

How to write a Control Panel Application

Writing a control panel app

Rod Burns, Developer Consultant, Symbian Ltd
Revision 1.0, July 2004

Summary

This paper and example code show how to implement a control panel application based on a dialog.

In addition to showing the essential characteristics of control panel applications, the paper aims to give readers an insight into good practice for implementing dialogs.

The example code was developed using Symbian OS version 7.0. The same code will also run on Symbian OS v7.0s and Symbian OS v8.0a. It uses the Techview UI, a test UI against which Symbian OS core development and testing is carried out. A Symbian OS v7.0s SDK which uses this UI is available with the new book [Symbian OS C++ for Mobile Phones - Volume 2](#). Note that the code can easily be ported to UIQ; details of the required changes are given in Appendix A.

1	PURPOSE AND SCOPE	3
2	WHAT IS A CONTROL PANEL APPLICATION?.....	3
3	WHAT DOES THE EXAMPLE CODE DO?.....	3
4	IMPLEMENTING THE CONTROL PANEL APP	4
4.1	CLASS STRUCTURE.....	4
4.2	THE .MMP FILE.....	4
4.3	THE FIRST ORDINAL EXPORTED FUNCTION.....	5
4.4	THE .HRH FILE	5
4.5	THE RESOURCE FILE	5
4.5.1	<i>DIALOG</i>	5
4.5.2	<i>DLG_BUTTONS</i>	6
4.5.3	<i>ARRAY</i>	7
4.6	CREATING A LIST DIALOG.....	7
4.7	CREATING A COLUMN DIALOG	8
4.8	PRE AND POST LAYOUT INITIALISATION	9
4.9	USING CONTROLS AND CATCHING EVENTS.....	9
5	SUMMARY	11
6	APPENDIX A – PORTING THE APPLICATION TO UIQ.....	11

Purpose and Scope

This paper is targeted towards developers looking to implement a Control Panel application. It shows the structure of the classes involved in writing a simple dialog, including how to set up a list dialog that uses columns, pages and scroll bars.

What is a Control Panel application?

Control panel applications are used to configure settings with system-wide scope, e.g. Internet IAP, sound settings, Bluetooth settings and FEP selection, etc. Control panel applications should not be used to configure settings with an application scope; these should be set within the owning application.

Figure 1 shows an example of a control panel application used for sound settings in the Techview UI.

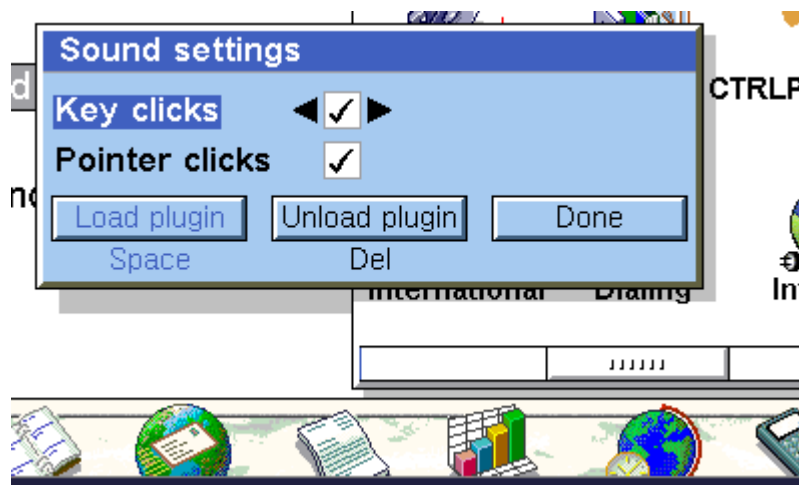


Figure 1

The control panel application framework is trivial:

- Control panel applications must export the following first ordinal function:

```
EXPORT_C void CreateControlL(const TDesC& aPath)
```
- The MMP file must specify a target type of CTL, a path of \SYSTEM\CONTROLS, and a UID2 of 0x10003A34.

In the example code, the bulk of the application is implemented as a dialog. This dialog is instantiated and executed from the function `CreateControlL()`.

What does the example code do?

The example code builds a simple Control Panel application.

The control panel application creates a dialog with two pages. The first page displays the list of applications available, while the second displays all the .ini files that are present on the C:\ drive.

The application allows items to be removed from the list; it does not actually remove them from memory but simply removes them from the display list. The pages are shown in figure 2.

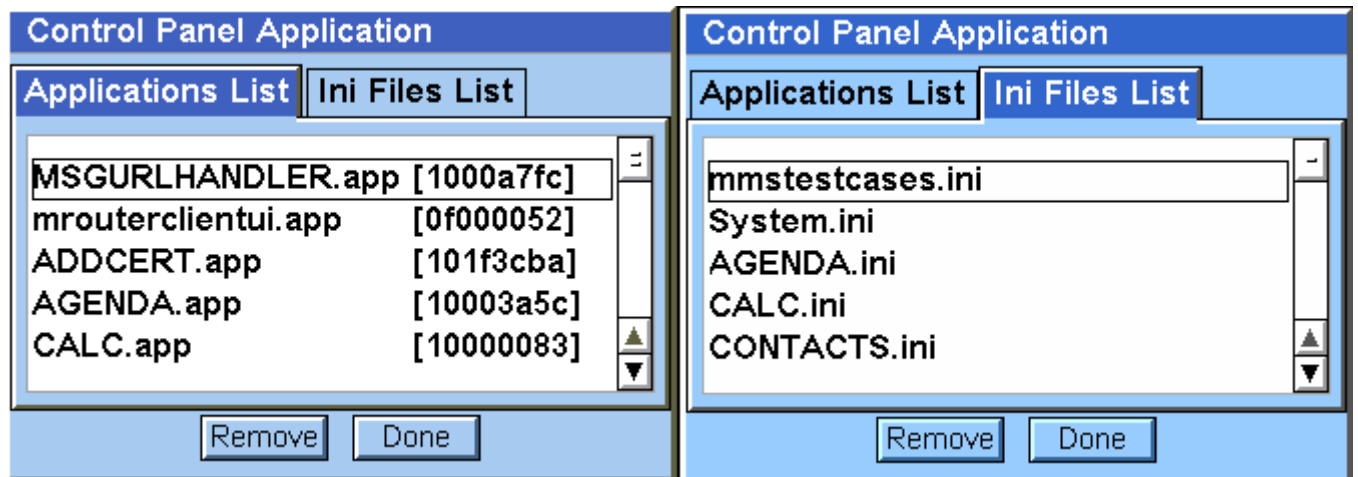


Figure 2

To build the code, change to the same directory as the bld.inf file and use the commands:

```
bldmake bldfiles
abld build wins udeb
```

When the emulator is re-started the control panel will have a new control panel application called "CtrPanelApp".

Implementing the control panel app

Class structure

All dialogs are derived from CEikDialog, the generic base class for dialogs. The dialog is dynamically constructed from a structure that is specified in a resource file.

The example dialog inherits from the observer class MEikListBoxObserver, and therefore implements the HandleListBoxEventL() method. The observer is used to detect control events that occur in the list boxes (such as double clicks).

Two arrays of descriptors (type CDesCArrayFlat) are used to store the data that is shown in the list boxes. The CColumnListBoxData class is used to set up the column settings for the applications list box, these being the font used, the alignment of the text and the width of each column in pixels.

The .mmp file

For a control panel app you must set the target type as "CTL" and the path as \SYSTEM\CONTROLS. The uid for a control panel application is 0x10003A34.

```
target          CtrIPanelApp.CTL
targettype     CTL
targetpath     \SYSTEM\CONTROLS
uid            0x10003A34 0x10201874
```

You will need to provide your own unique id for the second id; this can be obtained from Symbian by sending an email to: uids@symbian.com.

The first ordinal exported function

The main entry code is in the function `CreateControlL()`. It is here that the dialog is instantiated and executed. This method in the example code looks like this:

```
// Entry point for creating the dialog
EXPORT_C void CreateControlL(const TDesC& aPath)
{
    CControlPanelDialog* dialog = new (ELeave) CControlPanelDialog();
    CleanupStack::PushL(dialog);
    dialog->AddResourceFileL(aPath);
    CleanupStack::Pop(dialog);
    dialog->ExecuteLD(R_CTRL_PANEL_DIALOG);
}
```

The dialog is created and the structure of the dialog is read in from the resource file. The path that is passed in is the location of the `.CTL` file that the application uses. This is used to find the location of the resource file used to create the dialog. Finally the dialog is executed.

The standard dll entry point `E32Dll()` must also be implemented.

```
// Main entry point for dll
GLDEF_C TInt E32Dll(TDllReason)
{
    return (KErrNone);
}
```

The .hrh file

The `.hrh` file stores the control ids that are assigned to each of the dialogs and controls. These ids are used in code to distinguish between events from different controls.

In the example the ids are passed to the `HandleControlStateChangeL()` method.

The resource file

The structure of the dialog is defined in the resource file (`ControlPanelApplication.rss`). Here the main dialog, buttons and pages are defined, along with other additions to the dialog box.

Common to all Symbian OS resource files, the file must define the resource "NAME"; this is done using the following in the example code:

```
NAME CTRA
```

The following line must also be included otherwise a panic will occur when the dialog is invoked:

```
RESOURCE RSS_SIGNATURE { }
```

It is important to know the structures that are used for different types of dialogs. In the example code three types of structure are used:

DIALOG

The `DIALOG` structure is used to define a main dialog box for the application.

The main structure in the resource file is R_CTRLPANEL_DIALOG, of the DIALOG resource type. Inside the structure we define the title that is shown in the top bar, the buttons and the pages for the dialog.

The title STRING_R_CTRLPANEL_DIALOG is defined in the file ControlPanelApplication.rls along with some other strings that are used in the dialog. This is a common paradigm to allow localisation of program strings in a single location.

The “flags” field is set to EEikDialogFlagNotifyEsc, this means that the OkToExitL() method is called when either of the buttons are pressed or a cancel or escape command is invoked.

The following taken from the example code shows the main dialog’s structure:

```
RESOURCE DIALOG r_ctrlpanel_dialog
{
    title = STRING_r_ctrlpanel_dialog;
    buttons = r_ctrlpanel_dialog_buttons;
    flags = EEikDialogFlagNotifyEsc;
    pages = r_ctrlpanel_pages;
}
```

DLG_BUTTONS

The DLG_BUTTONS structure is used to define the buttons that are displayed on the dialog box. This is referenced by the main dialog structure (see R_CTRLPANEL_DIALOG_BUTTONS).

Setting the “button” field to CMBUT creates a CEikCommandButton for the dialog. It is worth noting that the “Done” button is assigned as a hotkey. This means it is predefined as the button used to exit and means when OkToExitL() is called, EEikBidOk is passed in as the button id argument. The code that defines the buttons is as follows:

```
// This defines the buttons for the dialog
RESOURCE DLG_BUTTONS r_ctrlpanel_dialog_buttons
{
    buttons =
    {
        DLG_BUTTON
        {
            id=ERemoveButton; // Button Id - defined in the .hrh file
            flags = 0;
            button=CMBUT // Defines a command button
            {
                txt="Remove"; // Text shown on the button
            };
        },
        DLG_BUTTON
        {
            id = EEikBidOk; // Predefined id for an "OK" type button
            button = CMBUT // Defines a command button
            {
                txt = "Done";
            };
            hotkey = EEikBidOk;
            flags = EEikLabeledButtonIsDefault; // sets this button as
default
        }
    }
}
```

```
};
}
```

ARRAY

The ARRAY structure is used to store one or more items referenced from another structure.

In the example code the structure R_CTRLPANEL_PAGES is used to define the two dialog pages; one page is a standard list box, the other is a list box with columns.

The column list box is defined by setting the “type” field to EEikCtColListBox and the other page is defined as being a standard list box by setting the “type” field to EEikCtListBox. The width of the listbox can also be independently defined inside this structure by setting the “width” field. The width is measured in characters.

The following extract (from the example code) shows how the list box with columns dialog is implemented.

```
// This defines the applications list page
RESOURCE ARRAY r_appslist_page
{
    items =
    {
        DLG_LINE
        {
            type = EEikCtColListBox; // Column list box
            id = EAppsListBox;
            control = LISTBOX
            {
                width = 10; // width in characters
            };
        }
    };
}
```

Creating a list dialog

A list dialog is a dialog box that shows a number of items. An example is shown in figure 3. The example code creates a list dialog by defining the “type” field to be EEikCtListBox in the relevant structure defined in the resource file.

The contents of the list box are stored in a descriptor array and the method SetItemTextArray() is used to set the array that will be displayed in the list box. Note that a call needs to be made to HandleItemAdditionL() or HandleItemRemovalL() if there are items added or removed from the list. This ensures the list is updated when the list has changed. After calling the remove or addition methods a call to DrawNow() is made in the refresh method to ensure the list box is drawn or re-drawn. This ensures the data is displayed correctly.

The method CreateScrollBarFrameL() is used to create a scroll bar. The method SetScrollBarVisibilityL() is used to set up visibility of the scroll bar; EAuto can be set to ensure the scroll bar is always shown when required.

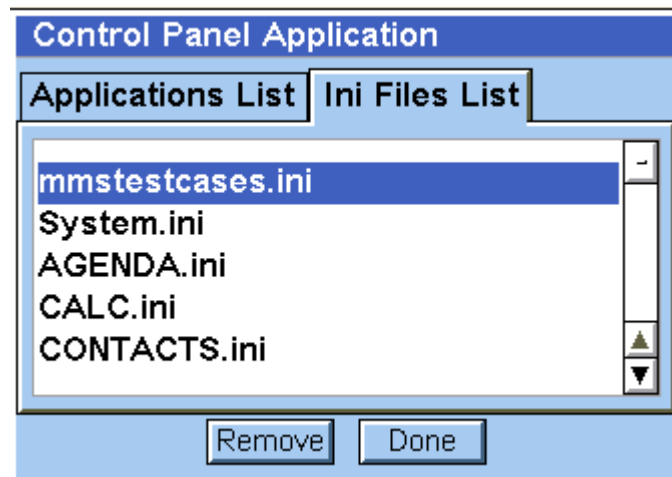


Figure 3

Creating a column dialog

The list dialog using columns is created similarly to the standard list dialog (some extra settings are required). The two main differences are setting the “type” field in the resource file to be `EikCtCollListBox`, and setting the column data using the class `CColumnListBoxData` in the `PreLayoutDynInitL()` method. The column data is obtained from the dialog’s model and is used to set font type, column width in pixels and alignment of the text in the column. The code snippet below shows how this is obtained and how the data is set using methods such as `SetColumnFontL()`:

```
iColumnData = appsListBox->ItemDrawer()->ColumnData();
iColumnData->SetColumnFontL(KPathColumn,iEikonEnv->NormalFont());
```

The `SetColumnFontL()` method takes the argument `KPathColumn` to specify which column the font is being set for.

The columns placement is set inside the array holding the list items. A constant `KColumnListSeparator` is inserted between the data in the descriptor you wish to display to define where the column will go. This extract of code shows how this is done in the example code:

```
TFileName fileName = parse.NameAndExt();
// Append the column separator for the list box
fileName.Append(KColumnListSeparator);
fileName.Append(info1.iUid.Name());
iAppsArray->AppendL(fileName);
```

This tells the list box where you would like the next column to start. The width of the columns is defined using the `SetColumnWidthPixelL()` method. The example code calculates the maximum size of all entries and gives the list box this value. Figure 4 below shows the dialog using columns for the different data.

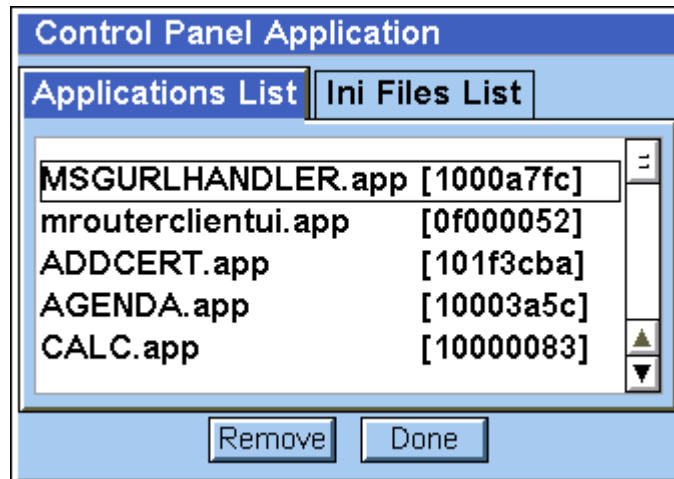


Figure 4

Pre and Post layout initialisation

The `PreLayoutDynInitL()` and `PostLayoutDynInitL()` layout methods are called before and after the dialog creates the layout and should be used to initialize data for the dialog when it is available. It is often only necessary to implement the pre layout method but both are implemented here as an example.

The following code shows the `PreLayoutDynInitL()` and `PostLayoutDynInitL()` methods. In the pre layout method the dialog data is initialized, the post layout method simply refreshes the contents of the dialog box.

```
void CControlPanelDialog::PreLayoutDynInitL()
{
    // Do pre drawing initialization
    InitializeAppsListboxL();
    InitializeInisListboxL();
}

void CControlPanelDialog::PostLayoutDynInitL()
{
    // Do post drawing initialization - just refreshes the list box
    RefreshAppsListboxL(EItemAdded);
}
```

Using controls and catching events

The `Control()` method is used get a reference to a list box based on its id (as defined in the `hrh` file). This code snippet shows how this is done:

```
CEikColumnListBox* appsListBox= static_cast <CEikColumnListBox*>
(Control(EAppsListBox));
```

In the example a pointer to the list box dialog is passed back allowing the list's array to be updated. The controls id is passed in as an argument.

The most important event to be caught is in the method `OkToExitL()`. This is called when one of the buttons in the dialog is pressed and the value that is returned defines whether the dialog closes or not. In this example the "Remove" button should not invoke an exit since its function is to remove an entry from the dialog's list. Returning `EFalse` ensures the dialog is not exited, returning `ETrue` causes the dialog to exit.

The `PrepareForcedExitL()` method can also be used to catch the event when dialog is forced to close due to an error occurring. This can be useful for saving important data in the event of an unexpected exit from the dialog.

The example code implements some of the virtual methods available for catching events. One such example is `PageChangedL(TInt aPagelId)` which is called when the user has switched to a different page. The `aPagelId` parameter references the page id assigned to a page in the resource file. Each page in the dialog has a page id associated to it. In the example code below this method is used to keep track of what page is currently selected.

```
void CControlPanelDialog::PageChangedL(TInt aPagelId)
{
    // Keep track of what page is currently being displayed
    if(aPagelId == EAppsListPage)
    {
        iPageNumber = 0;
        RefreshAppsListBoxL(EItemAdded);
    }
    else if(aPagelId == EInisListPage)
    {
        iPageNumber = 1;
        RefreshAppsListBoxL(EItemAdded);
    }
}
```

The observer `MEikListBoxObserver`, is set using the method `SetListBoxObserver`. Its method `HandleListBoxEventL` is called when an event happens in either of the list boxes. Events could be key-presses or when the selection in a list box changes. Some of the possible events are caught in the example code.

```
void CControlPanelDialog::HandleListBoxEventL
(CEikListBox* /*aListBox*/, TListBoxEvent aEventType)
{
    // catch events from the list boxes
    switch(aEventType)
    {
        case EEventEnterKeyPressed:
            RDebug::Print(_L("Key Press"));
            break; /** Keypress event. */
        case EEventItemClicked:
            RDebug::Print(_L("Click"));
            break; /** Item single-tap event. */
        case EEventItemDoubleClicked:
            RDebug::Print(_L("Double Click"));
            break; /** Item two-taps event. */
        case EEventSelectionChanged:
            RDebug::Print(_L("Selection Changed"));
            break;
        default: /** Selection in list changed */
            RDebug::Print(_L("Default"));
            break;
    }
}
```

In the above code it is possible to detect events such as double clicks using the enumeration values from the `TListBoxEvent` class.

The virtual method `HandleControlStateChangeL()` is implemented in this example in order to catch changes in the list box for example when the user has selected an element in the list. The argument `aControlId` gives the id of the control where the event originates. These ids are stored in the `.hrh` file.

Summary

This white paper has shown how to implement a simple control panel application. It also shows the fundamentals behind creating simple dialogs. The white paper has tried to show best practices in creating dialogs.

Appendix A – Porting the application to UIQ

The instructions for porting this application to UIQ are surprisingly simple.

- In the file `ControlPanelApplication.h` change the line:
`#include <eikdial.h>`
to be
`#include <eikdialg.h>`
- In the file `ControlPanelApplication.mmp` remove or comment out the line:
`SYSTEMINCLUDE \epoc32\include\techview`
- Build the code as normal using the commands
`Bldmake Bldfiles`
`Abld build winscw udeb`

Acknowledgements

The author wishes to thank the many Symbian staff who supplied valuable information and review comments.

Symbian licenses, develops and supports Symbian OS, the platform for next-generation data-enabled mobile phones. Symbian is headquartered in London, with offices worldwide. For more information see the Symbian website, <http://www.symbian.com/>.

Trademarks and copyright

'Symbian', 'Symbian OS' and other associated Symbian marks are all trademarks of Symbian Ltd. Symbian acknowledges the trademark rights of all third parties referred to in this material. © Copyright Symbian Ltd 2003. All rights reserved. No part of this material may be reproduced without the express written permission of Symbian Ltd.

Disclaimer

Symbian Ltd makes no warranty or guarantee about the suitability or accuracy of the information contained in this document. The information contained in this document is for general information purposes only and should not be used or relied upon for any other purpose whatsoever.

[Back to Developer Library](#)

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.

