



## **Handling Events in Silverlight for Windows Embedded Compact 7**

Writer: Frankie Anderson

Technical Reviewer: David Franklin

Published: January 2012

Applies To: Windows Embedded Compact 7

### **Abstract**

This paper describes window event handling in Silverlight for Windows Embedded and details how it differs from Silverlight for Windows. It covers the following topics:

- Event handling differences
- Default and user-defined event handling
- How to create a user-defined event handler
- How to modify UI objects on load

# Contents

Introduction .....	3
Silverlight Event Processing Differences .....	3
Silverlight for Windows Embedded Compact 7 Event Handling .....	4
Default Event Handling.....	4
User-Defined Event Handling .....	7
Create a User-Defined Event Handler .....	8
Step 1: Create Your Application UI by Using Microsoft Expression Blend 3 .....	8
Step 2: Import the Expression Blend Project into a Silverlight for Windows Embedded Compact 7 Application Subproject.....	8
Step 3: Add Custom Code to Your Application to Implement Your Event Handler .....	8
Step 4: Build the Application .....	11
User-Defined Event Handling Example.....	11
Creating or Modifying UI Objects in C++ When a Silverlight Application Loads .....	13
Conclusion .....	13
Additional Resources .....	13

# Introduction

---

Silverlight for Windows Embedded Compact 7 OEMs create rich user interfaces (UIs) for their devices by defining their visual appearance, functionality, and behavior with a combination of source XAML and native C++ application programs. OEM designers use Microsoft Expression Blend 3 to generate XAML to define UI elements, their fundamental appearance, and behavior. OEM developers use the C++ methods provided in the Silverlight for Windows Embedded Application Programming Interface (API) to independently write applications that:

- Parse XAML-defined UI elements into the C++ object tree.
- Create Silverlight controls, window controls and other UI elements and add them to the object tree.
- Load, display and animate the UI elements.
- Implement event handling for the UI elements.
- Further customize the UI at run-time.

To implement and handle events in Silverlight applications, you must be aware of some specific details of Windows Embedded Compact 7. Windows Embedded and Windows event processing models are not identical. You must understand the Silverlight for Windows Embedded Compact 7 model to avoid unanticipated behavior in your applications. This paper discusses the implications of the Silverlight for Windows Embedded implementation differences for user-defined event handling in applications.

## Silverlight Event Processing Differences

---

The major differences between Silverlight for Windows Embedded Compact 7 and Silverlight for Windows (that is, desktop computers) are detailed in the Windows Embedded Compact 7 documentation. Generally, Silverlight for Windows Embedded Compact 7 events are equivalent to events in Silverlight for Windows. Silverlight for Windows Embedded Compact 7 provides the same event handling and routing services as those provided by the desktop version, but the Windows Embedded Compact 7 implementation differences are significant for application developers. A summary of the differences between Silverlight for Windows and Silverlight for Windows Embedded Compact 7 that are relevant to event handling are shown in Table 1.

**Table 1 Event Handling Differences**

Silverlight for Windows	Silverlight for Windows Embedded Compact 7
To handle events yourself in code, override the method in a subclass of the object. Setting the handled flag in the event data will stop event routing to other objects in the visual tree.	To handle events yourself in code, use the <b>Add*Handler</b> and <b>Remove*Handler</b> public methods on UI objects. Setting the handled flag in the event data will stop event routing to other objects in the visual tree.

Silverlight for Windows	Silverlight for Windows Embedded Compact 7
To additionally handle events that are marked as handled (handled flag set), subclass from the object class and use the <b>AddHandler</b> method.	<b>AddHandler</b> not supported for subclasses.
User-defined overrides of event handlers are always invoked instead of default event handlers.	User provided event handlers are always invoked before default or internal object event handlers.
Event data is stored in classes that inherit from <b>System.EventArgs</b> .	Event data is stored in structures that inherit from <b>XREventArgs</b> , which are used to specify the type of data that is contained in the <b>IXRDelegate&lt;ArgType&gt;</b> objects used to handle specific types of events.
Supports event binding directly in XAML through attribute values.	Supports event binding only in C++ code by attaching event handlers to UI objects.
Raises the <b>GotFocus</b> event when the focus is automatically set on the first focusable element in the visual tree during application initialization.	Sets the focus to the first focusable element in the visual tree. However, the <b>GotFocus</b> event is not raised unless you explicitly call <b>IXRControl::Focus</b> in the initialization procedure.

## Silverlight for Windows Embedded Compact 7 Event Handling

Silverlight for Windows Embedded Compact 7 hosts the object tree in a Silverlight visual host with an underlying Win32 window. The Silverlight for Windows Embedded Compact 7 visual host provides event handling for all UI objects in the object tree. The Graphics, Windowing, and Events Subsystem (GWES) provides the interface between the OS, the Win32 API, the graphics device interface (GDI), and the UI objects in your Silverlight visual tree.

### Default Event Handling

The Silverlight for Windows Embedded Compact 7 visual host propagates unhandled events through the object tree. A default scenario, which illustrates how events that are generated as a result of a user left mouse button click are “bubbled up” from the object that raises the event to successive parent objects, is shown in Figure 1. This Silverlight example contains six UI elements in the object tree: a

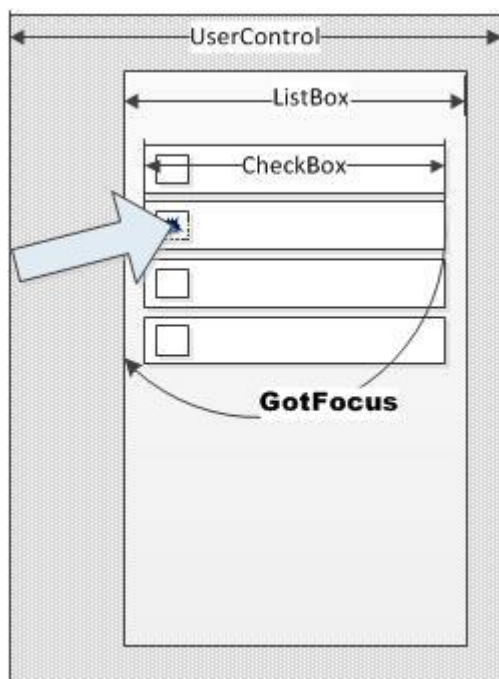
**UserControl** at the root, a **ListBox** child of the **UserControl**, and four **CheckBox** children of the **ListBox**. Assume the default **ClickMode** configuration for the check box controls in this example so that each **CheckBox** control automatically gets focus when a **MouseLeftButtonDown** event occurs within the control. Each control also generates a **Click** event if it has focus and there is a **MouseLeftButtonUp** event inside the control. No user-defined event handling occurs in this example.

Suppose the user moves the mouse into the second check box and presses down on the left mouse button as shown by the arrow in Figure 1. The relevant events for this action are **MouseEnter**, **MouseLeftButtonDown**, and **GotFocus**. In this case, the default event handling sequence is:

1. **MouseEnter** on the **ListBox**
2. **MouseEnter** on the **CheckBox**
3. **MouseLeftButtonDown** on the **CheckBox**
4. **GotFocus** on the **CheckBox**
5. **GotFocus** on the **ListBox**

The **GotFocus** event is routed up the visual tree from the **CheckBox** to the **ListBox**, as shown in Figure 1.

**Figure 1 MLBD Default Event Handling**



Default event handlers are commonly used to suppress routed events that a particular UI object implementation does not propagate further or to provide special handling of that routed event that is a feature of the object. For instance, a UI object might raise its own object-specific event that contains more specifics about what some user input condition means in the context of that particular object. The

object implementation might then mark the more general routed event as handled. Object event handlers are typically implemented so that they are not invoked for routed events where shared event data was already marked handled. There are some built-in Silverlight for Windows Embedded object event handlers that set the handled flag on the event data by default. In this case, the affected event will not be routed to the parent of the control.

Particular Silverlight for Windows Embedded event routing and event handlers are designed to work by default in the same way that they do on the desktop. Properties of the events and default event routing information can be found in the desktop documentation for the specific control and event.

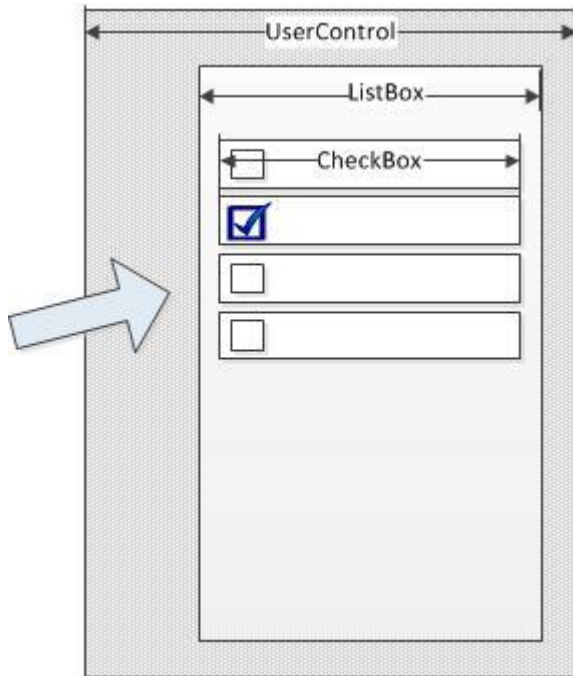
In the example illustrated in Figure 1, the **MouseLeftButtonDown** and **MouseLeftButtonUp** event handled flags are set to **true** by the default event handler, and depending specifically on the configured **ClickMode** of the **CheckBox** control, the more specific **Click** and **Checked** events are raised. Neither the **MouseLeftButtonDown** nor the **MouseLeftButtonUp** events are routed to the **ListBox**.

Figure 2 shows the continuation of this scenario. When the user releases the left mouse button and moves the mouse away from the **CheckBox** and **ListBox** elements, the default event handling sequence is:

1. **MouseLeftButtonUp** on the **CheckBox**
2. **Checked** on the **CheckBox**
3. **Click** on the **CheckBox**
4. **MouseLeave** on the **CheckBox**
5. **MouseLeave** on the **ListBox**

No additional events are bubbled up to the **ListBox** from the **CheckBox**, the **MouseLeave** events are generated as the mouse leaves the **CheckBox** and the **ListBox**, respectively.

Figure 2 MLBU Default Event Handling



## User-Defined Event Handling

You can handle UI events in Silverlight for Windows Embedded Compact 7 applications by using C++ delegates attached to UI objects. The Silverlight for Windows Embedded API includes a set of C++ **Add\*EventHandler** methods to define these delegates and to specify the user-generated methods that the delegates point to for handling events. Two implementation details are especially important in designing your own event handlers:

- User-defined object event handlers that are attached with **Add\*EventHandler** methods are always invoked in addition to and prior to internally defined or default object event handlers.
- If a user-defined event handler sets the handled flag of the **XREventArgs** structure to **true**, internal or default handlers detect that the event has been previously handled and will not perform default handling tasks. In addition, events with the handled flag set to **true** will not be routed to parent objects in the visual tree.

When you set the value of the **Handled** property to **true** in the event data for a routed event, this is referred to as "marking the event handled". There is no absolute rule for when you must mark routed events as handled. Basically, there are two reasons to mark an event handled:

- You do not want the default behavior that is provided by the internal event handling of the control.
- You do not want the behavior that is provided by the event handling of a parent of the control; that is, you do not want event routing.

Typically, only one handler implementation should exist for responding to any single routed event occurrence. If more responses are needed, then implement the necessary code through application logic that is chained within a single handler rather than by using the routed event system for forwarding.

To create an application that implements customized event handlers, use the techniques and procedures that are detailed in [A Sample Application Tutorial Using Windows Embedded Silverlight Tools](http://go.microsoft.com/fwlink/?LinkID=189508) (<http://go.microsoft.com/fwlink/?LinkID=189508>).

## Create a User-Defined Event Handler

To create an application that implements a user-defined event handler for the example, the following steps are required.

### Step 1: Create Your Application UI by Using Microsoft Expression Blend 3

Using the example from the previous section of this article, suppose you want to write a customized event handler for the **MouseLeftButtonDown** event on the **CheckBox** controls that are shown in Figures 1 and 2. Create a Windows Embedded Silverlight project in Microsoft Expression Blend 3 that contains the UI elements shown. Save your project so that you can import it into a Platform Builder project in Microsoft Visual Studio.

### Step 2: Import the Expression Blend Project into a Silverlight for Windows Embedded Compact 7 Application Subproject

Next, you import the Expression Blend project as an application subproject in Silverlight for Windows Embedded Compact 7.

#### To import an Expression Blend project into an application subproject

1. Open a Platform Builder OS design that supports Windows Embedded Silverlight.
2. On the **Tools** menu, click **Windows Embedded Silverlight Tools**, then click **Create Platform Builder Subproject** to open the Platform Builder Subproject Application Wizard.
3. Step through the wizard to create the Platform Builder application subproject directory and to import the UI project you saved in step 1.
4. Add this newly created subproject to your OS design.

### Step 3: Add Custom Code to Your Application to Implement Your Event Handler

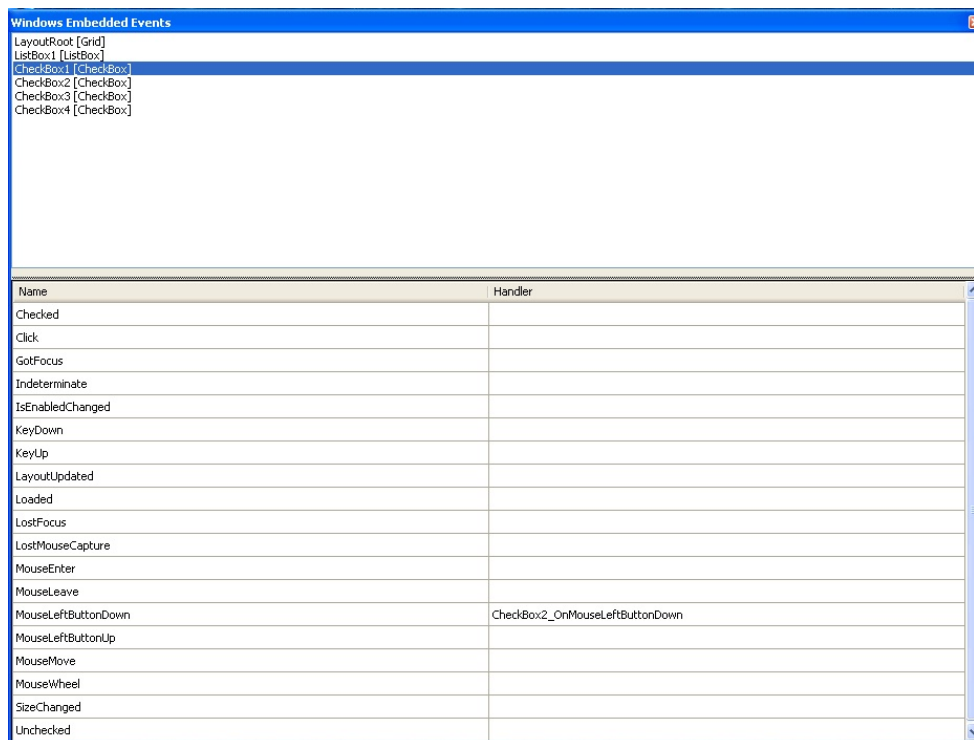
Next, you add the C++ code for your event handler to your Platform Builder application.

► **To add a customized event handler**

1. In Microsoft Visual Studio, in the Resource files folder of your application subproject, open the MainPage.xaml file.
2. On the **Tools** menu, click **Windows Embedded Silverlight Tools**, then click **Windows Embedded Events** to open the **Windows Embedded Events** window. The UI elements of the application are listed in the upper pane of the window and the possible events for the selected UI element are listed in the lower pane of the window.
3. To add code for your event handler using the **Windows Embedded Events** window, select a check box control in the upper pane of the **Windows Embedded Events** window and type the name of the event handler method in the **Handler** column for the **MouseLeftButtonDown** event.

When you press the ENTER key or otherwise move the mouse away from the Handler entry field, the system automatically adds C++ code to the MainPage.h and MainPage.cpp files for the named event handler. You then edit this code using Microsoft Visual Studio to implement your customized event handling functionality.

**Figure 3 Windows Embedded Events Window**



For the control and handler pictured in Figure 3, entering the handler name causes the system to add the following method definition to MainPage.h:

```
HRESULT CheckBox2_OnMouseLeftButtonDown (IXRDependencyObject* pSender,
XRMouseButtonEventArgs* pArgs);
```

The system adds the following call to **AddMouseLeftButtonDownEventHandler** for **CheckBox2** in the **MainPage::OnLoaded** event handler of MainPage.cpp:

```
if (m_pCheckBox2)
{
    m_pCheckBox2->AddClickEventHandler(CreateDelegate(this,
&MainPage::CheckBox2_OnMouseLeftButtonDown));
}
```

The system also adds a stub for the event handling method itself to the MainPage.cpp file:

```
// =====
// OnMouseLeftButtonDown
//
// Description: Event handler implementation
//
// Parameters: pSender - The dependency object that raised the click event.
//             pArgs - Event specific arguments.
// =====
HRESULT MainPage::CheckBox2_OnMouseLeftButtonDown (IXRDependencyObject* pSender,
XRMouseButtonEventArgs* pArgs)
{
    HRESULT hr = E_NOTIMPL;

    if ((NULL == pSender) || (NULL == pArgs))
    {
        hr = E_INVALIDARG;
    }

    return hr;
}
```

You then add to and modify the code to implement the functionality that you want for the event handler. Note that any modifications that you make to code in MainPage.h and MainPage.cpp will be overwritten

if you use the **Update Silverlight for Windows Embedded Project** menu option to update your application with changes made in the Expression Blend UI project. Also note that modifications you make to the application code by using Microsoft Visual Studio are not recognized by the Windows Embedded Events window.

For the purpose of our example here, add code to the above event handler that sets **Handled = true** in the **XEventArgs**-derived structure for the control. Assume the following line is in the code immediately before the return statement:

```
pArgs->Handled = true;
```

## Step 4: Build the Application

When all code changes are complete, the final step is to build your application.

### To build the application

1. On the **Build** menu, click **Build All Subprojects**.

## User-Defined Event Handling Example

Suppose you execute the sample application that you created in the last section, then you move your mouse into the second **CheckBox** and press the left mouse button, as depicted by the arrow in Figure 4. The relevant events for this action are **MouseEnter**, **MouseLeftButtonDown**, and **GotFocus**. In this case, the event sequence is:

1. **MouseEnter** on the **ListBox**
2. **MouseEnter** on the **CheckBox**
3. **MouseLeftButtonDown** on the **CheckBox**

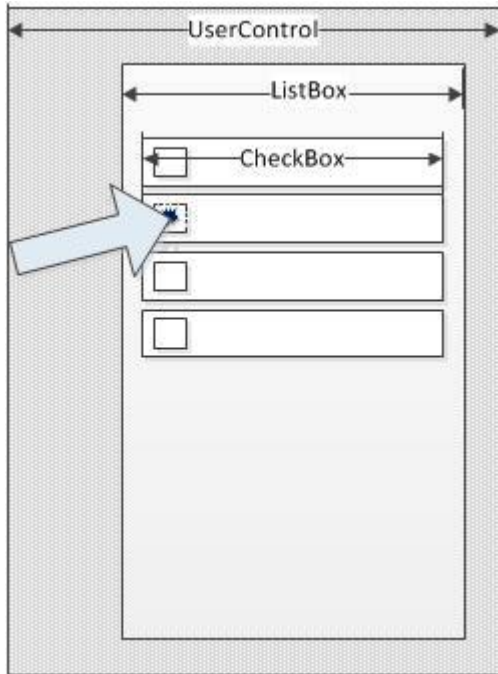
In this scenario, your event handler for **MouseLeftButtonDown** in the **CheckBox** executes first and sets the handled flag to **true**. The default **MouseLeftButtonDown** event handler then executes and attempts to process the event, but because the handled flag is already set, no further actions are taken. Note also that there is no change in focus in this scenario because the default event processing does not occur until after the handled flag is set.

If we continue the scenario, when the **MouseLeftButtonUp** event occurs on the **CheckBox**, the default object event handler executes, but since the handled flag is set, no **Click** or **Checked** events are generated. When the mouse is moved back out of the **CheckBox** and **ListBox**, only the following events occur:

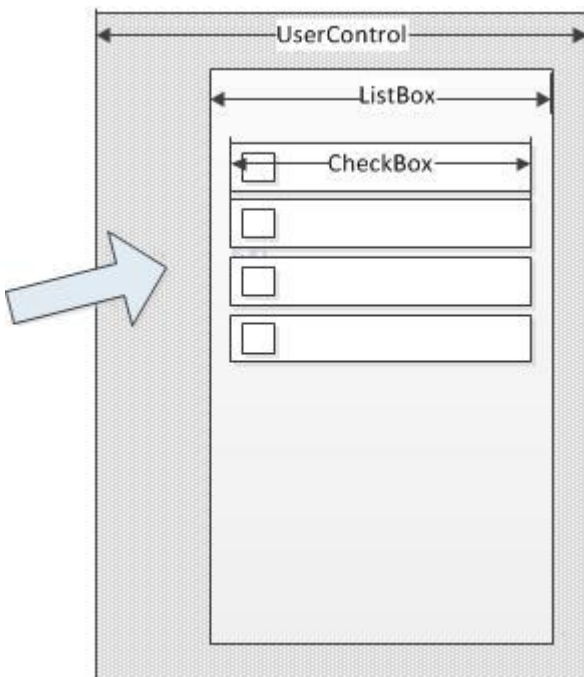
1. **MouseLeftButtonUp** on the **CheckBox**
2. **MouseLeave** on the **CheckBox**
3. **MouseLeave** on the **ListBox**

The state of the controls after this scenario ends is shown in Figure 5.

**Figure 4 MLBD User-Defined Event Handling**



**Figure 5 MLBU User-Defined Event Handling**



## Creating or Modifying UI Objects in C++ When a Silverlight Application Loads

If you want to create UI objects by using the Silverlight C++ class library and add them to your Silverlight for Windows Embedded application when the application loads, add the C++ code to the **LayoutUpdated** event handler. By adding the UI objects in response to the **LayoutUpdated** event, you can guarantee that the visual tree is fully instantiated before you attempt to add or modify UI objects in the visual tree.

Be aware that Silverlight raises the **LayoutUpdated** event each time that the visual layout is updated, so make sure that your code responds only to the first **LayoutUpdated** event that Silverlight raises when the application loads.

If you use the **Windows Embedded Events** window to add an event handler to this event, it automatically adds the **LayoutUpdated** event handler to the application source code and also adds code that calls the method **AddLayoutUpdatedEventHandler** to attach the **LayoutUpdated** event handler to the main Canvas object for the application.

## Conclusion

---

Event handling in Silverlight for Windows Embedded differs from event handling in Silverlight for Windows because of implementation differences. These differences are significant if you implement user-defined event handlers for your applications. Default event handling behavior is the same in both versions. This article walks through examples of event handling in default and user-defined scenarios. This article provides basic information about how to implement customized event handlers in Silverlight for Windows Embedded Compact 7.

## Additional Resources

---

- [Windows Embedded website](http://www.microsoft.com/windows/embedded/default.mspx) (http://www.microsoft.com/windows/embedded/default.mspx)
- [A Sample Application Tutorial Using Windows Embedded Silverlight Tools](http://go.microsoft.com/fwlink/?LinkID=189508) (http://go.microsoft.com/fwlink/?LinkID=189508)

This document is provided “as-is.” Information and views expressed in this document, including URL and other Internet Web site references, may change without notice. You bear the risk of using it.

This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.

© 2011 Microsoft. All rights reserved.