

# Designing and building portable UIs for Symbian OS: How to design dialog interfaces.

Sander van der Wal, mBrain Software

Revision 1.1, January 2004

## Introduction

Consider the problem of porting a program from one UI to another. If you have followed Symbian's programming guidelines, you will have divided the application into an engine and a user interface. The engine is UI-independent, while the user interface is obviously not.

One popular porting approach is to make a copy of the files that make up the user interface part of the program, and port that copy. This approach works, but has the disadvantage that you need to keep a number of files up-to-date when you make changes to the user interface. Also, a number of files will be identical between the different UI implementations. It would be a shame and bad engineering practice to keep multiple copies of these files.

In a more general sense, the purpose of the code will be identical in all implementations, even though the actual implementations might be different. As an example, I'll present an actual problem I had in one of my applications. Pdf+, an Adobe PDF file viewer, allows the user to go to a page in the document. The application asks for the page number in a dialog and, once the user has entered the page number, goes to that page.

- In the Symbian OS v5 (Eikon user interface) distribution, it was possible to use a built-in Eikon dialog. This is nice because using a built-in dialog reduces the memory footprint.
- In the Series 80/CKON distribution for the Nokia 9200 Communicator series, I had to provide my own dialog. This was not a major problem as the dialog is simple.
- In the Series 60/Avkon distribution, using a `CAknNumberQueryDialog` was the most natural way to ask for a page number. The user experience is best when using an Avkon-specific query, because the built-in apps also present the user with similar dialogs. As in the Symbian OS v5 case it also reduces the memory footprint.
- In the UIQ distribution, I again had to provide my own dialog. I could use the code from the CKON dialog, but for one thing: I had to use a UIQ-specific control to get at the page number editor in the dialog. The result was that I made a copy of the CKON implementation and changed it slightly.

Even a simple dialog like this presents a number of obstacles to writing code that is portable between the different UIs. The dialogs themselves aren't portable between the UIs for a number of reasons, and as a result, code that is calling the dialogs isn't portable either:

- In the Symbian OS v5 distribution, I included an Eikon header file and called the Eikon dialog.
- In the CKON and UIQ distributions, I included my own header file and called my own dialog
- In the Avkon distribution, I included an Avkon header file and called the Avkon query dialog.

I'll discuss in a number of articles some techniques that will allow you to write UI-level code that is mostly portable between different UIs. It should be obvious that the need for different dialogs is an obstacle, so I'll start with a technique for solving this particular obstacle.

## Symbian OS dialog programming

In the early days, the archetypical Symbian OS C++ dialog was relatively simple. You derived your dialog from `CEikDialog`. You implemented the `PreLayoutDynInitL()` method to initialize the values in the different fields,

and the `OkToExitL()` method to check the fields.

Using such a dialog wasn't hard either, you created one, and then called its `ExecuteLD()` method. You could pass parameters for the dialog to fill in, and you could act on these values depending on whether or not the user pressed 'OK'.

In code:

```
class CMyDialog: public CEikDialog
{
public:
    CMyDialog(TInt& aParameter): iParameter(aParameter) {}
private:
    void PreLayoutDynInitL();
    TBool OkToExitL(TInt aButtonId);
private:
    TInt& iParameter
};

// Using the dialog
TInt myParameter = 0;
CEikDialog* myDialog = new(ELeave) CMyDialog(myParameter);
if (myDialog->ExecuteLD(R_MY_DIALOG)
    {
    // Do something
    }
```

The CKON user interface for the Nokia 9200 Communicator series introduced something new: a single static method called `RunDlgLD()` that creates and runs the dialog in a single line. Most built-in CKON dialogs use this method of creating and running a dialog.

In code:

```
class CMyDialog: public CEikDialog
{
public:
    static TBool RunDlgLD(TInt& aParameter);
private:
    CMyDialog(TInt& aParameter): iParameter(aParameter) {}
    void PreLayoutDynInitL();
    TBool OkToExitL(TInt aButtonId);
private:
    TInt& iParameter
};

// Implementation
TBool CMyDialog::RunDlgLD(TInt aParameter)
{
    CMyDialog* self = new(ELeave) CMyDialog(aParameter);
    return self->ExecuteLD(R_MY_DIALOG);
}

// Using the dialog
TInt myParameter = 0;
if (CMyDialog::RunDlgLD(myParameter))
    {
    // Do something
    }
```

The Avkon UI for the Series 60 platform introduced even more possibilities:

- You can derive your dialog from `CAknDialog` and `CAknForm` as well as from `CEikDialog`
- A number of specialized dialogs, including notes, queries and waiting dialogs are available.

The UIQ User Interface also has some special dialogs, but no new approaches were introduced.

Finally, a quick look at the newer Series 90 UI from Nokia shows it appears to be similar to the Series 80 UI.

## The problem

Back to my “Go to page” dialog. Earlier I enumerated the four different dialogs I had written for asking a page number. Let’s now take a more detailed look at each one.

In the UI, in the Symbian OS v5 code, I used the `CEikPreviewGotoDialog` class. This dialog displays the current page and the number of pages. The current page can be changed, and is returned in the first parameter.

```
#include <eikon.rsg>
#include <eikprtpv.h>

void CPdfAppUi::CmdGoToPageL()
{
    TInt newPage = iPage; // Start with the current page

    CEikPreviewGotoDialog* dialog =
        new(ELeave) CEikPreviewGotoDialog(newPage, iDoc->getNumPages());
    if (dialog->ExecuteLD(R_EIK_DIALOG_PREVIEW_GOTO))
    {
        if (newPage != iPage)
            ChangePageL(newPage);
    }
}
```

To be able to use the code, I had to include the header files `eikprtpv.h` for the dialog and the header file `eikon.rsg` for the resource `R_EIK_DIALOG_PREVIEW_GOTO`.

As you can see below, the CKON code is almost identical to the Symbian OS v5 code, apart from the use of my own dialog:

```
#include "GoToPageDialog.h"

void CPdfAppUi::CmdGoToPageL()
{
    TInt newPage = iPage; // Start with the current page

    CEikDialog* dialog =
        new(ELeave) CGoToPageDialog(newPage, iDoc->getNumPages());
    if (dialog->ExecuteLD(R_GO_TO_PAGE_DIALOG))
    {
        if (newPage != iPage)
            ChangePageL(newPage);
    }
}
```

The dialog itself has the following interface and implementation:

GoToPageDialog.h, for CKON

```
#include <eikdialog.h>
class CGoToPageDialog: public CEikDialog
{
public:
    CGoToPageDialog(TInt& aPage, TInt aMaxPage)
        : iPage(aPage), iMaxPage(aMaxPage)
    {}
private:
    void PreLayoutDynInitL();
    TBool OkToExitL(TInt aButtonId);
private:
    TInt& iPage;
    TInt iMaxPage;
};
```

GoToPageDialog.cpp for CKON

```
#include "GoToPageDialog.h"
#include "Pdf.hrh"
#include <eikmfne.h>

void CGoToPageDialog::PreLayoutDynInitL()
{
```

```

CEikNumberEditor* pageEd = STATIC_CAST(CEikNumberEditor*,
    Control(EGoToPageDialogPage));
pageEd->SetMinimumAndMaximum(1, iMaxPage);
pageEd->SetNumber(iPage);

TBuf<10> buf;
buf.Num(iMaxPage);
STATIC_CAST(CEikLabel*,
    Control(EGoToPageDialogNumPages))->SetTextL(buf));
TBool CGoToPageDialog::OkToExitL(TInt /*aButtonId*/)
{
    iPage = STATIC_CAST(CEikNumberEditor*,
        Control(EGoToPageDialogPage))->Number();
    return ETrue;
}

```

Nothing spectacular, I would say!

The Avkon code looks like this:

```

void CPdfAppUi::CmdGoToPageL()
{
    TInt newPage = iPage; // Start with the current page

    CAknNumberQueryDialog* dlg = CAknNumberQueryDialog::NewL(newPage,
        CAknQueryDialog::ENoTone);
    dlg->PrepareLC(R_GO_TO_PAGE_DIALOG);
    dlg->SetMinimumAndMaximum(1, iDoc->getNumPages());
    if (dlg->RunLD())
    {
        if (newPage != iPage)
            ChangePageL(newPage);
    }
}

```

Finally, for the UIQ code, it was possible to reuse most of the CKON code. In UIQ, CEikNumberEditor had to be replaced by CQikNumberEditor, but that was the only change. For completeness, here's the complete code for UIQ:

```

#include "GoToPageDialog.h"

void CPdfAppUi::CmdGoToPageL()
{
    TInt newPage = iPage; // Start with the current page

    CEikDialog* dialog =
        new(ELeave) CGoToPageDialog(newPage, iDoc->getNumPages());
    if (dialog->ExecuteLD(R_GO_TO_PAGE_DIALOG))
    {
        if (newPage != iPage)
            ChangePageL(newPage);
    }
}

```

The dialog itself has an interface and implementation as follows.

GoToPageDialog.h for UIQ

```

#include <eikdialog.h>
class CGoToPageDialog: public CEikDialog
{
public:
    CGoToPageDialog(TInt& aPage, TInt aMaxPage)
        : iPage(aPage), iMaxPage(aMaxPage)
    {}
private:
    void PreLayoutDynInitL();
    TBool OkToExitL(TInt aButtonId);
private:
    TInt& iPage;
    TInt iMaxPage;
};

```

GoToPageDialog.cpp for UIQ

```
#include "GoToPageDialog.h"
#include "Pdf.hrh"
#include <QikNumberEditor.h>

void CGoToPageDialog::PreLayoutDynInitL()
{
    CEikNumberEditor* pageEd = STATIC_CAST(CQikNumberEditor*,
        Control(EGoToPageDialogPage));
    pageEd->SetMinimumAndMaximum(1, iMaxPage);
    pageEd->SetValueL(iPage);

    TBuf<10> buf;
    buf.Num(iMaxPage);
    STATIC_CAST(CEikLabel*,
        Control(EGoToPageDialogNumPages))->SetTextL(buf);
}

TBool CGoToPageDialog::OkToExitL(TInt /*aButtonId*/)
{
    iPage = STATIC_CAST(CQikNumberEditor*,
        Control(EGoToPageDialogPage))->Value();
    return ETrue;
}
```

## Analysis

When we examine the problem, it appears that there are in fact two sub-problems:

1. Using the dialog isn't portable.
2. The dialog's implementation isn't portable.

There is very little we can do about 2). If a UI insists on using its own controls, that's that. The dialog will not work without the UI-specific control.

Therefore, let's turn our attention to sub-problem 1). If we examine the code closely, we can see that there are two sets of differences. The first set consists of the include files used, and the second set are the calls. For reference, I have listed the call sites again

For Symbian OS v5:

```
CEikPreviewGotoDialog* dialog =
    new(ELeave) CEikPreviewGotoDialog(newPage, iDoc->getNumPages());
if (dialog->ExecuteLD(R_EIK_DIALOG_PREVIEW_GOTO))
```

For CKON and UIQ:

```
CEikDialog* dialog =
    new(ELeave) CGoToPageDialog(newPage, iDoc->getNumPages());
if (dialog->ExecuteLD(R_GO_TO_PAGE_DIALOG))
```

For Avkon:

```
CAknNumberQueryDialog* dlg = CAknNumberQueryDialog::NewL(newPage,
    CAknQueryDialog::ENoTone);
dlg->PrepareLC(R_GO_TO_PAGE_DIALOG);
dlg->SetMinimumAndMaximum(1, iDoc->getNumPages());
if (dlg->RunLD())
```

It is obvious that the calls in CKON and UIQ are identical as the interfaces of both CGoToPageDialog implementations are identical. The methods of the CEikPreviewGotoDialog are identical again, although their names are different. The reason is of course that the CGoToPageDialog was modeled after the CEikPreviewGotoDialog.

A solution therefore could be to use a class CGoToPageDialog. This class could be implemented by deriving it from CEikDialog in the CKON and UIQ case, from CEikPreviewGotoDialog in the Symbian OS v5 case, and from CAknNumberDialog in the Avkon case.

This solution has a number of disadvantages.

1. It might be impossible to derive `CGoToPageDialog` from `CAknNumberDialog` or `CEikPrevewGoToDialog`. These classes could have been implemented in such a way that user derivation is impossible.
2. Even if it were possible in this case, it is easy to imagine that it is impossible for other kind of dialogs. The solution cannot therefore be considered as a general solution.
3. Even if it is always possible, each implementation would need its own header file as for each UI, the `CGoToPageDialog`'s parent would be a different class. The result is that you would still need to manage a number of header files that must be kept in perfect synch.
4. Finally, the different header files will introduce extra identifiers in all compilation units they are used in. This could result in clashes between your identifiers and the Symbian OS ones.

## The solution

Fortunately it *is* possible to create a goto page dialog class that doesn't have these problems. The idea came when I was examining the CKON trick of using a single static method to run a dialog. For your convenience, I have copied the relevant parts below:

```
class CMYDialog: public CEikDialog
{
public:
    static TBool RunDlgLD(TInt& aParameter);
private:
    CMYDialog(TInt& aParameter): iParameter(aParameter)
    void PreLayoutDynInitL();
    TBool OkToExitL(TInt aButtonId);
private:
    TInt& iParameter
};

TBool CMYDialog::RunDlgLD(TInt aParameter)
{
    CMYDialog* self = new(ELeave) CMYDialog(aParameter);
    return self->ExecuteLD(R_MY_DIALOG);
}
```

If you examine the `RunDlgLD()` method carefully, you'll notice that it doesn't matter very much that it is a member of the `CMYDialog` class. It only uses public methods. It could have been a method in a different class, or even a global function. This means that it is possible to create an interface class that completely hides the actual implementation (for more details, read about "fully Insulated classes" in Lakos[1]). Finally, the `RunDlgLD()` method uses an automatic variable for pointing to the dialog. It is not necessary to make the pointer to the dialog a member variable. This idea led to the following class definition:

`GoToPageDialog.h`, generic header file

```
#include <e32std.h>

class GoToPageDialog
{
public:
    static TBool RunDlgLD(TInt& aPage, TInt aMaxPage);
};
```

The `GoToPageDialog::RunDlgLD()` method constructs and runs a specific goto page dialog, but this implementation is now completely hidden inside the `GoToPageDialog.cpp` file. If you wish, you can also use a global function, a struct or a namespace instead of a class.

Using the dialog looks like this:

```
#include "GoToPageDialog.h"

void CPdfAppUi::CmdGoToPageL()
```

```

{
    TInt newPage = iPage; // Start with the current page
    if (GoToPageDialog::RunDlgLD(newPage, iDoc->getNumPages())
    {
        if (newPage != iPage)
            ChangePageL(newPage);
    }
}

```

Let us now examine some concrete implementations:

GoToPageDialog.cpp, CKON implementation

```

#include "GoToPageDialog.h"

#include "Pdf.hrh"
#include <Pdf.rsg>

#include <eikdialog.h>

class CGoToPageDialogImp: public CEikDialog
{
public:
    CGoToPageDialogImp(TInt& aPage, TInt aMaxPage)
        : iPage(aPage), iMaxPage(aMaxPage)
    {}
private:
    void PreLayoutDynInitL();
    TBool OkToExitL(TInt aButtonId);
private:
    TInt& iPage;
    TInt iMaxPage;
};

void CGoToPageDialogImp::PreLayoutDynInitL()
{
    // etc.
}

// Implementation of the static method.
TBool GoToPageDialog::RunDlgLD(TInt& aPage, TInt aMaxPage)
{
    CEikDialog* imp = new(ELeave) CGoToPageDialogImp(aPage, aMaxPage);
    return imp->ExecuteLD(R_GO_TO_PAGE_DIALOG);
}

```

This means that we can now also use existing dialogs instead of rolling our own, like this:

GoToPageDialog.cpp, Avkon implementation

```

#include "GoToPageDialog.h"

// Ui
#include <Pdf.rsg>

// Avkon
#include <avkon.hrh>
#include <aknquerydialog.h>

TBool GoToPageDialog::RunDlgLD(TInt& aPage, TInt aMaxPage)
{
    CAknNumberQueryDialog* imp =
        CAknNumberQueryDialog::NewL(aPage, CAknQueryDialog::ENoTone);
    imp->PrepareLC(R_GO_TO_PAGE_DIALOG);
    imp->SetMinimumAndMaximum(1, aMaxPage);
    return imp->RunLD();
}

```

## Conclusion

We have seen that it is possible to create a dialog interface that completely hides its implementation. As a result, the dialog interface has become portable across multiple UIs. Using a common interface will enable us to write UI-level code that is independent of the actual UI. How to do that is the topic of the second paper in this series.

## Background

Before becoming a Symbian OS C++ developer, Sander van der Wal worked in Pascal as a systems designer and project leader for a Dutch software company building, maintaining and selling a hospital information system. The programs were of the traditional, non-event driven kind. Some of the projects he worked on include a subset SQL compiler and an almost complete rewrite of the company's text editing system.

After establishing mBrain Software, Sander published four highly successful programs publicly available: Pdf+ (an Adobe PDF file viewer), PdfPrinter (a PDF document creator), Fonts (a font file management tool) and FontMachine (an Open Font System font renderer). Sander has extensive experience with Symbian OS phones – Pdf+, for example, is available for all Symbian OS devices currently in the market.

## References

[1] John Lakos. *Large-scale C++ software design*. Addison-Wesley. ISBN: 0201633620.

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.