

ASP.NET Web Page Resources Overview

10/22/2014 • 12 minutes to read

In this article

[Resource Files](#)

[Creating Resource Files for ASP.NET Web Sites](#)

[Working with Resources in Web Pages](#)

[Selecting Resource Files for Different Languages](#)

[Working with Resources Programmatically](#)

[See Also](#)

If you create Web pages that will be read by speakers of different languages, you must provide a way for readers to view the page in their own language. One approach is to re-create the page in each language. However, that approach can be labor intensive, error prone, and difficult to maintain as you change the original page.

ASP.NET enables you to create a page that can obtain content and other data based on the preferred language setting for the browser or based on the user's explicit choice of language. Content and other data is referred to as resources and such data can be stored in resource files or other sources

In the ASP.NET Web page, you configure controls to get their property values from resources. At run time, the resource expressions are replaced by resources from the appropriate resource file.

Resource Files

A resource file is an XML file that contains the strings that you want to translate into different languages or paths to images. The resource file contains key/value pairs. Each pair is an individual resource. Key names are not case sensitive. For example, a resource file might contain a resource with the key Button1 and the value Submit.

You create a separate resource file for each language (for example, English and French) or for a language and culture (for example English [U.K.], English [U.S.]). Each localized resource file has the same key/value pairs; the only difference is that a localized resource file can contain fewer resources than the default resource file. The built-in language fallback process then handles loading the neutral or default resource.

Resource files in ASP.NET have an .resx extension. At run time, the .resx file is compiled into an assembly, which is sometimes referred to as a satellite assembly. Because the .resx files are compiled dynamically, like ASP.NET Web pages, you do not have to create the resource assemblies. The compilation condenses several similar-language resource files into the same assembly.

When you create resource files, you start by creating a base .resx file. For each language that you want to support, create a new file that has the same file name. But in the name, include the language or the language and culture (culture name). For a list of culture names, see the [CultureInfo](#) class. For example, you might create the following files:

- WebResources.resx

The base resource file. This is the default (fallback) resource file.

- WebResources.es.resx

A resource file for Spanish.

- WebResources.es-mx.resx

A resource file for Spanish (Mexico) specifically.

- WebResources.de.resx

A resource file for German.

At run time, ASP.NET uses the resource file that is the best match for the setting of the [CurrentUICulture](#) property. The UI culture for the thread is set according to the UI culture of the page. For example, if the current UI culture is Spanish, ASP.NET uses the compiled version of the WebResources.es.resx file. If there is no match for the current UI culture, ASP.NET uses resource fallback. It starts by searching for resources for a specific culture. If those are not available, it searches for the

resources for a neutral culture. If these are not found, ASP.NET loads the default resource file. In this example, the default resource file is `WebResource.resx`.

Creating Resource Files for ASP.NET Web Sites

In ASP.NET, you can create resource files that have different scope. You can create resource files that are global, which means that you can read the resource file from any page or code that is in the Web site. You can also create local resource files, which store resources for a single ASP.NET Web page (.aspx file).

Global Resource Files

You create a global resource file by putting it in the reserved folder `App_GlobalResources` at the root of the application. Any .resx file that is in the `App_GlobalResources` folder has global scope. Additionally, ASP.NET generates a strongly typed object that gives you a simple way to programmatically access global resources.

Local Resource Files

A local resources file is one that applies to only one ASP.NET page or user control (an ASP.NET file that has a file-name extension of .aspx, .ascx, or .master). You put local resource files in folders that have the reserved name `App_LocalResources`. Unlike the root `App_GlobalResources` folder, `App_LocalResources` folders can be in any folder in the application. You associate a set of resources files with a specific Web page by using the name of the resource file.

For example, if you have a page named `Default.aspx` in the `App_LocalResources` folder, you might create the following files:

- `Default.aspx.resx`. This is the default local resource file (the fallback resource file) if no language match is found.
- `Default.aspx.es.resx`. This is the resource file for Spanish, without culture information.
- `Default.aspx.es-mx.resx`. This is the resource file for Spanish (Mexico) specifically.
- `Default.aspx.fr.resx`. This is the resource file for French, without culture information.

The base name of the file matches the page file name, followed by a language and culture name, and ending with the extension .resx. For a list of culture names, see [CultureInfo](#).

Localizing Client Script Resources

Localization support for ASP.NET AJAX client script builds on the foundation of the ASP.NET 2.0 localization model. In this model, you embed script files and localized script resources in a hub-and-spoke organization of assemblies (satellite assemblies). You can then selectively use these embedded client scripts and resources for specific languages and regions. This model enables a single code base to support multiple cultures. There is also support for localized script files that are provided as .js files on disk. ASP.NET can serve localized client scripts and resources automatically for specific languages and regions.

For more information, see the following topics:

- [Localizing Resources for Component Libraries Overview](#)
- [Walkthrough: Embedding Localized Resources for a JavaScript File](#)
- [Walkthrough: Globalizing a Date by Using Client Script](#)

Choosing Between Global and Local Resource Files

You can use any combination of global and local resource files in the Web application. Generally, you add resources to a global resource file when you want to share the resources between pages. Resources in global resource files are also strongly typed for when you want to access the files programmatically.

However, global resource files can become large, if you store all localized resources in them. Global resource files can also be more difficult to manage, if more than one developer is working on different pages but in a single resource file.

Local resource files make it easier to manage resources for a single ASP.NET Web page. But you cannot share resources between pages. Additionally, you might create lots of local resource files, if you have many pages that must be localized into

many languages. If sites are large with many folders and languages, local resources can quickly expand the number of assemblies in the application domain.

When you make a change to a default resource file, either local or global, ASP.NET recompiles the resources and restarts the ASP.NET application. This can affect the overall performance of your site. If you add satellite resource files, it does not cause a recompilation of resources, but the ASP.NET application will restart.

Note

Linked resources are supported only in global resource files.

Working with Resources in Web Pages

After you create resource files, you can use them in ASP.NET Web pages. You typically use resources to fill the property values of controls on the page. For example, you might use resources to set the [Text](#) property of a [Button](#) control, instead of hard-coding the property to a specific string.

To use resources to set control property values, you can use implicit localization or explicit localization, as follows:

- Implicit localization works with local resources and lets you automatically set control properties to matching resources.
- Explicit localization lets you use a resource expression to set a control property to a specific resource in a local or global resource file.


Implicit Localization with Local Resources

If you have created local resource files for a specific page, you can use implicit localization to fill the property values for a control from the resource file. In implicit localization, ASP.NET reads a resource file and matches resources to property values.

To use implicit localization, you must use a naming convention for resources in the local resource file that uses the following pattern:

Key.Property

For example, if you are creating resources for a [Button](#) control named Button1, you might create the following key/value pairs in the local resource file:

	 Copy
<pre>Button1.Text Button1.BackColor Label1.Text</pre>	

You can use any name for Key, but Property must match the name of a property of the control that you are localizing.

In the page, you use a special meta attribute in the markup for the control to specify implicit localization. You do not have to explicitly specify which properties are localized. A [Button](#) control that is configured for implicit localization might resemble the following:


	 Copy
<pre><asp:Button ID="Button1" runat="server" Text="DefaultText" meta:resourcekey="Button1" /></pre>	

The resourcekey value matches a key in the corresponding resource file. At run time, ASP.NET matches resources to control properties using the control label as the resourcekey. If a property value is defined in the resource file, ASP.NET substitutes the resource value for the property.

Explicit Localization

Alternatively, you can use explicit localization, where you use a resource expression. Unlike implicit localization, you must use a resource expression for each property that you want to set.

A [Button](#) control that is configured to set the [Text](#) property from a global resource file might resemble the following:

	 Copy
<pre><asp:Button ID="Button1" runat="server" Text="<%= \$Resources:WebResources, Button1Caption %>" /></pre>	


The resource expression takes the following form, where Class is optional, unless the resource is a global one, and ResourceID is required:

```
<%= $Resources:Class,ResourceID %>
```

The Class value identifies the resource file to use when you use global resources. When .resx files are compiled, the base file name, without extensions, is used as the class name of the resulting assembly, explicitly. If you want to use resources from a local resource file (one that matches the current page name), you do not have to include a class name. This is because ASP.NET matches the page class to the resource class.

The ResourceID value is the identifier of the resource to read. In the previous example, the [Text](#) property for the button is read from the global resource file WebResources.resx (or the appropriate localized version). In that file, ASP.NET uses the value for the resource with the identifier Button1Caption and for the page itself. To set page properties, you can use resource expressions in the [@ Page](#) directive.

You can specify an explicit resource expression or an implicit resource expression for a control, but not both. The following declarative syntax on a Button control causes a parser error:

	 Copy
<pre><asp:Button ID="Button1" runat="server"</pre>	

```
meta:resourcekey="Button1Resource1"  
Text="<%%$ Resources:WebResources, Button1Caption %>" />
```

In this example, an implicit local resource file (one that matches the current page name) is specified as well as an explicit resource file that is named `WebResources`. To prevent a parser error for this control, remove one of the resource expressions.

Localizing Static Text

If a page includes static text, you can use ASP.NET localization by including it in a `Localize` control, and then using explicit localization to set the static text. The `Localize` control renders no markup; its only function is to act as a placeholder for localized text. The `Localize` control can be edited in **Design** view, not only in the property grid. At run time, ASP.NET treats the `Localize` control as a [Literal](#) control. For example, your page might include the following code:



```
<h1>  
  <asp:Localize runat=server  
    ID="WelcomeMessage"  
    Text="Welcome!" meta:resourcekey="LiteralResource1" />  
</h1>  
<br />  
<br />  
<asp:Localize runat="server"  
  ID="NameCaption"  
  Text="Name: " meta:resourcekey="LiteralResource2" />  
<asp:TextBox runat="server" ID="TextBox1"  
  meta:resourcekey="TextBox1Resource1" />
```

Security Note:

This example has a text box that accepts user input, which is a potential security threat. By default, ASP.NET Web pages validate that user input does not include script or HTML elements. For more information, see [Script Exploits Overview](#).

Implicit Localization in Templates

In templated controls, such as the [DataList](#), [GridView](#), and [Wizard](#) controls, you localize template style properties by accessing the properties through the parent control's implicit resource expression. You cannot use an implicit resource expression on the template itself.

To localize values for a template property, use the meta attribute and a resource key for the control that the template belongs to. Then use either the Property.Subproperty syntax or the Property-Subproperty syntax in the resource file. For example, the following declarative syntax is valid for a [Wizard](#) control:



```
<asp:Wizard ID="Wizard1"
  runat="server"
  meta:resourcekey="Wizard1Resource1">
  <NavigationStyle
    BorderWidth="<%= $resources.navBorderWidth %>" />
  <WizardSteps>
    <asp:WizardStep ID="WizardStep1"
      runat="server"
      Title="Step 1"
      meta:resourcekey="WizardStep1Resource1">
    </asp:WizardStep>
  </WizardSteps>
</asp:Wizard>
```

The following key/value pairs in the local resource file could be used for the previous example:



```
Wizard1Resource1.NavigationStyle.BackColor, Red
navborderWidth, 5
```

Or the following key/value pairs could be used:



```
Wizard1Resource1.NavigationStyle-BackColor, Red  
navborderWidth, 5
```

You can use an explicit resource expression for the [NavigationStyle](#) property of the [Wizard](#) control in the previous example. The explicit resource expression omits the Class name so that resources from a local resource file are used.

For more information about templated server controls, see [ASP.NET Web Server Controls Templates](#).

Selecting Resource Files for Different Languages

When a page runs, ASP.NET selects the version of the resource file that most closely matches the current [UICulture](#) setting for the page. If there is no match, ASP.NET uses resource fallback to obtain a resource. For example, if you are running the Default.aspx page and the current [UICulture](#) property is set to es (Spanish), ASP.NET uses the compiled version of the local resource file Default.aspx.es.resx.

ASP.NET can set the [UICulture](#) and [Culture](#) properties for the page to the language and culture values that are passed by the browser. Alternatively, you can set the [UICulture](#) and [Culture](#) properties explicitly, either declaratively or in code. You can also set the values declaratively in Web.config files. For detailed information, see [How to: Set the Culture and UI Culture for ASP.NET Web Page Globalization](#).

ⓘ Note

You should not rely exclusively on browser settings to set language and culture information, because users can be using a browser on a computer other than their own. Also, browsers frequently communicate only language information without a specific culture setting. In that case, the server has to deduce a specific culture for data formatting. A good strategy is to let users select a language explicitly.

Working with Resources Programmatically

Besides setting resource values in markup with resource expressions, you can retrieve resource values programmatically. You might do this if the resource value is not known at design time or if you want to set the resource value to a value that is obtained at run time. For more information, see [How to: Retrieve Resource Values Programmatically](#).

See Also

Other Resources

[ASP.NET Globalization and Localization](#)