MVA jumpstart

04 | Sharing Code Between Windows 8 and Windows Phone 8 in Visual Studio

Ben Riga
http://about.me/ben.riga

Microsoft

Course Topics

Agenda

Reuse techniques

Portable library

Shared source code

Inheritance

Conditional compilation

Partial classes and methods

Q&A

Reuse Techniques

Windows

Portable Library

Portable Class Library project template in Visual Studio 2012 (except Express)

C# or Visual Basic

Good for creating a cross-platform .dll with pure business logic

Only managed code

Single codebase for Windows, Silverlight, Windows Phone, or Xbox

Only shared assemblies are available for use (mostly restricted to System namespaces)

Overall, limited, but provides portable, modular, and encapsulated code

Demo 1:
Portable Class Library

Windows

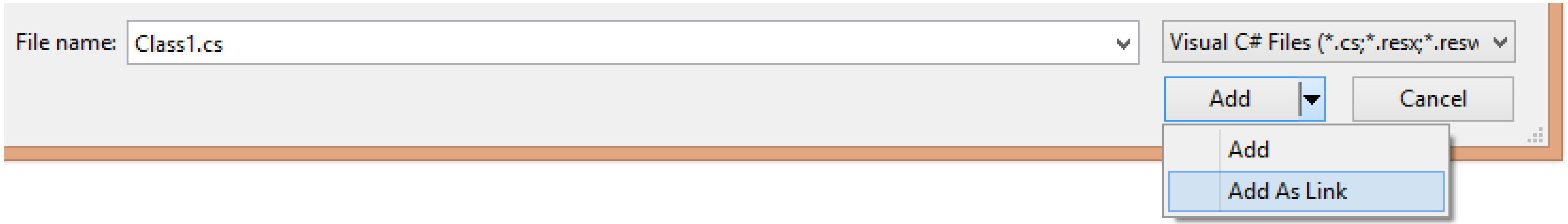
Shared Source Code

Share code files between multiple projects

A single change shows up in all projects

Use **Add As Link** to include an external code file in multiple projects

When the APIs don’t match, use another technique to separate platform-specific code, such as: inheritance, #if conditional blocks, or partial classes and methods.


File name: Class1.cs

Visual C# Files (*.cs;*.resx;*.resw

#if Conditional Blocks

Enable/disable pieces of code based on target platform (or any other compiler variable)

Platform-specific compilation constants:

NETFX_CORE	Windows 8
WINDOWS_PHONE	Windows Phone 8

Useful for when there are subtle differences in the APIs

If overused, code can become unreadable and hard to maintain

#if Conditional Block Examples (1/2)

```
#if NETFX_CORE
            using Windows.UI.Xaml.Media.Imaging;
#else
            using System.Windows.Media.Imaging;
#endif
```

#if Conditional Block Examples (2/2)

```
public object Convert(object value, Type targetType, object parameter,
           #if !NETFX_CORE
                      CultureInfo culture
           #else
                      string language
           #endif
)
```

Demo 2:
#if Conditional Blocks

Windows

Inheritance

Put shared functionality in base class and platform-specific code in sub-classes

Separates cross-platform features from platform-specific features

Base class can be abstract to enforce platform-specific implementations

Two main approaches:

The base class contains fully implemented logic. The sub-classes provide additional platform-specific functionality.

The base class is incomplete. Subclasses implement the abstract methods to provide platform-specific functionality.

Inheritance Examples (1/2)

```
/// <summary>
/// Base View Model used across both platforms
/// </summary>
public class AlbumPageViewModel : NotifyBase, IViewModel
{ ... }
/// <summary>
/// Sub-class containing additional information used in the Windows application
/// </summary>
public class WinAlbumPageViewModel : PhotoCollectionViewModel, ISupportsDesignTimeDataViaCode
{
}
```

Inheritance Examples (2/2)

```
/// <summary>
/// Abstract base class available across both platforms
/// </summary>
public abstract class StartPageViewModel : NotifyBase,IViewModel
{
    protected abstract bool NavigatedToPhoto();
}
/// <summary>
/// Sub-class implementing the abstract method
/// </summary>
public class WinStartPageViewModel : StartPageViewModel, ISupportsDesignTimeDataViaCode
{
    protected override bool NavigatedToPhoto()
    {
        NavigationService.Navigate<WinPhotoCollectionViewModel>(SelectedPicture.Path);
        return true;
    }
}
```

Demo 3: Inheritance

Partial Classes

Shared functionality in one code file

Ex: DataModel.cs

Platform-specific code in another code file

Ex: DataModel.WP8.cs

Partial class definitions compiled into a single class

Partial Methods

Can use partial methods as a mechanism to separate out platform-specific logic

Two parts: the definition and the implementation

Both parts must be in the same partial class

If no implementation exists, the compiler removes all calls to the method

Partial Classes Example

```
/// <summary>
/// DataModel.cs
/// </summary>
public partial class DataModel:IDataModel {
    public async Task<IEnumerable<IFolder>> GetFoldersAsync(IFolder root) {
        // ...
        var folders = await LoadFolders(root);
        // ...
        return folders
    }
}
/// <summary>
/// DataModel.WP8.cs
/// </summary>
public partial class DataModel {
    private async Task<IEnumerable<IFolder>> LoadFoldersAsync(IFolder root) {
        // ...
    }
}
```

Demo 4:
Partial Classes

Windows

Recap

Windows

Takeaways

Develop apps for both Windows Phone 8 and Windows 8!

Even if you do not initially target both platforms, it does not hurt to plan for the future.

Be aware of which APIs are common to both platforms and which are not.

Use DataBinding to make your life easier when creating complex UIs that need to be dynamically updated with data.

Use commands to execute a piece of code when a button is clicked

MVVM works well with data binding

Takeaways

Consider designing your apps around the MVVM pattern.

Even if you do not use MVVM, separate the data model from the UI. This lets you reuse the data model in the future.

Use strategies so that you can reuse source code

Portable libraries

Shared source files

#if conditional blocks

Inheritance

Partial classes, partial methods

Additional Resources

Windows 8, Windows Phone 8 APIs

Windows Phone API QuickStart

Data Binding

Data Binding Overview

XAML Data binding sample

MVVM

"Using the MVVM Pattern in Windows"

"MVVM Light Toolkit"

Sharing Code

Windows Phone 8 and Windows 8 app development

Sample Code

"Windows 8 app samples"

"Windows Phone samples"

All resources available at:
http://bit.ly/BuildForBoth

Q&A

Windows

Thank you!

Windows