.Net Resources

Agenda

- Why Use Resource Files
- Types of Resource Files
- URIs

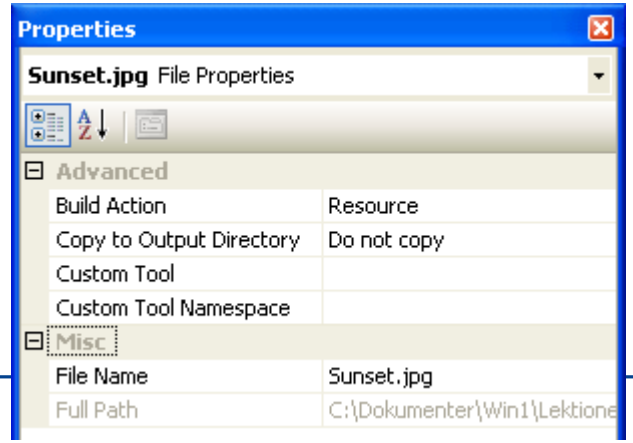
Why Resources

- If you want to build a graphically distinctive application, the resource system provides a straightforward way to *skin* your applications with customized yet consistent visuals

- The resource handling infrastructure is entirely dedicated to making it possible to get hold of the resource you require
  - It doesn't care what the resource is
  - It simply provides a mechanism for identifying and locating objects

Resource Files

- If an application data file must always be available to an application
  - then you can compile it into an application's main executable assembly or one of its referenced assemblies
  - This type of application data file is known as a ***resource file***
- You should use resource files when:
  - You don't need to update the resource file's content after it is compiled into an assembly
  - You want to simplify application distribution complexity by reducing the number of file dependencies
  - Your application data file needs to be localizable
- In Microsoft Visual Studio, you create a resource file by adding a file to a project and setting its **Build Action** to **Resource (**or **Embedded Resource)**



Types of Resource Files

Build Action	API	What it Does
Content	`Application.GetContentStream`	Copies each resource to the directory of the application
Resource	`Application.GetResourceStream`	Embeds each resource in a common resource in the application
EmbeddedResource	`Application.GetManifestResourceStream`	Embeds each resource in the application separately
Page *Only valid for:* – *Window* – *NavigationWindow* – *Page,* – *FlowDocument* – *ResourceDictionary*		The XAML items are converted to binary format and compiled into the associated assembly. These files can be used in the same way as typical resource files.

Using Resource Files

- All resources files are put into a single manifest resource stream called *appname*.g.resources
- To load a resource file, you can call the GetResourceStream method of the Application class, passing a pack URI that identifies the desired resource file
- GetResourceStream returns a StreamResourceInfo object, which exposes the resource file as a Stream and describes its content type

```
public Stream GetImageAsResource()
{
   Uri resourcePath = new Uri("Images/Sunset.jpg",
                                        UriKind.Relative);
   StreamResourceInfo ri =
            Application.GetResourceStream(resourcePath);
   Stream data = ri.Stream;
   return data;
}
```

Using Embedded Resource Files

- Files with build action set to “Embedded Resource” are build into the containing assembly as separate resource streams in the manifest section of the assembly
- From code you can retrieve those files using the Assembly class’s GetManifestResourceStream method

```
private void UseAssemblyManifestResource()
{
   Assembly asm = Assembly.GetExecutingAssembly();
   Stream s = asm.GetManifestResourceStream
                        ("BinaryResources.MyStream.txt");
   StreamReader r = new StreamReader(s);
   WriteLine(r.ReadToEnd());
}
```

Build Action Content

- A ***content file*** is distributed as a loose file alongside an executable assembly
- Although they are not compiled into an assembly, assemblies are compiled with metadata that establishes an association with each content file
- You should use content files when your application requires a specific set of application data files that you want to be able to update without recompiling the assembly that consumes them
- When the project is built, an attribute is compiled into the metadata of the assembly for each content file

```
[assembly: AssemblyAssociatedContentFile("ContentFile.xaml")]
```

Loading Loose Files

- If you don’t know the file name at compile time, then you have 2 options:
  - Specify the full path to the file, e.g.:
    - <Image Source="**file:///C:/DataFile.bmp**" />
    - <Image Source="**http://www.datafilewebsite.com/DataFile.bmp**" />
    - Note that this require your application to have full trust!
  - Load files only from the application's **site of origin** (launch location)

```
Uri uri = new Uri("/SiteOfOriginFile.xaml",
                  UriKind.Relative);
StreamResourceInfo info = Application.GetRemoteStream(uri);
```

```
Uri pageUri = new
       Uri("pack://siteoforigin:,,,/SiteOfOriginFile.xaml",
            UriKind.Absolute);
this.pageFrame.Source = pageUri;
```

Rebuilding After Changing Build Type

- After you change the build type of an application data file, you need to **rebuild** the entire application to ensure those changes are applied

- **If you only build the application, the changes are not applied!!!**

URIs

Uniform Resource Identifiers

URI

- A Uniform Resource Identifier (URI) is a string of characters used to identify a name or a resource
- You can classify URIs as:
  - Locators (URLs)
  - Names (URNs)
  - Or both
- A Uniform Resource Name (URN) defines an item's identity (a person's name)
- Uniform Resource Locator (URL) provides a method for finding an item (a person's street address)

URI Syntax

- The URI syntax consists of a URI scheme name followed by a colon character, and then by a scheme-specific part:

<scheme name> : <hierarchical part> [? <query>] [# <fragment>]

- Typical schemes:
  - http
  - ftp
  - file

- The **hierarchical part** of the URI is intended to hold identification information hierarchical in nature
  - Usually this part begins with a double forward slash ("//"), followed by an ***authority*** part and an optional ***path***

URIs in WPF

Uniform resource identifiers (URIs) are used to identify and load files in many ways:

- Specifying the window (UI) to show when an application first starts
- Loading images
- Navigating to pages
- Loading non-executable data files

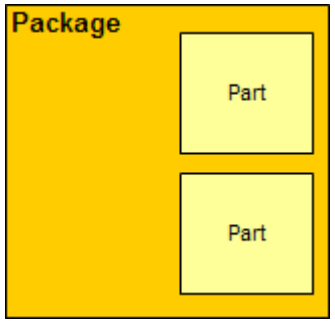
Possible Locations

URIs can be used to identify and load files from a variety of locations:

- The current assembly
- A referenced assembly
- A location relative to an assembly
- The application's site of origin
- An absolute path

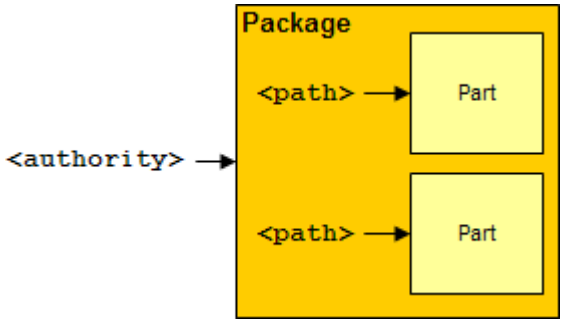
Pack URI Scheme

- The pack URI scheme is used by the Open Packaging Conventions (OPC) specification, which describes a model for organizing and identifying content
- The key elements of this model are packages and parts, where a package is a logical container for one or more logical parts



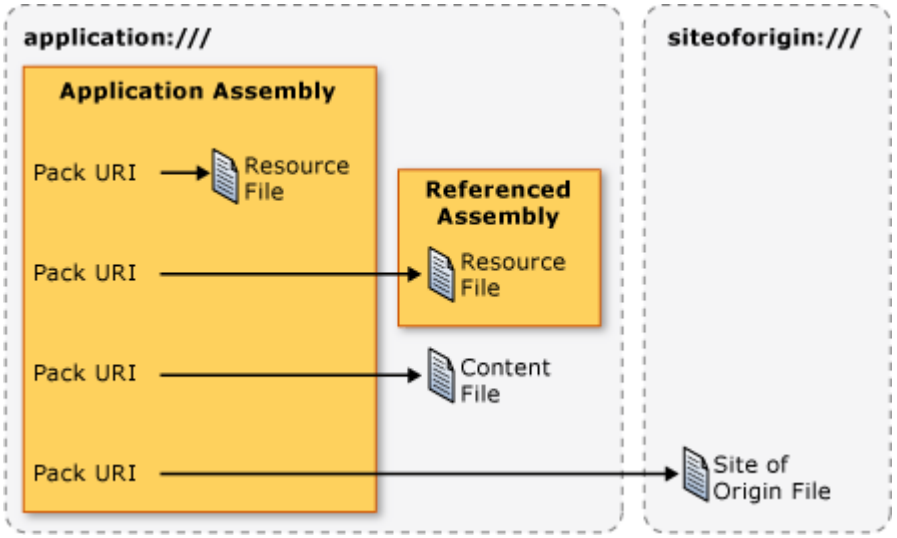
- The format for a pack URI:

```
pack://authority/path
```



Authorities

- WPF supports two authorities:
  - application:///
    - identifies application data files that are known at compile time, including resource and content files
  - siteoforigin:///
    - identifies site of origin files



Pack URI Examples

- The pack URI for a resource file that is compiled into the local assembly uses the following authority and path:

```
pack://application:,,,/FileName.xaml
```

- Content File Pack URIs

```
pack://application:,,,/FileName.xaml
```

- Site of Origin Pack URIs

```
pack://siteoforigin:,,,/FileName.xaml
```

Absolute vs. Relative Pack URIs

- A fully qualified pack URI includes the scheme, the authority, and the path

```
pack://application:,,,/FileName.xaml
```

- A relative pack URI includes only the path.

```
/FileName.xaml
```

- By default, a relative pack URI is considered relative to the location of the markup or code that contains the reference

- **If a leading slash is used, the relative pack URI reference is considered relative to the root of the application**

Pack URI Resolution

- A pack URI can refer to either a resource file in the local assembly or a content file.

```
pack://application:,,,/ResourceOrContentFile.xaml
/ResourceOrContentFile.xaml
```

- WPF resolves URIs by using the following heuristics:
  1. Probe the assembly metadata for an AssemblyAssociatedContentFileAttribute attribute that matches the pack URI
     - If the AssemblyAssociatedContentFileAttribute attribute is found, the path of the pack URI refers to a content file
  2. If the AssemblyAssociatedContentFileAttribute attribute is not found, probe the set resource files that are compiled into the local assembly
     - If a resource file that matches the path of the pack URI is found, the path of the pack URI refers to a resource file
  3. If the resource is not found, the Uri is invalid

References

- Windows Presentation Foundation Application Resource, Content, and Data Files
  http://msdn.microsoft.com/en-us/library/aa970494.aspx

- Pack URIs in Windows Presentation Foundation
  http://msdn.microsoft.com/en-us/library/aa970069.aspx

- Uniform Resource Identifiers (URI): Generic Syntax
  http://www.ietf.org/rfc/rfc2396.txt