

Porting applications from the Nokia Series 60 to the UIQ reference design

Martin de Jode
Revision 1.1, Aug 2002

1. Introduction

This tutorial is one of a series describing some of the issues involved in porting applications between various flavours of Symbian OS. In this paper we shall consider taking a simple HelloWorld GUI application written for the Nokia's Series 60 reference design and transforming it into a HelloWorld example for the UIQ reference design.

This article presumes a basic understanding of C++ and some familiarity with building and running simple C++ applications on Symbian OS. Since this tutorial starts where "Getting Started with C++ Development on the Series 60 SDK" finishes, it is strongly recommended that you read the above first. See <http://www.symbian.com/developer/techlib/papers/cpp.html>.

In this paper we shall first briefly review the Series 60 Hello World example. We will then look at the changes necessary to create a basic UIQ version of the above. In Section 4 we shall consider some embellishments to the application to achieve the real UIQ "Look and Feel". Finally, In the last section we shall briefly review what we have learnt.

2. Nokia Series 60 HelloWorld

The Nokia Series 60 platform is Nokia's smartphone reference design and is built on the generic technology of Symbian OS. The Series 60 platform supports a screen of dimensions 176x208 pixels and is the basis for the Nokia 7650 phone.

The starting point for this tutorial is the HelloWorld application developed in the "Getting Started with C++ Development on the Series 60 SDK". If we build and run this application we should see the following



Fig 1 HelloWorld on the Series 60 Emulator

The HelloWorld UI principally consists of a window with a label containing the famous text. This is encapsulated in the `CHelloContainer` class. The constructor creates a window and a `CEikLabel` instance. The `Draw(...)` method (called by the window server) draws a filled rectangle to the UI.

This example was generated by the Nokia Series 60 Application wizard. The Series 60 uses the AVKON framework, which is an extension to Symbian's application framework (for more information consult the references listed in the bibliography), providing the Series 60 "Look and Feel".

3. Basic UIQ HelloWorld

We shall now consider changes to the code that are necessary to port the application to UIQ. The extension to Symbian's generic application framework for the UIQ reference design is known as QIKON. Hence the major change required to port an application from the Series 60 to UIQ is simply to make use of the respective QIKON application framework classes, instead of the AVKON framework classes.

3.1. Changes to the C++ source code

First let us consider the declaration of the `CHelloApp` class in `HelloApp.h`.

```
include <aknapp.h>
...
class CHelloApp : public CAknApplication {...};
```

The first thing to note is that class `CHelloApp` extends the AVKON `CAknApplication` class. We can change this so that `CHelloApp` extends the `CQikApplication` class,

```
include <qikapplication.h>
...
class CHelloApp : public CQikApplication {...};
```

using the QIKON framework instead. Consequently we need to include `qikapplication.h` instead of `aknapp.h` (and link against `qikctl.lib` in the `Hello.mmp` project file). The implementation code (in `HelloApp.cpp`) shouldn't need changing.

We do a similar thing for the `CHelloDocument` class. The class definition in the `HelloDocument.h` header file is shown below:

```
#include <akndoc.h>
...
class CHelloDocument : public CAknDocument {...};
```

Again we find that the `CHelloDocument` class extends `CAknDocument`. So instead we change this to extend the `CQikDocument`.

```
#include <qikdocument.h>
...
class CHelloDocument : public CQikDocument {...};
```

Instead of including the `akndoc.h` header we now include the `qikdocument.h` (linking against the `qikctl.lib`). In this case we also need to change the signature of the constructor in the source file so that the `CQikDocument` super class constructor is called.

```
CHelloDocument::CHelloDocument(CEikApplication& aApp) : CQikDocument(aApp)
{
    ...
}
```

Now we shall turn our attention to the `CHelloAppUi` class definition

```

#include <aknappui.h>
class CHelloAppUi : public CAknAppUi
{
};

```

Again `CHelloAppUi` extends from the AVKON class `CAknAppUi`. Once again we change this to derive from the `CQikAppUi` class. Hence we should include the `qikappui.h` instead of `aknappui.h` and link against the `qikctl.lib` library.

```

#include <qikappui.h>
class CHelloAppUi : public CQikAppUi
{
};

```

A few changes are necessary to the `CHelloAppUi` source code (`HelloAppUi.cpp`). The second phase constructor shown below

```

void CMyHelloAppUi::ConstructL()
{
    BaseConstructL();
    iAppContainer = new (ELeave) CMyHelloContainer;
    iAppContainer->SetMopParent(this);
    iAppContainer->ConstructL( ClientRect() );
    AddToStackL( iAppContainer );
}

```

has the following statement

```
iAppContainer->SetMopParent(this).
```

This tells the Container that this AppUi owns it. In the CONE world, every UI object (app ui, view or control) knows which objects it owns, unfortunately they generally do not know which object owns it. To work around this the AVKON version of `CCoeControl` implements the `MObjectProvider` interface, which mandates the `SetMopParent(...)` function. Since this is specific to AVKON it should be removed for the UIQ variant.

Also we no longer need to include the `avkon.hrh` file so we remove the following line

```
#include <avkon.hrh>
```

from the include statements at the top of `HelloAppUi.cpp` and replace it with

```
#include <eikenv.h>
```

to ensure this header (previously include with `avkon.hrh`) is still available.

The `CHelloContainer` class needs no alteration. This class derives from the `CCoeControl` and `MCoeControlObserver` classes. These are control classes (a control being a unit of user interaction using any combination of screen, keyboard, and pointer) and are not specific to the AVKON framework. However we can make a small cosmetic change to the `Draw(...)` function (shown below) of `HelloContainer.cpp` to bring the application more into line with the UIQ style by changing the background color to white.

```

void CHelloContainer::Draw(const TRect& aRect) const
{

```

```
CWindowGc& gc = SystemGc();
gc.SetPenStyle(CGraphicsContext::ENullPen);
gc.SetBrushColor(KRgbGray);
gc.SetBrushStyle(CGraphicsContext::ESolidBrush);
gc.DrawRect(aRect);
}
```

To do this simply remove the

```
gc.SetBrushColor(KRgbGray);
```

statement.

3.2. Changes to the resource file Hello.rss

We shall now look at the resource file (Hello.rss). The resource file is used to define the main elements of the UI in a convenient format. By removing these definitions from the C++ source code, porting of applications between Symbian OS UIs is facilitated.

First consider the EIK_APP_INFO resource, which identifies the symbolic IDs of the main user interface resources used by the application.

```
RESOURCE EIK_APP_INFO
{
hotkeys=r_hello_hotkeys;
menubar=r_hello_menubar;
cba=R_AVKON_SOFTKEYS_OPTIONS_BACK;
}
```

Since UIQ is a pen driven tablet UI (rather than CBA/keyboard driven) we have no need to define any hotkeys. Hence we can remove this line from EIK_APP_INFO resource. We can therefore also remove the HOTKEYS resource as redundant. Similarly the EIK_APP_INFO resource identifies a CBA control. This again is a feature not used by the UIQ reference design (being a pointer driven UI) so we can remove this line from EIK_APP_INFO resource (in this case the CBA control, R_AVKON_SOFTKEYS_OPTIONS_BACK, is pre-defined in avkon.rh).

The r_hello_app_menu MENU_PANE resource was produced by the Series 60 Wizard and is not used in this example. So we can delete it too.

Finally we shall make a couple of cosmetic changes to the r_hello_menu resource (shown below).

```
RESOURCE MENU_PANE r_hello_menu
{
items=
{
MENU_ITEM { command=EAknCmdExit; txt="Exit"; },
MENU_ITEM { command=EHelloCmdAppTest; txt="Say Hello"; }
};
}
```

According to the UIQ style guide (see for instance Reference 2 "Migrating C++ Programs to Quartz") UIQ applications are closed by the system, never by the user, when another application is opened. Hence a UIQ application should never have a Close or Exit option. Accordingly we shall remove the EAknCmdExit menu item (which is an AVKON stock menu item anyway). Since this leaves our menu pane rather empty we shall replace it with a custom "Say Goodbye" menu item (shown below)

```
MENU_ITEM { command=EHelloCmdAppGoodbye; txt="Say Goodbye"; }
```

to complement our "Hello" message.

Hence a minimal resource file for the UIQ HelloWorld example looks like this:

```
// RESOURCE IDENTIFIER
NAME    AWIZ // 4 letter ID

#include <eikon.rh>
#include "hello.hrh"
#include "hello.loc"

// RESOURCE DEFINITIONS
RESOURCE RSS_SIGNATURE { }

RESOURCE TBUF { buf="Hello"; }

RESOURCE EIK_APP_INFO
{
    menubar=r_hello_menubar;
}

RESOURCE MENU_BAR r_hello_menubar
{
    titles=
    {
        MENU_TITLE { menu_pane=r_hello_menu; txt="Hello"; }
    };
}

RESOURCE MENU_PANE r_hello_menu
{
    items=
    {
        MENU_ITEM { command=EHelloCmdAppTest; txt="Say Hello"; },
        MENU_ITEM { command=EHelloCmdAppGoodbye; txt="Say Goodbye"; }
    };
}
```

In keeping with the UIQ style guide we have changed the title on the menu bar (defined in the `r_hello_menubar` resource) from “File”, to “Hello” (the name of the application).

We need to add the `EHelloCmdAppGoodbye` identifier to our enumeration in the `Hello.hrh` file. Therefore this becomes

```
#ifndef HELLO_HRH
#define HELLO_HRH

enum THelloCommandIds
{
    EHelloCmdAppGoodbye = 1,
    EHelloCmdAppTest
};

#endif // HELLO_HRH
```

3.3. Handling the menu items

Finally we need to change the `HandleCommandL(...)` function (shown below)

```
void CMyHelloAppUi::HandleCommandL(TInt aCommand)
{
    switch ( aCommand )
    {
        case EAknSoftkeyBack:
        case EEikCmdExit:
            {
                Exit();
                break;
            }
        case EMyHelloCmdAppTest:
            {
                iEikonEnv->InfoMsg(_L("Hello Again"));
            }
    }
}
```

```

        break;
    }
    default:
        break;
    }
}

```

in HelloAppui.cpp to accommodate the changes we have made. We remove the

```

case EAKnSoftkeyBack:
case EEikCmdExit:
    {
        Exit();
        break;
    }

```

case statements from the switch clause since these are now redundant and replace them with our Say Goodbye option

```

case EHelloCmdAppGoodbye:
    {
        iEikonEnv->InfoMsg(_L("Goodbye"));
        break;
    }

```

Hence our UIQ HandleCommandL(...) function now looks like this

```

void CMYHelloAppui::HandleCommandL(TInt aCommand)
{
    switch ( aCommand )
    {
        case EHelloCmdAppGoodbye:
            {
                iEikonEnv->InfoMsg(_L("Goodbye"));
                break;
            }
        case EHelloCmdAppTest:
            {
                iEikonEnv->InfoMsg(_L("Hello Again"));
                break;
            }
        default:
            break;
    }
}

```

If we build and run the application we should see this



Figure 2 A minimal UIQ Hello World application

It is worth noting that, apart from changes to the resource file, the essential changes to the C++ source code required to create a working UIQ version were to derive from the QIKON framework classes instead of AVKON. The other changes we have made were cosmetic in order to comply with the UIQ style guide. The functionality that was previously offered by the Series 60 CBA is now incorporated into the UIQ menu bar.

4. Adding the UIQ Toolbar

In the previous section we showed how to develop a minimal HelloWorld application running on UIQ. In fact this application was entirely generic, using no UIQ specific features. In this section we shall show how we can add some UIQ controls, giving the application more of a UIQ “Look and Feel”.

The Series 60 HelloWorld application used two soft key CBA buttons (“Options” and “Back”) as its principal controls. We shall now show how we can use the QIKON framework to add a UIQ button toolbar to replace the controls previously offered by the Series 60 CBA. We shall then show how we can facilitate the displaying of an “Options” dialog, when the “Options” button is selected.

Most of the changes needed to achieve this are restricted to the Hello.rss resource file. First we need to add a toolbar identifier to our EIK_APP_INFO resource as shown below.

```
RESOURCE EIK_APP_INFO
{
    menubar=r_hello_menubar;
    toolbar=r_hello_toolbar;
}
```

We then define this toolbar as follows in the QIK_TOOLBAR resource.

```
RESOURCE QIK_TOOLBAR r_hello_toolbar
```

```

{
  controls=
  {
    {
      id=EMyToolBarButtonOptions;
      txt="Options";
    },
    {
      id=EMyToolBarButtonDummy;
      txt="Dummy";
    }
  };
}

```

The “Back” CBA button of the Series 60 example closed the application. However, as discussed earlier, such functionality is inconsistent with the UIQ style guide. So this control has now been replaced by a dummy button for the UIQ HelloWorld application. We have also provided an “Options” button on the UIQ toolbar to furnish the control previously offered by the “Options” CBA button of the Series 60 example.

Since this is a QIKON toolbar we need to include the `qikon.rh` resource header file. So we add the following line to the include statements at the top of `Hello.rss`.

```
#include <qikon.rh>
```

To identify the buttons we have to add the following enumeration to our `Hello.hrh` file

```

enum TExampleToolBarButtons
{
  EMyToolBarButtonOptions = 10,
  EMyToolBarButtonDummy
};

```

This is all that is required to add a toolbar containing an “Options” and “Dummy” button to our application!

However it would be nice to do something when the buttons are pressed, so we will create an options dialog which we shall show when the “Options” button is pressed. In the case of our “Dummy” button we shall simply flash an Info message indicating the button has been pressed.

Since a dialog is a dynamic feature of the UI we have to cater for its creation and handling in the C++ source code. To this end we shall create a minimal dialog class by extending the `CEikDialog` class (part of the generic EIKON framework). The class header file `HelloDialog.h` is shown below:

```

#include <eikdiag.h>
// class CHelloDialog
class CHelloDialog : public CEikDialog
{
public:
    TBool OkToExitL(TInt aKeyCode);
};

```

It declares the `OKToExitL(...)` method, which is a virtual function of the `CEikDialog` class and is invoked when the user presses any button in the button panel of the dialog other than the “Cancel” button. The function definition is in the `HelloDialog.cpp` source file and shown below.

```

#include "HelloDialog.h"
#include "Hello.hrh"

TBool CHelloDialog::OkToExitL(TInt aKeyCode )
{
    switch ( aKeyCode )
    {

```

```

    case EHelloCmdAppGoodbye:
    {
        iEikonEnv->InfoMsg(_L("Goodbye"));
        break;
    }
    case EHelloCmdAppTest:
    {
        iEikonEnv->InfoMsg(_L("Hello Again"));
        break;
    }

    default:
        break;
}

return ETrue;
}

```

The implementation provides the same functionality as the menu, namely displaying a Hello or Goodbye message according to the option selected. It uses the same identifiers already enumerated in the `THElloCommandIds` enumeration of `Hello.hrh`.

Finally we have to add some code in the `HandleCommandL(...)` method of the `CHelloAppUi` class (source file `HelloAppUi.cpp`) to handle the toolbar buttons.

```

void CHelloAppUi::HandleCommandL(TInt aCommand)
{
    switch ( aCommand )
    {
        case EMyToolbarButtonDummy:
        {
            iEikonEnv->InfoMsg(_L("Button pressed"));
            break;
        }

        case EHelloCmdAppGoodbye:
        {
            iEikonEnv->InfoMsg(_L("Goodbye"));
            break;
        }

        case EHelloCmdAppTest:
        {
            iEikonEnv->InfoMsg(_L("Hello Again"));
            break;
        }

        case EMyToolbarButtonOptions:
        {
            CEikDialog* dialog = new(ELeave) CHelloDialog();
            dialog->ExecuteLD(R_HELLO_DIALOG);
            break;
        }

        default:
            break;
    }
}

```

We have added two extra cases to the `switch` statement. Pressing the “Dummy” button (case `EMyToolbarButtonDummy`) just displays an info message. Whilst pressing the “Options” button (case `EMyToolbarButtonOptions`) creates and displays the options dialog.

We must of course now include the `HelloDialog` header file, so we add the following line

```
#include "HelloDialog.h"
```

to our include statements at the top of `HelloAppUi.cpp`.

Finally we shall define the dialog controls in the Hello.rss file. We define a minimal DIALOG resource

```
RESOURCE DIALOG r_hello_dialog
{
    title="Options";
    buttons = r_hello_dialog_buttons;
}
```

with a title and some buttons. The buttons are defined in a DLG_BUTTONS resource as shown below.

```
RESOURCE DLG_BUTTONS r_hello_dialog_buttons
{
    buttons=
    {
        DLG_BUTTON
        {
            id = EHelloCmdAppTest;
            button = CMBUT {txt = "hello";};
        },

        DLG_BUTTON
        {
            id = EHelloCmdAppGoodbye;
            button = CMBUT {txt = "goodbye";};
        },

        DLG_BUTTON
        {
            id = EEikBidCancel;
            button = CMBUT {txt = "Back";};
        }
    };
}
```

The Hello and Goodbye buttons are identified by the previously defined `THElloCommandIds` enumeration, whilst the Cancel option uses the stock identifier enumerated in `EIKDIALOG.HRH` (which we have already included in Hello.rss via `eikon.rh`) ensuring our `OKToExitL(...)` handler of the `CHelloDialog` class works correctly.

If we now build and run the application, (remembering first to update our Hello.mmp project file to include the HelloDialog.cpp source and `eikdlg.lib` library files) we should see the following.



Figure 3 The UIQ HelloWorld with toolbar

Pressing the "Options" button brings up the options dialog as shown below.



Figure 4 Illustrating the "Options" Dialog

5. Summary

In this paper we have discussed porting a simple HelloWorld application generated using the Nokia Series 60 application wizard to the UIQ reference design. After briefly discussing the Series 60 version in Section 2, we then proceeded to strip out the series 60 specific features producing a minimal UIQ HelloWorld application using only generic (ie EIKON) controls. Section 4 of the paper showed how to embellish the application with some additional UIQ features such as the QIKON toolbar.

We have seen how the flexible application framework used by Symbian OS minimises the changes required to the C++ source code so that apart from minor changes to the handling of new controls it is sufficient to change the concrete application, document and appui classes from extending the respective AVKON parent classes to the respective QIKON parent classes. We have also seen how by defining the main components of the UI in an external resource file further minimises changes to the actual C++ source code.

The final source code accompanying this paper is available in the associated Hello.zip file.

6. Bibliography

1. Getting Started with C++ Development on the Series 60 SDK
<http://www.symbian.com/developer/techlib/papers/series60/series60.html>
2. Migrating C++ Programs to Quartz
<http://www.symbian.com/developer/techlib/papers/cpp.html>
3. Porting 9200 Series Applications to UIQ
http://www.symbian.com/developer/techlib/papers/portingXSTAL_UIQ/porting_xstal.html
4. Professional Symbian Programming, Chapter 9; Tasker M et al, Wrox Press (2000).

Want to be kept informed of new articles being made available on the Symbian Developer Network?
[Subscribe to the Symbian Community Newsletter.](#)

The Symbian Community Newsletter brings you, every month, the latest news and resources for Symbian OS.