Implementing the Qt plug-in interface

Overview

Qt applications can be extended with Qt plug-ins. This code snippet demonstrates how to implement the plug-in on top of the plug-in interface defined in the client application (the interface was described in the code snippet Defining a Qt plug-in interface).

The plug-in has to implement all methods defined in the plug-in interface. Additional methods and member variables can also be implemented. However, note that the client application only sees methods and data that are defined into the common interface `examplepluginterface.h`.

The defining of a plug-in interface and plug-in usage are described in additional snippet articles.

Important

- When using Qt libraries from the (`qt_libs_armv5_udeb.sisx`) on a device, be sure to build both the plug-in and the loading application in `DEBUG(UDEB)` mode. This is because Qt uses a build key for each plug-in to ensure that only compatible plug-ins are loaded, and the build mode `[debug` is part of that build key.
- In Symbian we have to define the `EPOCALLOWDLLDATA` variable as true for the plug-ins (library) because Qt macros have initialised global data; see plug-in project files for this.

Project .pro file

```
TEMPLATE        = lib

CONFIG         += plugin

# Define path where compiler can found your defined interface for the plugins
INCLUDEPATH    += ../../

HEADERS         = yourplugin.h
SOURCES         = yourplugin.cpp

TARGET          = $$qtLibraryTarget(yourplugin)

symbian: {
# Load predefined include paths (e.g. QT_PLUGINS_BASE_DIR) to be used in the pro-files
load(data_caging_paths)

# EPOCALLOWDLLDATA have to set true because Qt macros has initialised global data
TARGET.EPOCALLOWDLLDATA=1

# Defines plugin files into Symbian .pkg package
pluginDep.sources = yourplugin.dll
pluginDep.path = $$QT_PLUGINS_BASE_DIR/exampleplugins
DEPLOYMENT += pluginDep
}

target.path += $$[QT_INSTALL_PLUGINS]/exampleplugins
INSTALLS += target
```


(C) Copyright Nokia 2014. All rights reserved.

Header (yourplugin.h)

The header for your plug-in that implements your own `ExamplePluginInterface` plug-in interface.

```
// Your defined interface for the plugins
#include "examplepluginininterface.h"

class YourPlugin : public ExamplePluginInterface
    {
    // That is because we use Qt signal/slot feature
    // in "void someSlot(QString value)"
    Q_OBJECT

    // This macro tells Qt which interfaces the class implements
    // This is used when implementing plugins
    Q_INTERFACES(ExamplePluginInterface)

    public: // From ExamplePluginInterface
        ~LinkPlugin();
        QString sayHello();

    public slots: // From ExamplePluginInterface
        void someSlot(QString value);
    };
```

Source (yourplugin.cpp)

```
#include <QtCore/qplugin.h>
#include "yourplugin.h"

YourPlugin::~YourPlugin()
    {
    }

QString YourPlugin::sayHello()
    {
    return QString("Hello from YourPlugin: "+data);
    }

void YourPlugin::someSlot(QString value)
    {
    data = value;
    }

// This Qt macro exports the plugin class YourPlugin with the
// name "exampleplugins"
// There should be exactly one occurrence of this YourPlugin
// macro in a Qt plugin's source code.
Q_EXPORT_PLUGIN2(exampleplugins, YourPlugin);
```

See also

(C) Copyright Nokia 2014. All rights reserved.

- Defining a Qt plug-in interface
- Loading and initialising a Qt plug-in dynamically
- For more information about Qt plug-ins, see How to Create Qt Plugins.

Postconditions

The Qt plug-in interface is implemented.

(C) Copyright Nokia 2014. All rights reserved.