S60 View Architecture with UI Design

Overview

31 Aug 2008

This example shows how to develop S60 UI applications similar to the native “Settings” application available on every S60 device. This application was chosen because it has different views and uses tabs and a main view to switch between them. The example is based on the S60 UI view architecture.

The article discusses the view architecture and other architectures that can be used to develop UI applications. The example was developed using the Carbide S60 UI design. This article also gives some tips on using this tool.

This snippet can be self-signed.

Preconditions

This example requires using Carbide UI Design.

Background

There are three UI architecture options for developing applications: the traditional Symbian OS architecture, the S60 view architecture, and the dialog-based architecture.

Dialog-based architecture:

An architecture where most parts of the views are dialogs. In these cases multi-page dialogs act as multiple views.

Advantages:

- Content and layout can be changed in the resource file without rebuilding the C++ code.

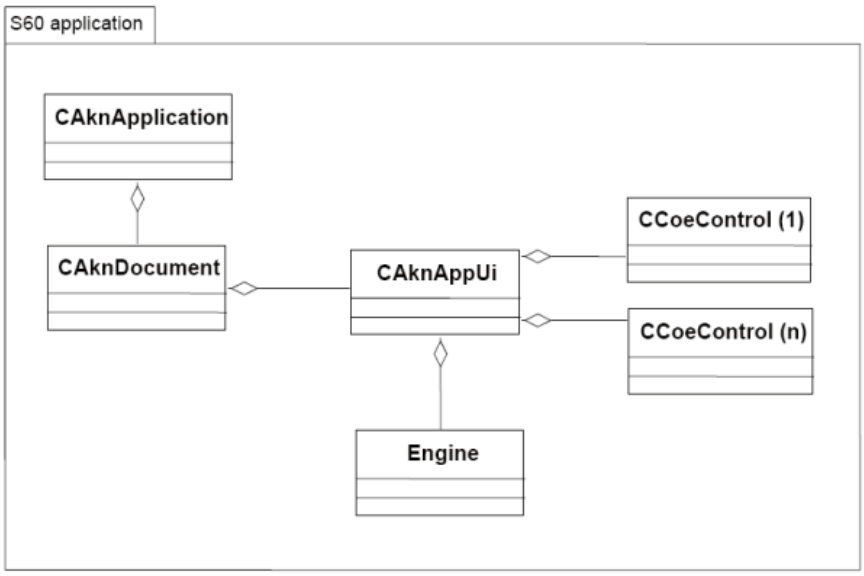
Disadvantages:

- The number of nested dialogs can create problems for stack.

Traditional Symbian OS UI architecture:

In this option you need to implement a UI Controller, based on CAknAppUI class and a view derived from the CCoeControl class.

The CAknAppUI-derived class acts as a controller and each CCoeControl object acts as a different view. The following figure shows the relation of these classes.



Advantages:

- The most flexible way to construct an application UI.
- Developers can freely design their own UI architecture.

Disadvantages:

- The developer has to do the view management. Tasks like view switching have to be implemented completely by the developer.

The S60 view architecture: Designed by S60, this option implements a view management mechanism in the CAknViewAppUi class. This class acts as a

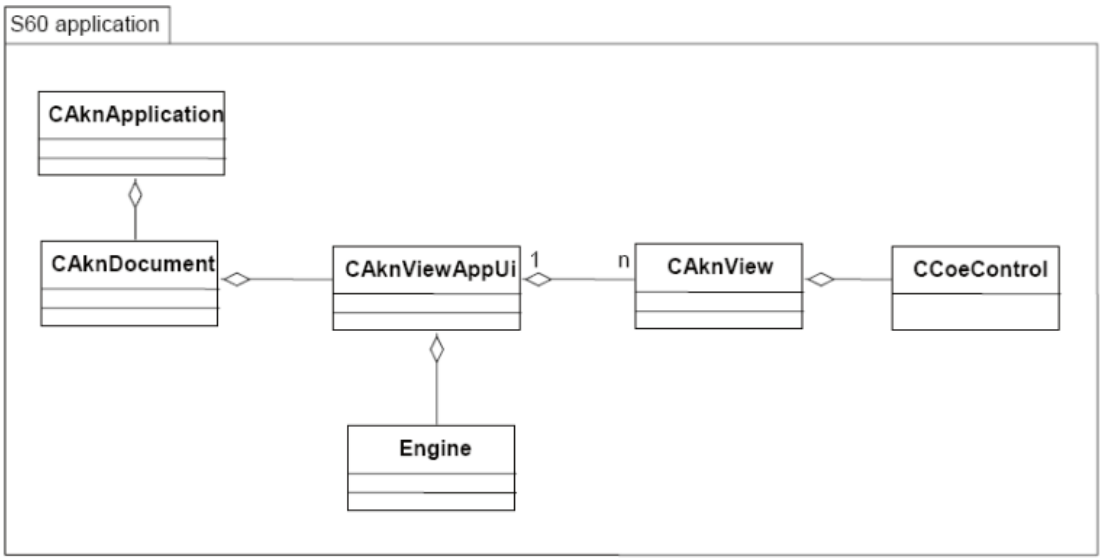
(C) Copyright Nokia 2013. All rights reserved.

UI controller, the CAknView class acts as a view controller, and the CCoeControl acts as a view. This architecture allows only one view to be active in each application.

The CAknViewAppUI class:

- Creates one or more CAknView-derived view controllers.
- Handles events that are not handled by the view controllers.
- Activates and deactivates views to switch between views.
- Handles menu commands passed to it by the view controllers.
- Receives events such as layout and foreground notifications from the run-time environment.

The relation of these classes can be seen in the following figure.



Advantages:

- This architecture is more object-oriented and adapts better to situations where changes are required.
- External server management of views which enables interdependencies between applications that share functionalities.
- Reduces complexity of individual applications.

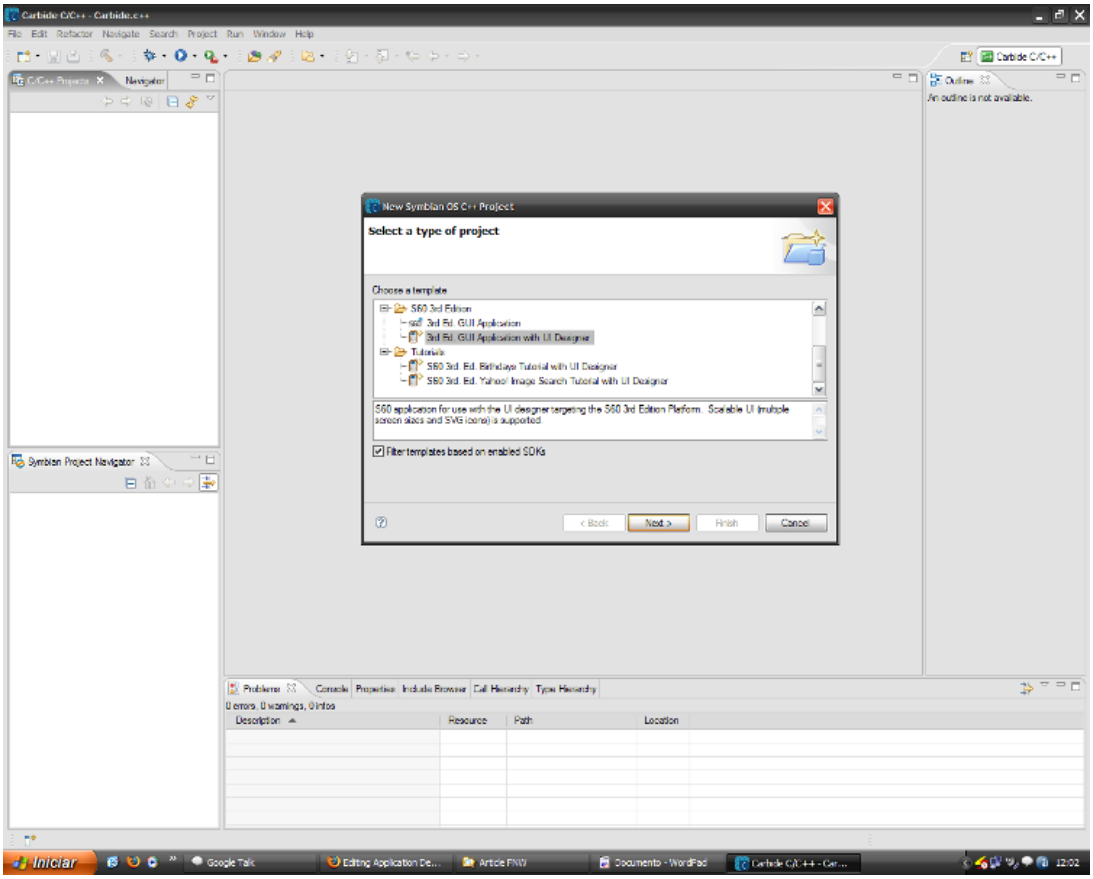
Disadvantages:

- Is more restrictive than the traditional Symbian OS view.
- Cannot be used in all cases.

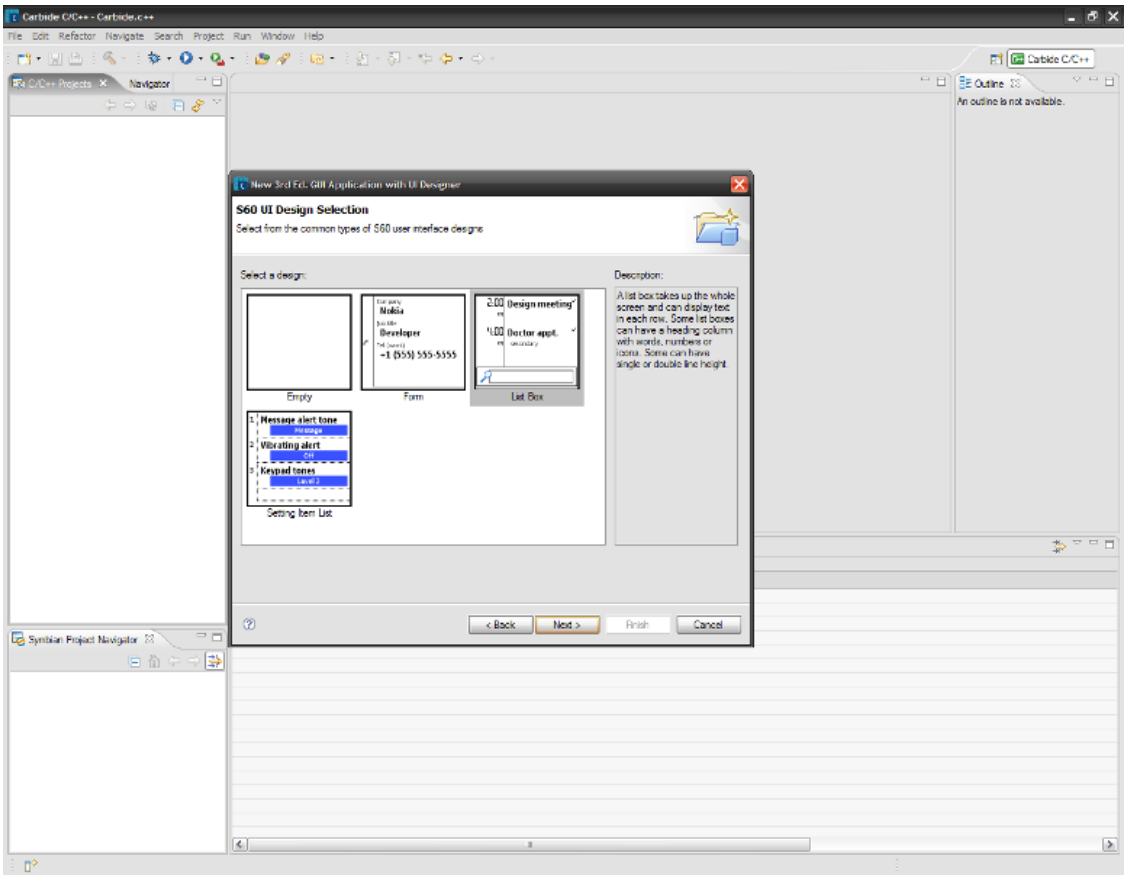
Creating your first view

Start a new project. Choose a Carbide template (*S60 3rd Ed GUI Application with UI Designer*) for this as shown in the following picture.

(C) Copyright Nokia 2013. All rights reserved.

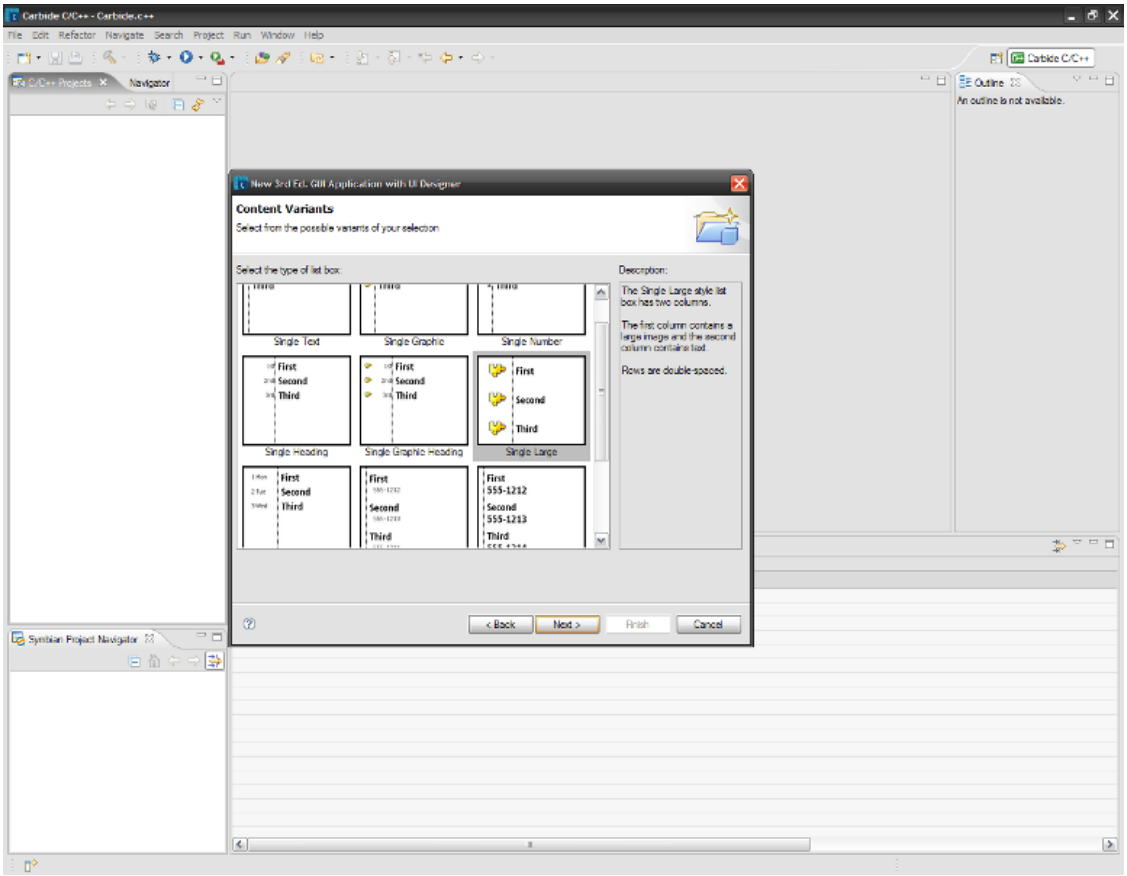


After choosing the template and giving the project a name, Carbide prompts a screen where it asks to select a design. In this case, List Box is selected.

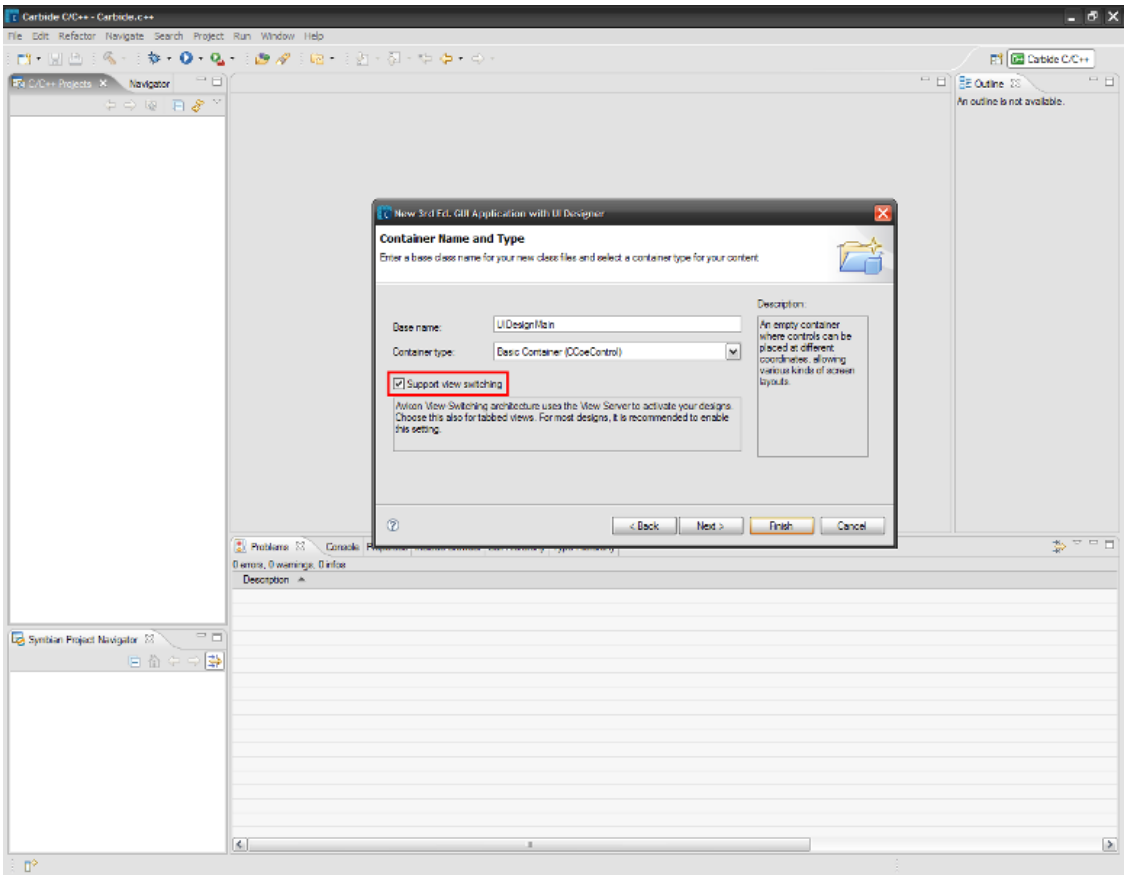


Next, select the type of the list box. This first design corresponds to the main screen of the application, so select a single large list for it.

(C) Copyright Nokia 2013. All rights reserved.

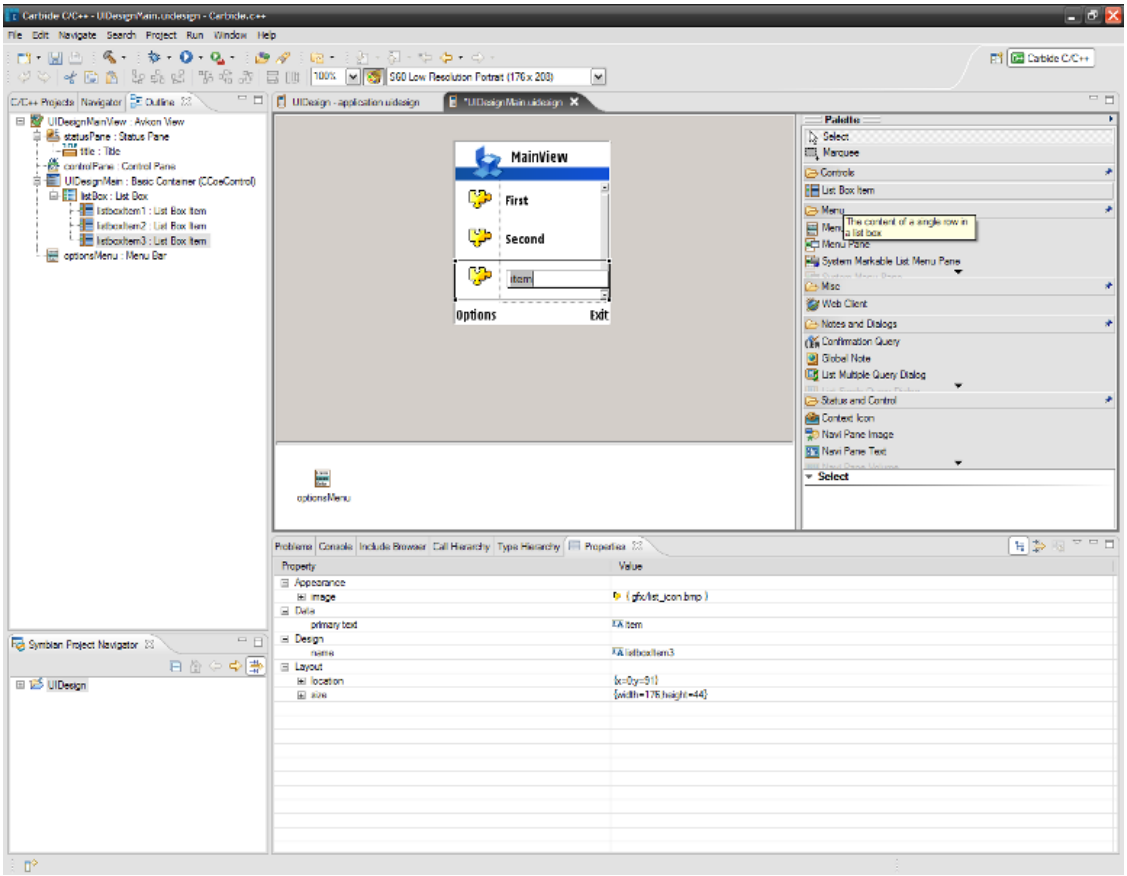


After this, give the list box a name and make sure that support for view switching is enabled (as seen in the following figure).



Now the main screen design is ready and you can add items to it. To do this, just drag and drop a List Box Item from the Palette to the design. A simple double-click on the item enables the editing mode.

(C) Copyright Nokia 2013. All rights reserved.



After creating the main view, three more views are created; one for each item available in the MainView. All the views are shown in the following figure.

Each view is registered with the AppUI class through the `AddViewL` method of its base class (CAknViewAppUI). The `InitializeContainersL` method should be called when constructing the AppUI object. This method creates the view objects of the application and registers then to the framework that takes its ownership. This method also sets the default view, that is, the first view to be shown when the application starts. This view is the MainView and the `SetDefaultViewL` is used.

```
void CUIDesignAppUi::InitializeContainersL()
 {
 iUIDesignMainView = CUIDesignMainView::NewL();
 AddViewL( iUIDesignMainView );
 SetDefaultViewL( *iUIDesignMainView );
 iUIDesignSettingItemListFirstView = CUIDesignSettingItemListFirstView::NewL();
 AddViewL( iUIDesignSettingItemListFirstView );
 iUIDesignFormSecondView = CUIDesignFormSecondView::NewL();
 AddViewL( iUIDesignFormSecondView );
 iUIDesignListThirdView = CUIDesignListThirdView::NewL();
 AddViewL( iUIDesignListThirdView );
 }
```

Enabling navigation

Now there are four views and we need to enable navigation among then. In this case, there are three views that are “secondary screens” and one view that is the “main screen”. The navigation on these screens is as follows:

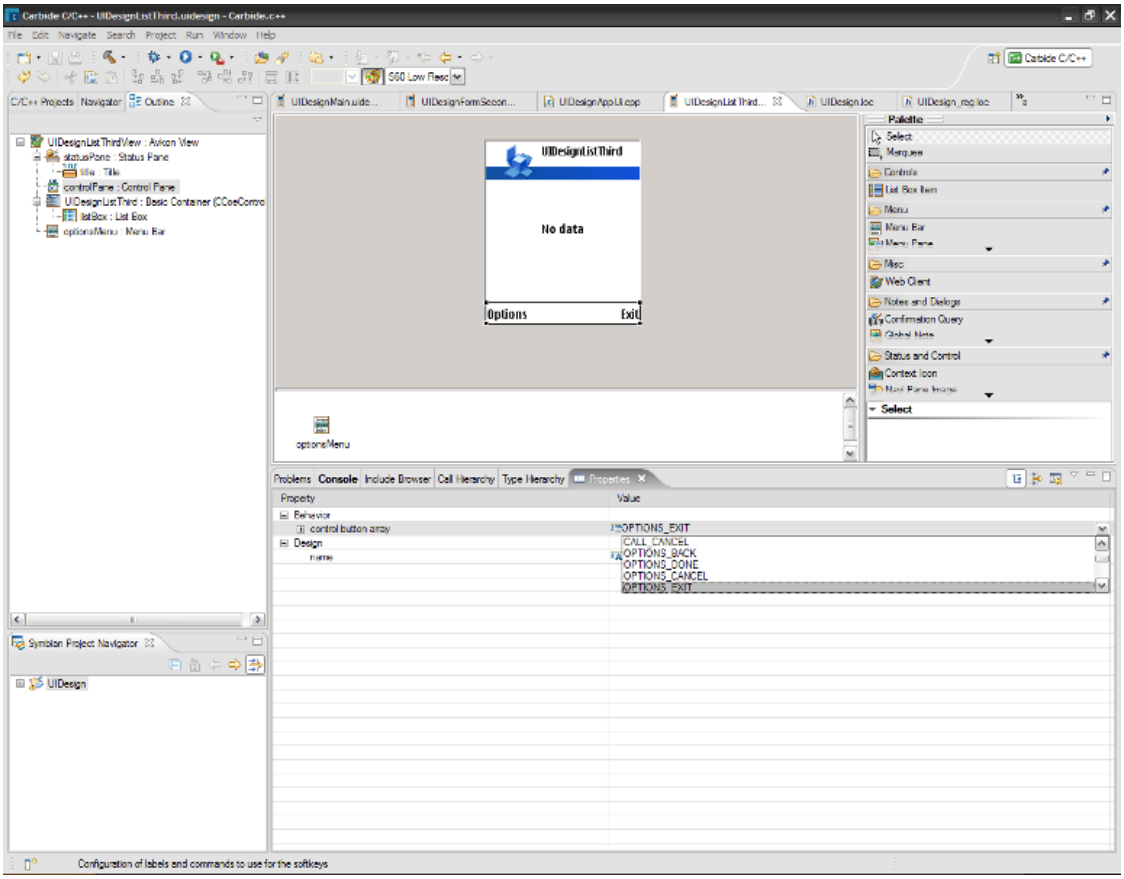
The application opens in the “main screen”. In this screen the user can exit the application by pressing the right softkey or go to one of the three views

(C) Copyright Nokia 2013. All rights reserved.

that are listed by pressing the action button. When the user does this, the view appears and the user can change to other secondary views by pressing the left and right keys. Each view has one tab. To go back to the main view, the user presses the right softkey.

The first thing to notice is that UI design creates all views as if they were the unique view of the application. Due to this, each view is created with a CBA sequence of OPTION_EXIT so that the user can exit the application by pressing the right softkey. However, the views that belong to the secondary screens should behave in a different way. Therefore, the CBA sequence of the "secondary screens" should be changed to OPTIONS_BACK so that the user is moved to the "main screen" and only in this screen show the user the OPTIONS_EXIT sequence.

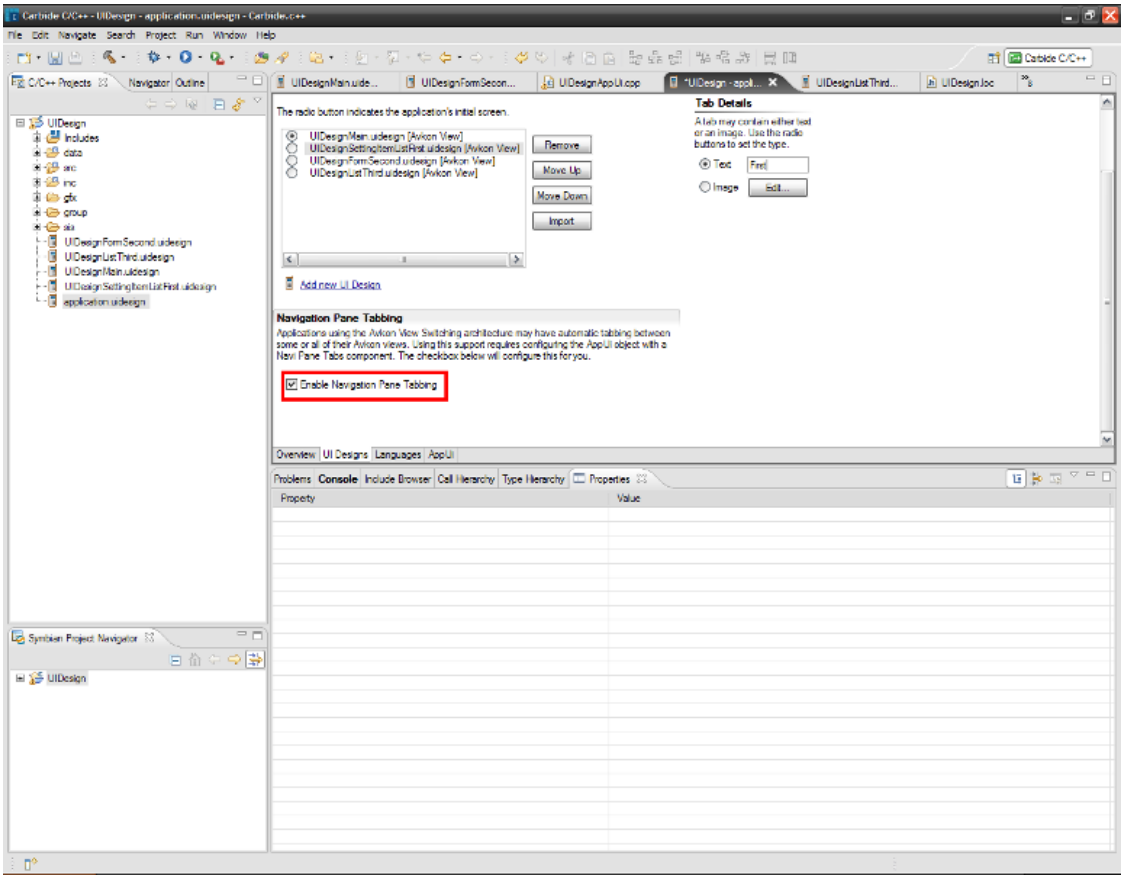
To do this, you need to combine two views in Carbide: the outline view and the Property view. Open the outline view and select the Control Pane of the design of a secondary screen. Doing this, the Property view shows the properties of this element. There you can see under the "Behaviour" property the "control button array" entry. You can change this entry from the CBA sequence OPTIONS_EXIT to OPTIONS_BACK. This is shown in the following picture.



However, this is not enough to get the desired results. You also need to change the entry "exit application on close" to false under the Behavior property of the Basic Container of the view. To do this, select the Basic Container in "Outline" and search for this entry in the Property view.

You also need to enable tab navigation. To do this, go to the "application.uidesign" file, open the tab UI Design, and mark the check box "Enable Navigation Pane Tabbing". After this, select a view and fill the text to appear in the tab of that view. This is shown in the following picture.

(C) Copyright Nokia 2013. All rights reserved.



This adds the following code to the `InitializeContainersL` method of the AppUi class.

```
void CUIDesignAppUi::InitializeContainersL()
 {
 …
 CAknNavigationControlContainer* naviPane = ( CAknNavigationControlContainer* )
  StatusPane()->ControlL( TUid::Uid( EEikStatusPaneUidNavi ) );
 iNaviDecorator_ = naviPane->ResourceDecorator();
 if ( iNaviDecorator_ != NULL )
  {
  iNaviTabs1 = ( CAknTabGroup* ) iNaviDecorator_->DecoratedControl();
  }
        …
```

This code enables accessing and controlling the tabs through the `iNaviTabs1` object.

Activating secondary screen views

Before continuing, you need to define a state for the UI controller (AppUi class) so it knows which view is activated at the moment. This state can be EMainView if the main screen is shown or ETabView if a secondary screen is shown.

In the main screen you need to be able to select the current view you want to see by pressing the action button. To be able to do this, you need to know which item in the item list is selected when the button is pressed. From the index of the selected item, activate the correct view through the method `ActivateLocalViewL` of the base class of AppUi. To handle the action button event, go to the `HandleKeyEventL`of the AppUi class. The following code shows how to do this.

```
TKeyResponse CUIDesignAppUi::HandleKeyEventL(
  const TKeyEvent& aKeyEvent,
  TEventCode aType )
 {
        if( iUIState == EMainView )
  {
  if( aKeyEvent.iCode == 63557 && iNaviTabs1 )
   {
                        TInt activeIndex = 1 + iUIDesignMainView->CurrentSelectedIndex();
   iNaviTabs1->SetActiveTabByIndex( activeIndex );
   ActivateLocalViewL( TUid::Uid( activeIndex ) );
   ShowTabs( ETrue );
   iUIState = ETabView;
   return EKeyWasConsumed;
   }
  else
   return EKeyWasNotConsumed;
                }
        }
…
```

You also need to reorder the UID of the views by placing the view of the main screen as the last view.

```
enum TUIDesignViewUids
 {
 EUIDesignSettingItemListFirstViewId = 1,
 EUIDesignFormSecondViewId,
```


(C) Copyright Nokia 2013. All rights reserved.

```
 EUIDesignListThirdViewId,
 EUIDesignMainViewId
 };
```

Activating main screen view

The previous section explained how to go from the main screen to the secondary screens. Now we need to know how to make the reverse path, going from a secondary screen to the main screen. The challenge in this step is that the item in the main screen has to correspond to the previous activated view. The wrong item can be obtained by navigating laterally among the secondary screens through tab switching.

The UI controller class handles the correct activation of the main screen. When the right softkey is pressed in a secondary screen view, the window server sends the command `EAknSoftKeyBack` to the `HandleCommandL` function of the view. If the view does not consume this event, it is sent to the UI controller (AppUi). The following piece of code shows the implementation of the `HandleCommandL` method in each view.

```
void CUIDesignListThirdView::HandleCommandL( TInt aCommand )
 {
       ...
       // code to send the right soft key event to AppUi
 if ( !commandHandled )
  {
  if ( aCommand == EAknSoftkeyBack )
   {
   AppUi()->HandleCommandL( EAknSoftkeyBack );
   }
  }
 }
```

In the `HandleCommandL` function of the UI controller, write the code that selects the correct item in the main screen list depending on the previous activated view. The first thing to do is to get the active tab index. This index is used to select the correct item in the main screen. The second thing to do is to send this information to the view. An elegant way to do this is to send a message to the view. The method `ActivateLocalViewL` has an overloaded version that allows doing this. This version has two parameters: a message identification code and the message itself. As the message to be sent is quite simple (an integer), the message identification parameter UI controller is used to send the message. The integer is converted to a UID. The following code shows how to do this.

```
void CUIDesignAppUi::HandleCommandL( TInt aCommand )
 {
 if ( iUIState == ETabView &&
   aCommand == EAknSoftkeyBack )
  {
  TInt active = iNaviTabs1->ActiveTabIndex();
  ActivateLocalViewL( TUid::Uid( (int)EUIDesignMainViewId ),
       TUid::Uid( active ),
       KNullDesC8
       );
  ShowTabs( EFalse );
  iUIState = EMainView;
  }
 ...
}
```

In the main screen view, the message is received. Select the correct item in the list. The following code shows the view activation code.

```
void CUIDesignMainView::DoActivateL(
  const TVwsViewId& /*aPrevViewId*/,
  TUid aCustomMessageId,
  const TDesC8& /*aCustomMessage*/ )
 {
 SetupStatusPaneL();

 CEikButtonGroupContainer* cba = AppUi()->Cba();
 if ( cba != NULL )
  {
  cba->MakeVisible( EFalse );
  }

 if ( iUIDesignMain == NULL )
  {
  iUIDesignMain = CUIDesignMain::NewL( ClientRect(), NULL, this );
  iUIDesignMain->SetMopParent( this );
  AppUi()->AddToStackL( *this, iUIDesignMain );
  }
 iUIDesignMain->SetSelectedItem( (int) aCustomMessageId.iUid );
 }
```

Controlling the Tab Group

The last thing that needs to be done is to hide and show the tab group in the status pane depending on the view that is activated at the moment. To do this, create a method in the UI controller. See the following code:

```
void CUIDesignAppUi::ShowTabs( TBool aState )
 {
 if( iNaviTabs1 && iNaviDecorator_)
  {
  iNaviDecorator_->MakeScrollButtonVisible( aState );
  iNaviTabs1->MakeVisible( aState );
  iNaviDecorator_->DrawDeferred();
  }
 }
```

This code should be called every time the state of the UI controller changes. This happens in the methods `ConstructL`, `HandleKeyEventL`, and `HandleCommandL` of the AppUi.

(C) Copyright Nokia 2013. All rights reserved.

Test application

Test application File:S60viewarchitecture.zip

See also

- S60 application views
- How to work with views and view architecture
- Model-View-Controller application architecture

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.