", tr("Notification text"));
notification.setImage("icon-m-common-done");
notification.publish();
...
```

The above snippet will show a system banner with "icon-m-common-done" icon and localized "Notification text".

The particular form of this notification (a banner) is set with the first `eventType` parameter of MNotification constructor. The full list of known options is given in MNotification class reference. Change `MNotification::DeviceEvent` type (which corresponds to the banner presentation) to other types to observe changes in look and behaviour of your notification. Some of notification types, unlike the banner showed here, also make use of the second constructor parameter, `summary`, so prior to experimenting with `eventType` it is advised to fill it in too so you can later determine whether you need it.

(C) Copyright Nokia 2014. All rights reserved.

Now you may want your application to react when a user interacts with your notification. For example, you may open a particular page that contains full information about the event a user has been just informed of.

Remember that since this is a system notification, it is NOT a part of your appliction's process and thus knows nothing about your methods.

`MNotification` class contains a method `setAction(const MRemoteAction &action)` that allows you to attach a D-Bus call to your notification. In fact, you can use this feature to perform an external call, but here we shall see how you can export your application method to D-Bus.

Implementation: D-Bus Adaptor

Below is the minimal D-Bus Adaptor you can create to make `myMethod` exposed to D-Bus:

```
// myadaptor.h
#include <QDBusConnection>
#include <QDBusAbstractAdaptor>
#include <QApplication>

class MyAdaptor: public QDBusAbstractAdaptor
{
    Q_OBJECT
    Q_CLASSINFO("D-Bus Interface", "com.nokia.myapp")
private:
    QApplication *app;
public:
    MyAdaptor(QApplication*);
public slots:
    void myMethod(const QString&);
};
```

```
//myadaptor.cpp
#include "myadaptor.h"

MyAdaptor::MyAdaptor(QApplication *application) : QDBusAbstractAdaptor(application),
app(application) {
}

Q_NOREPLY void MyAdaptor::myMethod(const QString &someString)
{
    // Do something here
}
```

And, finally, in your `main.cpp`:

```
// main.cpp
...
#include "myadaptor.h"
...
Q_DECL_EXPORT int main(int argc, char *argv[])
{
    QScopedPointer<QApplication> app(createApplication(argc, argv));
...
    new MyAdaptor(app.data());
    QDBusConnection::sessionBus().registerObject("/MyApp", app.data());
...
```


(C) Copyright Nokia 2014. All rights reserved.

```
}
```

Walkthrough: D-Bus Adaptor

In this example we intend to expose `myMethod`, which take one `QString` parameter, supposedly a file name, a phone number, or something present in our notification.

To do this, we must create a QDBusAbstractAdaptor-based class, where `myMethod` must be declared as a public slot. All methods declared as public slots in a QDBusAbstractAdaptor-based class automatically become invokable through D-Bus.

```
Q_CLASSINFO("D-Bus Interface", "com.nokia.myapp")
```

This line defines the name of the interface which will be used during the call.

Now we register our adaptor:

```
// main.cpp
 QDBusConnection::sessionBus().registerObject("/MyApp", app.data());
```

Here it's important to note that we register our object by "/MyApp" path, and with the session bus, which means that our method exposure will last until our application is terminated. The latter makes sense if our application has the main window, and a user may terminate it manually. If you are writing a console application, consider using `QDBusConnection::systemBus()` instead.

Further reading:

- Introduction to D-Bus
- Using QtDBus Adaptors

Implementation: MRemoteAction D-Bus Call

Now you can use MRemoteAction to call `myMethod` when a user interacts with your notification:

```
// myappclass.cpp
...
QList<QVariant> args;
args.append("string to be passed as someString parameter");
MRemoteAction opAction("com.nokia.myapp","/MyApp","com.nokia.myapp","myMethod", args);
MNotification notification(MNotification::DeviceEvent, "", tr("Notification text"));
notification.setAction(opAction);
notification.setImage("icon-m-common-done");
notification.publish();
...
```

Note `serviceName`, `objectPath` and `interface` parameters in `MRemoteAction` constructor which correspond to those declared above during D-Bus adaptor creation.

Also we supply `myMethod` with an argument through `QList<QVariant>`. Our method accepts only one argument, however, if it accepted more, we would append them to the list in the order they were declared.

(C) Copyright Nokia 2014. All rights reserved.

(C) Copyright Nokia 2014. All rights reserved.