
Series 60 Developer Platform: Application UI Customization

Version 1.0
October 19, 2004

S E R I E S **60** D E V E L O P E R P L A T F O R M

Legal Notice

Copyright © 2004 Nokia Corporation. All rights reserved.

Nokia and Nokia Connecting People are registered trademarks of Nokia Corporation. Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. Other product and company names mentioned herein may be trademarks or trade names of their respective owners.

Disclaimer

The information in this document is provided “as is,” with no warranties whatsoever, including any warranty of merchantability, fitness for any particular purpose, or any warranty otherwise arising out of any proposal, specification, or sample. Furthermore, information provided in this document is preliminary, and may be changed substantially prior to final release. This document is provided for informational purposes only.

Nokia Corporation disclaims all liability, including liability for infringement of any proprietary rights, relating to implementation of information presented in this document. Nokia Corporation does not warrant or represent that such use will not infringe such rights.

Nokia Corporation retains the right to make changes to this specification at any time, without notice.

License

A license is hereby granted to download and print a copy of this specification for personal use only. No other license to any other intellectual property rights is granted herein.

Table of Contents

1.	Introduction	5
1.1	Purpose and Scope.....	5
2.	Customization Levels Overview	6
3.	Standard Controls	9
3.1	Eikon Controls Customization	9
3.1.1	Common adjustable parameters	9
3.1.2	Listbox	10
3.1.3	Buttons.....	11
3.1.4	Menu.....	11
3.1.5	Command Button Array (CBA).....	12
3.1.6	Label	12
3.1.7	Editors.....	12
3.2	Avkon Controls Customization.....	13
3.2.1	Status pane.....	14
3.2.2	Lists	14
3.2.3	Grids	15
3.2.4	Note dialogs.....	16
3.2.5	Query dialogs.....	17
3.2.6	Forms.....	17
3.2.7	Setting items.....	18
4.	Custom Controls	19
4.1	Custom Control Using Eikon	19
4.2	Full-Screen Customization	20
4.3	Themes	20
5.	Summary	22
6.	Terms and Abbreviations	23
7.	References	24

Change History

October 19, 2004	Version 1.0	Initial document release

1. Introduction

1.1 Purpose and Scope

This document describes the different levels of application UI customization that are possible in the Series 60 Developer Platform for Symbian C++ developers. Features of each level are explained and code examples are provided for demonstration purposes. The relationship between application UI customization and its usability is discussed as well. Note that shell customization is not discussed in this document because it is not possible for third parties. However, the Skins API that is available from the Series 60 Developer Platform 2nd Edition onwards is discussed as one way of modifying the application UI.

The document begins with an overview of customization levels, and then continues with a description of standard controls. Custom components and their implementations are discussed at the end of the document, along with details on Direct Screen Access and the Skins API.

Note that in general custom UIs should only be used when standard Series 60 UI (Avkon) components cannot provide the needed functionality or desired user experience. Creating a custom UI is typically more demanding and fails easier than creating an application UI using the standard UI components. Therefore, the decision to build a custom UI must be considered carefully.

2. Customization Levels Overview

Application UI customization is needed in order to create unique, nonstandard application UIs. Typical applications that employ customization include games and entertainment applications. Custom UI elements replace standard components and help keeping the user experience consistent throughout the application. For example, it is important for a high-level menu or score table to be implemented in a style that is consistent with the game world itself. As such, it will not distract the user, and may provide a better user experience when implemented properly with usability in mind [UIGAME].

By contrast, business applications rarely focus attention on such details, and an application UI composed of plain and standard components is more appropriate. In addition, the use of standard components gives developers more time to work on application problems rather than on the UI, and improve application portability and scalability.

Whereas standard components provide a common navigation structure that is used in Series 60 devices, a very customized UI forces developers to create their own navigation scheme. However, when implementing their own UI elements, developers should try to stay close to the common navigation scheme in order to not confuse the user and provide a consistent user experience.

Application UI customization can be implemented in several ways depending on how much the UI needs to be changed. The approach can vary from using standard functions of Avkon components to creating a fully custom UI using Direct Screen Access (DSA). The Series 60 Developer Platform gives developers enough possibilities to build a custom UI, but the deeper it goes, the more responsibility the developer has to keep usability at an acceptable level.

The following levels of UI customization can be defined:

- **Customization of standard controls** (Avkon/Eikon) — Standard components are customized using available functions.
- **Custom controls** — Custom controls are created based on Uikon (Eikon) components (mainly `CCoeControl`) or from the ground up; the degree of customization can range from slight modifications of existing components to completely original bitmap-based components.
- **Full-screen UI** — The application UI is built without using the UI control framework, drawing directly on the full screen; this approach is usually used with games.

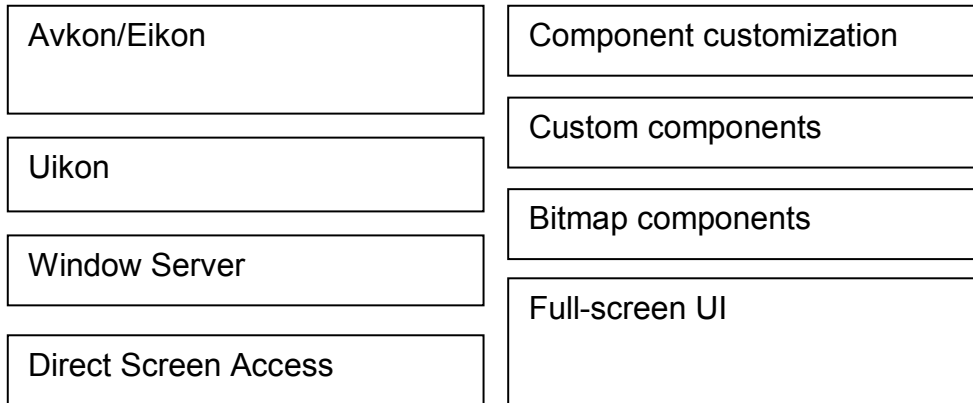


Figure 1: API hierarchy

The customization levels in the Series 60 Developer Platform directly correspond to the levels of the API used. Figure 1 represents their relationship.

Avkon is Nokia's UI library available in the Series 60 Developer Platform. Eikon (EikStd) is a legacy component library introduced in the EPOC 5; it has an implementation in each Symbian OS-based mobile device for portability reasons. Nokia has made Eikon available for the Series 60 Developer Platform as well. Uikon provides a common base for a concert UI library in all Symbian OS variations.

Avkon components were designed in a way that reduces the possibility of UI layout errors by the application that uses that UI library; as such, it makes the entire smartphone UI consistent and also reduces the workload of application screen designers by introducing the Style Guide [UISTYLE]. Of course, this means that there are only a few possibilities for changing the application's appearance, but that was the main idea behind Avkon. This strong restriction on Avkon components' visual appearance and layout is mainly for third-party applications. Series 60 devices, such as the Siemens SX1, have their own Avkon layouts. The built-in UI library components provide support for different languages, a lot of implemented functionality, and compatibility across different developer platform versions. Also, they provide navigation implementation to quickly build easy-to-use applications.

If something needs to be changed or adjusted, developers should implement their own component. It is possible to derive this component from an Avkon component and implement a new behavior, but this requires extensive knowledge of the parent control and extensive usage of the base class methods. Therefore, using Avkon components for derivation is not necessarily easy. Derivation is recommended when the purpose is to add more functionality to an existing component, but not for trying to overwrite the layout of the standard component. It is very likely that some Avkon-based custom components will work incorrectly if some inner behavior is changed. Careful consideration of the original design of the component is necessary for successful derivation.

Eikon components are the preferred basis for creating custom components because they increase portability. Although it requires more work to make custom components compatible with developer platform improvement features, such as the theme feature of Series 60 Developer Platform 2nd Edition, Eikon components may be simpler to work with, impose fewer restrictions, and be easier to modify than Avkon components. In fact, most Avkon components are built upon Eikon. However, there is always some risk when using Eikon for base classes because not all classes are fully supported in the Series 60 Developer Platform and they can be modified and changed in new releases. There are no restrictions when building your own custom components from such basic classes as `CCoeControl`, and it is possible to implement anything you need.

However, even `CCoeControl` can be heavy on memory for some applications. In extreme cases, `bitmap(s)` can be used to represent any image and/or form. In some cases, when a back-end framework is not necessary, it may be feasible to build UI elements without using a basis class—for example, in a game, during gameplay. It might be useful for an animation-intensive application to use DSA, which provides a safe way to avoid the use of communication with the Window server and speeds up drawing operations [SUCO].

In any case, use of nonstandard components should not confuse the user, and recognizable ways of interacting with the application should be provided. Developers should try to use common navigational elements — soft-key labels that are similar to the standard control pane, navigation keys, and so on. For more details, see [UISTYLE].

3. Standard Controls

3.1 Eikon Controls Customization

Eikon provides a rich set of controls for developers to use. SDK documentation describes the majority of Eikon controls; most, but not all, are available in the Series 60 Developer Platform for compatibility reasons. Even those that *are* available may have some functionality limited due to the different requirement of the Series 60 UI. For example, `CEikRichTextEditor` has an `InsertObjectL()` method that pops up a file dialog and is capable of inserting pictures based on the user's selection. However, because there are no file dialogs in the Series 60 Developer Platform, due to an Avkon design decision, this feature is not supported by Avkon UI libraries. Therefore, developers should perform preliminary testing before making any decisions about using components from the Eikon or Uikon UI libraries.

3.1.1 Common adjustable parameters

Several basic controls lie at the roots of control hierarchy and define the common interface. The most important is `CCoeControl`, which is used for derivations of every control and is the basis for custom controls. The `CCoeControl` class has several useful functions that can be used to adjust colors in the controls.

`CCoeControl::OverrideColor()` allows developers to alter the color palette used for control drawing. The function takes a logical color (`TLogicalColor`) value in the first parameter that specifies what element of the UI is drawn by this color, for example, the control text or background window. See the description of `TLogicalColor` in SDK documentation for the full list of possible values. The second parameter is a new color value (`TRgbColor`) used to set a new color for the logical color. When some logical color is overridden, the `CCoeControl::HandleResourceChange()` with the parameter value `KEikMessageColorSchemeChange` should be called to make changes effective. For example, the following code sets a new color for label text:

```
...
iLabel->OverrideColorL(EColorLabelText, KRgbRed);
iLabel->HandleResourceChange(KEikColorResourceChange);
...
```

Note that you do not need to call the `HandleResourceChange()` method if the colors are set before activating control.

In addition, the `CCoeControl::GetColorUseListL()` function can be used to find out the list of logical colors in use by a control. Overall, color overriding may work, but system controls implement their layout based on Avkon UI library standards instead of taking into account individual control functions. An override will work, but will not be visible when values are overridden; the new values will not be seen until the next redraw. As soon as the draw method is called, most of the system components will restore the color values inside the `Draw()` to the ones defined by the system. Therefore, it will appear as if the functions have not been affected at all, or only for a few logical colors. This should be checked and tested in advance by the developer. In general, changing colors and creating your own look is NOT recommended, as it may interfere with the theme feature (the user can change the theme at any time, and the theme can contain various color layouts) as soon as the application is ported to Series 60 Developer Platform 2nd Edition.

Another Symbian OS Uikon library class, `CEikBorderedControl` (derived from `CCoeControl`), provides functions to specify the type of border. The visual representation of a border is defined by the `TGulBorder` class. At present, this class is not a base class for the Avkon UI library controls. Therefore, apart from `CAknInputFrame`-, `CAknPopupList`-, `CAknPopupField`-,

`CAknPopupFieldList`-, and `CAknSlider`-based classes, calling `CEikBorderedControl::SetBorder()` has no meaning.

3.1.2 Listbox

Eikon listbox classes represent the solid foundation that was used to build the Avkon lists hierarchy. The base class `CEikListBox` defines the primary interface and the structure of listboxes. It contains a model (`MListBoxModel` interface); a view (`CListBoxView`), which is responsible for overall list appearance; and an item drawer (`CListItemDrawer`), which is responsible for individual item representation.

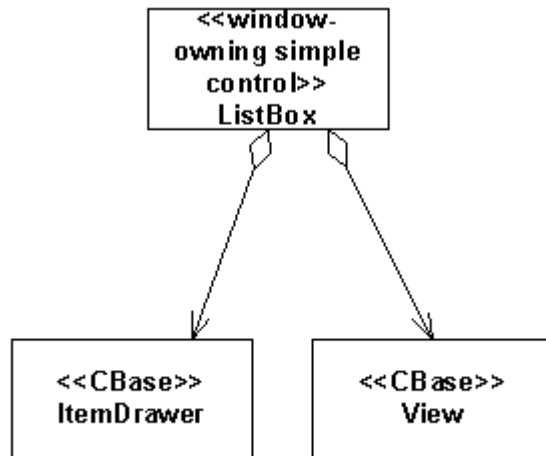


Figure 2: Listbox architecture

We will examine the parameters that can be altered in the list view and the item drawer. First, `CListItemDrawer` allows developers to change the following parameters:

- Item cell size
- Flag for draw mark
- Width of the mark column
- Gap between mark and item main content
- Mark color
- Font to use for mark drawing (symbol font)
- Background color
- Background color for highlighted items
- Color for dimmed text
- Background color for dimmed items
- Text color
- Text color for highlighted items

In general, it is not recommended to change the listbox look through the item drawer in Avkon because the layout can be distorted.

Additionally, an instance of the `CListBoxData` class can be set in the item drawer to specify the following font parameters:

- Text alignment
- Font height

The `CListBoxView` class has the following parameters that can be altered:

- Item height
- Background color
- Text color
- Flag to draw items emphasized
- Flag to draw items dimmed

Eikon provides several concrete listboxes. The text listbox (`CEikTextListBox`) has its own model and item drawer (`CTextListItemDrawer`) that adds the following parameters to change text appearance:

- Text font
- Item width in characters

The same drawer is used for a snaking listbox (`CEikSnakingTextListBox`). Note that the snaking listbox is not a part of the Avkon UI; using Avkon grids is recommended.

The `CEikColumnListBox` class derived from the `CEikTextListBox` class allows developers to build a list with items consisting of several columns, either text or graphics. The `CColumnListBoxData` class is used by the `CColumnListBoxItemDrawer` class for actual item drawing. It contains the following data, which defines the look of an item:

- Column width in pixels
- Column's horizontal gap
- Column's font
- Column's alignment
- Flag that indicates whether the column contains graphics

3.1.3 Buttons

This group contains classes (`CEikButtonBase`, `CEikBitmapButton`, `CEikCommandButton`, `CEikCommandButtonBase`, `CEikButtonGroupContainer`, and `CEikButtonGroupStack`) to build basic types of buttons, such as command buttons. Usually, a button can contain a picture and/or text. The `CEikImage` class is used to control the picture and the `CEikLabel` class is used to control the appearance of text. In cases where a button contains both picture and text, their layout could be specified. Most button classes provide functions to alter corresponding objects. Developers should not try to insert any button to a dialog in any other place but defined as a "button" in its resource. It is recommended to use `CCoeControl` to show buttons (for example, as the system application Calculator does).

3.1.4 Menu

Menu use in the Series 60 Developer Platform is usually limited to a two-level pop-up menu accessed through the control pane. For this reason, its maximum height is fixed and the recommended maximum number of items at one menu level is six items. A menu pane is presented and can be accessed through the `CEikMenuPane` class. The class provides the basis functions to manage menu

panes on the level of inserting, removing, and modifying menu items. The menu look is not designed to be customizable in any way.



Note: The dimmed menu items in the Series 60 Developer Platform are implemented in a way that menu items become invisible. Icons are not supported and, therefore, not displayed in menus of Series 60 devices.

3.1.5 Command Button Array (CBA)

The Command Button Array (CBA) is located at the control pane. It is possible to dynamically change it by calling the `CAknView::Cba()` method and obtaining the pointer to a `CEikButtonGroupContainer` object. The `CEikButtonGroupContainer` class provides functions to set new commands, replace or modify existing ones, and change text labels. The Avkon UI library is intended to look two-dimensional, therefore images are not supported as command buttons and are ignored in the Series 60 Developer Platform.

When using `CEikButtonGroupContainer` it is not recommended to use all of its functionality, such as positioning buttons to different parts of the screen where they appear for the original softkeys. It is not recommended to use more than one CBA in a dialog.

3.1.6 Label

The label component is implemented in the `CEikLabel` class that can display multiline text. The appearance of the text can be controlled by setting the following parameters:

- Font
- Flags text underlining or strike-through
- Gap between lines
- Emphasis

3.1.7 Editors

The text input control of the `CEikEdwin` class is a basis control for entering and manipulating text. It is used frequently in forms and dialogs.

Background color can be set in two ways:

- By calling `CEikEdwin::SetBackgroundColorL()`.
- By calling `CEikEdwin::OverrideColorL()` with `EColorControlBackground` for the first parameter.

The standard Uikon UI library's `CEikEdwin` uses the `CTextView` class, which provides basic control over text appearance. To provide text formatting functions, `CEikGlobalTextEditor` and `CEikRichTextEditor` classes were derived from `CEikEdwin`. The difference between them is that the rich-text editor supports picture storage and offers more possibilities for using styles. Character formatting is accomplished by setting a format layer (`CCharFormatLayer`) and modifying attributes with the help of `TCharFormat` and `TCharFormatMask` classes. `TCharFormat` is used to set various font attributes:

- Font representation, through the data member of `TFontPresentation` type (text color, highlight style and color, underline, strike-through, picture alignment).
- Font specification, through the data member of `TFontSpec` type (font style, height, typeface).

The `TCharFormatMask` class is used to define what attributes are set in the format layer.

The same approach is used to set the paragraph style by setting `CCharParagraphLayer` coupled with the `TParaFormat` and `TParaFormatMask` classes. The following paragraph parameters can be altered:

- Tabs
- Borders
- Background color
- Margins
- Indent
- Vertical and horizontal alignment
- Line spacing
- Wrapping
- Bullets

The `CEikEdwin::SetCharFormatLayerL()` and `CEikEdwin::SetParaFormatLayerL()` functions are used to set new layers. The following code shows how to change text color in Edwin control:

```
...
CCharFormatLayer* formatLayer =
CEikonEnv::NewDefaultCharFormatLayerL();

TCharFormat charFormat;
TCharFormatMask charFormatMask;
FormatLayer->Sense(charFormat, charFormatMask);

charFormat.iFontPresentation.iTextColor=KRgbRed;
charFormatMask.SetAttrib(EAttColor);

formatLayer->SetL(charFormat, charFormatMask);

iEdwin->SetCharFormatLayer(Format);
...
```

The extended versions of text input controls – `CEikGlobalTextEditor` and `CEikRichTextEditor` – apply the same formatting. Two rich-text examples with documentation (Type, RTx) showing the usage of these classes for customizing the text editor are available for download from the Forum Nokia Web site.

Eikon provides several specialized components derived from the `CEikMfne` class. They allow input of such types as time, date, number (fixed point and floating point), and range. In addition, Avkon extends the capabilities of `CEikMfne`-derived classes by providing a function to set the font (`CFont`). Please note that because there was no location information supplier accessory available in the Series 60 Developer Platform, the Longitude and Latitude editors were removed from the Eikon library of the Series 60 Developer Platform.

3.2 Avkon Controls Customization

Building an Avkon-based application UI is a primary choice for third-party add-on applications that don't need specific visuals. Such an approach guarantees compatibility across different Series 60 Developer Platform editions, but, in turn, puts some limitations on the application UI layout imposed by Avkon. Avkon provides a wide choice of well-designed and versatile components, but not all of these are easily modifiable. Here we

will look at each group and see what elements can be modified in the controls. A full description of UI elements appears in the *Series 60 UI Style Guide*, which is part of the Series 60 SDKs.

3.2.1 Status pane

The status pane is located in the region at the top of the screen; a normal layout usually contains an application icon, a title, a signal indicator, and navigation panes. In an idle layout, the battery signal pane and clock appear instead of the application icon. The look of the status pane can be changed by adjusting some of the subpanes, or it can be made invisible if the developer chooses. The following example code demonstrates this:

```
...
StatusPane()->MakeVisible(EFalse); //make pane invisible
...
```

The `StatusPane()` method is available in the `CAknAppUi` class. If access to the status pane is needed within any other `CCoeControl`-derived class, the following code is used:

```
...
CEikStatusPane* statP = iEikonEnv->AppUiFactory()-
>StatusPane();
...
```

If the status pane is visible, the following panes can be adjusted in the normal layout:

- The context pane (`CAknContextPane`) allows setting any bitmap instead of the application icon.
- The title pane (`CAknTitlePane`) allows a new text string or a bitmap to be set.
- The navigation pane (`CAknNavigationControlContainer`, `CAknNavigationDecorator`) allows changes in configuration and adjustment of the following elements: tabs, bitmap, text, and volume indicator.

The above-mentioned elements can be swapped for custom controls as well.

The signal level, volume level/universal indicator cannot be controlled by an application.



Figure 3: Status pane with normal layout

See the SDK example and SDK Status Pane help for more detailed instructions on manipulating the contents of the status pane.

3.2.2 Lists

List controls are widely used for building UI interfaces in the Series 60 Developer Platform; these controls offer conventions for showing similar data items. There are several predefined layouts implemented in a number of classes (derived from `CAknColumnListBox`), which allow data items to be represented in several ways. The variations mostly occur in the following parameters:

- Heading (combination of numbering heading, text heading, graphic heading)
- Item height (small/big)

- Number of lines (single/double)
- Setting option

Lists can also be shown in View mode, where the highlight is not visible.

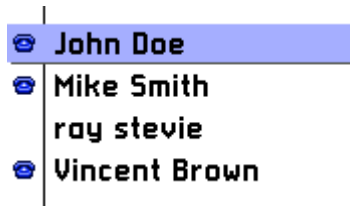


Figure 4: List

See the SDK example and SDK Listbox Control help for a full list of possible list types and instructions on how to construct and use them.

3.2.3 Grids

Grids (`CAknGrid`) allow items to be represented in a two-dimensional table. In fact, a grid is a modified version of a list. The text and/or bitmap can be displayed as an item. Because the size of the cell is usually small (due to the limited screen size of Series 60 devices), there is less possibility for putting complex content inside. More work is required from a developer to set up a grid from scratch. All layouts should be set manually. This approach is recommended only if more control over the look of the control cannot be achieved by using bitmaps.

The `CAknGrid` class is used to set up the basic parameters of a layout in terms of number of columns and rows, cell size, and orientation. After this, `AknListBoxLayouts` is used to set up detailed parameters for graphics and text in the cell. The following cell parameters can be set:

- Location and size of bitmaps
- Location and size of text string
- Font and its color
- Text align in the allocated space

Bitmaps are loaded manually, as an icon array.

Other parameters of visual elements (for example, highlighted text color) are used from the Series 60 Developer Platform standard look and feel module; these should not be overridden unless you've instantiated your own class.



Figure 5: Normal grid

Several specialized grid classes can be found in Avkon.

`AknCalMonthlyStyleGrid` is used to show a month view. The first row shows the day and the first column shows the week number. The following data can be set for an item:

- Outlined icon to show border, dimension 21 x 9 pixels
- Marking icon in the bottom right corner of a cell, dimension 5 x 5 pixels
- Two-digit number

`CAknPinbStyleGrid` is used to show application shortcuts in a 5 x 5 cells grid. For every item, an icon is specified that is displayed in the center of a cell. In addition, two small icons, 13 x 13 pixels, can be specified. The first one is displayed in the top-right corner for marking. The second one is displayed in the bottom-left corner to show the target application.

`CAknGMSGrid` is the most useful from the ready grids and it displays images, which can be marked. The main image of each cell should have dimensions of 74 x 28 pixels. Additionally, a marking icon can be provided and displayed in the top-right corner.

See the SDK example for Grid Control for information on how to construct and use grids.

3.2.4 Note dialogs

Avkon provides a set of classes for notification messages. All of them are derived from the `CAknNoteDlg` class. This class can be used to specify your own text as well as image/icon/animation. All parameters are specified in the resources, although they can be changed from code. Font parameters and colors will follow the standard Avkon look and apply the current theme settings automatically.

`CAknConfirmationNote`, `CAknInformationNote`, `CAknErrorNote`, and `CAknWarningNote` classes derived from `CAknNoteWrapper` can be used without specifying resources, and they contain a predefined layout and corresponding animated icons. Also, `CAknStaticNoteDialog` is available to show permanent notes that do not have a time out and that stay visible until they are removed by the application.

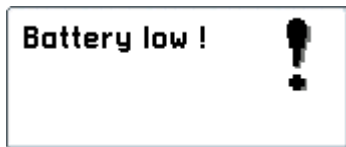


Figure 6: Warning note

There are also progress (`CAknProgressDialog`) and waiting (`CAknWaitDialog`) dialogs available for use. The `CAknProgressDialog` class uses a standard progress bar (`CEikProgressInfo`), while `CAknWaitDialog` uses a custom animation icon. The look of the bars in both classes is defined by the current color scheme.

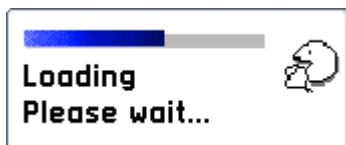


Figure 7: Progress dialog

There are also global variants of note dialogs (`CAknGlobalNote`). Unlike normal notes, global notes stay at the top, even if the application changes. They have their own resource data and in order to use them, developers should only call `CAknGlobalNote::ShowNoteL()` with the note type and text message specified. All possible types of global notes are listed in the `TAknGlobalNoteType` enumeration.

See the SDK's Note Control Examples.

3.2.5 Query dialogs

Query dialogs provide ready-to-use controls to query commonly used data types like text, number, date, and time. In addition, confirmation dialogs can be constructed for simple yes/no or OK/cancel queries. Single-selection and multiselection list query dialogs are also available for use. The following parts are available for modification from the `CAknQueryDialog` class:

- Data query
 - Header text
 - Header image/animation
 - Prompt text
 - Edit indicator (can be removed)
- Confirmation query
 - Prompt text
 - Image/animation
- List query
 - Header text
 - Header image/animation
 - List type (Single, SingleHeading, SingleGraphic, SingleGraphicHeading)

Note that in order to have a header pane, the `AVKON_HEADING` component should be specified as the first dialog line in the resource description. At runtime, the dialog's header pane (`CAknPopupHeadingPane`) can be obtained by calling the function `CAknQueryDialog::QueryHeading()`.



Figure 8: Data query

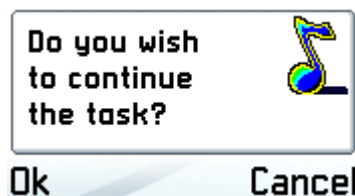


Figure 9: Confirmation query

3.2.6 Forms

Forms (`CAknForm`) are available exclusively in the Series 60 Developer Platform and provide the way to input text. Dialogs are not recommended in Avkon for that purpose, because filling the Edwin prompt results in a black box instead of text. See the Series 60 UI guidelines for a detailed description of forms. Concerning customization, the `CAknForm` class provides a single-line or double-line layout

and it is possible to use icons on forms. The following can be set in the resource file:

- Label type (text/bitmap)
- Single- or double-line format for items

Figure 10: Form with single-line layout

3.2.7 Setting items

A settings view (or screen) is a nice way to represent all application settings in a compact form. Usually this contains a list of setting items (`CAknSettingItemList`) that are derived from the `CAknSettingItem` class and can have different types, for example text, number, data, time, volume, slider, and binary (on/off). Every setting item can have an associated setting page for editing its value (based on `CAknSettingPage`). The look of the list view or setting page is not meant to be customized; the setting item title and default value can have their own value. It is possible to derive from `CAknSettingPage` to create a custom behavior, or restrict user input to a certain subset, which may not be achievable by setting different Edwin constants.

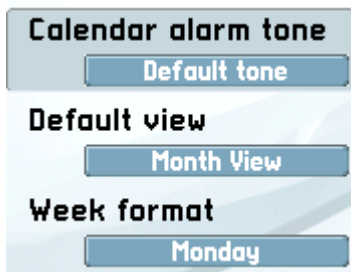


Figure 11: Setting list from Calendar

Settings lists are described in *Series 60 Developer Platform: Implementing Settings Screens* (with example), which can be downloaded from Forum Nokia, and in the SDK example Setting List.

4. Custom Controls

4.1 Custom Control Using Eikon

CONE provides a solid base for developers to build their own custom controls. Usually developers derive their class directly from the `CCoeControl` class and implement custom drawings and event handling. Two-phase construction can be implemented as well if there are some internal members in a custom control. The following virtual functions are important for a custom control:

- `Draw()` - Performs component drawings.
- `SizeChanged()` - Actions needed if component's size changes.
- `CountComponentControls()` and `ComponentControl()` - For compound controls to provide inner components to the framework.
- `OfferKeyEventL()` - Provides appropriate behavior for key events.
- `MinimumSize()` - Calculates control's minimum size; should be implemented if control is displayed inside a dialog.
- `MopSupplyObject()` - Sets the object provider chain (`MObjectProvider`) if theme support is planned.

Also, putting the custom control to the control stack may be necessary in order to provide key events to the control.

The rest depends on the developer's application needs. For a simple control, the developer would add necessary data members and functions. Drawing is mostly performed using internal data. Compound controls usually own their parts as data members, and drawings are delegated to them. However, the `Draw()` method of a compound control often contains code to clear its rectangle.

```
...
CWindowGc& gc=SystemGc();
gc.Clear(aRect);
...
```

The `AknLayoutUtils` class can be used to lay out Avkon components according to the standard Avkon look and feel, but it is not recommended to use this class without being familiar with the original layouts of the components.

Special actions are needed to use custom controls in dialogs. First, the appropriate `DLG_LINE` should be defined in the resource definition of the dialog:

```
RESOURCE DIALOG r_custom_dlg
{
    flags = EEikDialogFlagCbaButtons | EEikDialogFlagWait |
    EEikDialogFlagNoDrag;
    buttons = R_AVKON_SOFTKEYS_OK_CANCEL;
    items =
    {
        DLG_LINE
        {
            type = ECustomDlgCtInputFrame;
            id = ECustomDlgIdInputFrame;
        }
    };
}
```

where `ECustomDlgCtInputFrame` is a type of custom component and `ECustomDlgIdInputFrame` is its ID.

Both values are defined in the `.hrh` file. Next, `CreateCustomControlL()` has to be overridden and include code that will create a custom control. This function will be called during dialog construction for every `DLG_LINE` with a custom type.

```
SEikControlInfo CCustomDlg::CreateCustomControlL(
    TInt aControlType)
{
    SEikControlInfo controlInfo;
    controlInfo.iControl = NULL;
    controlInfo.iTrailerTextId = 0;
    controlInfo.iFlags = 0;

    switch (aControlType)
    {
        case ECustomDlgCtInputFrame:
            controlInfo.iControl = CInputFrame::NewL();
            break;
        default:
            break;
    }
    return controlInfo;
}
```

Note that the Series 60 Developer Platform 2nd Edition application wizard creates the correct custom control when the Dialog architecture with theme support is chosen.

Bitmap controls can be implemented in several ways. In the simplest case, developers can use the `CFbsBitmap` class to keep the image as a data member. Then it is displayed by drawing it in the `Draw()` method:

```
...
CWindowGc& gc=SystemGc();
gc.BitBlt(TPoint(0,0), iMainImage);
...
```

where `iMainImage` is the pointer to the instance of the `CFbsBitmap` class. Several bitmaps can be used to represent different states of control depending on the application logic.

4.2 Full-Screen Customization

It is advisable to omit the UI control framework and communicate directly with the Window server for graphic-intensive applications. Certainly the lack of functionality that is provided otherwise by the Avkon framework is compensated by the higher graphics performance achieved by optimizing requests to the Window server. See the SDK examples (Symbian OS/Window Server example projects) for implementation details. DSA in Symbian OS v7.0s and Symbian OS v6.1 would be an ideal choice for ultimate performance. It skips client-server communication with the Window server and the context switching associated with this, in order to obtain faster drawing speed. See [GAMES] for implementation details and example code.

Regardless of the method used for game drawing, it is recommended that developers build the UI from scratch. Implementation details depend on specific application requirements. With no constraints on how the UI looks, developers should not confuse users, and should try to follow basic UI guidelines for Series 60. The basic functionalities of the mobile device should be maintained, such as the red key goes to idle, the application key switches between applications, and the device receives incoming calls and messages, including visual and audible notifications.

4.3 Themes

Avkon themes (or skins) were introduced in Series 60 Developer Platform 2nd Edition. Themes allow a changing shell look by providing custom UI elements such as display

background, color palette, highlights, pop-up windows, and a common components look. Themes give the device UI a new brand look. Series 60 Theme Studio is available to produce such themes by graphical designers [THEME].

Avkon components have four different levels of skin support. See [SKINS] for more details. Developers can use the Skins API in several ways. First, it is recommended that developers add skin support to their application when it is ported to Series 60 Developer Platform 2nd Edition. Avkon provides a couple of helper classes (`AknsUtils`, `AknsDrawUtils`) that make it easy. Also, Series 60 AppWizard 2.0 can generate application templates with skin support.

An API is provided to add theme elements for use in custom controls. These local elements can be accessed through an appropriate context object – by using the Object Provider mechanism. Some of the local elements can be used to replace some predefined elements of the current theme, for example, the main background or the control panel's background [SKINS].

5. Summary

This document provides an overview of Series 60 UI customization approaches. It gives developers a general idea about each approach and how it can be used to build custom UIs. In general, custom UIs should be used only when Avkon components cannot provide the needed functionality. The safe and reliable way is for developers to build their own custom components. Modification of Avkon components is not recommended because these components are not meant to be used this way — and doing so will create more problems than benefits. Custom bitmap components and full-screen customization are recommended and efficient ways to build the highly customizable UIs needed for game and entertainment applications.

6. Terms and Abbreviations

Term or abbreviation	Meaning
CONE	The UI Control Framework.
DSA	Direct Screen Access.
UI	User interface.

7. References

[DIGIA] *Programming For The Series 60 Platform And Symbian OS*, Digia Inc., 2003

[GAMES] *Series 60 Developer Platform 1.0/2.0: Programming Games in C++*

[SKINS] *Series 60 Platform 2.x AVKON Skins User's Guide*

[SUCO] *Creating Suitable And Compatible Series 60 Application UI*, Eero Tunkelo, April 29, 2003

[THEME] *Series 60 Theme Studio Artist's Guide*

[UIGAME] *Series 60 Developer Platform 2.0: Usability Guidelines For Symbian C++ Games*

[UISTYLE] *Series 60 UI Style Guide*