Windows Embedded Compact

# A Sample Application Tutorial Using Windows Embedded Silverlight Tools

Writer: Mark McLemore

Technical Reviewers: Natalie Cervantes, Paul Monson

Published: March 2011

Applies To: Windows Embedded Compact 7

## Abstract

Describes how to use Windows Embedded Silverlight Tools with a simple Expression Blend sample project.

- Create an application template from the Expression Blend project
- Add C++ code to the template to create a functioning application
- Add event handlers to the application using the Windows Embedded Silverlight Tools event handler view
- Update the application with changes from an updated version of the Expression Blend project

# Contents

# Introduction

Windows Embedded Silverlight Tools is a plug-in for Visual Studio 2008 SP1 that helps you convert Expression Blend Silverlight projects into native (C++) Windows Embedded Compact applications. The Silverlight for Windows Embedded Platform Builder Subproject Application Wizard walks you through the steps for creating a template Platform Builder subproject from the XAML (eXtensible Application Markup Language) and resource files of an Expression Blend project. Starting with this generated template, you can easily add functionality to create a native Windows Embedded Compact application that has a Silverlight user interface (UI) designed in Expression Blend. When you are ready to create event handlers for your Silverlight UI objects, the Event Handler View allows you to select objects and events (such as KeyDown and MouseLeftButtonDown) to associate with those objects, automatically generating C++ stub event handlers when you add new events. You can later modify your Silverlight project in Expression Blend and merge these changes into your Platform Builder subproject using the Windows Embedded Silverlight Tools update tool. Windows Embedded Silverlight Tools automates the process of using Expression Blend files to create Silverlight-based Windows Embedded Compact applications. This automation eliminates difficult and time-consuming tasks, diminishes the likelihood of introducing human error, and frees you to focus on the business logic of your application. The ability to merge UI updates during and after application development makes it possible for the designer and the developer to work on an application in parallel, so the application can make it to market faster.

## How Windows Embedded Silverlight Tools Work

When a designer creates graphics, animations, and user interface elements in Expression Blend, the resulting Expression Blend project contains XAML files that describe the visual components and user interface characteristics of an application. However, these XAML files may include elements that are not supported by Silverlight for Windows Embedded. For each of these XAML files, Expression Blend generates a .NET code-behind file that may contain C# application code. However, since Windows Embedded Compact applications must be written in C++ to use the Windows Embedded Compact native Silverlight runtime, you cannot use these Expression Blend-generated code-behind files as an application template. To use an Expression Blend project as the basis for a Windows Embedded Compact application, you need some way to convert Expression Blend project files to the tools and native runtime of Windows Embedded Compact.

Windows Embedded Silverlight Tools bridges this gap between the Expression Blend project and the Windows Embedded Compact development environment by providing the following features:

- Verifies incoming XAML files for use with Silverlight for Windows Embedded, locating and reporting any unsupported elements.
- Automatically generates template application C++ code from XAML source files.
- Provides a way to easily select objects, hook up events, and add event handlers to your Expression Blend-derived application.

- Automatically identifies changes in the Expression Blend project and propagates these changes to your application's project resource files.

Without Windows Embedded Silverlight Tools, you would have to manually parse each incoming XAML file to identify and remove unsupported elements. For large Expression Blend projects, this porting task could be a tedious and error-prone process. Since the code-behind files generated by Expression Blend are not suitable for use with Windows Embedded Compact, you would have to manually write your own complex application code to manage the Silverlight runtime and attach event handlers to XAML objects. When the designer makes changes to the source Expression Blend project, you would have to manually find and merge these changes into your application project. Windows Embedded Silverlight Tools merges Expression Blend project updates with a mouse click

# Limitations

Although Windows Embedded Silverlight Tools provides many capabilities for accelerating your application development, it does have several limitations.

- Windows Embedded Silverlight Tools does not create XAML files.
- Windows Embedded Silverlight Tools does not generate managed source code. Windows Embedded Silverlight Tools only generates C++ code to use with Windows Embedded Compact.
- Windows Embedded Silverlight Tools does not generate C++ code for the following XAML elements or their children.
  - ControlTemplate
  - DataTemplate
  - FrameworkTemplate
  - ItemsPanelTemplate
  - MediaElement
  - Resources
  - RenderTransform
  - Storyboard
  - Style
  - VisualState
  - VisualStateGroups
- Windows Embedded Silverlight Tools does not support custom controls. You must manually register any custom controls in your application. For more information about custom controls, see the section "Create a Custom User Control in Silverlight for Windows Embedded" in the article Silverlight For Windows Embedded Developer's Guide (http://go.microsoft.com/fwlink/?LinkID=210403).
- Windows Embedded Silverlight Tools does not handle manual modifications that you make to your Silverlight project outside of Expression Blend.

Windows Embedded Silverlight Tools is not a replacement for Expression Blend, Visual Studio, or Platform Builder; it complements these tools by providing an efficient way to use Expression Blend project files as the basis for Windows Embedded Compact applications. For this reason, Windows Embedded Silverlight Tools does not offer overlapping functionality that is already available in these other tools.

## Development Process

Typically, a designer creates a Silverlight-based user interface project in Expression Blend and passes this UI project to a developer. The developer, in turn, creates a Windows Embedded Compact application from this Expression Blend project. The developer writes and tests the code that implements the functionality of the application with UI resources from the Expression Blend project. To enable the designer and the developer to work on the application in parallel, they create a Designer/Developer contract that defines the interface between the UI and the business logic of the application. If the designer and the developer properly follow this contract, they can assemble the XAML from the Expression Blend project and the C++ files from the developer at any time to view the current state of the application.

Windows Embedded Silverlight Tools works on the assumption that you and your designer are developing your application according to a Designer/Developer contract, and that there is a clearly-defined interface between the UI and the application implementation. For more on this development process and the Designer/Developer contract, see the article Introduction to Application Development with Silverlight for Windows Embedded (http://go.microsoft.com/fwlink/?LinkId=183252).

## Common Tasks

Windows Embedded Silverlight Tools provides support for three common Silverlight for Windows Embedded development tasks.

- Generating an application template from an Expression Blend project.
- Adding event handlers.
- Updating your application with changes made to the Expression Blend project.

Each of these tasks is described in this article, with simple tutorial examples that demonstrate how Windows Embedded Silverlight Tools can help you quickly create Silverlight-based applications for Windows Embedded Compact.

# Installation

You can automatically install Windows Embedded Silverlight Tools and application templates for Expression Blend as part of the standard installation for Windows Embedded Compact 7, or you can install Windows Embedded Silverlight Tools and the Expression Blend templates as stand-alone components.

# Installing Windows Embedded Silverlight Tools and Expression Blend with Windows Embedded Compact 7

When you install Windows Embedded Compact 7 and choose **Full Install (installs all options available)** in the **Installation Options** dialog box, Windows Embedded Silverlight Tools and the Expression Blend application templates are automatically installed. If you choose **Custom install (select options to install)** in the **Installation Options** dialog box, select the following options in the **Customize Installation** dialog box to install Windows Embedded Silverlight Tools.

- Platform Builder
- Windows Embedded Silverlight Tools

You can verify that Windows Embedded Silverlight Tools was installed successfully if the **Windows Embedded Silverlight Tools** option is present in the **Tools** menu in Visual Studio.

# Installing Only Windows Embedded Silverlight Tools or Expression Blend Templates

You can also install Windows Embedded Silverlight Tools and the Expression Blend application templates without installing Windows Embedded Compact 7. During the installation process, you can install only Windows Embedded Silverlight Tools, only the Expression Blend templates, or both Windows Embedded Silverlight Tools and the Expression Blend templates.

Before installation, ensure that you have already installed the prerequisites.

1. Windows Embedded Silverlight Tools requires Visual Studio 2008 SP1.
2. If you are installing the Expression Blend templates, Expression Blend 3 must be installed. You can install it from the Expression Blend 3 download website (http://go.microsoft.com/fwlink/?LinkId=204751).

   **Note**

   This is a requirement because the Windows Embedded Silverlight Tools installer also installs the Expression Blend templates and can install the templates without installing the tools. The installer verifies that Expression Blend 3 is installed before it installs the Expression Blend templates. If you install Windows Embedded Silverlight Tools on a computer that does not have Expression Blend installed, you must uninstall Windows Embedded Silverlight Tools and install Expression Blend before you can successfully install Windows Embedded Silverlight Tools and the Expression Blend templates together
3. To install Windows Embedded Silverlight Tools or Expression Blend templates, go to the Windows Embedded website (http://go.microsoft.com/fwlink/?LinkID=183524) and click **Downloads**.

**To install Windows Embedded Silverlight Tools and Expression Blend together**

- Choose **Full Install**.

**To install only Windows Embedded Silverlight Tools or only Expression Blend templates**

1. Choose **Custom Install**.

2. To install Windows Embedded Silverlight Tools, choose **Windows Embedded Silverlight Tools**, or, to install the Expression Blend templates, choose **Silverlight for Windows Embedded Application Template for Expression Blend 3.0**

The Expression Blend templates are installed at %ProgramFiles%\Microsoft Expression\Blend 3\ProjectTemplates\en\Windows Embedded Silverlight Tools\Application. If you plan to use a non-English version of Expression Blend 3, you must manually copy the template files from %ProgramFiles%\Microsoft Expression\Blend 3\ProjectTemplates\en\Windows Embedded Silverlight Tools\Application to a directory you create in your local user directory: %Users%*user-name*\Documents\Expression\Blend 3\ProjectTemplates\Windows Embedded Silverlight Tools\Applications. After you manually copy the files to the new directory path, Expression Blend can display the templates in its **New Project** dialog box.

# Creating an Application Template

Windows Embedded Compact 7 offers two ways to create a Silverlight for Windows Embedded application template.

- Create the project using a Silverlight for Windows Embedded SDK. This method works if you do not have Platform Builder installed.
- Import an Expression Blend project into Platform Builder.

## Import Your Expression Blend Project

To create an initial Silverlight for Windows Embedded application template from an Expression Blend project, use the Windows Embedded Silverlight Tools Platform Builder Subproject Application Wizard. The Subproject Application Wizard generates a Platform Builder subproject from the resources and XAML files of the source Expression Blend project. After you incorporate this Platform Builder subproject into your OS design, you can develop, test, and deploy this application as you would any other Windows Embedded Compact application.

When you import your Expression Blend project into a Platform Builder subproject, follow these guidelines.

- Avoid making manual changes to the Expression Blend XAML files; always edit XAML files from within Expression Blend.
- If your Expression Blend project contains user controls, ensure that each user control has a unique name, even across namespaces.
- Verify that your Expression Blend project includes a default App.xaml file for the application class and MainPage.xaml (or Page.xaml) for the startup; do not rename these files.
- Name all XAML elements that you will attach to event handlers. If you want Windows Embedded Silverlight Tools to generate empty stub code for your event handlers, you must explicitly name these XAML elements from within Expression Blend.

- If you want to add XRPack identifiers to user controls in your Expression Blend project, add these identifiers before you import the project. For more on application packaging with XRPack, see Using the XAML Resource Packager (http://go.microsoft.com/fwlink/?LinkId=199905) in Windows Embedded Compact 7 documentation. For special XAML programming considerations related to XRPack, see Appendix: XAML Markup for XRPack.
- Finalize the Expression Blend project name and location, XAML file names, the .csproj file name, and all namespace names before you launch the Subproject Application Wizard. Changing the names or locations of any of these files and folders can break build dependencies between your subproject and the Expression Blend project.

A Silverlight Clock sample project is included in the Windows Embedded Silverlight Tools installation. This sample project provides the basis for the tutorials in this article.

**To prepare the Silverlight Clock sample project**

1. Browse to the WindowsEmbeddedSilverlightClock sample folder located at %ProgramFiles%\Windows Embedded Silverlight Tools.
2. Create a C:\ExpressionBlend working folder and copy the WindowsEmbeddedSilverlightClock sample project to this folder.

You will use the accompanying WindowsEmbeddedSilverlightClockUpdate folder later in this article to apply updates. When you merge these updates into your sample project, you will overwrite some files. You create a working copy of the sample folder in C:\ExpressionBlend to avoid overwriting the original sample project.

In the following steps you create and run a Silverlight-based Windows Embedded application, MyClock.exe, which is based on the Silverlight Clock sample project.

# Include Silverlight in Your OS Design

Before you can run Silverlight-based applications in Windows Embedded Compact, you must include the native Silverlight runtime in your device image.

**To include Silverlight in your OS design**

1. Launch Visual Studio.
2. In Platform Builder, open the OS design for your target device.

   If you do not have a device, you can use a virtual CEPC as your development target. For more about Virtual CEPC, see the article Getting Started with Virtual CEPC (http://go.microsoft.com/fwlink/?LinkID=190470) also available at %ProgramFiles%\Windows Embedded Compact 7\Documentation. If you use a virtual CEPC for your development target, configure your OS design using the Enterprise Device Handheld template as explained in Getting Started with Virtual CEPC (http://go.microsoft.com/fwlink/?LinkID=190470).
3. In Platform Builder, open the **Catalog Items View** for your OS design. If the **Catalog Items View** is not visible, follow these steps to make it visible:
   a. In Visual Studio, click the **View** menu.

b. Select **Other Windows** and then select **Catalog Items View**.

The **Catalog Items View** pane appears in Visual Studio.

4. Expand **Core OS**, **Windows Embedded Compact**, **Shell and User Interface**, and then **Silverlight for Windows Embedded**.
5. Under **Silverlight for Windows Embedded**, select the check box for **Silverlight for Windows Embedded**.

   This enables the Sysgen variable SYSGEN_XAML_RUNTIME in your OS design.
6. Expand **Core OS**, **Core OS Services**, and then **Kernel Functionality**.
7. Under **Kernel Functionality**, verify that the check box for **Target Control Support (Shell.exe)** is selected.

   This enables you to use the Target Control functionality in Platform Builder to communicate with an attached device, such as a virtual CEPC.
8. Build your OS design.

# Create a Platform Builder Subproject

To create an initial Windows Embedded Compact application template from the Expression Blend sample project, you use the Windows Embedded Silverlight Tools to Platform Builder Subproject Application Wizard. Perform the following steps to create a Platform Builder subproject for the MyClock application.

**To create the initial MyClock subproject from the Silverlight Clock sample**

1. Create an empty folder called MyClock to hold your Platform Builder subproject. For example, if the name of your OS design is VCEPC, create a MyClock folder in this location:

   C:\WINCE700\OSDesigns\VCEPC\MyClock
2. With your OS Design project open, on the **Tools** menu, click **Windows Embedded Silverlight Tools** and then click **Create Platform Builder Subproject**.
3. When the **Create Platform Builder Subproject** dialog box appears and displays the **Overview**, click **Next**.
4. In the **Project Name** dialog box, type “MyClock” for the name.
5. Browse to the location of your empty Platform Builder subproject MyClock folder. For example:

   C:\WINCE700\OSDesigns\VCEPC\MyClock

   Click **Next**.
6. In the **Project Selection** dialog box, click **Browse** and then navigate to the .csproj file of the Silverlight Clock sample that you copied to C:\ExpressionBlend earlier. This file is at the following location:

   C:\ExpressionBlend\WindowsEmbeddedSilverlightClock\WindowsEmbeddedSilverlightClock.csproj
7. Select the .csproj file and then click **Open**.

8. Under **Project Contents**, in the **Project Selection** dialog box, verify that the **Project Selection** dialog box displays MainPage.xaml and then click **Next**.
9. Verify that the **Validation** dialog box shows **No Errors Reported** to indicate that Windows Embedded Silverlight Tools imported the Expression Blend project successfully.

   If you see the error message **The project file you selected is either invalid or corrupt**, then Windows Embedded Silverlight Tools was unable to create a Platform Builder subproject from the Expression Blend project. For more information about resolving validation errors, see the Troubleshooting section.
10. Click **Finish** to view the **Summary Report Change Log**.

**Important**

After you have created a Platform Builder subproject, do not move or rename the associated Expression Blend project folder, XAML files, or .csproj file. Your Platform Builder subproject depends upon the .csproj and XAML files of the associated Expression Blend project.

# Add the Subproject to Your OS Design

To build and test the application template produced by the Subproject Application Wizard, you must add the generated Platform Builder subproject to your OS design.

**To add the MyClock subproject to your OS design**

1. Open your OS design project in Platform Builder.
2. On the **Project** menu, click **Add Existing Subproject**.
3. In the new Windows Explorer window, browse to the location of your MyClock project. For example:

   C:\WINCE700\OSDesigns\VCEPC\MyClock\MyClock
4. Select the .pbpxml file associated with the MyClock subproject, MyClock.pbpxml. Click **Open**.

# Build the Application Template

Once you have incorporated this subproject into your OS design, you can compile and link the generated application template to run on your target device.

**To build MyClock.exe**

1. Select your OS design in Platform Builder. In the **Solution Explorer** pane, expand **Subprojects** to view all subprojects.
2. Locate and right-click the **MyClock** subproject that you created earlier, then click **Build**.
3. Verify that the output window displays the following message:

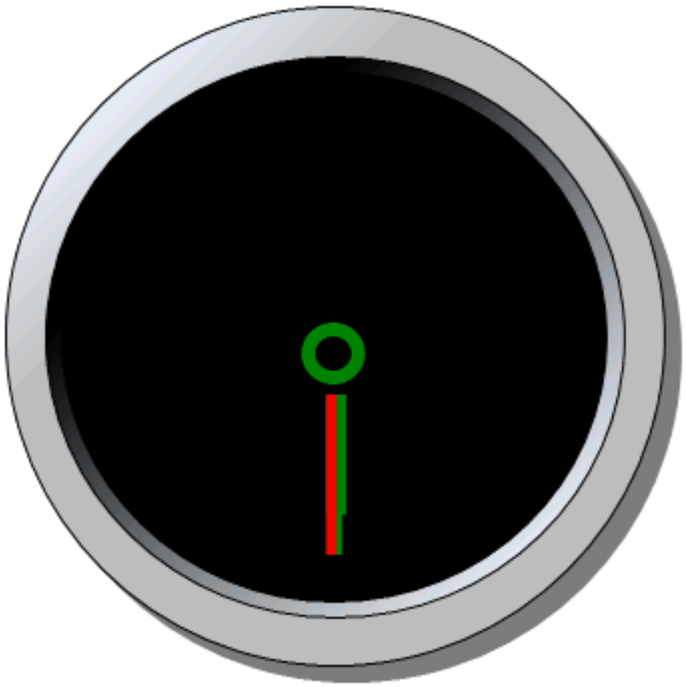
# Run the Application Template

When the subproject builds successfully, you can run the resulting application, MyClock.exe, on your target device by performing the following steps.

**To run MyClock.exe on your target device**

1. Connect and boot up your device in Platform Builder. If you are using virtual CEPC, be sure that your virtual CEPC is running and connected to Platform Builder as described in the “Download and Boot Your OS Image” section of Getting Started with Virtual CEPC (http://go.microsoft.com/fwlink/?LinkID=190470).
2. On the **Target** menu, click **Run Programs**.
3. In the **Run Program** dialog box, under **Available Programs**, select **MyClock.exe** and click **Run**.
4. Verify that your target device displays the face of the Silverlight Clock with stationary hour, minute, and second hands pointing downward at six o’clock as shown in the following figure.

**Running the Application Template**



The hands of MyClock do not yet move; you will add functionality later in this article so that MyClock can display the current time.

The Subproject Application Wizard creates enough C++ code in the application template to do the following:

- Initialize the Windows Embedded Compact Silverlight runtime.
- Instantiate XAML-declared objects in the Silverlight runtime.

Although the C# code-behind files of the source Expression Blend project contain functionality to animate the hands of the clock, Windows Embedded Silverlight Tools does not port this code for you. In the next section, you will add C++ code to animate the hands of MyClock.exe so that it can keep time.

# Develop Your Application

After you have imported the Expression Blend project into a Platform Builder subproject and verified that the resulting application template builds and executes on your target device, you can add C++ code to make the application work.

## Examine the Generated Application Template

When the Platform Builder Subproject Application Wizard creates the initial MyClock application template, it generates a source tree that contains files associated with the application. You can examine this source tree at the following location:

C:\WINCE700\OSDesigns\VCEPC\MyClock\MyClock

The following table summarizes the important source files that make up the MyClock application.

| Name | Description |
|---|---|
| WinMain.cpp | This file contains the main entry point of the application and normally requires no changes. |
| App.cpp and App.h | These files contain the derived application class. |
| MainPage.cpp and MainPage.h | These files implement the minimum code required to use the `<UserControl>` defined in MainPage.xaml. These files may also contain member variables that are mapped to named objects and stubs for event handlers. You must complete these event handler implementations to get a working application. |
| MyClock.rc | This is a listing of all of the Windows resources that the program uses when compiling for a platform that uses the standard Windows Embedded Compact shell. It includes the XAML files, images, and other assets that are stored |

| Name | Description |
| --- | --- |
| | in the resources subdirectory. MyClock.rc can be edited directly in Visual Studio. |
| resource.h | This is the standard header file, which defines new resource IDs. Visual Studio reads and updates this file automatically. |

## Port Code-Behind Files

This section describes how to modify App.h to add application functionality to the Silverlight Clock.

Typically, the source Expression Blend project will have existing .NET code-behind files that the designer uses for demonstrating and testing the user-interface design. You can save development time by porting this existing functionality to your Windows Embedded Compact application.

The Silverlight Clock sample contains several C# code-behind files that animate the hands of the clock face. To add functionality to MyClock, you can port functionality from the Expression Blend project C# code-behind file associated with MainPage.xaml (MainPage.xaml.cs) to C++ code that works with the native Silverlight runtime of Windows Embedded Compact.

The Silverlight Clock sample uses the storyboard and animation features of Silverlight to move the hands around the clock face. In MainPage.xaml (in the folder WindowsEmbeddedSilverlightClock), a `Storyboard` object is declared with three child `DoubleAnimation` objects, one for each hand of the clock:

```
<Storyboard x:Name="ClockStoryboard">
    <!-- This animation targets the hour hand transform. -->
    <DoubleAnimation x:Name="HourAnimation"
            Storyboard.TargetName="HourHandTransform"
            Storyboard.TargetProperty="Angle"
            Duration="12:0:0" RepeatBehavior="Forever" />
    <!-- This animation targets the minute hand transform. -->
    <DoubleAnimation x:Name="MinuteAnimation"
            Storyboard.TargetName="MinuteHandTransform"
            Storyboard.TargetProperty="Angle"
            Duration="1:0:0" RepeatBehavior="Forever" />
    <!-- This animation targets the second hand transform.  -->
    <DoubleAnimation x:Name="SecondAnimation"
             Storyboard.TargetName="SecondHandTransform"
```

```
            Storyboard.TargetProperty="Angle"
            Duration="0:1:0" RepeatBehavior="Forever" />
</Storyboard>
```

As shown in the XAML excerpt above, each animation declares a duration time for the animation of the associated clock hand. The hour hand animation has a `Duration` of 12 hours, the minute hand animation has a `Duration` of one hour, and the second hand has a `Duration` of one minute. The `RepeatBehavior` of each animation is set to `Forever`, which means that each animation will start over at the end of its cycle and repeat indefinitely, moving the hands continuously around the clock face with durations that correspond to the hours, minutes, and seconds as appropriate for each hand.

Silverlight does the work of animating the hand objects declared in this XAML file. Therefore, the core logic of your Windows Embedded Compact application only has to perform the following tasks:

1. Determine the current time.
2. Calculate start angles for each clock hand.
3. Find the clock face objects in the Silverlight visual tree.
4. Initialize the clock face objects and start the animation.

The following subsections describe how to port this functionality to Windows Embedded Compact. We begin by showing you snippets of the original Silverlight Clock sample source in C#, explain what each code snippet does, and then show you the equivalent C++ version of that code for use in Windows Embedded Compact. The final code example lists the complete C++ code sample so that you can cut and paste it into a MyClock source file to try it out.

**Important**

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

## Determine The Current Time

In the code-behind file of the Silverlight Clock sample, **DateTime.Now** provides the current time:

```
// Get the current date and time.
System.DateTime date = DateTime.Now;
```

In Windows Embedded Compact, you can get the current time by calling the **GetLocalTime** function:

```
// Get the current time.
SYSTEMTIME st;
GetLocalTime(&st);
```

After you call **GetLocalTime**, you read the current time in hours, minutes, and seconds from the **SYSTEMTIME** structure. Next, you will use these values to calculate the start angles of the clock hands.

## Calculate Start Angles

The code-behind file of the sample Silverlight Clock has C# source code that calculates the starting angle, in degrees, of each hand corresponding to the current time. Since the XAML declares the initial position of each hand in a pointing-downward position (at six o'clock) when the application starts, the sample code adds 180 degrees to each hand so that the time/angle calculation is made relative to the twelve o'clock position on the clock face.

```
// Calculate the initial angle (in degrees) for the hour
// hand based on the current time:
double hourAngle = (((float)date.Hour) / 12) * 360 +
                    date.Minute / 2;
hourAngle += 180;

// Similarly, calculate the initial angle of the minute hand:
double minuteAngle = (((float)date.Minute) / 60) * 360;
minuteAngle += 180;

// Similarly, calculate the initial angle of the second hand:
double secondAngle = (((float)date.Second) / 60) * 360;
secondAngle += 180;
```

This code can be readily ported to C++ for use in the MyClock Windows Embedded Compact application. The **SYSTEMTIME** structure provides the current hours, minutes, and seconds to use in the same calculations as shown previously:

```
// Initialize hand angles to their current position at 6 o'clock:
float hourAngle = 180.0, minuteAngle = 180.0, secondAngle = 180.0;

// Calculate the initial angle (in degrees) for each
// hand based on the current time.
hourAngle   += (((float)st.wHour) / 12) * 360 + (float)st.wMinute / 2;
minuteAngle += (((float)st.wMinute) / 60) * 360;
secondAngle += (((float)st.wSecond) / 60) * 360;
```

Once these initial angle values are calculated, you use them to position the hands of the clock to the current time before starting the animation of the clock face.

## Find the Storyboard and Animation Objects

When the Platform Builder Subproject Application Wizard creates the initial application template, it does not generate named variables for all of the objects declared in the XAML. Instead, you must manually declare **IXRStoryboardPtr** and **IXRDoubleAnimationPtr** smart pointers and use the **FindName** method of the root element to locate the objects declared in the XAML file. In the case of the MyClock example, you use **FindName** to locate the `ClockStoryboard` object and the three animation objects corresponding to each of the clock hands. For each object, you pass the `x:Name` property declared in the XAML for that object to the first argument of **FindName**:

```
IXRFrameworkElementPtr pRoot;
IXRStoryboardPtr pStoryboard;
IXRDoubleAnimationPtr pSecondAnimation;
IXRDoubleAnimationPtr pMinuteAnimation;
IXRDoubleAnimationPtr pHourAnimation;

// Get the root element of the visual host:
m_pVisualHost->GetRootElement(&pRoot);

// Find the storyboard and animation objects:
pRoot->FindName(L"ClockStoryboard",&pStoryboard);
pRoot->FindName(L"SecondAnimation", &pSecondAnimation);
pRoot->FindName(L"MinuteAnimation", &pMinuteAnimation);
pRoot->FindName(L"HourAnimation", &pHourAnimation);
```

Note that the Silverlight Clock sample code-behind does not use a **FindName** method to locate the `Storyboard` and animation objects. In the C# source code of the Silverlight Clock sample, Expression Blend creates a named variable for each object declared in the XAML, so the Silverlight Clock sample code simply references each of these objects by name without having to perform a lookup of each object. The application template produced by Windows Embedded Silverlight Tools does, however, requires that you locate each object in the Silverlight visual tree and assign it to a smart pointer before use.

## Initialize and Start the Animation

Each clock hand animation is associated with a `RotateTransform` in MainPage.xaml. For example, here is the XAML declaration for the `RotateTransform` of the clock's second hand:

```
<Rectangle.RenderTransform>
    <RotateTransform x:Name="SecondHandTransform" />
</Rectangle.RenderTransform>
```

The XAML declaration for the animation object of the second hand sets the `Storyboard.TargetName` to the `SecondHandTransform` declared above:

```
<!-- This animation targets the second hand transform.  -->
    <DoubleAnimation x:Name="SecondAnimation"
             Storyboard.TargetName="SecondHandTransform"
             Storyboard.TargetProperty="Angle"
             Duration="0:1:0" RepeatBehavior="Forever" />
```

Each clock hand animation declares its `TargetProperty` to be the `Angle` property of the associated `RotateTransform`. Therefore, changing the `From` and `To` properties of a clock hand animation will set the `Angle` property of its associated `RotateTransform`. To prepare the clock for animation, initialize the `From` property to the initial angle of the animation for that clock hand, and set the `To` property of that hand to one complete sweep around the clock face from the starting point (that is, add 360 degrees). The C# code in the Silverlight Clock sample sets each of these properties directly and starts the animation by calling the **Begin** method on the parent object, **ClockStoryboard**:

```
// Set the beginning and end of the hour hand animation.
HourAnimation.From = hourAngle;
HourAnimation.To = hourAngle + 360;

// Set the beginning and end of the minute hand animation.
MinuteAnimation.From = minuteAngle;
MinuteAnimation.To = minuteAngle + 360;

// Set the beginning and end of the second hand animation.
SecondAnimation.From = secondAngle;
SecondAnimation.To = secondAngle + 360;

// Start the storyboard.
ClockStoryboard.Begin();
```

Notice that the C# sample code sets the `From` and `To` properties of each `DoubleAnimation` instance directly. In the C++ environment of Windows Embedded Compact, you use the **SetFrom** and **SetTo** methods to initialize instances of the `DoubleAnimation` class:

```
// Set the beginning and end of the hour hand animation.
```

```
pHourAnimation->SetFrom(hourAngle);
pHourAnimation->SetTo(hourAngle + 360);

// Set the beginning and end of the minute hand animation.
pMinuteAnimation->SetFrom(minuteAngle);
pMinuteAnimation->SetTo(minuteAngle + 360);

// Set the beginning and end of the second hand animation.
pSecondAnimation->SetFrom(secondAngle);
pSecondAnimation->SetTo(secondAngle + 360);

// Start the animation:
pStoryboard->Begin();
```

Once the animations are initialized with these beginning and end values, the call to the **Begin** method of the **Storyboard** starts the animation of all three clock hands.

In the next section, you will add this ported code to the application template created by the Windows Embedded Silverlight Tools using Platform Builder Subproject Application Wizard.

## Add Functionality to the Application Template

When the application template starts (see WinMain.cpp), it instantiates the **App** class, calls its **Initialize** method, then calls its **Run** method to execute the functionality of the application:

```
INT WINAPI WinMain(. . .)
{
    App AppInstance;
    HRESULT hr = AppInstance.Initialize(hInstance);
    if(SUCCEEDED(hr))
    {
        hr = AppInstance.Run();
    }
    return AppInstance.GetWinMainResultCode();
}
```

The **App::Initialize** method prepares the application to run your code by completing the following preliminary tasks:

1. Initializing the XAML runtime.

2. Retrieving the XAML runtime application instance.
3. Initializing the application's resource dictionary.
4. Creating the visual host object.

When **App::Initialize** completes successfully, the XAML runtime is ready and the application message loop for receiving events has started. You can add additional startup code by modifying the methods called by **App::Initialize** or by inserting calls to your own methods (see App.h).

After **App::Initialize** completes successfully, the application passes control to the **App::Run** method (see App.h). You use the **App::Run** method to launch the core functionality of your application. For the Silverlight Clock example, you will add code directly to **App::Run** to animate the second, minute, and hour hands of MyClock.exe. The code below shows what **App::Run** looks like after you add the code that you ported in the previous section:

```
inline HRESULT App::Run()
{
    HRESULT hr = E_FAIL;
    UINT uiExitCode = 0;

    IXRFrameworkElementPtr pRoot;
    IXRStoryboardPtr pStoryboard;
    IXRDoubleAnimationPtr pSecondAnimation;
    IXRDoubleAnimationPtr pMinuteAnimation;
    IXRDoubleAnimationPtr pHourAnimation;

    if(m_pVisualHost != NULL)
    {
        // Get the root element of the visual host:

        m_pVisualHost->GetRootElement(&pRoot);

        // Find the storyboard and animation objects:
        pRoot->FindName(L"ClockStoryboard",&pStoryboard);
        pRoot->FindName(L"SecondAnimation", &pSecondAnimation);
        pRoot->FindName(L"MinuteAnimation", &pMinuteAnimation);
        pRoot->FindName(L"HourAnimation", &pHourAnimation);
```

```
// Get the current time:
SYSTEMTIME st;
GetLocalTime(&st);

// Initialize hand angles to 6 o'clock:
float hourAngle = 180.0, minuteAngle = 180.0;
float secondAngle = 180.0;

// Calculate the initial angle (in degrees) for each
// hand based on the current time.
hourAngle   += (((float)st.wHour) / 12) * 360 +
               (float)st.wMinute / 2;
minuteAngle += (((float)st.wMinute) / 60) * 360;
secondAngle += (((float)st.wSecond) / 60) * 360;

// Set the beginning and end of the hour hand animation.
pHourAnimation->SetFrom(hourAngle);
pHourAnimation->SetTo(hourAngle + 360);

// Set the beginning and end of the minute hand animation.
pMinuteAnimation->SetFrom(minuteAngle);
pMinuteAnimation->SetTo(minuteAngle + 360);

// Set the beginning and end of the second hand animation.
pSecondAnimation->SetFrom(secondAngle);
pSecondAnimation->SetTo(secondAngle + 360);

// Start the animation:
pStoryboard->Begin();

// save the exit code for WinMain
hr = m_pVisualHost->StartDialog(&uiExitCode);
SetWinMainResultCode(uiExitCode);
```

```
    }

    . . .
} // Run
```

Error-handling code paths have been omitted in these examples for clarity. In production code, you should check the return code from **FindName** so that your application can recover gracefully in the event that **FindName** returns an error.

# Run the Application

After you make the above changes to the **App::Run** method in App.h, you can build and run MyClock.exe again.

1. If the previous instance of MyClock is still running, right-click the **MyClock** taskbar button on your Windows Embedded Compact device, and then click **Close**.
2. In Visual Studio, locate and right-click the **MyClock** subproject that you created earlier, and then click **Build**.
3. On the **Target** menu, click **Run Programs**.
4. In the **Run Program** dialog box, under **Available Programs**, select **MyClock.exe** and click **Run**.

On the display of your device, you should see the second hand moving while the hour and minute hands report the current time.

**Running the Application**

The hands of the MyClock application should continue to report the current time (see above figure for an example), with the second hand moving around the clock face once per minute, updating the animation at the default Silverlight frame rate.

# Add Event Handlers

Although the MyClock example shown previously does not require user interaction to operate, most Silverlight applications have a user interface that invites interaction with the application by generating events from the keyboard, mouse, or other input devices. Your application responds to these events through event handlers that you write and attach to user-interface objects.

The Windows Embedded Event Tool Window allows you to easily add event handlers for each object in your Silverlight user interface. This section explains how to use the Event Handler View to add new events to your application. You will continue to extend the functionality of MyClock by adding an event handler to respond to a mouse click on the clock's bezel.

# Open the Event Handler View

The Event Handler View presents a list of objects for the current XAML file that you are working with. The top object in the list is selected by default. Below the list of objects, the Event Handler View displays a list of applicable events and any associated handlers for the selected object. When you select a different object in the object list, the event list refreshes to display all possible events that can be associated with that object. You can select only one object at a time.

**To open the MyClock Event Handler View**

1. In the **Solution Explorer** pane, locate the **MyClock** subproject and open **Resource files**.
2. Open the file MainPage.xaml.
3. On the **Tools** menu, click **Windows Embedded Silverlight Tools** and click **Windows Embedded Events**.

The **Windows Embedded Events** pane, located in the top right region of the Visual Studio development environment, consists of a list of objects in the top portion of the display and a list of event handlers in the bottom portion of the display. When you click on an object name in the upper portion of the display, the event list changes to indicate which events (such as **GetFocus**, **KeyDown**, **MouseEnter**) can be attached to that object. The **Name** of each event appears in the left-hand column of this display, while the **Handler** for that event is listed in the right-hand column.

Click each object to see which events you can attach to that object and notice if there are any event handlers that are defined for that object. For example, when you click **OuterCircle[Ellipse]**, you can see that no handlers have been defined for that object because the entire **Handler** column is blank. In the next section, you will create a new event handler for the **OuterCircle** object.

# Create an Event Handler

You can associate a new event handler with an event on a particular object by simply typing the name of the event handler in the text box field to the right of the event. When you add a new event handler name, the Event Handler View automatically stubs out the event handler code for you.

**To add an event to the OuterCircle object in the MyClock example**

1. In the **Windows Embedded Events** pane, click the **OuterCircle [Ellipse]** object.
2. In the **Name** column of the event list, find the **MouseLeftButtonDown** event and type the handler name `ChangeBezel` in the text field to the right and press ENTER.

Windows Embedded Silverlight Tools automatically generates the stub method **MainPage::ChangeBezel** and opens the source file MainPage.cpp at the location of this new method:

```
HRESULT MainPage::ChangeBezel (IXRDependencyObject *pSender,
                               XRMouseButtonEventArgs *pArgs)
{
    HRESULT hr = E_NOTIMPL;
    if((NULL == pSender) || (NULL == pArgs))
```

```
    {
        hr = E_INVALIDARG;
    }
    return hr;
}
```

In MainPage.h, Windows Embedded Silverlight Tools automatically adds this new method to the list of public methods for the class **MainPage**:

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

Because Windows Embedded Silverlight Tools creates the event handler for you and automatically integrates this event handler into your application, you only have to add application-specific code to the event handler for responding to the event. You spend far less time managing the framework so that you can focus on what your application does with these events.

In the next section, you will add one line of code to the **MainPage::ChangeBezel** event handler to modify the appearance of MyClock when a user clicks the outer bezel of the clock face.

## Add Code to the Event Handler

When a user clicks the outer bezel of MyClock, **MainPage::ChangeBezel** will run. For this example, you will add one line of code to change the thickness of the outer border of the clock face whenever you receive this event:

```
m_pOuterCircle->SetStrokeThickness(20.0);
```

This call to the **SetStrokeThickness** method will increase the width of the outer circle border to 20 pixels. Here is what the modified event handler looks like (see MainPage.cpp):

```
HRESULT MainPage::ChangeBezel (IXRDependencyObject *pSender,
                               XRMouseButtonEventArgs *pArgs)
{
    return m_pOuterCircle->SetStrokeThickness(20.0);
}
```

Since this simplified event handler ignores **pSender** or **pArgs**, you can remove the generated code that checks for a NULL **pSender** or a NULL **pArgs**.

## Test the Event Handler

After you make the above changes to the **MainPage::ChangeBezel** event handler method, save all of your changes then build and run MyClock.exe again. When you click the bezel of the clock, you should see the outer border increase in thickness, resulting in the display shown in the following figure.

**Testing the Event Handler**

You can modify this event handler to make more sophisticated changes to the clock bezel. For example, you could easily modify this method to toggle border thickness between two different settings on each new mouse click, or create a new **IXRLinearGradientBrush** to change the bezel to a different color or different subtle shadow effect on each new mouse click.

# How to Merge Updates

When a designer makes changes to the source Expression Blend project, you use the Windows Embedded Silverlight Tools update tool to identify and merge these changes into your Platform Builder subproject. If the designer adds new user interface functionality, you should find new code stubs in your subproject when the update process completes.

This section continues with the MyClock example to show you how to merge Silverlight Clock user interface changes into your MyClock application. After completing the update process, you should see an improved clock face that uses the application functionality you added in the previous sections.

# Modify Silverlight Project Files

To simulate the changes that a designer would make to your source Expression Blend project, you will overwrite several files in your Expression Blend project with newer files from the WindowsEmbeddedSilverlightClockUpdate folder.

**To modify your Silverlight Clock sample project**

1. Locate the WindowsEmbeddedSilverlightClockUpdate folder at the following location:

   %ProgramFiles%\Windows Embedded Silverlight Tools\WindowsEmbeddedSilverlightClockUpdate
2. Select and copy all of the files in this folder to your source Expression Blend project located at C:\ExpressionBlend\WindowsEmbeddedsilverlightembClock, overwriting MainPage.xaml, Readme.txt, and WindowsEmbeddedSilverlightClock.csproj.

This designer modification adds a new resource image (Windows.png), and adds new elements to MainPage.xaml for an improved clock face.

# Update Your Subproject

To update your subproject, you select the subproject to receive the updates and launch the Windows Embedded Silverlight Tools update tool. The Windows Embedded Silverlight Tools update tool automatically updates your source files to reflect changes made to the source Expression Blend project. Although Windows Embedded Silverlight Tools may add new code stubs to your source files, Windows Embedded Silverlight Tools will never delete or remove your code during update. Follow these steps to update your MyClock.

**To update the MyClock project**

1. Open your OS design project in Platform Builder.
2. In the **MyClock** subproject, open the file MainPage.xaml.

   Visual Studio may open a dialog box with the message “This file has been modified outside of the source editor. Do you want to reload it?” If so, click **Yes to All**.
3. On the **Tools** menu item, click **Windows Embedded Silverlight Tools** and click **Update Silverlight for Windows Embedded Subproject**.
4. Wait for the **Update Windows Embedded Silverlight Tools Project** progress window to complete.
5. Verify that the “Summary Report” displays the message **No Errors Reported** to indicate that the update completed successfully.

If the update does not succeed, see the Troubleshooting section for more information.

# Run the Updated Application

After you complete the updates to MyClock, build and run MyClock.exe again. You should see a different clock face as shown in the following figure.

**Updated Clock Face**


The hands will continue to move and keep time as before, without requiring you to make any code changes to accommodate the user-interface updates. When you click the clock's bezel, the **MainPage::ChangeBezel** event handler will run and increase the thickness of the border of this new clock face as it did with the previous version of the clock face.

**After Mouse Click Event**


Although the designer made fundamental changes to the user interface design of the Silverlight Clock, your application code continues to drive the clock without your having to make any code changes for the redesigned user interface.

# Troubleshooting

The following sections provide solutions to common problems that you may experience while importing an Expression Blend project into a Platform Builder subproject, creating event handlers, or merging changes from the Expression Blend project.

## Import Issues

**The Validation Report displays one or more validation errors**

If the Subproject Application Wizard is unable to import your Expression Blend project, it produces a "Validation Report" that details any errors that it finds during the import process. Typically, validation errors are caused by the presence of controls and other elements in the Expression Blend project that are not supported by Silverlight for Windows Embedded. For more information, see Differences Between Silverlight for Desktop Browser and Silverlight for Windows Embedded

(http://go.microsoft.com/fwlink/?LinkId=192019) and “Unsupported Controls and Features in Expression Blend” in Expression Blend and Silverlight for Windows Embedded (http://go.microsoft.com/fwlink/?LinkId=204753).

**The Subproject Application Wizard displays a “file not found” error message**

Windows Embedded Silverlight Tools does not support image or XAML files with special characters in the name. For example, a XAML file name containing an ampersand (&) will cause the import of the Expression Blend project to stop with an error message. For best results, name your image and XAML files using only letters, numerals, and underscores (_).

**The Subproject Application Wizard seems to be frozen**

Windows Embedded Silverlight Tools supports only one running instance of Visual Studio. If you have multiple instances of Visual Studio running Windows Embedded Silverlight Tools at the same time, one of these instances could freeze while it waits for input from the other instance. To remedy this situation, use only one Visual Studio instance while running Windows Embedded Silverlight Tools.

**Windows Embedded Silverlight Tools displays an error message about an invalid template file**

If your Expression Blend project has one or more linked XAML files, Windows Embedded Silverlight Tools will not be able to import these files into your subproject. To remedy this problem, add XAML files directly to your Expression Blend project rather than creating links.

**Adding the subproject results in an “illegal characters in path” error**

If your Expression Blend project has one or more file names that contain double-byte characters, Windows Embedded Silverlight Tools will be unable to use your project to create a valid Platform Builder subproject. To remedy this problem, convert these file names to Unicode format.

**Windows Embedded Silverlight Tools displays an “Invalid File Name” error**

If the directory path to the application project contains white spaces or other invalid characters, this error message will appear. To remedy this problem, remove white spaces or other invalid characters from the directory path to the application project.

# Event Handler Issues

**Newly created event handler code does not compile**

When you specify a new event handler name, this name must be a valid C++ identifier or the event handler will not compile. The event handler name must begin with a letter or an underscore (_) followed by any number of letters, underscores, or numerals.

# Update Issues

Update displays the message, “Please open a Silverlight for Windows Embedded project document.”

Before selecting the **Update Silverlight for Windows Embedded Project** menu item, you must first open a file in the Platform Builder subproject that you want to update (for example, MainPage.xaml). This selection ensures that the Windows Embedded Silverlight Tools update tool updates the appropriate subproject in your OS design.

# Appendix: XAML Markup for XRPack

In the Windows Embedded Compact 7 build system, the XAML Resource Packager (XRPack) extracts all of the necessary resources (strings, images, and XAML) for a Silverlight for Windows Embedded project and compiles them into a resource file. The build system then binds the resources in the file into the executable (.EXE) or dynamic-link library (.DLL) file. For more about application packaging with XRPack, see XAML Resource Packager (http://go.microsoft.com/fwlink/?LinkId=204756) at MSDN.

The code examples below cover some programming considerations for specific XAML elements in code that XRPack compiles.

## xrpack:String

**Xrpack:String** attributes are processed and translated into resource bindings, creating resource string tables that are used in the localization process.

The following examples show single-string resources with generated labels.

```
<ContentControl x:Name="ResourceStringBindingContentControl1"
xrpack:String="Content:100"  Content="Also 'My Nifty String' at runtime"/>
<Button x:Name="ResourceStringBindingButton1" xrpack:String="Content:101"
Content="Some other String"/>
<CheckBox x:Name="ResourceStringBindingCheckBox1" xrpack:String="IsChecked:102"
IsChecked="True"/>
```

The following example shows how to define a single string in the string table that can replace multiple strings in the XAML.

```
<TextBlock xrpack:String="Text:100" Text="100" />
<TextBlock xrpack:String="Text:100" Text="101" />
<TextBlock xrpack:String="Text:102" Text="102" />
```

**XRPack** generates the following string table from the XAML in the preceding example.

```
STRINGTABLE
BEGIN
    100 "100"
    102 "102"
END
```

The technique in the preceding example is useful when a project has multiple XAML files containing controls with the same text that need to be localized. In this case, the text can be localized once in the string table to update all instances of the text that appear on controls.

The following example shows how to use multiple string resources for a control.

```
<TextBlock x:Name="ResourceStringBindingTextBlock1"
xrpack:String="Text:100;Visibility:103"  Text="My Nifty String"
Visibility="Collapsed"/>
```

The following example shows how to define a resource ID label to use in code.

```
<Button x:Name="ResourceStringBindingButton1"
xrpack:String="Content:101(IDR_MYLABEL)" Content="Some other String"/>
```

## xrpack:ClassResourceId

An `xrpack:ClassResourceId` attribute is a class resource definition. You can use it with or without the `x:class` attribute to control resource identifier generation. If a file has no user-specified resource identifier, **XRPack** automatically generates one based on the `x:Class` or `xrpack:ClassResourceId` attributes. If the class resource definition contains full markup, as in the following example, you do not need to use the `x:Class` attribute.

```
xrpack:ClassResourceId="Class:101(IDR_USERLABEL)"
```

If MainPage.xaml contains the code in the preceding example, **XRPack** generates the following resource identifier and label.

```
#define IDR_USERLABEL 101 // MainPage.xaml
```

If the class resource definition contains partial markup, as in the following example, **XRPack** uses the specified identifier for the resource identifier and a generated name from the `x:Class` attribute.

```
xrpack:ClassResourceId="Class:192"
x:Class="TextBlock.MainPage"
```

If MainPage.xaml contains the code from the previous example, **XRPack** generates the following resource identifier and label.

```
#define IDR_TEXTBLOCK_MAINPAGE 192 // MainPage.xaml
```

If you do not use an `xrpack:ClassResourceId` attribute, then **XRPack** generates a resource identifier and creates the label based on the `x:Class` attribute. For example, if MainPage.xaml contains the following code,

```
x:Class="Assets.Tulips"
```

then, **XRPack** generates the following code:

```
#define IDR_ASSETS_TULIPS <generated number> // Tulips.xaml
```

In this case, if no `x:Class` attribute is present to supply the label, **XRPack** logs error **XRPack0010, XRPACK_E_INVALID_ATTRIBUTE_MARKUP**.

## xrpack:Resource

`Xrpack:Resource` attributes map a resource ID to a specific resource as shown in the following example.

```
<Image xrpack:Resource="Source:902(IDR_MYIMG)" Source="Images/My Image.png" />
```

**XRPack** compresses bitmaps that you embed in the resource package by default or if you set the **/ImageCompress** option on the command line. (The resource package is contained in the resource (.RES) file that is linked into the executable (.EXE) or dynamic-link library (.DLL) file). The compressed image improves efficiency when Silverlight for Windows Embedded is rendering a scene containing the bitmap. However, user applications cannot retrieve a compressed image from the resource package (for example, by calling **LoadResource**) in its original, uncompressed format.

## References to Resources

When you use **XRPack**, all references to resources inside XAML tags must use their Uniform Resource Identifiers (URIs) rather than the identifiers assigned in the resource file. Expression Blend 3 uses this syntax, but legacy XAML files may contain code that does not use the URI. **XRPack** resolves all URIs and locates them in the file system using the relative path locations of the XAML file and the resources. If it cannot resolve a URI, **XRPack** generates an error.

The following example shows the syntax that Silverlight for Windows Embedded supports but that **XRPack** cannot compile.

**Resources.rc:**

```
AUTO_GIF_RES1 XAML_RESOURCE DISCARDABLE  "..\\data\\Images\\Resource1_GIF.gif"
```

**MyXaml.xaml:**

```
<Image Height="100.00" Width="100.00" x:Name="GIF" Source="AUTO_GIF_RES1"
Opacity="100" VerticalAlignment="Stretch" />
```

# Conclusion

Windows Embedded Silverlight Tools speeds Windows Embedded Compact application development time by automating the manual steps that would be required to connect C++ application functionality to Silverlight UI XAML files created in Expression Blend. Windows Embedded Silverlight Tools automatically validates Expression Blend project files for compatibility with Silverlight for Windows Embedded, generates application template code and event handler stubs for the Silverlight runtime, and merges in any changes that the designer makes to the source Expression Blend project.

This article described how to import a sample Expression Blend project into a Platform Builder subproject, add Windows Embedded Compact C++ functionality to the resulting application template, create new event handlers using the Event Handler View, and update your application with Silverlight XAML and resource changes by using the update tool. You created a simple Windows Embedded Compact MyClock application from a Silverlight Clock example project, ported application code from Expression Blend C# code-behind files to the native C++ Silverlight runtime of MyClock, created new event handlers for MyClock, and updated the MyClock application with a new user interface without having to make any code changes.

# Additional Resources

To learn more about Silverlight for Windows Embedded, see the following links.

- Microsoft Silverlight (http://go.microsoft.com/fwlink/?LinkId=191041)
- Microsoft Silverlight for Windows Embedded (http://go.microsoft.com/fwlink/?LinkId=163763)
- Introduction to Application Development with Silverlight for Windows Embedded (http://go.microsoft.com/fwlink/?LinkId=183252)
- Handle Events in Silverlight for Windows Embedded, in the Silverlight for Windows Embedded Developer's Guide (http://go.microsoft.com/fwlink/?LinkId=191039)
- Silverlight for Windows Embedded Reference (http://go.microsoft.com/fwlink/?LinkId=209193)
- Differences Between Silverlight for the Web and Silverlight for Windows Embedded (http://go.microsoft.com/fwlink/?LinkId=209194)

To learn more about the XAML Resource Packager, XRPack, see the following links.

- XAML Resource Packager (http://go.microsoft.com/fwlink/?LinkId=204756)
- XRPack Errors (http://go.microsoft.com/fwlink/?LinkID=198975)

To learn more about the Virtual CEPC development platform, see the following links.

- Getting Started with Virtual CEPC (http://go.microsoft.com/fwlink/?LinkID=190470)
- Advanced Virtual CEPC (http://go.microsoft.com/fwlink/?LinkID=206041)

To learn more about Windows Embedded Compact, see the following link.

- Windows Embedded (http://go.microsoft.com/fwlink/?LinkId=183524)

© 2011 Microsoft. All rights reserved.