WPF Assembly Resources ve Content Files Kavramı

WPF uygulamasında assembly resources kullanımı diğer .NET uygulamalarındaki gibidir. Temel konsept olarak projeye bir dosya eklediğimiz zaman, uygulama derlendiğinde bu dosya EXE veya DLL dosyasına gömülür.

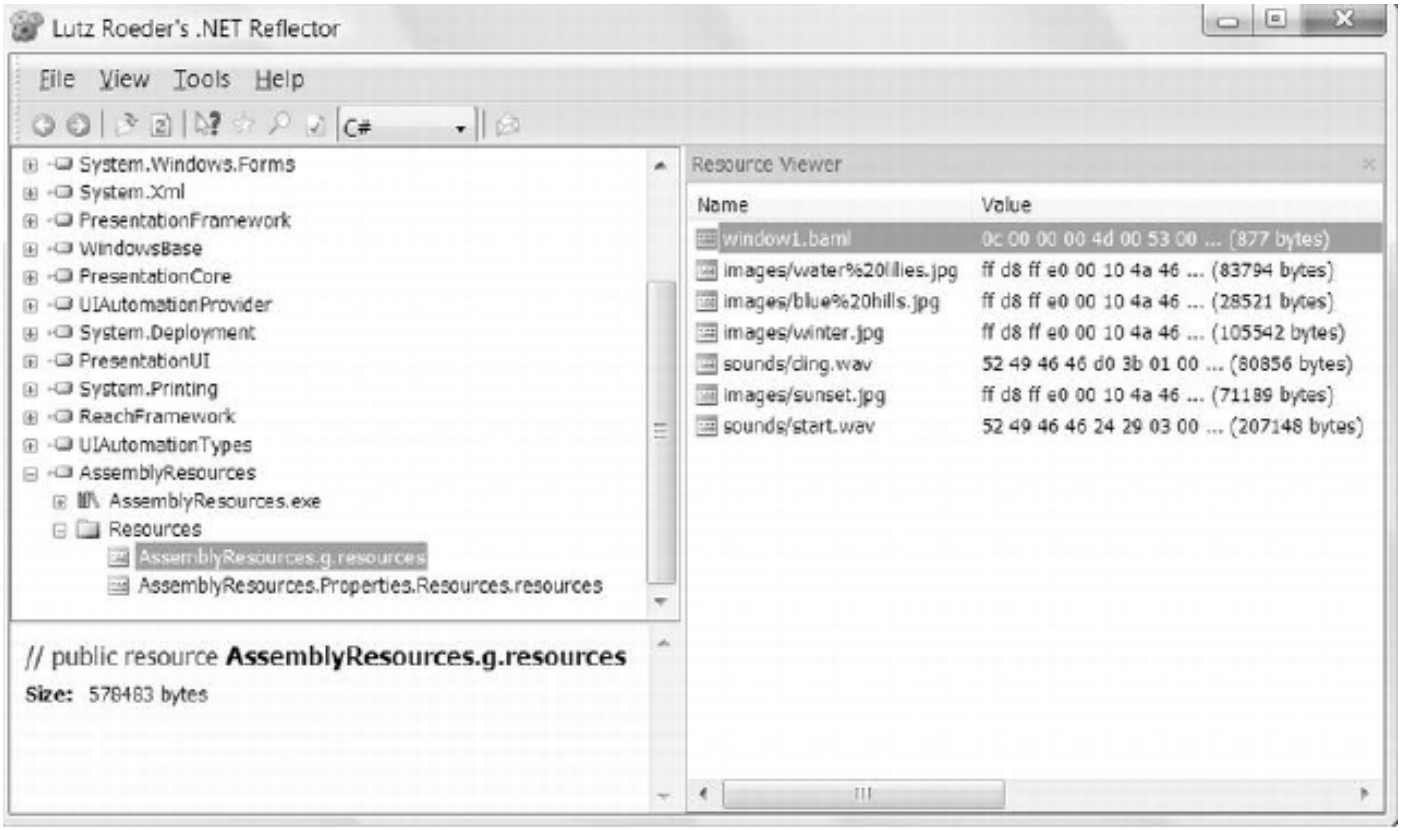
Uygulamamıza bir dosya eklediğimiz zaman dosyanın, Visual Studio'da Properties penceresinde bulunan **Build Action** property'isini **Resource** yapmak gerekmektedir.

Aynı şekilde daha iyi organizasyon yapmak için bir klasör eklediğimizde dosyaya yaptığımız işlemi bu klasöre de yapabiliriz.

Not: WPF, Visual Studio'nun Properties penceresinde çıkan **Resources** kısmını assembly resources olarak kullanmadığı için bu kısma girilen bir url assembly resource olarak algılanmayacaktır.

Not: Eklenen bir dosyanın Build Action property'sini **Embedded Resource** şeklinde **belirtmemeliyiz.** Çünkü bu değer seçildiği zaman dosya erişmesi zor olan bir yere binary olarak kaydedilecektir. Sonuç olarak Build Action property değeri Resource **yapılmalıdır.**

Uygulama derlendiği zaman WPF tüm resource dosyalarını ve BAML (XAML dosyalarının derlenmiş halini) dosyalarını **AssemblyName.g.resources** şeklinde isimlendirilen bir formatta tutar. Ayrıca eklediğimiz dosyaların isimleri de **lowercase** yapılır. Dosyalarda boşluk karakterleri de %20 şeklinde doldurulur.



Retrieving Resources

Aşağıdaki kodlarla eklediğimiz dosyayı okuyabiliriz:

```
1   StreamResourceInfo sri = Application.GetResourceStream(
2       new Uri("images/winter.jpg", UriKind.Relative));
```

StreamResourceInfo nesnesi iki tür bilgi barındırır. Bunlardan birisi verinin tipini içeren **ContentType** property'dir. Bu property verinin tipini string olarak dönderir. Örnekte bir JPG dosyası okunduğu için **content-type** image/jpg olur. İkincisi ise **Stream** property'dir. Bu property **UnmanagedMemoryStream** nesnesi dönderir. Bu nesne ile byte byte dosyayı okuyabiliriz.

Yukarıdaki yöntem low-level olduğu için WPF bize daha gelişmiş sınıflar sunmuştur. Bunlardan birisi **BitmapImage** sınıfıdır:

```
1   img.Source= new BitmapImage(new Uri(@"d:\Photo\Backgrounds\arch.jpg"));
2   //veya
3   img.Source= new BitmapImage(new Uri("images/winter.jpg", UriKind.Relative));
```

Not: XAML dosyasında bir dosyayı kullanmak için aşağıdaki gibi url verebiliriz:

```
<Image Source="Images/Blue hills.jpg"></Image>
```

Pack URIs

WPF uygulamasın BAML gibi derlenmiş kaynaklara erişmek için pack URI sentaksını kullanabiliriz. Örneğin images/winter.jpg relative URL'ye sahip bir dosyaya şu şekilde de erişmek mümkündür: `pack://application:,,,/images/winter.jpg`

```
img.Source= new BitmapImage(new Uri("pack://application:,,,/images/winter.jpg"));
```

Örnekte kullanılan üç tane virgül üç tane escaped slash (/) karakterini ifade etmektedir. Şu şekilde de yazabilirdik: `pack://application:///...`

Not: Absolute url kullandığımız zaman external bir web sitesindeki resmi de uygulamamızda kullanabiliriz. Aynı şekilde bilgisayarımızdaki herhangi bir dosyanın adresini vererek de ulaşabiliriz.

Başka bir assembly içerisinde bulunan bir dosyaya erişmek için şu sentaks kullanılır: `pack://application:,,,/AssemblyName;component/ResourceName` Örneğin ImageLibrary assembly içinde bulunan winter.jpg isimli bir resimi şu şekilde okuruz:

```
img.Source= new BitmapImage(
     new Uri("pack://application:,,,/ImageLibrary;component/images/winter.jpg"));
```

Daha pratik olarak şu şekilde de erişebiliriz:

```
img.Source= new BitmapImage(
     new Uri("ImageLibrary;component/images/winter.jpg", UriKind.Relative));
```

Eğer assembly dosyamız versiyon, public key token bilgilerini de kullanıyorsa şu şekilde kullanmalıyız:

```
img.Source= new BitmapImage(
     new Uri("ImageLibrary;v1.25;component/images/winter.jpg",
     UriKind.Relative));
```

Yukarıdaki örnekte versiyon numarasına sahip bir assembly dosyasındaki winter.jpg isimli dosyaya erişilmiştir. Hem versiyona hem de public-key token'a sahip olduğu zaman:

```
img.Source= new BitmapImage(
     new Uri("ImageLibrary;v1.25;dc642a7f5bd64912;component/images/winter.jpg",
     UriKind.Relative));
```

Content Files

Bazı durumlarda dosyaları WPF uygulamasına resources olarak ekleyemeyiz. Bunun nedeni dosyanın çok büyük olması, ses dosyası olması, dosyanın compile edilmesi yerine direkt erişme isteği gibi durumlarda, bu tip dosyaları content dosyası olarak işaretlemek gerekir.

Bunu yapabilmek için Visual Studio'da Build Action değerini **Content** yapmalıyız. Örnek olarak bir ses dosyasını kullanmak için şu tarz bir kod yazılabilir:

```
<MediaElement Name="Sound" Source="Sounds/start.wav"
LoadedBehavior="Manual"></MediaElement>
```

Not: Ses dosyasının açılabilmesi için ayrıca **Copy to Output Directory** ayarının **Copy Always** yapılması gereklidir.

WPF bu tarz dosyaların varlığında assembly'e **AssemblyAssociatedContentFile** attribute ekler. Bu attribute içerisinde dosya eklenme biçimi **Content** olan dosyaların lokasyonlarını tutulur.