Windows Embedded Compact

Populating a Silverlight for Windows Embedded UI with Collections of Data

Published: January 2012

Applies To: Windows Embedded Compact 7

Abstract

This paper demonstrates data binding in a Silverlight for Windows Embedded application. Data binding is the process of linking data stored in data source objects with elements in a UI, so that when the data is updated, the UI elements display the new values. This paper describes the following:

- Differences between the implementation of data binding in Silverlight for Windows Embedded and Microsoft Silverlight 3
- How to support hierarchical data in collections
- How to create a list box that is connected to a data source collection
- How to respond to notifications from the data provider
- How to add support for updating the data source from the UI

The example project in this paper requires Microsoft Expression Blend 3, Visual Studio 2008 with SP1, Windows Embedded Compact 7, and a virtual CEPC.

Contents

Introduction

Many types of Windows Embedded Compact 7 applications must display collections of data that are maintained separately from the UI. Often, these collections are large and constantly changing. For example, an email application on a Windows Embedded Compact device connects to a Microsoft Exchange Server to populate its inbox with a collection of email messages. As new data becomes available, the email data from Microsoft Exchange Server that is displayed in the inbox is updated.

When you want an application to display data that is maintained separately from the UI, you can use Silverlight for Windows Embedded classes and interfaces to bind data in a collection to elements in the UI. When the data changes, you can optionally update the data in the collection so that the UI displays the updated data.

Data binding is the process of linking data stored in data source objects with elements in a UI, so that when the data is updated, the UI elements display the new values.

Some types of applications that you can populate with dynamic data are:

- Email applications that display data from a Microsoft Exchange Server
- Multimedia applications that display data from a user's media library
- Database applications that display data from SQL Server Compact Edition
- Short Message Service (SMS) message applications that display data from SMS messages
- Set-top box applications that display data from a cable television channel listing or video-on-demand (VOD) server
- Global Positioning System (GPS) applications that display GPS data based on the current location of the device.
- Applications that display the status of nodes on a home automation or industrial automation network, such as Heating, Ventilation, and Air Conditioning (HVAC), or lights

If you are familiar with Windows Embedded Compact 7, Silverlight for Windows Embedded, and Microsoft Expression Blend 3, and your application must display data that is maintained separately from the UI, you can use Silverlight for Windows Embedded classes and interfaces to bind data in a data collection to elements in the UI. When the data changes, you can update the data in the collection so that the UI can display the updated data.

The primary components that you use to implement data binding functionality are:

- A data source collection — an object that contains a set of C++ data source objects, which store strings, property values, or other nested objects in a property bag
- The data provider — the external source of dynamic data, such as a server, database, or computer

When the data provider has new data, you can design your application to access the data source collection, and update, delete, or add new objects to it. Then, the UI elements can display updated data stored in the collection.

In some cases, you may want to display hierarchical data in a Silverlight for Windows Embedded UI. When hierarchical data is supported in a data source collection, items in the collection can also contain their own sub-collections of data. For example, consider an application that displays a collection of inventory items. When a user selects an item, the user might also want to see a collection of calendar dates that shows when the item was produced, inspected, shipped, and purchased. To display a collection that belongs to an item in a collection, you must support hierarchical data in your Expression Blend project and C++ code.

The data-binding implementation for using data collections in Silverlight for Windows Embedded resembles the implementation in Microsoft Silverlight 3, but differs in the following respects:

- Silverlight for Windows Embedded provides one primary derived collection class, XRObservableCollection (http://go.microsoft.com/fwlink/?LinkId=223714), which you use to create data source collections. In contrast, Microsoft Silverlight 3 provides approximately fourteen classes that inherit from **System.Collections.ObjectModel::Collection<T>**.
- Silverlight for Windows Embedded does not support reflection, but instead provides a property bag interface that handles the functionality of resolving property names.
- Silverlight for Windows Embedded provides helper classes to support functionality that resembles the Component Object Model (COM) for managing collection objects.
- Silverlight for Windows Embedded supports interface maps for extending data binding objects to support additional interfaces.

The major steps you perform to populate the UI with collections of dynamic data are:

1. Create the Expression Blend Project and the Windows Embedded Silverlight Tools Subproject
2. Populate the List Box with Data
3. Update the Data (Optional)
4. Add Support for Users to Update the Data (Optional)
5. Update and Build the Project

Before beginning the steps to populate the UI with collections of data, it is helpful to understand how data binding works in Silverlight for Windows Embedded. The next section provides in-depth information about how data binding works with data source collections.

Data Source Collections in Silverlight for Windows Embedded

To maintain data collections separately from the UI, you can use the Silverlight for Windows Embedded class library to implement data source objects and a data source collection. You use the library classes and interfaces to populate the UI with collections of data.

Implementation for Binding Data in Silverlight for Windows Embedded

In Silverlight for Windows Embedded, a ***data source collection*** is an **XRObservableCollection<ItemType>** object that contains a set of C++ ***data source objects***, each of which implements **TPropertyBag<Derived>**. Each data source object stores its strings, property values, and other nested objects in a property bag. You can display the property values, strings, and nested objects in a Silverlight for Windows Embedded UI by binding their data to UI elements. Then, you can set the data source collection as both the data context and the items source for a list box that contains data-bound elements that you defined in the XAML UI.

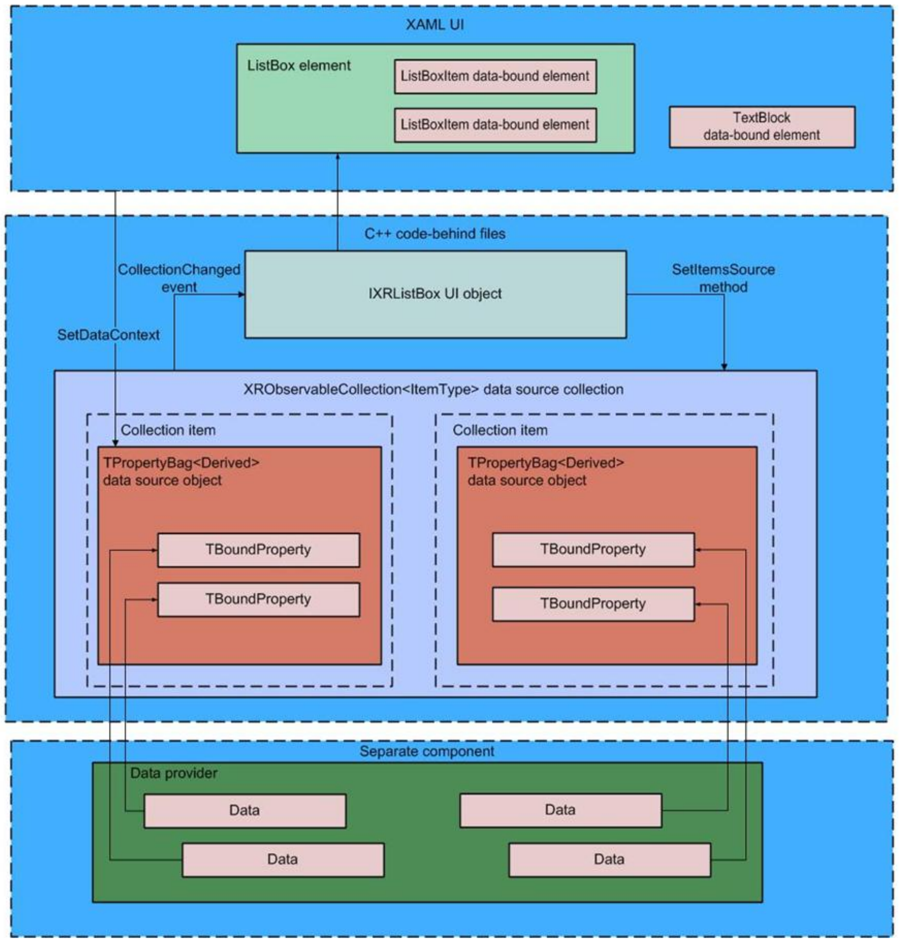
Note

Although this white paper uses an IXRListBox (http://go.microsoft.com/fwlink/?LinkId=208908) object to illustrate how to bind data in collections to the UI, you can also use an IXRComboBox (http://go.microsoft.com/fwlink/?LinkId=208907) object, which inherits from **IXRItemsControl** and also supports the IXRItemsControl::SetItemsSource (http://go.microsoft.com/fwlink/?LinkID=223719) method. To define a combo box in XAML, see ComboBox Class (http://go.microsoft.com/fwlink/?LinkId=142209) on MSDN.

A ***data provider*** is an external source of dynamic data, such as a mail folder, that exists in a separate component, such as another DLL on the device or a server, database, or computer. Data is maintained in the data provider, while the data source objects and data source collection that store the data are written in the C++ code-behind for your Windows Embedded Silverlight Tools subproject. You can design your application to access the data source collection and update, delete, or add new objects to it when the data provider has new data. Then, when the collection raises the **CollectionChanged** event, the list box that you defined in the XAML UI can display updated data stored in the collection.

The following diagram shows the relationships between components in the Silverlight for Windows Embedded implementation for binding data.

Figure 1: Implementation for Binding Data in Silverlight for Windows Embedded



Differences Between Microsoft Silverlight 3 and Silverlight for Windows Embedded

The implementation for using data source collections in Silverlight for Windows Embedded resembles the implementation in Microsoft Silverlight 3. For example, XRObservableCollection<ItemType> (http://go.microsoft.com/fwlink/?LinkId=223714) is the C++ equivalent of ObservableCollection<T> (http://go.microsoft.com/fwlink/?LinkId=226354) in Microsoft Silverlight 3. And, XRObservableCollection<ItemType> (http://go.microsoft.com/fwlink/?LinkId=223714) implements IXRNotifyCollectionChanged (http://go.microsoft.com/fwlink/?LinkId=226356), which is the C++ equivalent of INotifyCollectionChanged (http://go.microsoft.com/fwlink/?LinkId=226355). The primary differences in implementation between Microsoft Silverlight 3 and Silverlight for Windows Embedded for data source collections are:

- Silverlight for Windows Embedded provides XRObservableCollection<ItemType> (http://go.microsoft.com/fwlink/?LinkId=223714) as the primary derived class for data source collections.
- Because Silverlight for Windows Embedded does not provide equivalent support for the **System.Reflection** C# namespace, it instead provides the TPropertyBag<Derived> (http://go.microsoft.com/fwlink/?LinkId=223715) C++ interface for you to implement on your custom class type.
- **TPropertyBag<Derived>** maintains a property list for an object. It provides methods that you use to retrieve property values by specifying the name of a property, instead of specifying the name of an object that represents a property.
- **XRObservableCollection<Derived>** does not support C++ equivalent methods for ICollection::IsSynchronized (http://go.microsoft.com/fwlink/?LinkId=27027) or ICollection::SyncRoot (http://go.microsoft.com/fwlink/?LinkId=227485), which are used to synchronize access to a collection that is by default not synchronized (that is, not thread-safe). Instead, use an XRAutoCriticalSection (http://go.microsoft.com/fwlink/?LinkId=227480) object, or implement your own thread safety mechanism for accessing a data source collection.
- To support functionality that resembles COM for managing objects, Silverlight for Windows Embedded provides the helper class XRObject<Base> (http://go.microsoft.com/fwlink/?LinkId=224270).
- **XRObject<Base>** also supports interface maps, so that you can support requesting an interface pointer to a collection class object by using the COM method **IUnknown::QueryInterface**.
- Silverlight for Windows Embedded supports populating the **ListBox** and **ComboBox** Silverlight 3 XAML elements with data from a collection.

  The following Silverlight 3 XAML elements support displaying data from a collection in Microsoft Silverlight 3, but are unsupported in Silverlight for Windows Embedded:

  - **AutoCompleteBox**
  - **DataGrid**
  - **HeaderedItemsControl**
  - **TabControl**
  - **TreeView**
- Silverlight for Windows Embedded does not support UI-to-UI binding, in which you use a UI element as the data source for another UI element. Instead, you can identify properties in two different UI elements that are data-bound to the same data source property. For more information, see Data Binding in Silverlight for Windows Embedded Virtual Lab (http://go.microsoft.com/fwlink/?LinkId=226353).

TPropertyBag<Derived> in Silverlight for Windows Embedded

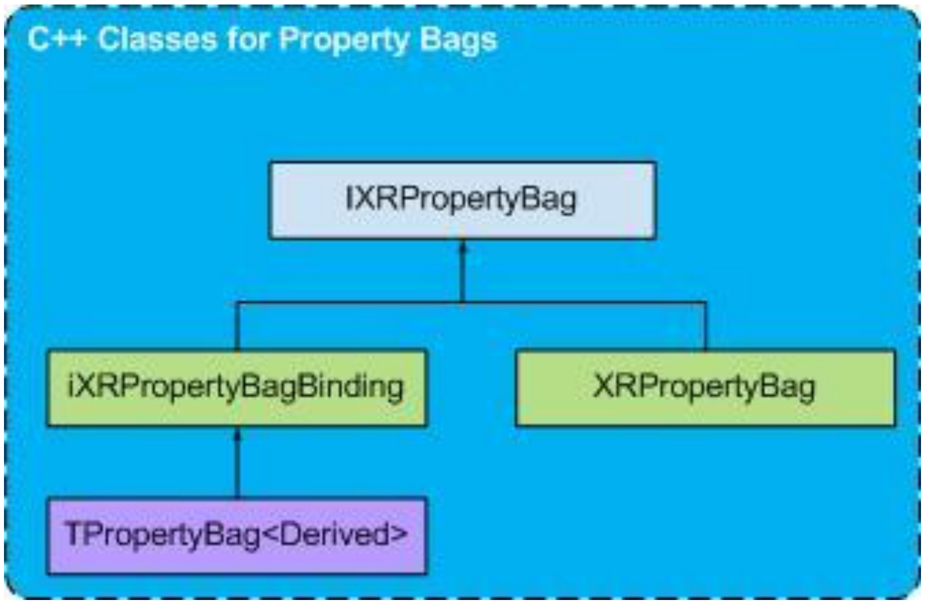
To manage access to the values of data source properties in a data source object, you can implement the TPropertyBag<Derived> (http://go.microsoft.com/fwlink/?LinkId=223715) interface on the object.

We recommend using **TPropertyBag<Derived>** instead of IXRPropertyBag (http://go.microsoft.com/fwlink/?LinkId=223720) because **TPropertyBag<Derived>** provides the following additional functionality in Silverlight for Windows Embedded:

- It supports registering properties so that you can obtain property values by name. By implementing **TPropertyBag<Derived>**, you will not have to provide a custom implementation of **GetValue** and **SetValue** methods in your data class.
- It raises the **PropertyChanged** event when you set a new value for a property in the property bag.
- It provides default functionality for owning and registering properties represented by objects of type **iXRPropertyBinding**.

The following diagram shows a class diagram for **TPropertyBag<Derived>**.

Figure 2: TPropertyBag<Derived> Class Diagram



Support for Hierarchical Data in Collections

You can display hierarchical data in a Silverlight for Windows Embedded application.

The example from the Introduction describes an inventory tracking application that can also display a sub-collection of calendar dates that indicate when each inventory item was produced, inspected, shipped, and purchased. When the user selects an inventory item, the UI can display calendar data from the item's sub-collection in another list box, or in text elements. This functionality is known as supporting hierarchical data in a UI.

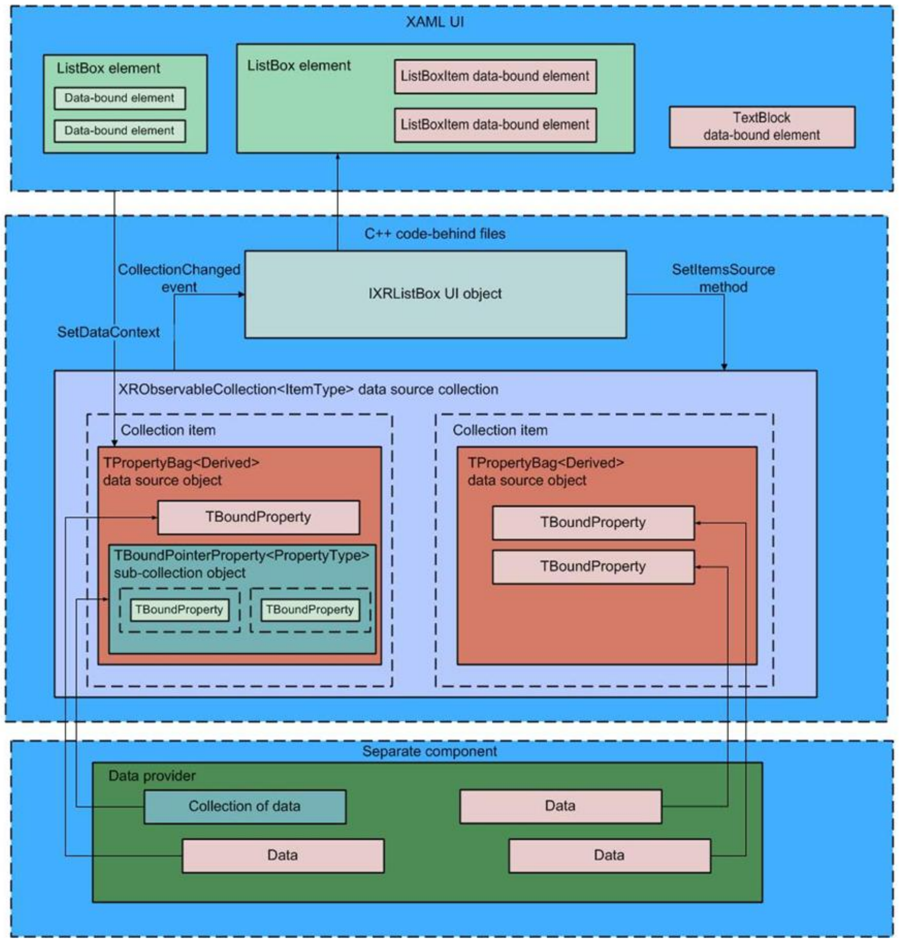
Silverlight for Windows Embedded represents hierarchical data in a property of a C++ data source object.

To display a sub-collection of data that belongs to an item in a collection, you must support hierarchical data in both your Expression Blend project and C++ code. To support hierarchical data in the data source object, you add a data field of type TBoundPointerProperty<PropertyType> (http://go.microsoft.com/fwlink/?LinkId=227476) to the data source object. Then, you can set the value of the **TBoundPointerProperty<PropertyType>** data field to an interface pointer to a hierarchical data collection in the form of another XRObservableCollection<ItemType> (http://go.microsoft.com/fwlink/?LinkId=223714) object.

To support hierarchical data, follow the optional steps in each procedure in this white paper that are labeled “To support displaying hierarchical data.”

The following diagram shows the differences in the Silverlight for Windows Embedded implementation for binding data when your application must support hierarchical data.

Figure 3: Implementation for Binding Data and Supporting Hierarchical Data



Prerequisites

To populate a Silverlight for Windows Embedded UI with dynamic collections of data by following the steps in this white paper, you must install the following items on your computer in the order listed:

1. Microsoft Expression Blend 3. For more information, see Microsoft Expression Blend 3 (http://go.microsoft.com/fwlink/?LinkId=204751) at the Microsoft Download Center.
2. Visual Studio 2008 with SP1.
3. Windows Embedded Compact 7.

   Important

   By default, Windows Embedded Silverlight Tools is installed when you install Windows Embedded Compact 7. To verify this during installation, choose **Custom Install** and confirm that the **Windows Embedded Silverlight Tools** option is selected.
4. A virtual CEPC, correctly installed and configured. For more information, see Getting Started with Virtual CEPC (http://go.microsoft.com/fwlink/?LinkId=199788).

When you install Windows Embedded Silverlight Tools, the Expression Blend templates are installed at %ProgramFiles%\Microsoft Expression\Blend 3\ProjectTemplates\en\Windows Embedded Silverlight Tools\Application.

If you are using a non-U.S. English language 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 an Applications folder that you create in your local user directory at the following location: %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 the **New Project** dialog box.

In addition, we recommend the following:

- (Optional) Determine which data provider (for example, a Microsoft Exchange Server, a Video on Demand (VOD) server, or a user's media library of songs) that you want to use to populate the UI with data. Make sure that you can access the data; for example, by creating an Ethernet network connection to a test server, or by implementing the media library in your OS design project.
- Build a run-time image that supports Silverlight for Windows Embedded (using SYSGEN variable SYSGEN_XAMLRUNTIME), Target Control Shell (using SYSGEN variable SYSGEN_SHELL), and the functionality that your data provider requires. For example, if your data provider is a media library, the run-time image must support Media Library (using SYSGEN variable SYSGEN_MEDIAAPPS_MEDIALIBRARY).

Create the Expression Blend Project and the Windows Embedded Silverlight Tools Subproject

To create an application that can display data from a data source, you must create a XAML UI that supports data from a data source collection. Then, you use Windows Embedded Silverlight Tools to create an application subproject that is founded on your XAML project so you can add it to your OS design.

In this step, you design a list-box UI element for the data-bound elements by using Microsoft Expression Blend 3. The Expression Blend project must include the following XAML elements:

- A ListBox (http://go.microsoft.com/fwlink/?LinkId=141725) XAML element to contain the data-bound elements
- A DataTemplate (http://go.microsoft.com/fwlink/?LinkId=221481) XAML element that defines the presentation and appearance of the data-bound elements

Before you create the project and subproject, you must determine the data source properties that will populate UI elements with data. To populate the list box with data, you add names of properties from the C++ data-source object to Binding Markup Extensions (http://go.microsoft.com/fwlink/?LinkID=219141).

To determine which data-source properties will populate UI elements with data

1. Determine the type of data to display in your application. For example, a media application might display the following data for each item in a data source collection:
   - The name of the album
   - The name of the artist
   - The number of times that a user plays the media
   - A collection of other albums also produced by the artist
2. Choose property names for the data. For example, the media application described earlier might use the following property names:
   - **AlbumName**
   - **AlbumArtist**
   - **PlayCount**
   - **OtherAlbumsList**

   Note

   You will reuse these property names when you register properties with data source objects in the collection.

To create an Expression Blend 3 project for a UI that you will populate with collections of data

1. In Expression Blend, on the **File** menu, click **New Project**.
2. In **Project types**, select **Windows Embedded**.
3. Type a project name, and then click **OK**.
4. Add a **ListBox** element.
5. Add a **DataTemplate** element based on how you want to bind data:
   - To bind data to items in one element, such as a **ListBox**, add the **DataTemplate** element to the **UserControl** root element in MainPage.xaml.
   - To bind data to several different elements on the page, define the **DataTemplate** element as an application resource in App.xaml.
6. In the **ListBox** element, add an **ItemTemplate** attribute and set its value as the name of the **DataTemplate**. For more information, see Data Templating Overview (http://go.microsoft.com/fwlink/?LinkId=220675) on MSDN.
7. Transform elements into data-bound elements by doing the following:
   a. Identify the element attribute to populate with data.
   b. For the attribute value, add a binding markup extension with the property name that you chose earlier in this section. For more information, see Binding Declarations Overview (http://go.microsoft.com/fwlink/?LinkId=219553) on MSDN.
8. (Optional) To support displaying hierarchical data, you can define another **ListBox** element and transform it into a data-bound element. Add an **ItemsSource** attribute to the additional **ListBox** element, and use a binding markup extension that has a property name. Later, you will register the property name of the hierarchical data with a **TBoundPointerProperty** object.

   For example, in the media application described earlier, the hierarchical data would be represented as the **OtherAlbumsList** property.

   For more information about hierarchical data, see Data Source Collections in Silverlight for Windows Embedded. For an example in XAML, see How to: Use the Master-Detail Pattern with Hierarchical Data (http://go.microsoft.com/fwlink/?LinkId=221489) on MSDN.
9. On the **File** menu, click **Save**.

To create a Windows Embedded Silverlight Tools subproject that is founded on your Expression Blend project

- For instructions on how to create a Windows Embedded Silverlight Tools subproject, see A Sample Application Tutorial Using Windows Embedded Silverlight Tools (http://go.microsoft.com/fwlink/?linkid=189508) on MSDN.

Populate the List Box with Data

After you create the Windows Embedded Silverlight Tools subproject that uses your Expression Blend project as its foundation, you can add C++ code to the subproject files. This code populates the UI with data from a data source collection. The list box in your application displays the items from your data source collection in a list. Your application can populate each list-box item with a data source object from the collection.

When the user selects an item in the list, you can also populate data-bound elements with data-source properties of the object that is bound to the item. For example, when a user selects an item in a media playlist, the media application can populate **TextBlock** data-bound elements with the **AlbumName** and **AlbumArtist** data-source property values for the selected item.

To populate a list box with data, you must create a data class and a data source collection, and then establish a data connection between the data source collection and the list box.

Create a Data Class

To populate a list box with data from a collection, you must first create a class or set of classes to represent your collection data.

The data provider may organize its data in a variety of ways, including data in event structures, property values, classes, data fields, and so on.

To prepare the collection data for a Silverlight for Windows Embedded application, you must represent it by creating a data class that uses the programming elements in the Silverlight for Windows Embedded class library.

To create a data class

1. In Platform Builder, open the subproject you created in Create the Expression Blend Project and the Windows Embedded Silverlight Tools Subproject, earlier in this white paper.
2. Implement a custom data class that implements the TPropertyBag<Derived> (http://go.microsoft.com/fwlink/?LinkId=223715) interface.
3. In the data class, add a property field for each data-source property you identified in Create the Expression Blend Project and the Windows Embedded Silverlight Tools Subproject. The property fields must have data types of Silverlight for Windows Embedded classes that represent properties, which include TBoundProperty<PropertyType> (http://go.microsoft.com/fwlink/?LinkId=227477), TBoundProperty<BSTR> (http://go.microsoft.com/fwlink/?LinkId=227478), and TBoundPointerProperty<PropertyType> (http://go.microsoft.com/fwlink/?LinkId=227476).

   Note

   TBoundProperty<PropertyType>, **TBoundProperty<BSTR>**, and **TBoundPointerProperty<PropertyType>** provide `Get` and `Set` methods, so you do not have to implement your own `Get` and `Set` methods in the data class, as you do for classes that implement IXRPropertyBag (http://go.microsoft.com/fwlink/?LinkId=223720).

4. Implement functionality in the data class to register the properties by using **TPropertyBag.BeginRegisterProperties**, **TPropertyBag.RegisterBoundProperty**, and **TPropertyBag.EndRegisterProperties**, as follows:
   a. Block other threads by calling **TPropertyBag.BeginRegisterProperties**.
   b. Call **TPropertyBag.RegisterBoundProperty** for each property field, and pass the data-source property name that you identified earlier in this tutorial into the *PropertyName* input parameter.
   c. Stop blocking other threads by calling **TPropertyBag.EndRegisterProperties**.
5. (Optional) To support displaying hierarchical data, define another property field of type **TBoundPointerProperty<PropertyType>**. The property field represents a pointer to a collection object that belongs to the data class.

   Important

   Set the *PropertyType* template parameter to an interface that **XRObservableCollection<Derived>** supports, such as **IXRList**, **IXREnumerable**, or **IXRValueCollection**.

The following example code shows a data class that has a property field, **m_pManager**, for storing hierarchical data.

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.

```
#include "XRCollection.h"

#include "XRPropertyBag.h"


class _declspec(uuid("{557BFE61-D018-41ce-AF28-06EB6812DBFC}")) MyEmployee : public
TPropertyBag<MyEmployee>

{

protected:

    MyEmployee() {};

public:

    HRESULT Initialize(int ID, const WCHAR* pFirstName)

    {

        HRESULT hr = InitializeProperties();
        m_ID = ID;
        m_FirstName = SysAllocString(pFirstName);
        return hr;
    }

    TBoundProperty<int> m_ID;
    TBoundProperty<BSTR> m_FirstName;
    TBoundPointerProperty<TPropertyBag> m_pManager;
    TBoundPointerProperty<IXRList> m_pDirectReports;

    HRESULT InitializeProperties()
    {
        HRESULT hr = S_OK;
        hr = BeginRegisterProperties();
        if (FAILED(hr))
            return hr;

        hr = RegisterBoundProperty(L"ID", m_ID);
        if (FAILED(hr))
            return hr;

        hr = RegisterBoundProperty(L"FirstName", m_FirstName);
        if (FAILED(hr))
            return hr;

        hr = RegisterBoundProperty(L"Manager", m_pManager);
        if (FAILED(hr))
            return hr;

        hr = RegisterBoundProperty(L"DirectReports", m_pDirectReports);
        if (FAILED(hr))
            return hr;
        hr = EndRegisterProperties();


        return hr;
    }
};
```

Create a Data Source Collection

After you create a data class, you must use it to instantiate data source objects and add them to an **XRObservableCollection<ItemType>** object.

To create a data source collection

1. In your subproject, open the source code file where you want to add functionality to create the data source collection. For example, in **Solution Explorer**, browse to Subprojects\< *Subproject Name*>\Source files, and open MainPage.cpp.
2. Define an XRObservableCollection<ItemType> (http://go.microsoft.com/fwlink/?LinkId=223714) object variable and create an instance by calling **XRObservableCollection.CreateInstance(XRObservableCollection**)**.
3. Create data source objects to add to the collection. For each data source object, do the following:
   a. Define an object variable.
   b. To block other threads from accessing the collection object, implement a thread safety mechanism. For example, initialize an XRAutoCriticalSection (http://go.microsoft.com/fwlink/?LinkId=227480) object and call EnterCriticalSection (http://go.microsoft.com/fwlink/?LinkId=226720).
   c. To create an object instance of the data class, call **TPropertyBag.CreateInstance(Derived**)**.
   d. On the collection instance, call its derived method **XRValueCollectionT.Add(const ItemType&)** to add the data source object to the collection.
   e. Release ownership of the thread. For example, call LeaveCriticalSection (http://go.microsoft.com/fwlink/?LinkId=226721).
4. (Optional) To support hierarchical data, do the following:
   a. Identify a data source object that must support hierarchical data and that includes a **TBoundPointerProperty<PropertyType>** property field.
   b. Create an XRObservableCollection<ItemType> (http://go.microsoft.com/fwlink/?LinkId=223714) sub-collection of items by repeating steps 1 and 2.
   c. To convert the sub-collection object into an XRValue

(http://go.microsoft.com/fwlink/?LinkId=223716) object, create an XRValue (http://go.microsoft.com/fwlink/?LinkId=223716) object and call XRValue::SetValue(IXREnumerable, bool).

   d. To block other threads from accessing the collection object, implement a thread safety mechanism. For example, initialize an XRAutoCriticalSection (http://go.microsoft.com/fwlink/?LinkId=227480) object and call EnterCriticalSection (http://go.microsoft.com/fwlink/?LinkId=226720).
   e. On the data source object, call **TPropertyBag.SetValue(const WCHAR *,XRValue *)** to set the XRObservableCollection<ItemType> (http://go.microsoft.com/fwlink/?LinkId=223714) sub-collection as the value of a property field of type TBoundPointerProperty<PropertyType> (http://go.microsoft.com/fwlink/?LinkId=227476).
   f. On the collection instance, call its derived method **XRValueCollectionT.Add(const ItemType&)** to add the data source object to the collection.
   g. Release ownership of the thread. For example, call LeaveCriticalSection (http://go.microsoft.com/fwlink/?LinkId=226721).

5. In the shutdown code for your application, call the inherited method **_XRValueCollectionBaseT.Clear** to release references to objects in the data source collection. For example, add the code to the helper method **App::OnExit** in App.cpp.

The following code example creates a data source collection of data source objects. The code example assigns initial values to data source objects by using a custom method, `Initialize`, that initializes the `ID` and `FirstName` registered properties.

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.

```
#include "XamlRuntime.h"

#include "XRPtr.h"

#include "XRCollection.h"


static XRPtr<MyEmployee> m_pModel;

    static HRESULT CreateModel()

    {

        HRESULT hr = S_OK;


        XRPtr<XRObservableCollection<MyEmployee*>,IXREnumerable> pCollection;

        XRObservableCollection<MyEmployee*>::CreateInstance(&pCollection);
        XRPtr<MyEmployee> pEmployee1;
        MyEmployee::CreateInstance(&pEmployee1);
        pEmployee1->Initialize(1, L"EmployeeOne");

        XRPtr<MyEmployee> pEmployee2;
        MyEmployee::CreateInstance(&pEmployee2);
        pEmployee2->Initialize(2, L"EmployeeTwo");

        pCollection->Add(pEmployee1);
        pCollection->Add(pEmployee2);

        MyEmployee::CreateInstance(&m_pModel);
        m_pModel->Initialize(3, L"EmployeeModel");
        m_pModel->m_pDirectReports = pCollection;

        return hr;
    }
```

Set up a Data Connection Between the List Box and the Data Source Collection

After you create a data source collection, you must establish a data connection by expanding the responsibilities of the **XRObservableCollection<ItemType>** object to include the following roles:

- The data context for your application
- The items source for the list box

To set up a data connection between the list box and the data source collection

1. In your subproject, open the source code file where you want to add functionality for setting up the data connection. For example, in Solution Explorer, browse to Subprojects\<*Subproject Name*>\Source files, and open MainPage.cpp.
2. To obtain a pointer to the root element of the visual host, call **IXRVisualHost::GetRootElement**.
3. To obtain a pointer to the list box that you designed earlier in this tutorial, call **IXRFrameworkElement::FindName** on the root element.
4. Create an XRValue (http://go.microsoft.com/fwlink/?LinkId=223716) object instance of type VTYPE_ENUMERABLE.

5. To convert the XRObservableCollection<ItemType> (http://go.microsoft.com/fwlink/?LinkId=223714) object instance into an XRValue (http://go.microsoft.com/fwlink/?LinkId=223716) object, call **XRValue::SetValue(IXREnumerable, bool)**.
6. To set the data context, call IXRFrameworkElement::SetDataContext (http://go.microsoft.com/fwlink/?LinkId=223718) on the root element of the visual host, and pass the XRValue (http://go.microsoft.com/fwlink/?LinkId=223716) object as the *Value* parameter.

   Note

   For applications that display data from multiple data source collections, call IXRFrameworkElement::SetDataContext (http://go.microsoft.com/fwlink/?LinkId=223718) on each list box, user control, or content holder that will display the data stored in each collection.
7. To set the data collection as the source of items displayed in the list box, call the derived method IXRItemsControl::SetItemsSource (http://go.microsoft.com/fwlink/?LinkID=223719) on the **IXRListBox** pointer.

   Note

   This step is equivalent to adding an **ItemsSource** attribute to a **ListBox** element in XAML.

Update the Data (Optional)

You can extend your application to support dynamic data collections.

Data from a data provider can change periodically. For example, new email messages appear in a user's Inbox, television program listings change each day, and a user might add songs to a media playlist. When new data is available in the data provider, the application can update the collection.

You can programmatically update data by doing the following:

- Updating properties in an existing object
- Adding, removing, or replacing objects in an existing collection

The programming steps necessary to implement checking for new data or receiving notifications when new data is available are beyond the scope of this white paper. These steps will be specific to your implementation and your purpose for populating elements in your Silverlight for Windows Embedded UI with dynamic data. General guidelines for connecting to a data provider to check for new data are provided in Connect to a Data Provider.

If you decide to implement functionality for dynamic data collections, make sure that you implement a mechanism that retrieves update data synchronously from the data provider. The mechanism must work correctly with the UI threading model so that the UI thread is blocked while the application is updating data.

Connect to a Data Provider

Before you can update a data source collection with new data from the provider, you must connect the Windows Embedded Silverlight Tools subproject to the provider.

The specific steps to accomplish this depend on the data provider that your application uses. In general, for any type of Silverlight for Windows Embedded application, you must do the following to connect to a data provider:

1. Add the Data Provider's Supporting Files to your Application Subproject
2. Add Functionality to Respond to Notifications from the Data Provider
3. Convert Data Types for Properties in TPropertyBag<Derived>

To illustrate this task, this section of the white paper refers to examples for an email application that uses the Windows Embedded Compact Messaging API (CE MAPI).

Add the Data Provider's Supporting Files to your Application Subproject

To connect to a data provider, you include the header (.h) files and library (.lib) files that the provider uses to represent its data.

Note

The header files to include, and the order in which you define the `#include` statements, are specific to the functionality of your data provider.

For example, for a CEMAPI email message application, follow the steps below to add the supporting files to your application subproject.

To add a CEMAPI .h file to a Windows Embedded Silverlight Tools subproject

1. In Platform Builder, open the application subproject that you created in Create the Expression Blend Project and the Windows Embedded Silverlight Tools Subproject.
2. In MainPage.cpp, add `#include "mapidefs.h"` immediately following the `#include "stdafx.h"` statement.
3. Repeat step 2 for any other source code file in which you will use CEMAPI programming elements combined with Silverlight for Windows Embedded programming elements.

To link the CEMAPI .lib file to a Windows Embedded Silverlight Tools subproject

1. In the Solution Explorer in Platform Builder, right-click the subproject, and then choose

Properties.

2. Click **Link**.
3. In the **Additional Libraries** text box, append the following text to the text string, and then click OK:

   ```
   $(SG_OUTPUT_ROOT)\sdk\lib\$(_CPUINDPATH)\cemapi.lib
   ```

After you append the text, the contents of the text box are as follows:

```
$(SG_OUTPUT_ROOT)\sdk\lib\$(_CPUINDPATH)\coredll.lib
$(SG_OUTPUT_ROOT)\sdk\lib\$(_CPUINDPATH)\xamlruntime.lib
$(SG_OUTPUT_ROOT)\sdk\lib\$(_CPUINDPATH)\oleaut32.lib
$(SG_OUTPUT_ROOT)\sdk\lib\$(_CPUINDPATH)\cemapi.lib
```

After you add the required .h and .lib files, you are ready to start adding functionality that responds to notifications that the data provider sends when it updates its data.

Add Functionality to Respond to Notifications from the Data Provider

When a data provider's data changes, it must notify applications to which it is connected. Then, those applications can update their data source objects and display the new data in the UI.

To connect to a data provider, you must add functionality to your application that responds to notifications and obtains new data after each notification is received.

For example, you can add the functionality to an event handler for a UI element such as a "Check for Updated Data" button.

The functionality that responds to notifications and updates data must implement thread safety. Otherwise, a user updating the data by using the UI and a data provider updating the data might occur simultaneously, which can compromise data integrity. For example, you can use the XRAutoCriticalSection (http://go.microsoft.com/fwlink/?LinkId=227480) class to create a critical section object that blocks access to the thread until the updated data is obtained.

Note

Different data providers have different implementations of sending notifications to applications.

For example, for a CEMAPI email message application, you can implement functionality that registers for and responds to notifications by creating a MAPI advise sink object and obtaining event data from a NOTIFICATION programming element. For more information, see Event Notification in MAPI (http://go.microsoft.com/fwlink/?LinkId=228769), Handling Notifications (http://go.microsoft.com/fwlink/?LinkId=228772), and Handling an Incoming Message (http://go.microsoft.com/fwlink/?LinkId=228770).

After you add functionality to respond to notifications and obtain new data from the data provider, you must then verify that the data is in the correct format before you add it to a **TPropertyBag<Derived>** object.

Convert Data Types for Properties in TPropertyBag<Derived>

When you receive new data from a data provider, you must verify that the data is represented by a data type that is appropriate for a **TBoundPropertyBase<PropertyType,StoreType>** derived object. Data from a provider is not guaranteed to be represented by a data type that is compatible with the **TBoundPropertyBase<PropertyType,StoreType>** derived objects that represent properties in Silverlight for Windows Embedded. For example, not all strings in all data providers are of type **BSTR**. Then, you can add the data to a **TBoundPropertyBase<PropertyType,StoreType>** derived object.

The following list describes how to treat the various types of data:

- For data that uses common types such as integer, float, and Boolean, type conversion is not required. You can add the new value directly to a TBoundProperty<PropertyType> (http://go.microsoft.com/fwlink/?LinkId=227477) object by calling its derived method **TBoundPropertyBase.Set(XRValue *)** or **TBoundPropertyBase.Set(const PropertyType&)**.
- For text data in strings, you must first convert the string to a **BSTR** data type before you add it to a TBoundProperty<BSTR> (http://go.microsoft.com/fwlink/?LinkId=227478) object. For example, provide the string as the input parameter to the **SysAllocString** function when you allocate the **BSTR** string.
- For data that is represented as an object, you must add the value to a TBoundPointerProperty<PropertyType> (http://go.microsoft.com/fwlink/?LinkId=227476) object. For example, you can use **TBoundPointerProperty.operator=(const PropertyType *)** to assign a new value to the object.

The following example code defines a text string that emulates the property value of a PR_SENDER_NAME property from a MAPI message object, converts it to a **BSTR**, and sets a new value of the **Name** property in a **TPropertyBag<Derived>** data object.

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.

```
#include “XRPropertyBag.h”

#include “XamlRuntime.h”


LPWSTR EmailSenderName = TEXT("Sanjay Patel");


XRValue updatedxrvalue;

updatedxrvalue.bstrStringVal = SysAllocString(EmailSenderName);
updatedxrvalue.vType = VTYPE_BSTR;


m_DataObject->SetValue(L"Name", &updatedxrvalue);
```

The following example code converts an interface pointer to an XRPtr<Interface> (http://go.microsoft.com/fwlink/?LinkId=229300) smart pointer, and then sets the smart pointer as the value of a **TBoundPointerProperty<PropertyType>** object in a **TPropertyBag<Derived>** object called `ClassName`.

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.

```
#include “XRPropertyBag.h”

#include “XamlRuntime.h”

#include “XRPtr.h”

#include “Data.h”


// Define global variable for data source object.

XRPtr<ClassName> m_Object;


HRESULT MainPage::ConvertType(IXRPropertyBag* pData)

{

      // Initialize smart pointer and assign it the value of an interface pointer.

     XRPtr<IXRPropertyBag> pXRObject;

     pXRObject = pData;


     // Set the value of a property field in m_Object to the smart pointer.

     m_Object->pObject = pXRObject;


     return S_OK;

}
```

`Contents of Data.h file:`

```
class _declspec(uuid("{0ED08C01-200A-42ed-BB7C-A4ED016B65B7}")) ClassName : public
TPropertyBag<ClassName>
```

```
{
protected:
     // To create an instance of this class, use the TPropertyBag.CreateInstance
method and not the default constructor.
     ClassName() {};

public:
     TBoundProperty<BSTR> bstrProp1;
     TBoundProperty<BSTR> bstrProp2;
     TBoundPointerProperty<IXRPropertyBag> pObject;

     HRESULT InitializeProperties()
     {
          HRESULT hr = S_OK;
          hr = BeginRegisterProperties();
          if (FAILED(hr))
          {
               return hr;
          }
          hr = RegisterBoundProperty(L"Name", bstrProp1);
          hr = RegisterBoundProperty(L"Address", bstrProp2);
          hr = RegisterBoundProperty(L"Object1", pObject);
          hr = EndRegisterProperties();
          return hr;
     }
};
```

After you verify that the data is of the appropriate type, you can update a data source property or modify items in a data source collection.

Update Data Source Properties

You can update the data source properties of an existing object, such as an object that represents a media file, in the data source collection that you created in Create a Data Source Collection.

For example, consider a user who edits media files in a playlist and adds metadata for album name and album artist. When the user reloads the playlist on a Windows Embedded Compact device, the media application detects the changes and updates the **AlbumName** and **AlbumArtist** properties of items in an existing collection.

To update data source properties

1. In Platform Builder, open your OS design project.
2. In your subproject, open the source code file where you want to add functionality for updating the data. For example, in Solution Explorer, browse to Subprojects\<*Subproject Name*>\Source files, and open MainPage.cpp.
3. Obtain a pointer to the XRObservableCollection<ItemType> (http://go.microsoft.com/fwlink/?LinkId=223714) collection.
4. To block other threads from accessing the collection object, implement a thread safety mechanism. For example, create an XRAutoCriticalSection (http://go.microsoft.com/fwlink/?LinkId=227480) object, and call EnterCriticalSection (http://go.microsoft.com/fwlink/?LinkId=226720).
5. Obtain a pointer to a data-source object in the collection. For example, call the derived method **XRValueCollectionT.GetItem(int,Obj**)** to obtain a pointer to an item in the collection.
6. To obtain the current value of a property in the data source object, call **TPropertyBag.GetValue(const WCHAR *,XRValue *)**.
7. To set a new value for a property, call **TPropertyBag.SetValue(const WCHAR *,XRValue *)**.
8. Release ownership of the thread. For example, call LeaveCriticalSection (http://go.microsoft.com/fwlink/?LinkId=226721).

The following example code obtains the value of a property of an item in a data source collection and updates its value.

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.

```
#include "XRPropertyBag.h"

#include "XRCollection.h"

#include "oleauto.h"


HRESULT MainPage::UpdateData()

{

       // Create a data source collection with one item

       XRPtr<XRObservableCollection<ClassName*>,IXREnumerable> pCollection;

       XRObservableCollection<ClassName*>::CreateInstance(&pCollection);
```

```
XRPtr<ClassName> pItem;
pItem->InitializeProperties();
pCollection->Add(pItem);

// Define variables for the data source object and property.
XRPtr<ClassName> pObj;
XRValue xrvalueCurrent;
XRValue xrvalueNew;
TBoundProperty<BSTR> bstrProp;

// Set a BSTR string for the new XRValue object.
bstrProp = SysAllocString(L"UpdatedStringValue");
xrvalueNew.bstrStringVal = bstrProp;

// Define a critical section object.
XRAutoCriticalSection csObject;

// Obtain an item from the collection.
pCollection->GetItem(0, &pObj);

// Request mutually exclusive access to the thread.
EnterCriticalSection(&csObject);

// Update a data source property.
pObj->GetValue(L"PropName1", &xrvalueCurrent);

if (xrvalueCurrent.bstrStringVal != bstrProp)
{
   pObj->SetValue(L"PropName1", &xrvalueNew);
}

// Release the critical section object.
LeaveCriticalSection(&csObject);
```

```
        return S_OK;
}
```

Update a Data Source Collection

You can update the items in a data source collection. For example, if a user adds or deletes media files in a playlist and then reloads the playlist on a device, the media application can update the collection.

To update a data source collection

1. In Platform Builder, open your OS design project.
2. In your subproject, open the source code file where you want to add functionality for updating the data source collection. For example, in Solution Explorer, browse to Subprojects\<*Subproject Name*>\Source files, and open MainPage.cpp.
3. To block other threads from accessing the collection object, implement a thread safety mechanism. For example, create an XRAutoCriticalSection (http://go.microsoft.com/fwlink/?LinkId=227480) object and call EnterCriticalSection (http://go.microsoft.com/fwlink/?LinkId=226721).
4. Obtain a pointer to the XRObservableCollection<ItemType> (http://go.microsoft.com/fwlink/?LinkId=223714) collection. For example, call the derived method **IXRItemsControl::GetItemsSource**.
5. Update the data source collection.
    a. To add a new item, call the derived method **XRValueCollectionT.Add(const ItemType&)**.
    b. To remove an item, call the derived method **XRValueCollectionT.Remove(const ItemType&)**.
    c. To replace an item, call the derived method **XRValueCollectionT.SetItem(int,const ItemType&)**.

1. To raise the **CollectionChanged** event, call **XRObservableCollection.OnCollectionChanged(XRCollectionChangedCustomEventArgs *)**.
2. Release ownership of the thread. For example, call LeaveCriticalSection (http://go.microsoft.com/fwlink/?LinkId=226721).

Add Support for Users to Update the Data (Optional)

In your application's UI, you can add support that allows users to update data in the data source collection.

For example, an inventory tracking application might display a collection of items. When the user selects an item, the application can populate text boxes or text elements with data about the item. In order for the user to update this data, the application must support user updates to the data.

For editable data-bound elements, such as **TextBox** elements, you can add support for bidirectional updates so that a user's changes to a data-bound element are transferred to the data source and update the value of the C++ data source property. In Microsoft Silverlight 3, this functionality is known as ***two-way binding***.

For data-bound elements that cannot be edited, such as a **TextBlock**, you can add C++ functionality that calls **TBoundPropertyBase.Set** in a derived class to update the data source property based on user interaction.

Add C++ event handling functionality for users to update the data

1. In Platform Builder, open your subproject.
2. In Solution Explorer, expand the subproject, expand **Resource files**, and then open MainPage.xaml.
3. Create an event-handling method that changes a property of an item in the data source collection.
   a. On the **Tools** menu, point to **Windows Embedded Silverlight Tools**, and then click **Windows Embedded Events**.
   b. In the **Windows Embedded Events** window, select the UI element to which you want to add an event-handling method.
   c. Locate the event in the list, and double-click in the text field on the right.
   d. In MainPage.cpp, add functionality that retrieves a data-source object instance and updates its property value based on user interaction.

The following example code shows an event handler that increments the value of a **TBoundProperty<int>** data source property when the user clicks a button.

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.

```
#include "MainPage.h"

#include “XRPropertyBag.h”
```

```
XRPtr<IntegerPropertyBag>  m_DataSourceObject;


HRESULT MainPage::Increment_Click (IXRDependencyObject* pSender,
XRMouseButtonEventArgs* pArgs)
{
    HRESULT hr = E_NOTIMPL;

    if ((NULL == pSender) || (NULL == pArgs))
    {
        hr = E_INVALIDARG;
    }
     TBoundProperty<int>* IntProp;
     iXRPropertyBinding* pProperty = NULL;
     pProperty = m_DataSourceObject->GetPropertyByName(L"IntegerData1");

     IntProp = (TBoundProperty<int>*)pProperty;
     int intValue = IntProp->Get();
     intValue = intValue + 1;
     IntProp->Set(intValue);

     // Update layout to see whether the data-bound element is updated
     m_pLayoutRoot->UpdateLayout();

    return hr;
}
```

Add support for bidirectional updates between a UI element and a property

1. In Expression Blend 3, open the project you created in Create the Expression Blend Project and the Windows Embedded Silverlight Tools Subproject.
2. Find the UI element that you want users to edit with an updated property value (for example, a **TextBox**).
3. In the Binding Markup Extension (http://go.microsoft.com/fwlink/?LinkID=219141) for the UI element, add `Mode=TwoWay`. For example:

```
<TextBox x:Name="UserTextData" Height="47" HorizontalAlignment="Right"
```

```
Margin="0,167,75,0" VerticalAlignment="Top" Width="231"
Text="{Binding BinaryStringData1, Mode=TwoWay}" TextWrapping="Wrap"
FontSize="16"/>
```

4. Save your changes.

After you update the XAML or the C++ source code to add support for user updates, you must update and build the project in Platform Builder.

Update and Build the Project

After you create the Expression Blend project and the Windows Embedded Silverlight Tools subproject, and add C++ code to populate the list box with data, you can build and run the Silverlight for Windows Embedded application.

If you made changes to the Expression Blend project after you created the subproject in Platform Builder, you must update the subproject by using Windows Embedded Silverlight Tools.

To update the subproject

1. In Platform Builder, open the subproject.
2. Open a file (for example, MainPage.cpp) in the subproject.
3. On the **Tools** menu, point to **Windows Embedded Silverlight Tools**, and then click **Update Silverlight for Windows Embedded Project**.
4. On the **File** menu, click **Save All**.

To build and run the subproject

1. In Solution Explorer, expand **Subprojects**, right-click the name of your subproject, and then click **Build**.
2. Verify that no errors or warnings are reported in the **Output** window.
3. Start the virtual CEPC that you already preconfigured. If you have not done so already, see Getting Started with Virtual CEPC (http://go.microsoft.com/fwlink/?LinkId=199788).
4. On the **Target** menu, click **Attach Device**.
5. When the run-time image loads on the virtual CEPC, run your application.
   a. On the **Target** menu, click **Run Programs**.
   b. In the **Run Program** dialog box, click the file name of your application with the .exe extension, and then click **Run**.

   c. In virtual CEPC, check the UI to verify that the data that you added to the XRObservableCollection<ItemType> (http://go.microsoft.com/fwlink/?LinkId=223714) is displayed in the list box and in any additional data-bound elements.

Conclusion

In an application based on Silverlight for Windows Embedded, you can implement C++ functionality that populates UI elements with data from a data source collection. First, you create an Expression Blend project that has a **ListBox** element and data-bound elements that use Binding Markup Extensions (http://go.microsoft.com/fwlink/?LinkID=219141). Then, you use Windows Embedded Silverlight Tools to create a subproject in Platform Builder that uses your Expression Blend project as its foundation. Next, you add C++ code to the subproject to create a data class and a data source collection, and to establish a connection between the collection and the list box. You can optionally add functionality to update the data with new data from the data provider.

By using the data-binding classes and interfaces in Silverlight for Windows Embedded, you can build email applications, set-top box applications, media player applications, or any type of application that you want to display data that is maintained separately from the UI.

Additional Resources

To learn more about Silverlight for Windows Embedded, see the following resources:

- A Sample Application Tutorial Using Windows Embedded Silverlight Tools (http://go.microsoft.com/fwlink/?linkid=189508)
- Performance Tuning Guide for Silverlight for Windows Embedded (http://go.microsoft.com/fwlink/?LinkId=205643)
- Video: Create a Silverlight for Windows Embedded Application Part 1 (http://go.microsoft.com/fwlink/?LinkId=223337)
- Video: Create a Silverlight for Windows Embedded Application Part 2 (http://go.microsoft.com/fwlink/?LinkId=223339)