Create a Custom User Control in Silverlight for Windows Embedded

Windows Embedded CE 6.0 R3 Technical Article

Published: September 2009

Applies To: Windows Embedded CE 6.0 R3

Abstract

You can create a custom user control that provides niche functionality and unique visual characteristics by using the **XRCustomUserControlImpl<Class[,Interface]>** class in Silverlight for Windows Embedded. For example, you might need a control that is customized for a particular type of application, such as a control for navigating a map, a control for browsing a three-dimensional menu, or a control that shows progress or status using advanced visual indicators instead of a simple progress bar.

You define the appearance of a custom user control in XAML, implement its functionality in C++ by using Platform Builder, and access it from the visual tree by using the **FindName** method.

When you create a custom user control, you can reuse it across multiple Silverlight for Windows Embedded applications.

The **XRCustomUserControlImpl<Class[,Interface]>** class is the current, supported version of the template class for creating a custom user control. However, an earlier version that is no longer supported, **XRCustomUserControl<Interface,Class>**, is still documented in the reference.

Introduction

You can create a custom user control that provides niche functionality and unique visual characteristics by using the **XRCustomUserControlImpl<Class[,Interface]>** class in Silverlight for Windows Embedded.

When you create a custom user control, you can reuse it across multiple Silverlight for Windows Embedded applications.

The **XRCustomUserControlImpl<Class[,Interface]>** class is the current, supported version of the template class for creating a custom user control. However, an earlier version that is no longer supported,**XRCustomUserControl<Interface,Class>**, is still documented in the reference. The following methods of this class are no longer public methods: **Create**, **Initialize**, **IsInitialize**. The current version has an additional overloaded version of **Register** that you can use to register a control created entirely in XAML, without any C++ code. This overload version of **Register** has the following function signature:

```
static HRESULT Register(const WCHAR* pControlName, const WCHAR*
pNamespace)
```

Prerequisites

- A Silverlight for Windows Embedded application. For more information, see “Create a Silverlight for Windows Embedded Application” in the Windows Embedded CE 6.0 R3 documentation.

How to Create a Custom User Control

Creating a custom user control in Silverlight for Windows Embedded is a six-step process. First, you must define the graphical user interface (GUI) in XAML. Then, create a source file in C++ for the custom user control. Then you can implement the custom user control class. Then, you must register the control and add it to the visual tree. Finally, build your application and OS. The process to create a custom user control is described as follows.

Define the GUI for the Custom User Control in XAML

Use Expression Blend 2 or another XAML editor to create a new .xaml file that contains the GUI definition and the namespace declaration for the control. Then, to add the user control to other .xaml files, you add its custom namespace declaration to the root element. For more information about XAML namespaces, see MSDN. For more information about doing this with Expression Blend 3, see the Microsoft Expression Web site.

> **Note** Silverlight for Windows Embedded does not support code-behind in C# for user controls defined in XAML. If you have code-behind in C# for a Silverlight custom user control that you would like to reuse, you can port it to the C++ class that you create in this tutorial.

Create a C++ Source File for the Custom User Control

In your application subproject in Platform Builder, create a C++ source file for your user control that includes XamlRuntime.h and XRCustomControl.h.

If your application calls methods in the custom user control, you must create an interface for the control in the sources file that derives from **IXRCustomUserControl**. Then, define any custom methods or fields on the interface.

The following example template code provides a starting point for creating an interface:

```
_interface __declspec(uuid("{F01D249B-89EC-4134-9CF7-822CB326D5A3}"))
class ICustomCtrl : public IXRCustomUserControl
{
public:
   virtual HRESULT GetProperty() = 0;
};
```

As illustrated in the previous code, you must create a new unique identifier (ID) for the interface by using the Guidgen tool and the **__declspec** keyword. For more information, see **__declspec** at MSDN.

To obtain a unique ID for the __declspec keyword, do the following:

1. From the command prompt build window in Windows, change the directory to C:\Program Files\Microsoft Visual Studio 9.0\Common7\Tools and then type `guidgen`.
2. In the **Create GUID** dialog box, select **4 Registry Format**, and then click **Copy**.
3. Open the new source file by double-clicking it in Solution Explorer, and then paste the GUID into either the **__declspec** keyword, or into the second parameter of the **DEFINE_XR_IID** macro.

   Note You can also use the **DEFINE_XR_IID** macro to assign a unique ID to the interface. **DEFINE_XR_IID** is described in the "DEFINE_XR_IID" topic in the Windows Embedded CE 6.0 R3 documentation.

Implement the Custom User Control Class

Implement a custom user control class that inherits from the template wrapper class **XRCustomUserControlImpl<Class[,Interface]>**.

If you created a custom interface in the previous step, the following must be true:

- The optional **Interface** parameter of the template must be the name of that interface.
- The GUID you supply in the required **__declspec** keyword must be different from the one you associated with the custom interface.

The following code shows a starting point for implementing a custom user control class:

```
class __declspec(uuid("{91C2F5FF-8FCC-4626-AC67-850283620C09}"))
CustomCtrl : public XRCustomUserControlImpl<CustomCtrl,ICustomCtrl>
{
```

```
public:
  static HRESULT GetXamlSource(XRXamlSource* pXamlSource)
  {
    // add implementation
  }
  static HRESULT Register(HINSTANCE hInstance)
  {
    // add implementation
  }
};
```

If you derived the control class from the wrapper class **XRCustomUserControlImpl<Class[,Interface]>**, you must implement **Register** and **GetXamlSource**.

In addition to Register and GetXamlSource, you can also define and implement any additional methods, fields, or events that you want in your custom user control class implementation. If you will register a dependency property or an attached property, you can implement **Get*** and **Set*** methods that pass **XRValue** objects in their parameters.

When you define a custom user control class that inherits from **XRCustomUserControlImpl<Class[,Interface]>**, you must also provide custom implementations of the **Register** and **GetXamlSource** methods.

Register Method

In your application's **WinMain** procedure, after you retrieve the application instance and, optionally, add a resource module, you will call your custom **Register** method. Then, create the visual host by calling **IXRApplication::ParseXaml**.

At minimum, you must include code that calls the **XRCustomUserControlImpl::Register** method in your implementation. This method registers a custom user control by associating a specific element name, which was defined in a specific XAML namespace with the x:Class attribute, with a specific interface ID (IID). **XRCustomUserControlImpl::Register** is a static method and you can call it without creating an instance of **XRCustomUserControlImpl**.

If the custom user control has a dependency property or an attached property, your **Register** method implementation must also include code that registers the property. In this case, the **Register** method implementation must obtain an application instance by calling **GetXRApplicationInstance** and then call either **IXRApplication::RegisterAttachedProperty** or **IXRApplication::RegisterDependencyProperty** method, depending on the type of property included.

Typically, you also implement a **GetCustomProperty** and **SetCustomProperty** method in your control class that provides access to the value of the property.

The following example code shows an example implementation of the **Register** method that is intended to be used in the declaration of your implementation of CustomCtrl.

> **Important** For readability, the following code example does not contain security checking or error handling. Do not use the following code in a production environment.

```
static HRESULT Register()
    {
        HRESULT hr = S_OK;

        hr = XRCustomUserControlImpl::Register(__uuidof(CustomCtrl)),
L"CustomUserControl", L"clr-namespace:CustomUserControlNamespace");
        if (FAILED(hr))
        {
            goto Exit;
        }
   Exit:
        return hr;
}
```

GetXamlSource Method

Before you can parse and load the control into the visual tree, you have to call the GetXamlSource method to get the source XAML file that defines the GUI for the control. The **GetXamlSource** method retrieves the control-definition XAML file so that Silverlight can parse its XAML and load the control into the visual tree.

You write this method so that the **PFN_CREATE_CONTROL** callback function can call it internally when Silverlight parses XAML for the application. This callback function is provided by the wrapper class.

At minimum, your **GetXamlSource** implementation must include code that populates the **XRXamlSource** structure, which is supplied in the input parameter of this method, with information about the source XAML for your control. To add this code, do one of the following:

- Call the member function **XRXamlSource.SetResource**. This member function takes an application instance returned by an earlier call to **GetXRApplicationInstance**, a string that describes the XAML resource type, and the ID of the resource, which is a .xaml file. If you define IDs for XAML resources in a resource (.rc) file, you can convert the ID into a resource type by using the **MAKEINTRESOURCE** macro inline. The following code example shows how to use this member function in a **GetXamlSource** implementation.

  > **Important** For readability, the following code example does not contain security checking or error handling. Do not use the following code in a production environment.

  ```
  #define RT_XAML         L"XAML"
  ```

```
static HRESULT GetXamlSource(XRXamlSource* pXamlSource)
    {
        pXamlSource->SetResource(s_hInstance, RT_XAML,
MAKEINTRESOURCE(ID_XAML_2DLISTVIEW));
        return S_OK;
    }
```

- Call the member function **XRXamlSource.SetFile**. Using this approach, you can set the source XAML file by providing the file name. The following code example shows how to use this member function in a **GetXamlSource** implementation.

  Important For readability, the following code example does not contain security checking or error handling. Do not use the following code in a production environment.

```
static HRESULT GetXamlSource(XRXamlSource* pXamlSource)
    {
        pXamlSource->SetFile(RESOURCE_DIR
L"CustomControlDefinition.xaml");
        return S_OK;
    }
```

Register the Custom User Control

To register the custom user control in your application startup code, call the **Register** method you implemented before the application startup code calls **CreateHostFromXaml**. Then, the wrapper class's default implementation of **PFN_CREATE_CONTROL** calls **GetXamlSource** to provide the correct XAML file to parse and load.

Add the Custom User Control to the Visual Tree

If you want to reuse the custom user control from C++ code in Silverlight applications, add an instance of the custom user control to the visual tree at run time in C++. Do this after you have prepared the Silverlight visual tree, which is described in Step 4 in the topic “Create a Silverlight for Windows Embedded Application” in the Windows Embedded CE 6.0 R3 documentation.

To create a custom user-control instance and add it to the visual tree

1. Initialize an object variable using the class type of the custom user control.

   ```
   ICustomCtrl* pControl = NULL;
   ```

2. Call **IXRApplication::CreateObject** to convert the class you registered with the **Register** method into an object and to return a reference to the new object.

   ```
   pApplication->CreateObject(__uuidof(CustomCtrl), &pControl);
   ```

3. (Optional) To locate this object by **Name** after you add it to the visual tree, call **SetName**.
4. Define the new object instance by using its methods.

```
XRThickness Margin = {25, 25, 100, 200};

pControl->SetMargin(&Margin);

pControl->SetCustomPropValue(100);

pControl->SetName(L"ControlInstance1");
```

5. Attach delegates to the object so it can respond to events.
6. Obtain an **IXRFrameworkElement** smart pointer to the root of the visual tree. The custom user control will be integrated with the layout system in Silverlight and displayed in the graphical window. When you use a smart pointer, you don't have to call **Release** when you are done with it.

```
IXRFrameworkElementPtr pRoot;

pVisualHost->GetRootElement(&pRoot);
```

7. Locate the name of the panel element on which UI elements are positioned, such as an **IXRPanel**-derived object.
8. Find the element in the tree. On the root element, call **IXRFrameworkElement::FindName**, passing in the desired panel element's name, and an object pointer to reference the panel.

```
IXRCanvasPtr pCanvas;

pRoot->FindName(L"MainCanvas", &pCanvas);
```

9. Retrieve the panel's **IXRUIElementCollection** collection by calling **IXRPanel::GetChildren**.

```
IXRUIElementCollection* pCollection;

pCanvas->GetChildren(&pCollection);
```

10. Add the custom user control instance to the collection by calling the collection's inherited method **IXRCollection<In_T, Out_T>::Add**.

```
pCollection->Add(&pControl, NULL)
```

To retrieve a custom user control instance from the visual tree

1. Obtain an **IXRFrameworkElement** smart pointer to the root of the visual tree. When you use a smart pointer, you don't have to call **Release** when you are done with it.

```
IXRFrameworkElementPtr pRoot;

pVisualHost->GetRootElement(&pRoot);
```

2. Initialize an object variable using the class type of the custom user control.

```
CustomCtrl* pControl;
```

3. Find the control in the tree. On the root element, call **IXRFrameworkElement::FindName**, passing in the desired control's name and an object variable to receive the pointer.

```
pRoot->FindName(L"ControlInstance1", &pControl);
```

4. Execute functionality of the custom user control or change its property values by using the object pointer to call methods implemented control class.

```
// retrive a control template from a list box
```

```
    IXRControlTemplatePtr pListBoxControlTemplate);
    pListBox->GetTemplate(&pListBoxControlTemplate);

    // set this as the control template for the user control
    pControl->SetTemplate(&pListBoxControlTemplate);
```

Build the Application and Your OS Design

In **Solution Explorer**, expand **Subprojects**, right-click the subproject you created, and click **Build (build)**.

The build progress is displayed in the **Output** tab of the **Output** window.

For more information, see “Use Subprojects in an OS Design” in the Windows Embedded CE 6.0 R3 documentation.

Rebuild the run-time image, and download it to the device through the connection that you have already configured.

For more information, see “Build a Run-Time Image” and “Download a Run-Time Image” in the Windows Embedded CE 6.0 R3 documentation.

Conclusion

After you have completed this tutorial, you will have a custom user control that uses XAML to define its appearance, is implemented in C++, and is accessible from the visual tree by using the **FindName** method. You can reuse your custom user control across multiple Silverlight for Windows Embedded applications.

Custom controls provide a way for you to design and implement a unique control that meets the particular needs of a Silverlight application for a Windows Embedded CE powered device.

To reuse your custom control, you can create another C++ instance of the custom user control in a Silverlight for Windows Embedded application.

For more information:

Windows Embedded Web site

Copyright