

# Migrating to S60 5th Edition

EMCC Software Ltd

Published by the Symbian Developer Network

Version: 1.0 – April 2009

<b>1</b>	<b>INTRODUCTION</b> .....	<b>2</b>
<b>2</b>	<b>MIGRATING FROM S60 3RD EDITION TO S60 5TH EDITION</b> .....	<b>2</b>
	2.1 NEW FEATURES IN S60 5TH EDITION .....	2
	2.2 APPLICATIONS.....	4
	2.3 SYMBIAN OS V9.4 .....	4
	2.4 BINARY COMPATIBILITY ISSUES .....	5
<b>3</b>	<b>MIGRATING FROM UIQ TO S60</b> .....	<b>5</b>
	3.1 OVERVIEW .....	5
	3.2 ARCHITECTURE .....	6
	3.3 S60 MENUS VERSUS UIQ COMMANDS .....	8
	3.4 COMPOUND CONTROLS .....	10
	3.5 COMMON CONTROLS .....	10
	3.6 LOCALIZATION IN S60 .....	11
	3.7 RESOURCE FILES .....	11
<b>4</b>	<b>S60 5TH EDITION TOUCH UI</b> .....	<b>12</b>
	4.1 IMPLEMENTING HANDLEPOINTEREVENTL() IN CUSTOM CONTROLS .....	13
	4.2 DRAG EVENTS .....	14
	4.3 ADDING SUPPORT FOR LAYOUT CHANGE EVENTS .....	15
	4.4 SCROLLBARS .....	16
	4.5 ADDING SUPPORT TO EXISTING LIST BOXES AND GRIDS .....	17
	4.6 GENERIC BUTTON API.....	17
	4.7 TOOLBAR API.....	18
	4.8 STYLUS POP-UP MENU API .....	19
	4.9 CHOICE LIST API.....	20
	4.10 STATUS PANE API.....	20
	<b>APPENDIX A</b> .....	<b>23</b>

# 1 Introduction

This paper focuses on the migration of your existing applications to S60 5th Edition. It is directed at two audiences, with experience in different areas:

1. those wishing to port C++ applications from UIQ 3 to S60 5th Edition
2. those wishing to port C++ applications from S60 3rd Edition to S60 5th Edition.

Audience 1 is considered to be the group with the more complicated task and as such will be treated preferentially with respect to content. However, since S60 5th Edition is mostly backwards compatible, S60 3rd Edition code will largely compile and run correctly in an S60 5th Edition environment. Therefore, the main content for audience 2 will be the new Touch UI and the changes necessary to take full advantage of the new features of the environment.

The paper comprises three main sections:

- Section 2, 'Migrating from S60 3rd Edition to S60 5th Edition', provides an overview of the major changes and enhancements of S60 5th Edition with a focus on developers.
- Section 3, 'Migrating from UIQ to S60', provides a detailed look at the major differences between UIQ and S60 development and covers topics such as S60 Menus vs. UIQ Commands, migrating from UIQ Building Blocks to S60 Compound Controls and Common S60/UIQ System Controls.
- Section 4, 'S60 5th Edition Touch UI', provides a detailed look at the new functionality and APIs that are available to C++ developers for adding Touch UI capabilities to their applications.

References to S60 3rd Edition should be taken to mean S60 3rd Edition Feature Pack 2 unless otherwise stated. Links to online documentation have been made to the UIQ 3.1 Developer's Library and to version 1.1 of the S60 5th Edition C++ Developer's Library.

This paper contains code snippets for illustrative purposes only. Code snippets cannot be guaranteed to be defect free and may not be suitable for commercial quality software.

## 2 Migrating from S60 3rd Edition to S60 5th Edition

This section provides a developer-focussed overview of the major changes and enhancements of S60 5th Edition. The focus is on new APIs, frameworks and important applications available in S60 5th Edition that replace previous functionality. In many cases, details will not be included here as they can be found later on in this document or in the online S60 5th Edition C++ Developer's Library<sup>1</sup>, in which case the relevant section of the documentation will be pointed out.

### 2.1 New features in S60 5th Edition

#### 2.1.1 Touch UI

With the introduction of S60 5th Edition, S60 now supports user interaction via a number of input mechanisms:

- stylus or finger touch
- on-screen keyboard (accessed by stylus or finger touch);
- traditional softkey and keypad.

---

<sup>1</sup> The online version of the S60 5<sup>th</sup> Edition C++ Developer's Library can be found at [library.forum.nokia.com](http://library.forum.nokia.com).

This ability to interact via stylus or finger touch represents the most significant enhancement in S60 5th Edition compared to previous editions. As such, the Touch UI and associated APIs that are used to give an application touch input capabilities are covered in detail in Section 4.

## 2.1.2 Standard C/C++ library support

Open C/C++<sup>2</sup> is available as an add-on to S60 3rd Edition, but comes as a standard part of S60 5th Edition. It includes various standard C and C++ APIs that allow portable, platform-independent C and C++ code to be written.

### 2.1.2.1 Open C

Open C consists of a number of libraries giving access to standard C APIs, such as the C Standard Library, POSIX C APIs for communications, threads, etc., as well as APIs for Cryptography and Secure Sockets Layer (SSL) communication. It fulfils a similar role to [P.I.P.S.](#) ('P.I.P.S. Is POSIX on Symbian OS'), but covers a greater range of APIs and is S60-specific. There are some [limitations](#) that should be taken into account before porting an application to S60 using Open C. For example, `fork()` and `exec()` are not supported at the time of writing.

### 2.1.2.2 Open C++

Open C++ is made up of libraries that give access to Standard C++ APIs, allowing developers to write easily-portable code. The following libraries are included:

1. IOStreams, which is the C++ counterpart to `stdio.h` and allows input and output via streams using `istream`, `ostream`, `istream`, `cout`, `cin`, `fstream`.
2. The Standard Template Library (STL), which provides a rich set of generic C++ components, consisting of algorithms, containers, iterators, function objects, memory allocation classes and adaptors.
3. The Boost libraries, which build upon the STL to provide further portable C++ components. The S60 Open C++ implementation contains Containers, Math and Smart Pointer libraries.

It should be noted that at the time of writing, the real-time graphics and audio (RGA) APIs mentioned here, do not form part of the standard S60 5th Edition SDK and there is no add-on available.

For more information about Open C/C++, these links should prove useful:

- [an overview of Open C/C++ and where they fit into the S60 framework](#)
- [a detailed overview of what features are supported in Open C/C++](#)
- [online S60 5th Edition SDK Documentation covering Open C](#)
- [online S60 5th Edition SDK Documentation covering Open C++](#)
- online S60 5th Edition SDK Documentation covering the known limitations of the Open C/C++ libraries:
  - [limitations of Open C](#)
  - [limitations of Standard C++ Library \(IOStream and STL\)](#)
  - [limitations of Open C++ Boost Libraries.](#)

---

<sup>2</sup> Summarized at [www.forum.nokia.com/Resources\\_and\\_Information/Tools/Runtimes/C++/Open\\_C\\_and\\_C++\\_Plug-ins\\_for\\_S60\\_3rd\\_Edition/Features.xhtml](http://www.forum.nokia.com/Resources_and_Information/Tools/Runtimes/C++/Open_C_and_C++_Plug-ins_for_S60_3rd_Edition/Features.xhtml).

### 2.1.3 New C++ APIs

A number of new APIs are available on S60 5th Edition and a summary of the new C++ APIs can be found [here](#). The following list contains all new C++ APIs along with a link, if available, to the relevant location in the online S60 5th Edition Developer's Library:

- [Accessory Monitoring API](#)
- [Choice List API](#)
- [Generic Button API](#)
- [Hierarchical Lists API](#)
- [Hostlet Connection API](#)
- [Incoming Call Monitor API](#)
- [Messaging Integration API](#)
- [Sensor APIs](#)
- [Stylus Pop-up Menu API](#)
- [Tactile Feedback Client API](#)
- [Title Pane Touch Observer API](#)
- [Toolbar API](#)
- [Touch UI Utilities API](#)
- [Web Service Messaging API](#)
- [WLAN SDK Info API](#)
- [XML Engine DOM API](#)
- [XML Fragment API](#)

## 2.2 Applications

Appendix A contains a table of applications that are available on S60 5th Edition and their equivalents, where available, on S60 3rd Edition and UIQ 3. All applications have been updated to support Touch interaction and certain applications such as Gallery have had more significant changes made to them.

The role of the Control Panel has been expanded and it can be seen in Appendix A that a number of items that were previously standalone applications, such as Themes and Profiles, are now found in the Control Panel. In addition, certain applications' settings screens, such as RealPlayer, Calendar and Messaging, can be accessed via the Control Panel<sup>3</sup> as well as from within the applications themselves.

## 2.3 Symbian OS v9.4

S60 5th Edition is based on Symbian OS v9.4, whereas S60 3rd Edition FP2 is based on Symbian OS v9.3. As a result, S60 5th Edition benefits from the new features of Symbian OS v9.4. These include:

---

<sup>3</sup> These can be found in Control Panel\Phone\Application Settings.

- improvements to the performance of the Kernel and File System
- performance improvements for the SQL Server that was introduced in Symbian OS v9.3
- support for Multimedia Transfer Protocol (MTP) over USB.

Further details of what is available in Symbian OS v9.4 can be found in the specification sheet, at [www.symbian.com/files/rx/file9468.pdf](http://www.symbian.com/files/rx/file9468.pdf).

## 2.4 Binary compatibility issues

Although the standard S60 UI components have touch support built-in on 5th Edition, S60 3rd Edition applications that use custom controls or that rely on particular hardware keys may need to be updated to be compatible with all touch devices. Other applications should not pose significant migration problems except for any relevant binary compatibility issues. The S60 5th Edition SDK Documentation covers Binary Compatibility changes between S60 3rd Edition Feature Pack 2 and 5th Edition [here](#). Recompilation or re-linking of 3rd Edition code may be necessary for applications that use the affected APIs.

## 3 Migrating from UIQ to S60

This section discusses migration of existing UIQ 3.x user interfaces to S60 5th Edition.

### 3.1 Overview

UIQ and S60 implement different interaction philosophies. UIQ devices tend to focus on stylus-based or touch operation whereas S60 originally concentrated on ease of one-handed operation on devices that have hardware keys but no touch support. The UI layer in S60 5th Edition introduces touch functionality but retains full support for hardware keys, increasing the range of device types which S60 can support effectively and consequently increasing the range of devices that applications need to work on.

S60 tends to emphasize softkey-controlled options and menus. Since S60 3rd Edition FP2, up to three softkeys can be used. The softkeys can be triggered by hardware keys or by touch. One softkey is usually dedicated to negation (e.g., 'Back', 'Cancel', or 'Exit'), one to the available positive action or actions in the current view and a third to showing the current action or selection. When using hardware keys, the user controls the focus and selection of UI elements using a five-way navigation key; on a touch-based device, the user can tap controls on the display to shift focus and select items. S60 touch, with some exceptions, acts similarly to the navigation key from the user's point of view. A tap on a control selects it, and a second tap activates the action it represents.

On UIQ-based platforms, the user should never have to explicitly close an application; applications save their state when sent to the background and are closed as needed by the system depending on memory usage. On S60 the user decides when to close applications and so all applications must have an 'Exit' option.

Some important points to remember when designing an S60 5th Edition-compatible user interface are that:

- It must be possible for the user to use all application features through touch. S60 5th Edition touch devices are not required to implement most of the hardware keys found on non-touch S60 devices.

- The user should not have to switch between interaction methods (between touch and hardware keys) to complete an action.

More information on UI design considerations for S60 5th Edition applications can be found in Nokia's [Design and User Experience Library](#).

## 3.2 Architecture

The AVKON (S60) and QIKON (UIQ) UI layers start from the common UIKON architecture defined by Symbian OS. Both support effective user interfaces based around the Model-View-Controller (MVC) pattern in a set-up which allows easy switching between views offered by different applications or between different views in the same application. For AVKON, details about the supported architectures are available from the S60 5th Edition documentation on [UI architectures](#). S60 view-based architecture should be used for applications that need to support launching from external views, while the traditional view architecture most resembles the architecture used in UIQ3 and can be used for applications that do not need this facility.

Out of the architectures supported on S60, single view-based architecture is most similar to the architecture used in UIQ3, as the 'traditional' architecture does not implement the `MCoeView` concept.

Outline class diagrams for AVKON and QIKON are shown in Figure 1 and Figure 2 respectively, omitting incidental classes to maintain clarity. The AVKON and QIKON Application and Document classes are very similar. For minimal implementations almost no porting work is needed aside from deriving from the appropriate base classes for the UI layer being used. The frameworks really start to diverge at the UI controller level.

AVKON defines two UI controller base classes: `CAknViewAppUi`, which is suitable for S60 view-based applications, and `CAknAppUi`, which is used for the other supported UI architectures. `CAknViewAppUi` delegates some controller responsibilities to 'view controllers' derived from `CAknView`, such as the handling of view-specific menu options. In contrast, QIKON has `CQikAppUi` as its only UI controller base. The AppUIs are responsible for creating the program views on both platforms. In AVKON in particular, derived AppUI classes are also responsible for deleting the views they create, whilst in QIKON ownership of views is handed over to the base class.

The frameworks differ somewhat in their implementation of the View concept (defined by `CONE` in the `MCoeView` class). QIKON applications generally derive their view classes from `CQikViewBase` or `CQikMultiPageViewBase`, which are both derived from `CQikContainer` (which is a `CCoeControl`) as well as `MCoeView`. QIKON views are therefore control containers and can manage and lay out their child controls themselves. In AVKON the view class (`CAknView`) is a controller (known in S60 parlance as a view controller) and is not derived from `CCoeControl`. Usually on S60, a separate `CCoeControl`-derived container class must be implemented to own and lay out the view controls. Implementation of container classes for S60 5th Edition is discussed further in Section 3.4 of this paper, 'Compound controls'.

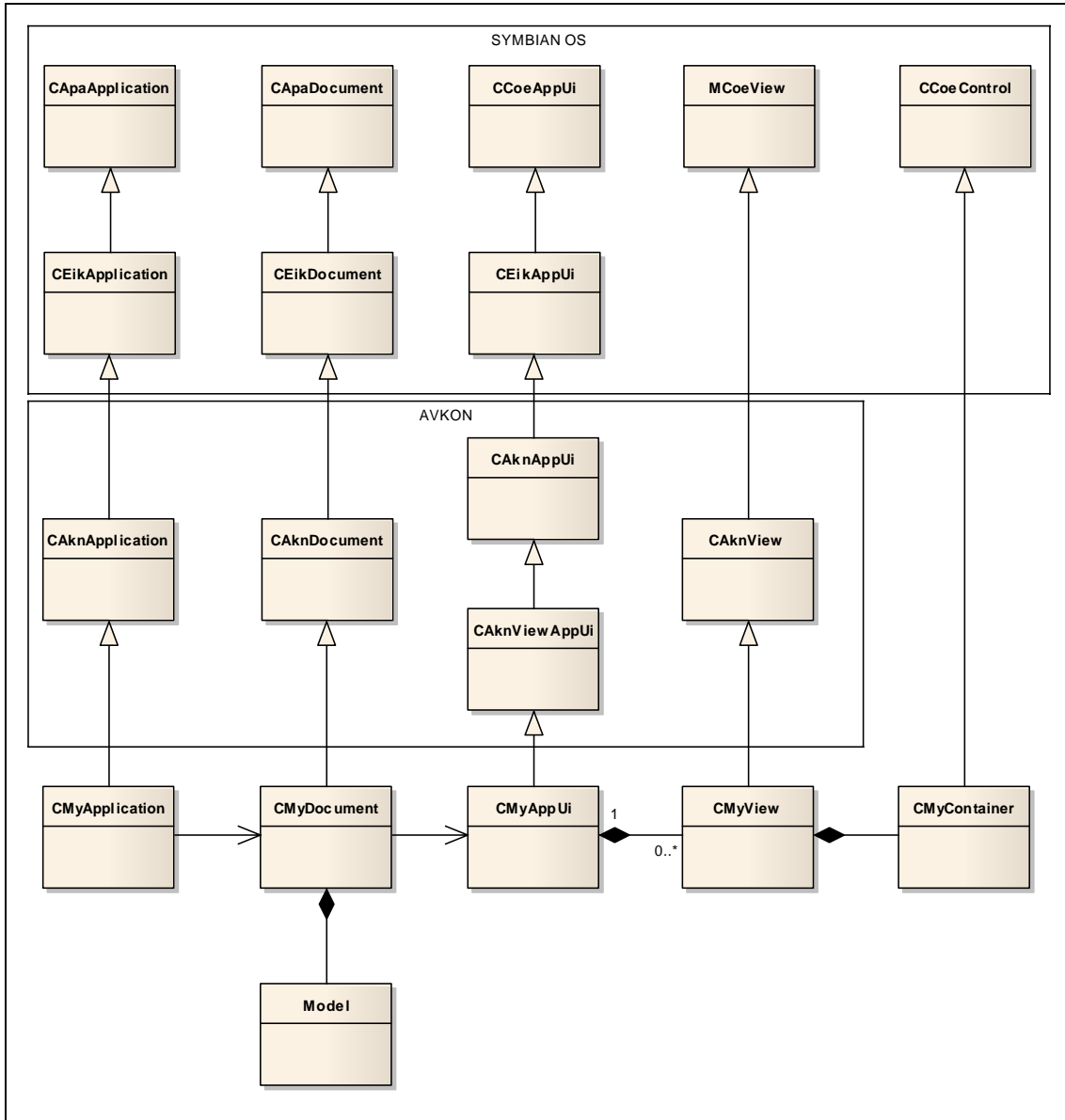


Figure 1: AVKON/S60 UI architecture overview

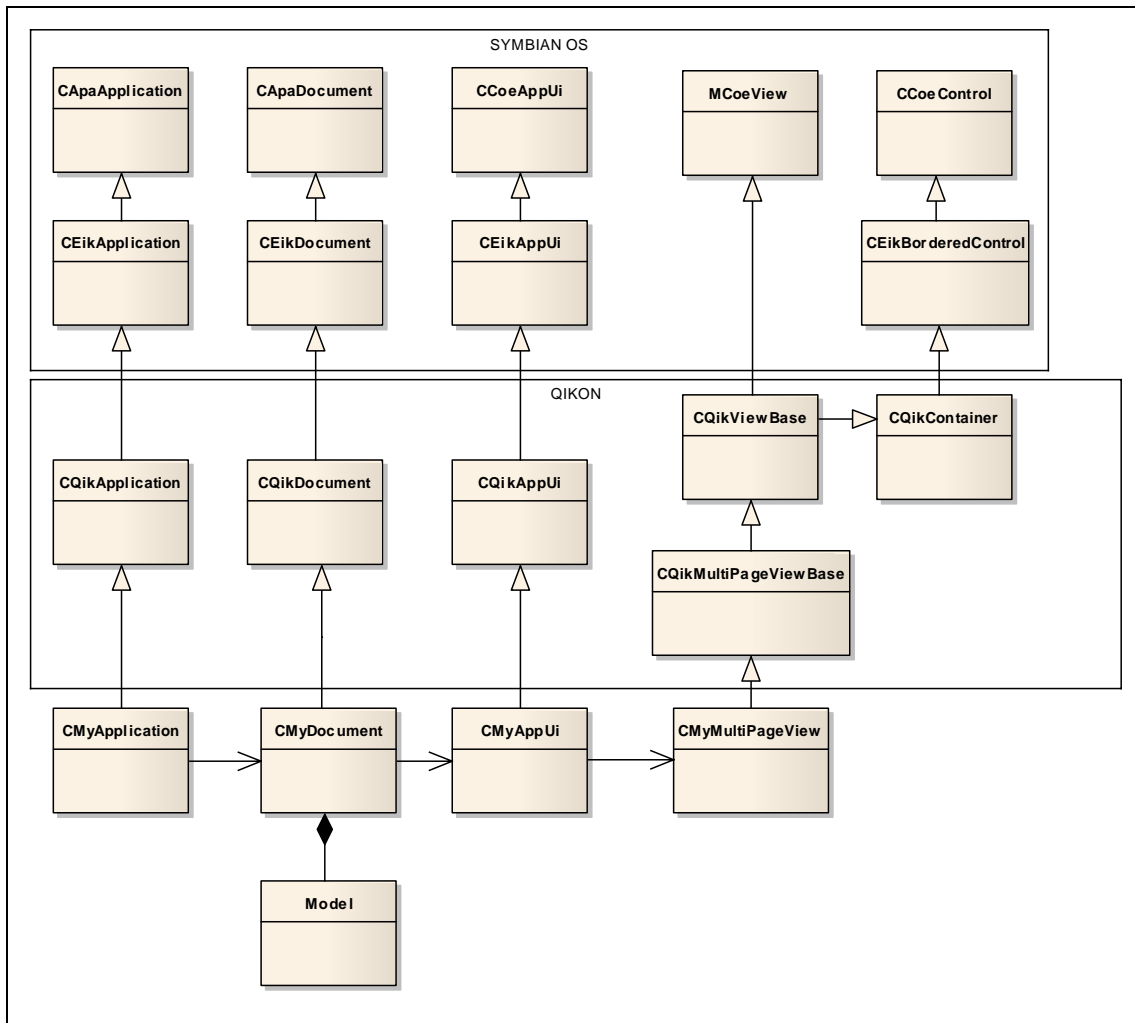


Figure 2: QIKON/UIQ UI architecture overview

QIKON uses a relatively consistent source-level representation to pass information to a destination view during view switching, typically consisting of a T class instance carrying view-specific information that can be wrapped in a descriptor and sent through `ActivateViewL()`. For example, `TQWebDNLURLEntry` can be used to pass a URL to be opened over to the web browser.

On S60, individual applications may accept context information for views they supply but the method for passing the context information can vary somewhat. The context information accepted by the various S60 system applications is described in [this resource](#).

### 3.3 S60 menus versus UIQ commands

UIQ applications offer optional actions to the user through generic command objects (instances of `CQikCommand`), which are typically defined in resources. The UIQ framework takes care of the detailed placement of commands, mapping them to softkeys, command buttons, menu options or hardware keys.

In S60, the programmer has direct control over the placement of actions on the UI; there is no common representation in the style of `CQikCommand` and the `QIK_COMMAND` resource. The appropriate APIs or resource definitions must be used to place a command on the toolbar or on

generic buttons (especially for touch devices) or under softkey menus. The implementation of each is somewhat different but Table shows the conversion of a simple set of UIQ commands to S60 menu items that could be used in a softkey-triggered Options menu. Note the addition of a menu item to close the application.

Generic buttons and toolbars are described further in Sections 4.6 and 4.7 respectively.

UIQ 3 Command Resource	Equivalent S60 5th Edition Menu Resource
<pre>#i ncl ude "myAppCommands. hrh"  RESOURCE QI K_COMMAND_LI ST r_my_cmds {   i tems =   {     QI K_COMMAND     {       i d = EMyExamp l eCommand1;       type = EQi kCommandTypeScreen;       text = STRI NG_Cmd1;       shortText = STRI NG_Short_Cmd1;     },     QI K_COMMAND     {       i d = EMyExamp l eCommand2;       type = EQi kCommandTypeScreen;       text = STRI NG_Cmd2;       shortText = STRI NG_Short_Cmd2;     }   } }; }</pre>	<pre>#i ncl ude "myAppCommands. hrh"  RESOURCE MENU_BAR r_my_menubar {   ti tles =   {     MENU_TI TLE     {       menu_pane = r_my_menupane;     }   }; }  RESOURCE MENU_PANE r_my_menupane {   i tems =   {     MENU_I TEM     {       command = EMyExamp l eCommand1;       txt = STRI NG_Cmd1;     },     MENU_I TEM     {       command = EMyExamp l eCommand2;       txt = STRI NG_Cmd2;     },     MENU_I TEM     {       command = EEi kCmdExi t;       txt = STRI NG_CI ose;     }   } }; }</pre>

**Table 1: Converting a UIQ 3.x command list to an S60 menu**

Commands are provided to your application through its implementation of `HandleCommandL()` in the `AppUI`.

### 3.4 Compound controls

UIQ provides building blocks and layout managers for arranging controls on the display. The building blocks break a view or part of a view down into groups of controls, and layout managers organise the controls within the building blocks and the building blocks within the view. [Predefined building blocks](#) make preparation of complex, self-consistent view layouts straightforward.

S60 leaves layout functionality in the hands of the developer; container controls must explicitly lay out any child controls themselves without the assistance of layout managers and containers themselves must be implemented by the developer. When implementing such containers the following points should be borne in mind:

- For efficiency reasons, your top-level container should usually own a window for use by its child controls. Calling `CCoeControl::CreateWindowL()` in your constructor will manage this. Your container's constructor should, as for any custom control, set its size and call `CCoeControl::ActivateL()` at the end of construction. Child controls should be made lodgers by calling `CCoeControl::SetContainerWindowL()` on them during construction.
- Re-implement `CCoeControl::CountComponentControls()` and `CCoeControl::ComponentControl()` to return the number of contained controls and a pointer to a selected child control respectively.
- As with any custom control in S60, you should re-implement `CCoeControl::SizeChanged()` to size the child controls and `CCoeControl::HandleResourceChangeL()` to pick up changes in view orientation and pass them on.
- The container should observe its child controls using `CCoeControl::SetObserver()` and implement `HandleControlEventL()` to act on the resulting event notifications.

S60 5th Edition provides some compound controls to meet common requirements, such as the [Form API](#) and the [Setting Pages API](#). The details of a particular container implementation will depend on the desired layout and purpose of the view. You can find other built-in compound controls in the [Classic UI services](#) list.

### 3.5 Common controls

This section lists some common types of UI components together with the UIQ and S60 5th Edition APIs that implement them. Further information on particular APIs should be obtained from the [UIQ 3.1 SDK documentation](#) and the [S60 5th Edition Developer's Library](#).

Type of Control	API (UIQ 3.x)	API (S60 5th Edition)
Choice list	CEikChoiceList	<a href="#">Choice List API</a>
Command button	CEikCommandButton	<a href="#">Generic Button API</a>
Custom dialog support	CQikSimpleDialog	<a href="#">Dialogs API</a>
Editors	CQikDateEditor CEikEdwin CEikRichTextEditor	<a href="#">Editors API</a>
Listbox	CQikListBox	<a href="#">Lists API</a>

Type of Control	API (UIQ 3.x)	API (S60 5th Edition)
Notification dialogs	CEi kEnv: : I nfoWi nL() CQi kSi mpl eDi al og	Notes API
Progress information display	CEi kProgressI nfo (bar control)	Progress note (dialog)
Slider control	CQi kSI i der	CAknSI i der Note that there is a dedicated Volume Control API
Text label	CEi kLabel	Labels API (including CEi kLabel )
Toolbar	CQi kTool bar	Toolbar API

### 3.6 Localization in S60

Like UIQ, S60 has support for localisation through files containing Resource Localizable Strings (RLS). The S60 build tools do support an alternative naming convention for RLS files, <appl i cati on\_name>. l xy, where xy is the two-digit language code, but otherwise the localization processes are the same on both platforms.

### 3.7 Resource files

Due to the differences in architecture described earlier, S60 and UIQ also differ in their use of resource files. A typical UIQ resource file will define the application's supported view configurations, which indicate the supported hardware configurations (pen or softkey, portrait or landscape) and the view and command list used for each. It will then go on to list the pages (tabs) associated with each view and then the controls for each page. Here is a comparison of the general structure of S60 and UIQ 3.x resource files:

UIQ 3.x Application Resource File	S60 5th Edition Application Resource File
<pre> /*   Standard QI KON resource #i ncl udes */  NAME EXMP RESOURCE RSS_SI GNA TURE { } RESOURCE TBUF { buf = ""; } RESOURCE EI K_APP_I NFO { }  /* QI K_COMMAND_LI ST (command defi ni ti ons)  QI K_VI EW_CONFI GURATI ONS (supported vi ew confi gurati ons) </pre>	<pre> /*   Standard AVKON resource #i ncl udes */  NAME EXMP RESOURCE RSS_SI GNA TURE { } RESOURCE TBUF { buf = ""; } RESOURCE EI K_APP_I NFO { }  #i ncl ude "my_l ocal i sed_stri ngs. l oc"  /* MENU_BAR, MENU_PANE (define menus for the appl i cati on) */ </pre>

UIQ 3.x Application Resource File	S60 5th Edition Application Resource File
<pre> */  /* Define the view in terms of its pages (containers) */ RESOURCE QIK_VI EW r_my_vie w_l ayout { pages = r_my_vie w_l ayout_pages; }  /* Defini ti ons of pages using QIK_CONTAI NER resources fol low */  /* Defini ti ons of child control s fol low */ </pre>	<pre> /* Define the view in terms of the commands it supports and its softkeys */ RESOURCE AVKON_VI EW r_my_vie w { menubar = r_my_menubar; cba = R_AVKON_SOFTKEYS_OPTI ONS_EXI T; }  /* Defini ti ons of resources for contai ners and child control s fol low */ </pre>

In S60, commands are replaced by menu items or softkey labels as described in Section 3.3 of this paper. Views for the application are defined using the AVKON\_VI EW resource structure. Applications construct views using these resources by calling `CAknView::BaseConstructL()` in the second-phase constructor of the derived view class:

```

#include <myApp.rsg>

void CMyView::ConstructL()
{
BaseConstructL(R_MY_VI EW);
}

```

The call identifies the resource to use to construct the view.

## 4 S60 5th Edition Touch UI

S60 5th Edition adds support for touch interaction. Framework-provided standard UI controls have been modified so that, generally, code utilizing them does not need to be changed to support such interaction. However, custom UI controls coded for previous S60 editions will normally need to be modified to receive and process touch events. Touch devices are also not required to have the complement of hardware keys that a non-touch device has, so code that relies on trapping those keys may never be called on touch devices. The following sections explain what changes need to be made to custom controls to ensure touch compatibility, and outline some of the new UI features that can be used to enhance applications in moving to S60 5th Edition.

## 4.1 Implementing `HandlePointerEventL()` in custom controls

Custom UI controls that are intended to support touch events in S60 5th Edition should override the virtual `HandlePointerEventL()` method inherited from `CCoeControl`. Custom implementations in container controls should call `CCoeControl::HandlePointerEventL()` to ensure that pointer events are passed on to any child controls.

In this container's `HandlePointerEventL()`, the base class implementation is called to pass on the events to child controls:

```
void CMyContainer::HandlePointerEventL(const TPointerEvent& aEvent)
{
    // Pass on the event to the correct child control.
    CCoeControl::HandlePointerEventL(aEvent);

    // Any additional logic
}
```

If no additional logic is required (e.g., focus control and scrollbar management) then it may not be necessary to override `HandlePointerEventL()` in the container.

A child control should have its own implementation of `HandlePointerEventL()`:

```
void CMyChildControl::HandlePointerEventL(const TPointerEvent& aEvent)
{
    switch(aEvent.iType)
    {
        case TPointerEvent::EButton1Down:
        {
            // Note that iPosition gives the pointer event location
            // in coordinates relative to the window origin.
            TPoint eventStartPosition = aEvent.iPosition;
            // Do something with eventStartPosition.
            [...]
            break;
        }

        case TPointerEvent::EButton1Up:
        {
            // iParentPosition is given relative to the parent window origin.
            TPoint eventStopPosition(aEvent.iParentPosition);
            // Do something with eventStopPosition.
            [...]
            break;
        }

        /* Handle other cases */
    }
}
```

Also see [Handling pointer events in custom controls](#) in the S60 5th Edition documentation.

## 4.2 Drag events

There are several different kinds of touch event that can be passed to a control but by default only the button up and button down events are passed by the window server. If you require drag events then you need to call `CCoeControl::EnableDragEvents()` in your view container's `ConstructL()`:

```
void CMyContainer::ConstructL(const TRect& aRect)
{
    // Create the top-level window.
    CreateWindowL();
    // Call to receive advanced pointer events in HandlePointerEventL.
    EnableDragEvents();

    /* Construct child controls here */

    // Size and activate the container.
    SetRect(aRect);
    ActivateL();
}
```

If a window-owning class calls `CCoeControl::EnableDragEvents()`, it and its child controls can receive them through `HandlePointerEventL()`. By handling the `TPointerEvent::EDrag` case, your application can implement sophisticated user actions such as dragging UI elements or responding to gestures. The following snippet sketches a possible way to capture the motion of the stylus or finger while pressed against the touchscreen. An instance of [CAknPointerEventSuppressor](#), `iEventSuppressor`, is used to keep the number of pointer locations stored manageable by suppressing events using distance moved and time interval criteria, while `iMyArrayOfPointerLocations` is of an array class type such as `RArray<TPoint>`:

```
void CMyChildControl::HandlePointerEventL(const TPointerEvent& aEvent)
{
    // Use CAknPointerEventSuppressor to regulate the capture rate.
    // See the platform documentation for more information on
    // CAknPointerEventSuppressor.
    if(!iEventSuppressor->SuppressPointerEvent(aEvent))
    {
        // The event was not suppressed
        switch(aEvent.iType)
        {
            case TPointerEvent::EDrag:
            {
                // Save the stylus position when we receive a drag event.
                iMyArrayOfPointerLocations.AppendL(aEvent.iPosition);
                break;
            }

            case TPointerEvent::EButton1Down:
            {
                // Clear the array of positions ready for a new drag path.
                iMyArrayOfPointerLocations.Reset();
                ReportEventL(EEventStateChanged);
                break;
            }

            case TPointerEvent::EButton1Up:
```

```

        {
            // Add this last point to the drag path and notify observer.
            i MyArrayOfPoi nterLocati ons. AppendL(aEvent. i Posi ti on);
            ReportEventL(EEventStateChanged);
            break;
        }
    }
}

```

The SDK documentation discusses enabling drag events [here](#).

### 4.3 Adding support for layout change events

S60 devices are available with a variety of display sizes and orientations. Some devices can dynamically change the display orientation, or switch display to a secondary, smaller screen. S60 application UIs must therefore be scalable to multiple resolutions and must be able to respond to changes in active display orientation or size (layout change events) by dynamically laying out controls to fit the available space. Standard AVKON components all support scalable UI but custom controls and AppUIs may need to be modified to respond to layout changes correctly.

Layout change events are reported to your application through the `HandleResourceChangeL()` method of your AppUI as events with the `KEi kDynamicalLayoutVariantSwitch` ID. Similarly, `CCoeControl`-derived objects on the control stack have their `HandleResourceChange()` implementations called automatically by the framework:

```

void CMyAppUI::HandleResourceChangeL(TInt aEventType)
{
    CAknViewAppUI::HandleResourceChangeL(aEventType);

    if(aEventType == KEi kDynamicalLayoutVariantSwitch)
    {
        /*
         * Any special handling for your application-wide resources.
         */
    }
    /*
     * If your view's container has not been added to the control stack,
     * you will need to call HandleResourceChange on it manually.
     */
    //i MyViewContainer->HandleResourceChange(aEventType);
}

```

The container's `HandleResourceChange()` implementation resizes the control to fit the main pane:

```

void CMyContainerControl::HandleResourceChange(TInt aEventType)
{
    CCoeControl::HandleResourceChangeL(aEventType);
    if(aEventType == KEi kDynamicalLayoutVariantSwitch)
    {
        // Resize this container to fill the main pane.
        TRect rect;
        AknLayoutUtils::LayoutMetricsRect(AknLayoutUtils::EMainPane, rect);
        SetRect(rect);
    }
}

```

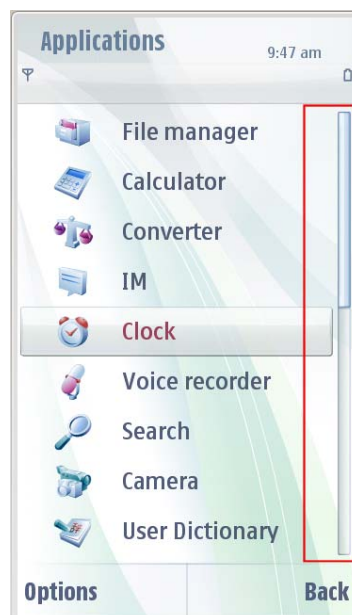
The call to `SetRect` will cause `CMyContainerControl::SizeChanged` to be called, and from there the container's child controls can be laid out to fit the new container size.

Supported display resolutions have changed between S60 3rd Edition and 5th Edition. The S60 emulator is supplied pre-configured with a list of these supported resolutions, and allows you to simulate switching between them and the different display orientations while your program is running. Note that it is possible to fix the display to a particular orientation in code; see the [relevant S60 Technical Solution](#). This technique may be useful if you have a view that must be displayed in fixed orientation, or if you wish to spend minimum time on porting a legacy UI; it is not recommended as a way to avoid dealing with scalable UI support and may impact the usability of your application. It may also have an impact on the [Symbian Signed](#) process for your application.

The S60 5th Edition platform documentation discusses this topic under [Implementing support for scalable UI](#).

## 4.4 Scrollbars

S60 scrollbars can be used directly through touch in S60 5th Edition. The graphical scrollbar frames are relatively narrow since they are only visual guides on non-touch devices. To make them easier to press on touch devices, the touch-sensitive area allocated to scrollbar frames is made larger than their graphical representation. This means that controls adjacent to scrollbars may not receive touch events if the user taps too close to the scrollbar. The SDK documentation discusses this further, together with other touch-specific issues, under the topic [Touch components](#).



**Figure 3: A scrollbar in the S60 emulator. The red rectangle indicates the approximate extent of the touch sensitive region allocated to the scrollbar**

To use scrollbars, your container control should create a scrollbar frame (`CEikScrollbarFrame`), which can be used to create the scrollbar instances (for example, of class `CAknDoubleSpanScrollbar`). Notifications of both touch and non-touch scrollbar events can be received through the `MEikScrollbarObserver` class.

Scrollbar-using custom controls written for S60 3rd Edition should already have implemented this observer interface in order to listen for non-touch scrollbar events. Since touch events are delivered through this interface in 5th Edition, only minor additions are necessary for existing code.

Further information on handling scrollbar touch events is available from the S60 5th Edition documentation in [Listening for scrollbar touch events](#).

## 4.5 Adding support to existing list boxes and grids

S60 5th Edition's touch support extends to the AVKON List and Grid APIs. Tapping on an item shifts the focus to it and tapping again causes AVKON to issue a `MEikListBoxObserver::EEventEnterKeyPressed` event. Existing code should already catch and process this event and therefore is unlikely to need modification. However, the S60 5th Edition documentation recommends that some modifications be made if you have a requirement to distinguish between key input and touch.

Passing the `EKnTouchCompatible` flag to `BaseConstructL()` in your `AppUI` causes the framework to generate the new event listbox observer event `EEventItemDoubleClicked` when a selected item is tapped:

```
void CMyAppUi::ConstructL()
{
    BaseConstructL(EKnTouchCompatible | EKnEnableSkin);
    [...]
}

void CMyContainer::HandleListBoxEventL(CEikListBox* aListBox, TListBoxEvent
aEventType)
{
    switch(aEventType)
    {
        case EEventItemDoubleClicked:
        {
            // User tapped on an item (touch event)
            // Handle touch-specific behaviour here.
            [...]
            break;
        }

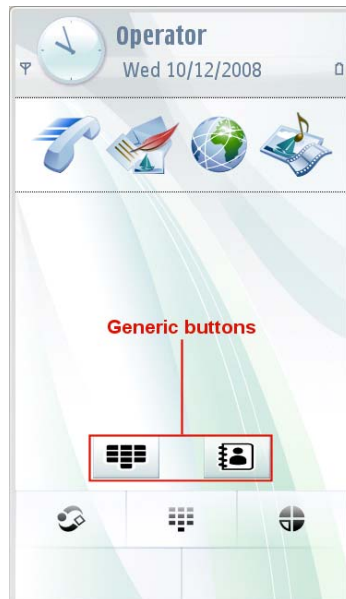
        case EEventEnterKeyPressed:
        {
            // User clicked an item (key event)
            // Handle non-touch behaviour here.
            [...]
            break;
        }
    }
}
```

The SDK documentation discusses this topic [here](#).

## 4.6 Generic Button API

S60 5th Edition introduces general touch-sensitive button controls. Generic buttons can be used directly to provide quick touch access to one or two application functions, or as part of a toolbar (see Section 4.7 for information on the Toolbar API, which itself makes use of generic buttons). S60 already supports a CBA button pane displayed above softkeys but, apart from being designed

for touch-based use, the Generic Button API also allows floating buttons that are independent of the softkeys and toolbar.



**Figure 4: Generic buttons on the idle screen**

Generic buttons can have a number of different states defined. For each state they can display a different icon, tooltip and flag values. They can also be disabled (dimmed).

Generic buttons can be configured from resources with the `AVKON_BUTTON` and `AVKON_BUTTON_STATE` structures or directly in code using the methods supported by `CAknButton`. Container controls should observe their child buttons and handle button events in their implementations of `MCoeControlObserver::HandleControlEventL()`. Further details are available in the S60 5th Edition documentation under [Generic button API](#).

## 4.7 Toolbar API

Important actions that a user may need to use regularly can be added to a toolbar for easy access. The toolbar is built on the generic button API, as shown in Figure 5.

Each view can have its own toolbar. Note that commands in the toolbar should always be accessible through the Options menu as well. The toolbar can be fixed, in which case it cannot have focus, or floating away from the CBA button pane, in which case it can receive focus and be used without touch support. The number of buttons displayed depends on display resolution and whether the toolbar is fixed or floating, but the minimum is three.

A toolbar is represented by an instance of `CAknToolBar`. Your view should implement `MAknToolBarObserver` to receive toolbar events. For more information on using the toolbar, including the API and user experience guidelines, see the [summary](#) provided by the S60 5th Edition platform documentation.

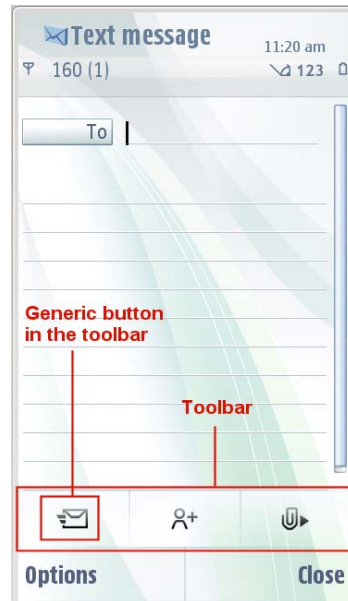


Figure 5: Toolbar in the S60 text message editor

#### 4.8 Stylus pop-up menu API

This API offers support for a pop-up menu that can be displayed in response to a touch event (often a long tap). An example use case would be to add a few options to a list view when the user taps on an item. The menu can display a limited number of options and does not scroll; it also does not dim the background view or use focus. The menu will disappear if the user taps outside it or picks an option. If left open, the menu will automatically close itself after a short interval.

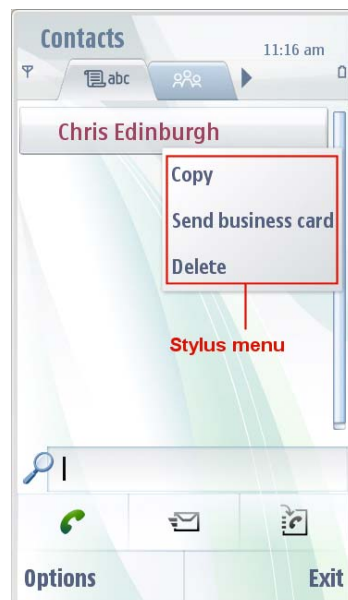
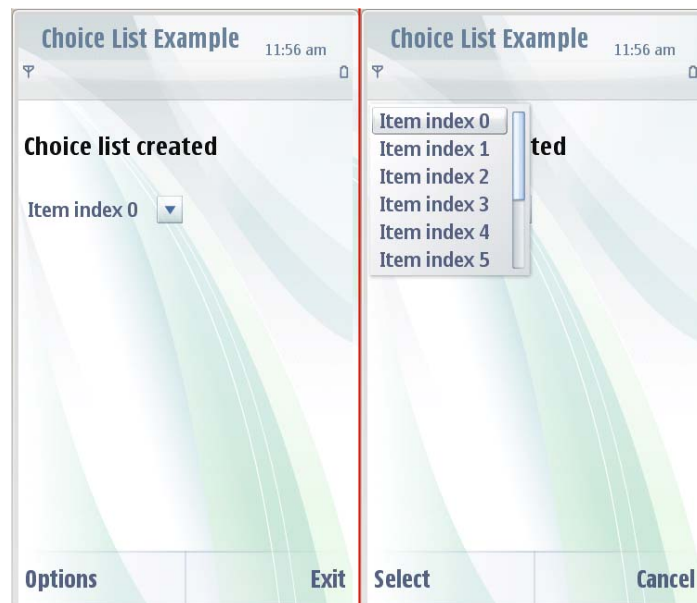


Figure 6: The stylus menu in use in Contacts

The `CAknStylusPopupMenu` class implements the menu. The associated AVKON resource is `STYLUS_POPUP_MENU`. Items can be pre-filled by resources or entered dynamically through the class interface. User selections are returned to your observer via the `MEikMenuObserver` interface. More information is available from the platform documentation on the [Stylus API](#).

## 4.9 Choice List API

This API displays a vertical list of items from which a selection can be made:



**Figure 7: Choice list when closed (left) and after being tapped (right)**

Taken from an S60 5th Edition SDK example application (Choi ceLi stEx)

This is particularly useful if you must directly replace a CEi kChoi ceLi st-based control from a UIQ application.

CAknChoi ceLi st is the implementation class for the control. Selection information is made available to an observer through the MCoeControlObserver interface; selection changes are indicated by the EEventStateChanged value, and the currently selected index can be obtained from CAknChoi ceLi st::SelectedIndex(). More details are available from the S60 5th Edition documentation on the [Choice list API](#).

## 4.10 Status Pane API

S60 supports application customization of some elements of the status pane through a dedicated API. An application can use this API to modify the contents of some of the sub-panes that make up the status pane (other sub-panes are global and cannot be customized). The status pane itself can be customized dynamically.



**Figure 8: Layout of the status pane**

The base classes for the status pane are CEi kStatusPaneBase and CEi kStatusPane, accessible through a CAknAppUi -derived AppUI using the CAknAppUi::StatusPane() member function.

The most useful sub-panes for customization are the navigation and title panes. The navigation pane can be customized to add decorators such as tab groups and scroll arrows. The title pane can contain a title label and a small image.

Both the navigation and title panes support touch interaction and can notify observer classes of touch events, which can be useful if custom navigation controls are to be used. An observer could look like this:

```
class CMyPaneObserver : public CBase,
                      public MAknTitlePaneObserver,
                      public MAknNavigatorDecoratorObserver
{
public: // Implement MAknTitlePaneObserver
    virtual void HandleTitlePaneEventL(TInt aEventId);

public: // Implement MAknNavigatorDecoratorObserver
    virtual void HandleNavigatorDecoratorEventL(TInt aEventId);
    [...]
    void ConstructL(CEikStatusPane& aStatusPane);

private:
    CAknNavigatorControlContainer* iNavPane;
    CAknNavigatorDecorator* iDecorator;
    CAknTitlePane* iTITLEPane;
};
```

The observer registers itself to receive tap events during construction:

```
void CMyPaneObserver::ConstructL(CEikStatusPane& aStatusPane)
{
    // Set the navigation pane observer (assumes the pane exists)
    iNavPane = static_cast<CAknNavigatorControlContainer*>
        ( aStatusPane. Control L(TUi d: : Ui d(EEikStatusPaneUi dNavi)) );

    iDecorator = iNavPane->ResourceDecorator();
    iDecorator->SetNavigatorDecoratorObserver(this);

    // Set the title pane observer (assumes the pane exists)
    iTITLEPane = static_cast<CAknTitlePane*>
        ( aStatusPane. Control L(TUi d: : Ui d(EEikStatusPaneUi dTitle)) );

    iTITLEPane->SetTitlePaneObserver(this);
}
```

The observer methods will now be called when the title or navigation pane controls are tapped.

While dynamic modification of panes through APIs such as CAknTitlePane or CAknNavigatorControlContainer can be useful, it is often equally useful to construct a custom pane using resources. The following snippet defines a custom status pane with a localizable label in the navigation sub-pane:

```
RESOURCE_EIK_APP_INFO
{
    status_pane = r_my_status_pane;
}
```

```

RESOURCE STATUS_PANE_APP_MODEL r_my_status_pane
{
    panes =
    {
        SPANE_PANE
        {
            id = EEi kStatusPaneUi dNavi ;
            type = EAknCtNavi Pane;
            resource = r_my_nav_l abel ;
        }
    };
}

RESOURCE NAVI_DECORATOR r_my_nav_l abel
{
    type = ENavi DecoratorLabel ;
    control = NAVI_LABEL
    {
        txt = stri ng_navi gati on_pane_l abel ;
    };
}

```

The status pane API allows resources to be loaded from a file, so the pane defined above could be swapped for another one during program execution.

The SDK documentation has further information on the status pane.

## Appendix A

The following table summarizes the applications that are available in the UIQ 3, S60 3rd Edition and S60 5th Edition SDKs.

Application (S60 5th Edition)	Application (S60 3rd Edition FP2)	Application (UIQ 3)
Calendar	Organizer\Calendar	Agenda
Contacts	Contacts	Contacts
Log	Log	-
Web	Web	Web
Messaging	Messaging	Messaging
Gallery	Media\Gallery	-
RealPlayer	Media\RealPlayer	-
Music player	Media\Music Player	- <sup>4</sup>
Radio	Media\Radio	-
Ctrl. Panel <sup>5</sup>	Settings	Control Panel/Tools
Ctrl. Panel\Personal\Voice commands	Tools\Voice Comm.	-
Ctrl. Panel\Personal\Profiles	Profiles	Control Panel\Sounds & alerts
Ctrl. Panel\Personal\Themes	Tools\Themes	Control Panel\Themes
Ctrl. Panel\Calling\Speed dialing	Tools\Speed dial	-
Ctrl. Panel\Phone\Phone mgmt.\Device updates	Tools\Device manager	Control Panel\Device\Device Information
Ctrl. Panel\Phone\Phone mgmt.\About	Tools\About	More->System Information
Ctrl. Panel\Application mgr.	Installations\App. mgr.	Control Panel\Install Control Panel\Uninstall
Ctrl. Panel\Connectivity	Connectivity	Control Panel\Connections

<sup>4</sup> Certain audio file types can be played by opening them in the File manager.

<sup>5</sup> Not all sub-categories of the Control Panel are shown here, just categories that have replaced standalone applications in S60 3rd Edition or UIQ 3.

Application (S60 5th Edition)	Application (S60 3rd Edition FP2)	Application (UIQ 3)
-	Tools\Licenses	-
Help	Tools\Help	More->Help
Applications	Installations	-
Applications\File Mgr.	Organizer\File Mgr,	Tools\File Manager
Applications\Calculator	Organizer\Calculator	Tools\Calculator
Applications\Convertor	Organizer\Convertor	-
Applications\IM	Connectivity\IM	-
Applications\Clock	Organizer\Clock	Tools\Time
Applications\Recorder	Media\Recorder	Tools\Voice
Applications\Search	-	-
Applications\Camera	Media\Camera	-
Applications\User Dict.	-	-
Applications\Debug Agent	Installations\Debug Agent	Debug Agent
Applications\Bilingual Dictionary	-	-
Applications\Eshell	Installations\eshell	eshell
Applications\Location	-	-
Applications\Location\Landmarks	Tools\Landmarks	-
Applications\Location\GPS Data	Tools\GPS Data	-
Applications\Location\Positioning	Tools\GPS Data\Options\Positioning Settings	-
Applications\Notes	Organizer\Notes	Jotter
Applications\Oper. Menu	Oper. Menu	-

**Table 2: A list of applications and their equivalents on the platforms covered in this paper**