Split view input in Symbian C++ applications

Introduction

17 Oct 2010

The Symbian^3 UI Style Guide document available on this site introduces a new input method, the "split view" (section 6.2.7). Unlike the traditional full screen virtual keyboard used in Symbian^1 (S60 5.0) , which hides completely the application when opened, the new input mode allows developers to retain control over the upper side of the screen, where they can show their own UI,optimized so that the view is focused on the editor which is currently receiving the keyboard input. If the real estate available for the application can accommodate

How can this input mode be enabled for your application?

- Step one: identify the editor component which will receive input in split mode and set the **EAknEditorFlagEnablePartialScreen** flag for it. If the flag is not set to the editor (**iEditor1**'s case), the default mode for opening the virtual keyboard is in full screen.

```
void CSplitScreenKeyboardContainer::ConstructL(
  const TRect& aRect,
  const CCoeControl* aParent,
  MEikCommandObserver* aCommandObserver )
 {
 if ( aParent == NULL )
     {
  CreateWindowL();
     }
 else
     {
     SetContainerWindowL( *aParent );
     }
 iFocusedControl = NULL;
 iCommandObserver = aCommandObserver;
 InitializeControlsL();
 iEdit1->SetObserver(this);
 iEdit2->SetObserver(this);

        // enable split view input for this editor
 iEdit2->SetAknEditorFlags(EAknEditorFlagEnablePartialScreen);

 SetRect( aRect );
 ActivateL();
 // [[[ begin generated region: do not modify [Post-ActivateL initializations]
 // ]]] end generated region [Post-ActivateL initializations]

 }
```

- Step two: be ready to handle the resource change notifications with types **KAknSplitInputEnabled** and **KAknSplitInputDisabled**. These change notifications are received when the split view keyboard is shown/hidden on the screen, and it is your responsibility to ensure that the application is aware if the client area resize.

```
void CSplitScreenKeyboardAppUi::HandleResourceChangeL( TInt aType )
 {
 CAknViewAppUi::HandleResourceChangeL( aType );

 // [[[ begin generated region: do not modify [Generated Code]

 // ]]] end generated region [Generated Code]

        if ( aType == KAknSplitInputEnabled )
     {
              // notify the view that client rectangle has changed, set the app in split kb mode
  iSplitScreenKeyboardContainerView->SetHalfKB(ETrue, ClientRect());
              // hiding the status pane gives more space for the application above the VKB
      StatusPane()->MakeVisible( EFalse );
     }
        else if ( KAknSplitInputDisabled == aType )
     {
              // restore normal view, with status pane and full size client rectangle
      StatusPane()->MakeVisible( ETrue );
      iSplitScreenKeyboardContainerView->SetHalfKB(EFalse, ClientRect());
     }

 }
```

Warning: It is important to note that you are not allowed to change the focus state of your components when handling the change notifications of these two resource types. A focus change at this point will generate an infinite loop in the FEP.

- Step three: update the layout of your current view, to take best advantage of the new real estate. Remember to place the editor which is currently receiving the focus in first plan, so that the user understands that the keyboard is linked to the highlighted editor, and that input will be directed there. You can otherwise use the rest of the client area as you see fit in your application.

```
void CSplitScreenKeyboardContainer::SizeChanged()
 {
 CCoeControl::SizeChanged();
 LayoutControls();

 // [[[ begin generated region: do not modify [Generated Contents]

 // ]]] end generated region [Generated Contents]

 if(!iHalfKBOn)
  {
  // create normal layout

  TRect rect = Rect();
  rect.Shrink(20,20);
```


(C) Copyright Nokia 2013. All rights reserved.

```
  iEdit1->MakeVisible(ETrue);
  iEdit1->SetExtent( rect.iTl, TSize( rect.Width(), rect.Height()/2-5) );
  iEdit2->SetExtent( TPoint( rect.iTl.iX, rect.iTl.iY + rect.Height()/2+5 ), TSize( rect.Width(), rect.Height()/2-5) );
  }
 else
  {
  // create a layout focused on the editor receiving the input

  TRect rect = Rect();
  rect.Shrink(10,10);

                // not visible in this mode
   iEdit1->MakeVisible(EFalse);

  iEdit2->SetRect( rect );
  }

  DrawNow();
 }
```

APIs used

There are no new APIs needed for supporting this experimental feature. The flag **EAknEditorFlagEnablePartialScreen** is defined in **eikon.hrh**, but the two resource types are defined in non-SDK file so if you decide this feature you will have to define them in your app.

```
#ifndef AKNPRIV_HRH
// Message used to inform split input status
#define KAknSplitInputEnabled 0x2001E2C0
#define KAknSplitInputDisabled 0x2001E2C1
#endif
```

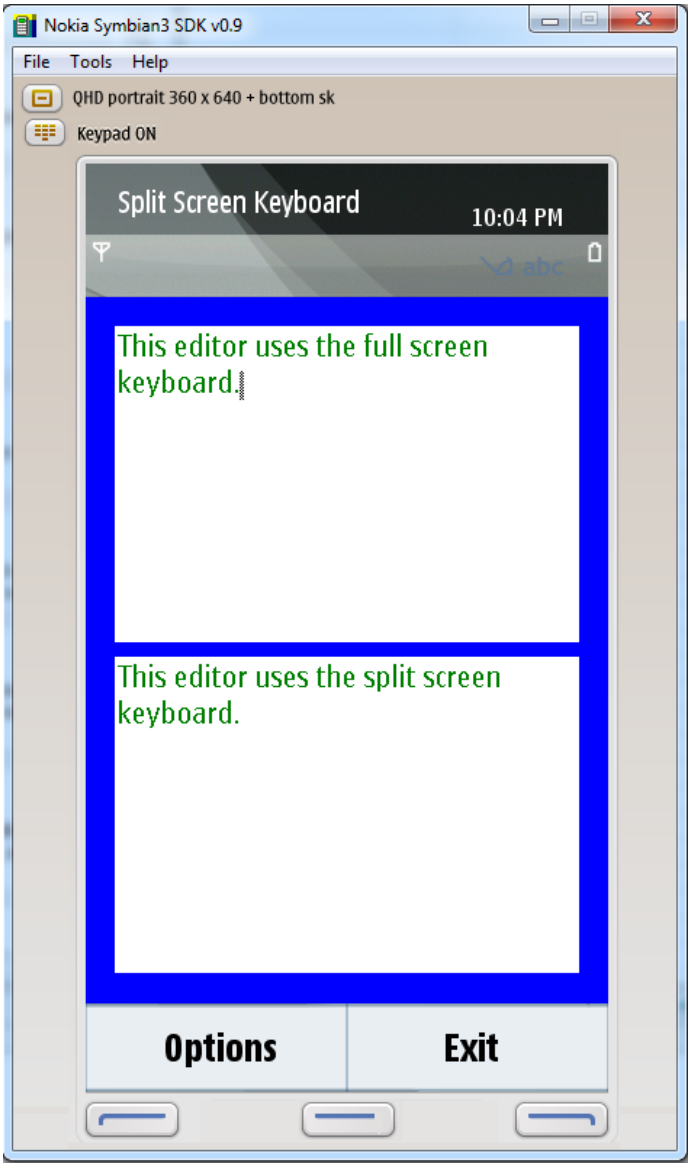
Example project

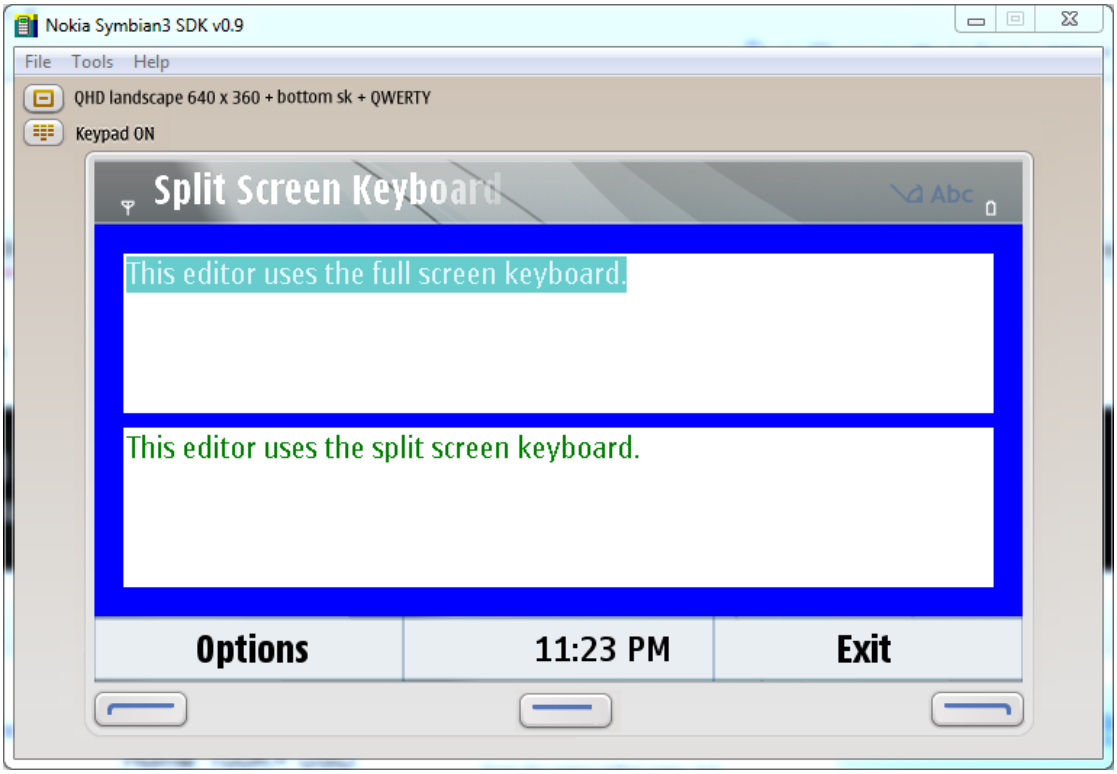
Download File:SplitScreenKeyboard.zip

Screenshots

a) The application in normal view,with both editors visible. iEdit2 is the text editor component at the bottom of the screen.

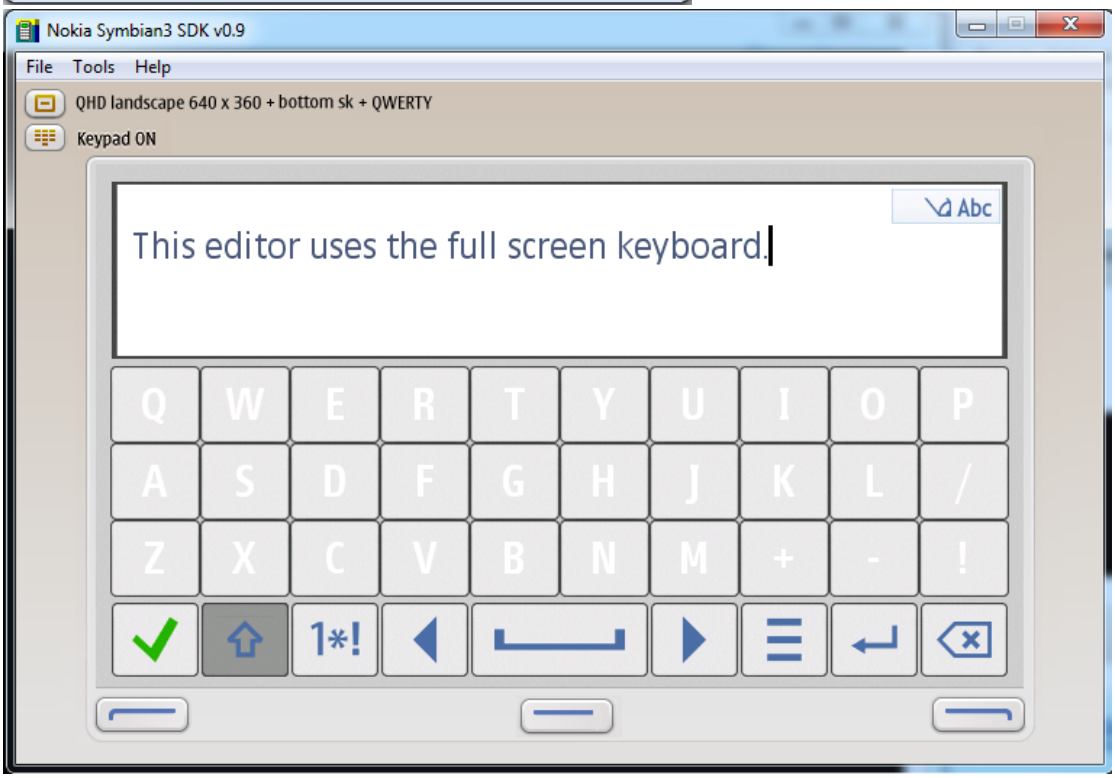
(C) Copyright Nokia 2013. All rights reserved.





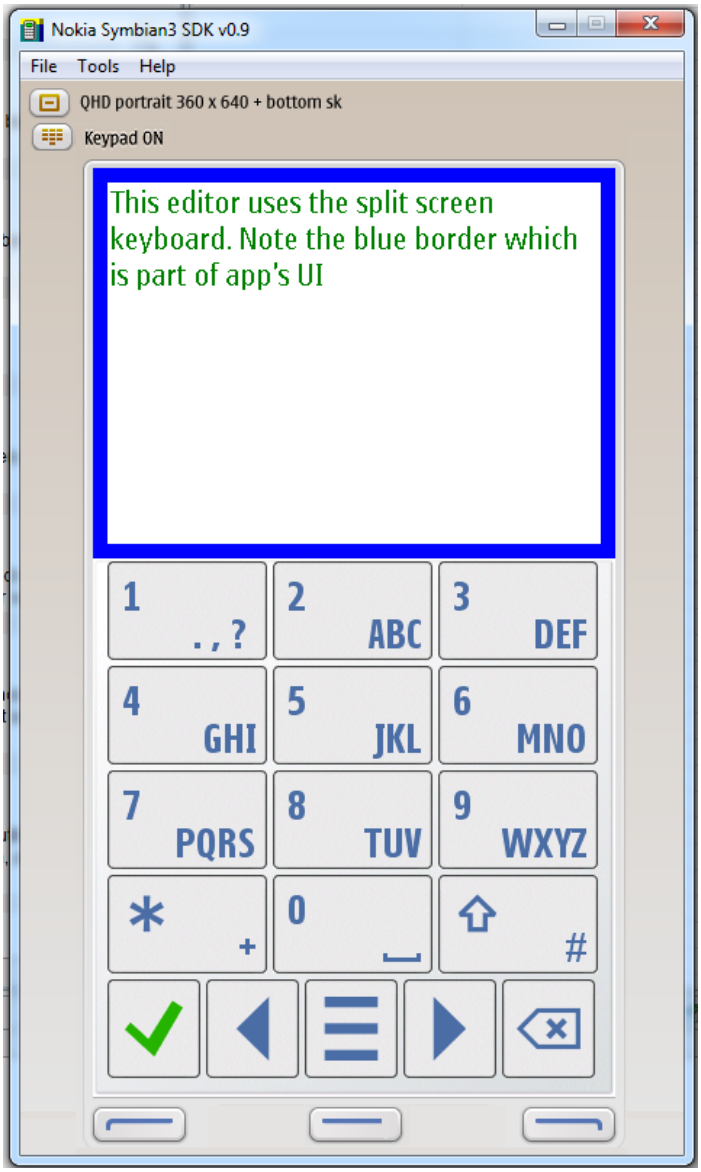
b) The virtual keyboard opened (in full screen mode) for editing the text in iEdit1

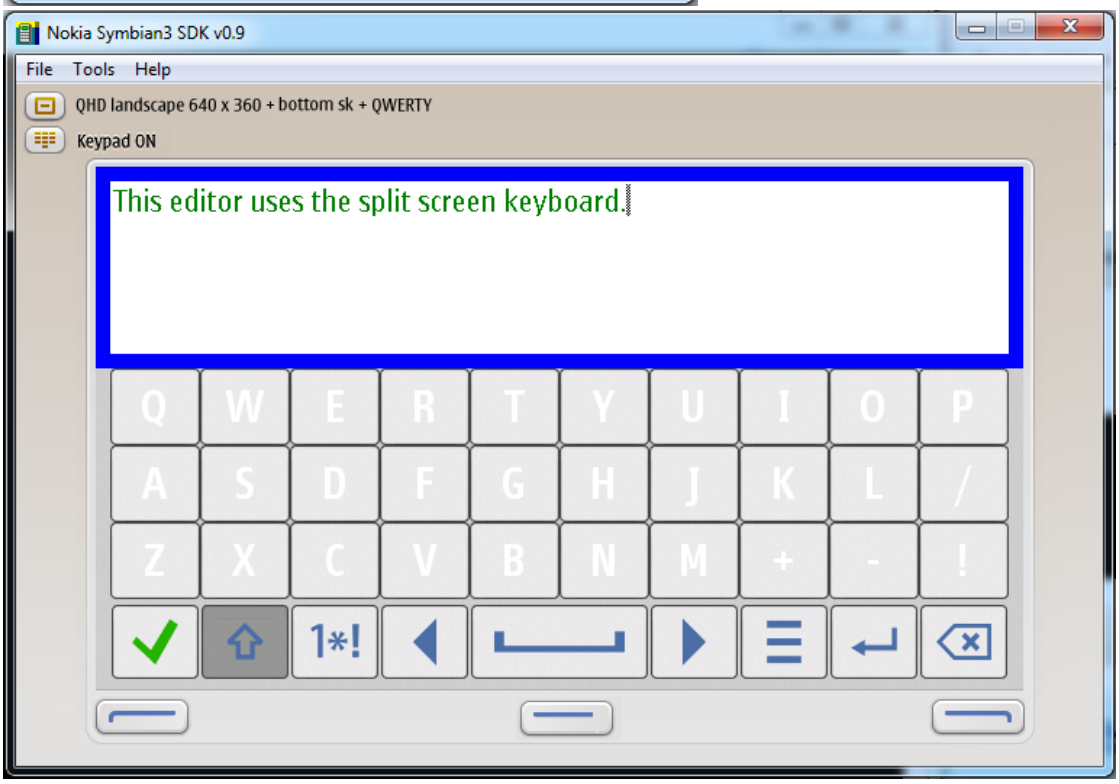
(C) Copyright Nokia 2013. All rights reserved.



c) The virtual keyboard opened (in split mode) for editing the content of iEdit2. The application is now responsible for displaying content in the top area of the screen.

(C) Copyright Nokia 2013. All rights reserved.





(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.