Resources

Sharing .NET objects

The WPF resource mechanism provides a standard mechanism for packaging and accessing arbitrary .NET objects. All `FrameworkElement` objects and the `Application` object have a `Resources` property of type `ResourceDictionary` into which objects can be stored and accessed.

Resources are referenced explicitly in in XAML using `StaticResource` or `DynamicResource` markup extensions. The resource resolution mechanisms can walk back up the logical tree enabling any FrameworkElement to access resources stored in ancestor nodes' resource dictionaries or in the application objects resource dictionary (or even system level resource dictionaries). It's is precisely this tree walking feature that makes WPF resources so powerful. Objects can be inserted into the application or window resource dictionaries and referenced from leaf nodes in the logical tree.

Both storing and referencing resources can be done in XAML or procedural code. Since XAML is the most common use case we focus on that first and then later show how the same results can be achieved procedurally.

Every object stored in a resource dictionary must have an associated key. The key can be either explicitly provided using the x:key attribute or implicitly evaluated using some other property of the object being stored. Implicit keys are fundamental to the application of styles and data templates in WPF. A style is stored using an implicit key by omitting the x:key and adding the `TargetType` attribute. A DataTemplate is stored using an implicit key by omitting the x:key attribute and adding the `DataType` attribute.

Referencing resources in XAML is achieved by using either a StaticResource or DynamicResource markup extension. Fundamentally the main difference between the two is that static references are only evaluated once at load time whereas dynamic resources are expressions that are evaluated at runtime enabling them to pick up runtime changes to referenced resource objects. Each has advantages and disadvantages. Static resources are slightly more efficient at runtime but lead to start-up cost and do not support forward references. Dynamic resources are evaluated at runtime and so can pick up runtime changes to stored objects. They are slightly less efficient at runtime but reduce start up overhead in comparison to StaticResource references. Also, DynamicResources can only be used to set the value of DependencyProperties or Freezables

Adding object to resource dictionaries (XAML)

Typically, explicit keys in XAML tend to be strings although the following are all valid

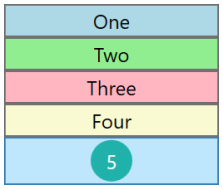
- String
- TypeExtension
- StaticExtension

When adding objects to a resource dictionary in XAML using implicit keys the x:Key attribute is not used. Instead the key is derived from another property of the object being added. Control Styles and DataTemplates are two important places where implicit styles are used. The implicit key for a control's style is set when defining the resource using the `TargetType` attribute and when resolving the resource using the type of the control. The implicit key for a DataTemplate is set when defining the resource using the `DataType` property on the template and when resolving the type of the object.

XAML

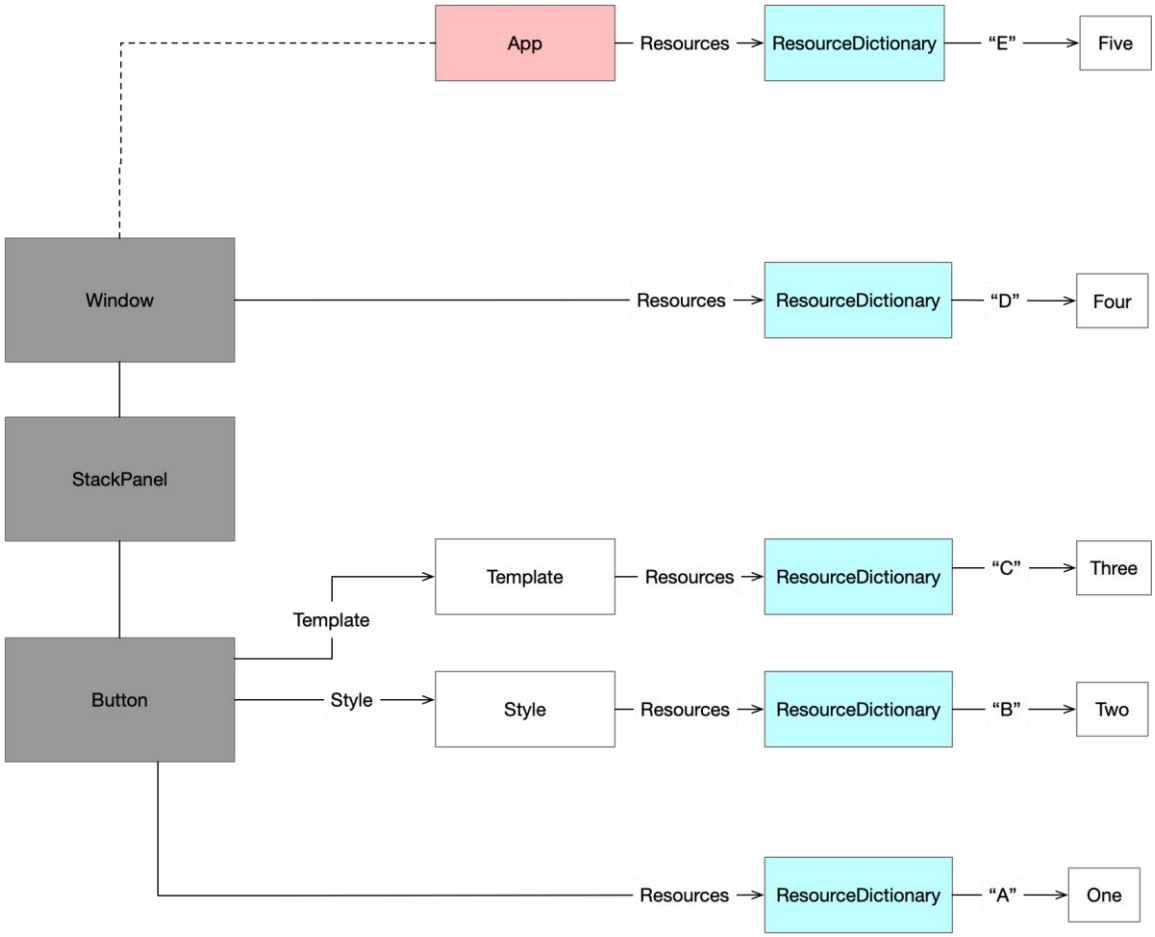
```
<Window.Resources>
    <!-- 1. Explicit string key -->
    <SolidColorBrush x:Key="ResourceOne" Color="LightBlue"/>

    <!-- 2. Explicit key using TypeExtension -->
    <SolidColorBrush x:Key="{x:Type system:String}" Color="LightGreen"/>

    <!-- 3. Explicit key using StaticExtension -->
    <SolidColorBrush x:Key="{x:Static local:Keys.Kenny}" Color="LightPink"/>

    <!-- 4. Implicit Key for style uses TargetType -->
    <Style TargetType="Button">
        <Style.Setters>
            <Setter Property="Background" Value="LightGoldenrodYellow"></Setter>
        </Style.Setters>
    </Style>

    <!-- 5. Implicit Key for datatemplate -->
    <DataTemplate DataType="{x:Type viewModels:Person}">
        <Grid>
            <Ellipse Width="25" Height="25" Fill="LightSeaGreen"></Ellipse>
            <TextBlock HorizontalAlignment="Center" VerticalAlignment="Center"
                       Foreground="White">5</TextBlock>
        </Grid>
    </DataTemplate>
</Window.Resources>

<StackPanel>
    <Button Content="One" Background="{StaticResource ResourceOne}"/>
    <Button Content="Two" Background="{StaticResource {x:Type system:String}}" />
    <Button Content="Three" Background="{StaticResource {x:Static local:Keys.Kenny}}" />
    <Button>Four</Button>
    <Button Content="{viewModels:PersonMarkup}" />
</StackPanel>
```

RENDERED WINDOW



Resolving resource references (XAML)

Consider the following structure of resource



DynamicResource resolution

When a DynamicResource processes a key, it generates an expression that will be evaluated at runtime. DynamicResources can only be used to set Dependency Properties or properties whose type is `Freezable`. The resolution of the markup follows the following steps

1. Look in the current elements resource dictionary
2. If the element has a style set look in the style's resource dictionary
3. If the element has a template set look in the template's resource dictionary
4. Walk back up the logical tree looking in ancestors' resource dictionaries

5. Finally the Application resources are checked (Resources in the ResourceDictionary defined by the WPF Application object for the WPF application)

The following are all valid DynamicResource resolutions with this structure

```
<Button Content="{DynamicResource A}">
    <Button.Resources>
        <system:String x:Key="A">One</system:String>
    </Button.Resources>
</Button>
<Button Content="{DynamicResource B}" />
<Button Content="{DynamicResource C}" />
<Button Content="{DynamicResource D}" />
<Button Content="{DynamicResource E}" />
```

StaticResource resolution

With StaticResource resolution we cannot access resource stored in the buttons template resource collection or its style' resource collection. Also, because forward references are not supported with StaticResource resolutions we can access the resource in our own resource dictionary using XAML property attribute syntax.

1. Look in the current elements resource dictionary
2. Walk back up the logical tree looking in ancestors' resource dictionaries
3. Finally the Application resources are checked (Resources in the ResourceDictionary defined by the WPF Application object for the WPF application)

We need to restructure our Button XAML to use property element syntax instead.

```
<Button>
    <Button.Resources>
        <system:String x:Key="A">One</system:String>
    </Button.Resources>
    <Button.Content>
        <StaticResource ResourceKey="A"></StaticResource>
    </Button.Content>
</Button>
<!-- Runtime error -->
<!--<Button Content="{StaticResource B}" />-->
<!--<Button Content="{StaticResource C}" />-->
<Button Content="{DynamicResource D}" />
<Button Content="{DynamicResource E}" />
```

Static versus Dynamic resolution

STATICRESOURCE MARKUP EXTENSION

- Processes key by searching for it in all parent resource dictionaries
- Does not support forward references from resource dictionaries
- Evaluated at load time

DYNAMICRESOURCE MARKUP EXTENSION

- Processes key by creating an expression that is evaluated at runtime
- Evaluated at runtime
- Can only be used to set values on DependencyProperties or Freezables

Procedural Code

Adding to a resource dictionary

```
this.Resources.Add("SolidColorBrush",new SolidColorBrush() {Color = Colors.Green});
```

Static Lookup

```
SolidColorBrush s = Button2.FindResource("SolidColorBrush1") as SolidColorBrush;
Button2.Background = s;
```

Dynamic Lookup

```
Button3.SetResourceReference(Button.BackgroundProperty,"Brush3");
```

Component Resource Keys

```
    <Window.Resources>
        <SolidColorBrush Color="Aqua"
            x:Key="{ComponentResourceKey TypeInTargetAssembly=local:ComponentResourceKey,
ResourceId=MyBrush}"/>
    </Window.Resources>

    <Grid>
        <Button
            Background="{DynamicResource {ComponentResourceKey
TypeInTargetAssembly=local:ComponentResourceKey, ResourceId=MyBrush}}">
        </Button>
    </Grid>
```

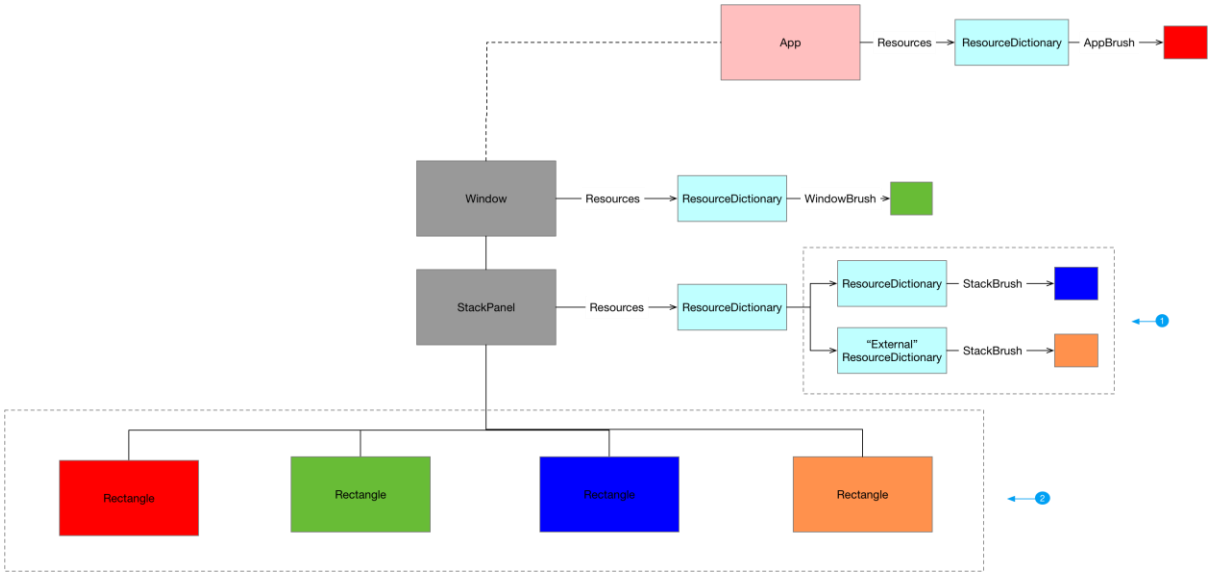
Often a provider of resources will create static fields that hold the ComponentResourceKeys we can use to access that resource colletions resources.

```
public static System.Windows.ComponentResourceKey MyBrushKey
   => new System.Windows.ComponentResourceKey(typeof(Keys),"MyBackground");
```

```
    <Window.Resources>
        <SolidColorBrush Color="Aqua"
            x:Key="{ComponentResourceKey TypeInTargetAssembly=local:ComponentResourceKey,
ResourceId=MyBrush}"/>
        <SolidColorBrush Color="Navy"
                         x:Key="{x:Static local:Keys.MyBrushKey}"/>
    </Window.Resources>

    <StackPanel>
        <Button
            Background="{DynamicResource {ComponentResourceKey
TypeInTargetAssembly=local:ComponentResourceKey, ResourceId=MyBrush}}">
        </Button>
        <Button Content="Click Me" Foreground="White"
            Background="{DynamicResource {x:Static local:Keys.MyBrushKey}}">
        </Button>
```

Resource dictionary tree traversal example

Resources are accessed in XAML using StaticResource extensions or DynamicResource extensions. Before we look at the specifics of each, we should be aware that any given framework element can access the resources of its ancestors in the logical tree as well as any resources in the application resource dictionary. Dictionaries can also be merged to pull in dictionaries from external files. The following diagram and XAML show these features of the resource system.



Listing 1MainWindow.xaml

```
<Window x:Class="Resources.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" SizeToContent="WidthAndHeight">
    <Window.Resources>
        <SolidColorBrush x:Key="WindowBrush" Color="Green"></SolidColorBrush>
    </Window.Resources>

    <StackPanel Orientation="Horizontal">
        <StackPanel.Resources>
            <!-- 1. Resources can be merged -->
            <ResourceDictionary>
                <ResourceDictionary.MergedDictionaries>
                    <ResourceDictionary>
                        <SolidColorBrush x:Key="StackBrush" Color="Blue"/>
                    </ResourceDictionary>
                    <ResourceDictionary Source="ExternalResourceDict.xaml"/>
                </ResourceDictionary.MergedDictionaries>
            </ResourceDictionary>
        </StackPanel.Resources>
        <!-- FrameworkElements can access resources from the resource collections
        of any ancestor in the logical tree and the App resources itself -->
        <Rectangle Width="100" Height="100" Fill="{StaticResource AppBrush}"></Rectangle>
        <Rectangle Width="100" Height="100" Fill="{StaticResource WindowBrush}"></Rectangle>
        <Rectangle Width="100" Height="100" Fill="{StaticResource StackBrush}"></Rectangle>
        <Rectangle Width="100" Height="100" Fill="{StaticResource ExternalResourceBrush}"></Rectangle>
    </StackPanel>
</Window>
```

Resources -Questions

What is the WPF resource mechanism?

The WPF resource mechanism provides a standard mechanism for packaging and accessing arbitrary .NET objects.

Which types have a Resources property?

FrameworkElement

FrameworkContentElement

Application.

What is the type of the Resources property?

ResourceDictionary

What kind of objects are typically stored in ResourceDictionaries?

Templates, Styles, Brushes

What kind of object can be stored in Resource Dictionaries?

Any .NET objects. If we want to instantiate it from XAML it needs to be XAML friendly (Default public constructor etc.)

What makes the resource mechanism so powerful?

The resource resolution mechanisms walks up the logical tree enabling any FrameworkElement to access resources stored in ancestor nodes' resource dictionaries or in the application objects resource dictionary (or even system level resource dictionaries).

Objects can be inserted into the application or window resource dictionaries and referenced from leaf nodes in the logical tree.

Compare and contracts Dynamic and Static lookup?

Static

1. *Only evaluated once at load time*
2. *One time start up cost then more efficient at runtime*
3. *Do not support forward references*

Dynamic

1. *Evaluated at runtime so pick up changes*
2. *Slightly less efficient*
3. *Can only set the value of DependencyProperties*

What is the dynamic resource lookup mechanism.

1. Look in the current elements resource dictionary
2. If the element has a style set look in the style's resource dictionary
3. If the element has a template set look in the template's resource dictionary
4. Walk back up the logical tree looking in ancestors' resource dictionaries
5. Finally the Application resources are checked (Resources in the ResourceDictionary defined by the WPF Application object for the WPF application)

What are implicit keys and where are they used?

The implicit key for a control's style is set when defining the resource using `TargetType`

The implicit key for a DataTemplate is set when defining the resource using the `DataType` property on the template and when resolving the type of the object.

Resolution