9

Application Globalization

In Depth

As Internet is making its global presence in almost all walks of life, it becomes quite imperative to move away from the traditional approach of application development to a more advanced one. The more advanced approach specifies that the content of a Web page is not restricted to a few international languages, so that a Web page is understandable to masses from all over the world. In .NET, the concept of application globalization in application development facilitates displaying the content of an application according to different cultures from all over the world. Displaying the date and time according to the local culture of a region is an example of application globalization. It provides a culture-specific Web experience to the Internet users.

ASP.NET 4.0 provides built-in mechanisms to globalize Web applications that can adapt to different languages. However, it is the built-in mechanism of ASP.NET that reconfigures the output of a Web application, so that the content, such as date and time objects, is presented in the appropriate culture-specific format. Basically, application globalization involves two sub-processes, internationalization and localization. Internationalization is the process of internationalizing application code as per some specific standards, and localization is the process of rendering the output of an application according to a specific language and culture.

In this chapter, you learn about the process of application globalization, which includes internationalization and localization. You then explore how to generate local and global resources to create globalization-enabled Web applications. You also learn to perform implicit and explicit localization and localize the script files. The chapter also describes how to perform localization of Hyper Text Markup Language (HTML) elements and static content.

Introducing Globalization

Globalization of Web application is the process of designing and developing Web applications that can display Web page contents in different languages and cultures. The process of globalizing a Web application involves separating the resources of the Web application that are culture-specific and need to be modified for different cultures. The next step is to customize the Web application for specific cultures.

The following two namespaces are used to globalize Web applications in .NET Framework:

- `System.Globalization`—Contains the classes defining information related to culture, such as language, country, format, and pattern for date, currency, and numbers
- `System.Resources`—Contains the classes and interfaces that you can use to create, store, and manage culture-specific resources used in a Web application

Table 9.1 lists the classes contained in the `System.Globalization` namespace:

Table 9.1: Classes in the System.Globalization Namespace

Class	Description
`Calendar`	Represents time in different parts, such as weeks, months, and years.
`CharUnicodeInfo`	Gets information related to the Unicode character.
`ChineseLunisolarCalendar`	Represents time in different parts, such as months, days, and years. Days and months are calculated with a lunisolar calendar; whereas, years are calculated using a Chinese calendar.
`CompareInfo`	Provides methods to perform culture-specific string comparisons.
`CultureAndRegionInfoBuilder`	Specifies a customized culture that is new or based on another culture and country/region.
`CultureInfo`	Represents culture-specific information, such as name, region, and language.
`DateTimeFormatInfo`	Represents the formats for the culture-specific date and time.
`DaylightTime`	Represents the daylight-saving time period.

Table 9.1: Classes in the System.Globalization Namespace	
Class	**Description**
`EastAsianLunisolarCalendar`	Represents division of time into months, days, years, and eras. Their dates are based on the cycles of the sun and the moon.
`GregorianCalendar`	Represents the Gregorian calendar.
`HebrewCalendar`	Represents the Hebrew calendar.
`HijriCalendar`	Represents the Hijri calendar.
`JapaneseCalendar`	Represents the Japanese calendar.
`JapaneseLunisolarCalendar`	Represents time in different parts, such as months, days, and years. Days and months are calculated with a lunisolar calendar; whereas, years are calculated using a Japanese calendar.
`JulianCalendar`	Represents the Julian calendar.
`KoreanCalendar`	Represents the Korean calendar.
`KoreanLunisolarCalendar`	Represents time in different parts, such as months, days, and years. Days and months are calculated with a lunisolar calendar; whereas, years are calculated using a Gregorian calendar.
`NumberFormatInfo`	Defines the manner in which culture-specific numeric information is formatted and displayed.
`PersianCalendar`	Represents the `Persian` calendar.
`RegionInfo`	Represents information about a country or region.
`SortKey`	Represents the mapping results obtained after mapping a string with its sort key.
`StringInfo`	Provides different string manipulation methods using which you can split a string into separate text elements and then iterate through those elements.
`TaiwanCalendar`	Represents the Taiwan calendar.
`TaiwanLunisolarCalendar`	Represents a Taiwan lunisolar calendar in which days and months are calculated with a lunisolar calendar; whereas, years are calculated using a Gregorian calendar.
`TextElementEnumerator`	Enumerates the text elements of a string.
`TextInfo`	Defines the properties and behavior of a writing system.
`ThaiBuddhistCalendar`	Represents the Thai Buddhist calendar.
`UmAlQuraCalendar`	Represents the Saudi Hijri (Um Al Qura) calendar.

Table 9.2 lists the classes contained in the `System.Resources` namespace:

Table 9.2: Classes in the System.Resources Namespace	
Class	**Description**
`MissingManifestResourceException`	Represents the exception that is thrown when the main assembly does not contain neutral culture resources
`MissingSatelliteAssemblyException`	Represents the exception that is thrown when the satellite assembly for the neutral culture resources is not present
`NeutralResourcesLanguageAttribute`	Communicates to the `ResourceManager` class about the language used to write the neutral culture resources

Table 9.2: Classes in the System.Resources Namespace	
Class	**Description**
`ResourceManager`	Provides dynamic access to the resources of a specific culture
`ResourceReader`	Allows you to read the resource names and the resource value pairs from the `.resources` files
`ResourceSet`	Stores the set of all localized resources corresponding to a specific culture, while neglecting resources of other cultures
`ResourceWriter`	Writes resources to an output stream or file in system's default format
`ResXDataNode`	Represents an element or data types in a resource file
`ResXFileRef`	Includes references of an external resource (`.resx`) file
`ResXFileRef.Converter`	Converts a `ResXFileRef` reference link to a string and vice versa
`ResXResourceReader`	Allows you to read the resource names and the resource value pairs from the `.resx` files
`ResXResourceSet`	Collects all items to create a single object of the `.resx` file
`ResXResourceWriter`	Writes resources in the `.resx` file or an output stream
`SatelliteContractVersionAttribute`	Ensures that the `ResourceManager` class retrieves the required version of a satellite assembly to simplify the updating process of the main assembly

Globalization of a Web application uses resources for user interface (UI) and static content of a Web application. As a result, different functional experts are required to build a globalized Web application. The different functional roles that are required to build a Web application are as follows:

- **Web designer**—Determines the Graphical User Interface (GUI) of a Web application. The Web designers decide how the content and controls are grouped on each Web page.
- **Developers**—Develop ASP.NET pages for the content and controls designed by Web designers. The developers also perform tasks, such as determining the structure for content storage, designing databases, and handling access to the content at runtime.
- **Translators**—Translate the content of the Web application to other languages. Translators are also required to work with Resource editors and XML editors to modify database content.
- **Stakeholders**—Coordinates between Web designers, developers, and translators. In addition, the role requires storing the content of a Web application such that the translators can access and modify the content. The stakeholders also ensure that the Web application supports new cultures without modifying the GUI of the Web application or database structure and other components of a Web application.

Globalization of Web application is related to the following two processes:

- Internationalization
- Localization

Now, let's discuss these in detail.

Internationalization

The process of Internationalization involves identifying different contents for different locales, such as date and time formats. To serve the purpose, you need to write the source code for the Web application such that the code is dynamically formatted to present the content of the Web application in the desired locale. The process of Internationalization ensures that the source code of a Web application is not modified when the application needs to be customized for specific culture or regions.

The goals of the process of Internationalization are as follows:

- Presenting the content in a single UI.
- Placing the content at a location so that it can be easily translated to another language. The location should also be programmatically accessible so that the content can be added to the UI of the Web application.
- Storing the content and user input as per the required culture.

Localization

Localization refers to the process of making changes in a Web application to meet the language and cultural requirements of users from a specific geographical location. The localization process involves configuring a Web application for a given culture. This implies translating and formatting content, such as time and date, according to the culture for which the Web application is to be localized. The elements, such as numeric data, date formats, and currency, are customized for a given culture in Web applications.

Prior to ASP.NET 2.0, it was difficult to localize a Web application. As previously, the resource files and the `ResourceManager` class were used, which required considerable amount of effort and code. However, ASP.NET 2.0 simplifies the localization process by introducing following new features:

- Auto-detection of preferred culture of the client browser
- Accessing resources programmatically
- Compilation of files with the `.resx` or `.resource` extension
- Supporting the creation of resources at design-time
- Declarative resource expressions (implicit and explicit) to bind the controls or the properties of the controls to the resources

While building a multilingual Web application, it is beneficial to maintain the source code for the entire application in a single page, as it is easier to maintain a single page in comparison to maintaining separate sites for different languages. ASP.NET enables localization of a Web application into different languages without any changes to the source code through resource expressions. Resource expressions are the subfeatures of the overall expressions, such as the data-binding expressions feature of earlier versions of ASP.NET.

An amazing feature of ASP.NET 4.0 is that it supports the usage of different resource files in a Web application. These resource files follow a unique naming convention and have the `.resx` file extension. All the resource files related to a Web application are stored in a separate folder, which enables us to add more resource files to a Web application without performing the compilation process. Further, you learn about the samples executed for en-US (English-US), fr (French), and de (German) language. You can also use other languages in a Web application after creating resource files for that language and setting the language as a preferred language in your browser.

Now, you will learn how to deploy all the resources used in Globalization.

Introducing Resources

Resources, such as images and strings, are used in various elements, such as the toolbar and menu icons of a Web application to display different sets of information to the user. If you want to make changes in the strings or image resources used in a Web application, you need to search the entire source code of the Web application for the resource to be modified. To simplify the process of making changes to the resources used in a Web application, you can place the source code for strings and images used in the Web application in a separate file. You can then modify the code in the file to change the used resources. Such a file that contains source code for the various resources used in a Web application is called the resource file.

In the earlier versions of ASP.NET, the resources were deployed using the hub and spoke model. The hub refers to the main assembly that contains the non-localizable executable code and the resources for a default culture. Each spoke connects to a satellite assembly that contains the resources for a single culture, but does not include any code.

However, ASP.NET 2.0 supports site pre-compilation. This implies that the Web pages and the local resources are compiled into deployable assemblies. ASP.NET also supports dynamic runtime compilation. In dynamic runtime compilation, the source code, local and global resources are deployed in source format. The runtime compiler then parses the Web pages and generates assemblies for deployment.

A hybrid of site pre-compilation and dynamic runtime compilation is also possible in ASP.NET 4.0. This implies that ASP.NET 4.0 provides with the ability to deploy Web pages and resources as source and compile all these source files into binary assemblies. Compiling files into binary assemblies enables you to edit the Web page layout and resource content. You do not need to dynamically recompile the assemblies of unaffected applications.

Now let's consider a practical scenario on different concepts of Globalization in the *Immediate Solutions* section.

Immediate Solutions

Listing the Available Cultures

.NET Framework provides the `CultureInfo` class residing in the `System.Globalization` namespace. The `CultureInfo` class contains culture-specific information, such as the language, country, and cultural conventions that are associated with a specific culture. The `CultureInfo` class also provides information required for performing culture-specific operations, such as formatting dates and numbers, casing, and comparing strings. In addition, you also need to use the `System.Resources` and `System.Threading` namespaces to set culture-specific information of a Web application.

You can use the `CultureInfo.GetCultures` method in the `CultureInfo` class to view the complete list of all the culture options according to which you can customize a Web application.

To use the `CultureInfo.GetCultures` method, follow these steps:

1. Create a console application, `CultureCS`, in Visual Studio 2010. This application is also available on CD-ROM.
2. Modify the code, given in Listing 9.1, of the Program.cs file of the CultureCS application:

Listing 9.1: Showing the Code of the Program.cs File of the CultureCS Application

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Globalization;
namespace CultureCS{
    class Program
    {
        static void Main(string[] args)
        {
            foreach (CultureInfo cultinfo in
              CultureInfo.GetCultures(CultureTypes.AllCultures))
            {
                Console.WriteLine(cultinfo);
            }
                Console.ReadLine();
        }

    }
}
```

3. Press the F5 key to execute the CultureCS application. The output of the application is shown in Figure 9.1:

Figure 9.1: Showing the Output of the CultureCS Application

A Web page in a Web application consists of two culture values, `Culture` and `UICulture`. The `Culture` value determines the functions, such as date and currency. These `Culture` values are used to format data and numbers in a Web page. The `UICulture` value determines the resources that are loaded on a Web page of a Web application. Resources in a Web application represent data, such as strings or images, displayed in the UI of the Web application.

You can set the `Culture` and `UICulture` values in a Web application by using any one of the following methods:

- Through the `web.config` file
- In the code-inline page
- In the code-behind (CB) page

NOTE

To translate the content of a Web application from one language to another, you need to place the string value in a resource file.

The `Culture` value for a Web application is specified by the preceding methods so that the UI of the Web application is appropriate for the culture for which it is created. You can set the `Culture` and `UICulture` values in a Web application to override the user settings or operating system settings.

To set the `Culture` and `UICulture` values through the `web.config` file, write the following code snippet in the Globalization section of the `web.config` file:

```
culture="en-US"
uiCulture="de-DE"
```

In the preceding code snippet, the combination of the language and culture values for the Web application is set to US English (en-US). The `UICulture` value is set to German language (de).

To set the `Culture` and `UICulture` values in the source code of the Web page, you need to include the following values to the Page directive:

```
<%@ Page UICulture="de-DE" Culture="en-US"...%>
```

In the code-behind file, you can set the default culture for a Web application. ASP.NET provides the `CurrentCulture` and `CurrentUICulture` properties to set the default values for the `Culture` and `UICulture` attributes in the code-behind file. For this, you need to include the following code in the code-behind file to set the default culture to German.

The code to be added to the code-behind page is given as follows:

```
using System.Globalization;
using System.Threading;
Thread.CurrentThread.CurrentCulture = CultureInfo.CreateSpecificCulture ("de-DE");
Thread.CurrentThread.CurrentUICulture=new CultureInfo("de-DE");
```

ASP.NET also provides an auto-culture handling feature that enables you to easily localize a Web application. You need to include the `Culture="auto"` and `UICulture ="auto"` attributes in the Page directive of each Web page of the Web application to enable the auto-culture handling feature.

Working with Resources

To create resource files, first you need to identify the sections specific to a culture in a Web application. You then need to make different resource files for the cultures for which you need to develop a Web application. Based on the culture settings made by a user in the browser, the resource files are loaded in the Web application. For example, consider that you need to display the content of a Web application in French language. For this, you need to create resource file for the French language. You can create the resource file in either the `App_LocalResources` folder or the `App_GlobalResources` folder of a Web application. The language settings of the browser that are used to view the output of a Web application are then changed to load the resource file in the Web application.

In .NET Framework, you can create global resources as well as local resources. The global resources are accessible from all the Web pages of a Web application, while the local resources are accessible from a single Web page of the Web application. Now, let's discuss these resources in detail.

Local Resources

The local resources for a specific Web page in the Web application are created in the `App_LocalResources` folder. Unlike the `App_GlobalResources` folder that resides in the root directory, the `App_LocalResources` folder can exist in any directory in the Web application. You can make a local resource file accessible to a Web page by using the name of the resource file. The naming convention of the resource file should match the base name of the Web page that accesses the resource file. The base name should be followed by the language name and the culture name for which the resource file is created. The name of the resource file should end with the extension `.resx`. For example, to associate a resource file for German language to the `Default.aspx` page, the naming convention for the resource file is given as follows:

```
Default.aspx.de.resx
```

In the preceding name for the resource file, the base name, `Default.aspx`, matches the name of the Web page. The `de` is the International Organization for Standardization (ISO) language code of the German language. The name of the resource file ends with the `.resx` extension.

Perform the following steps to add local resources to a website:

1. Select the `File`→`New`→`Web Site` option from the menu bar to create a new website. The New Web Site dialog box opens, as shown in Figure 9.2:

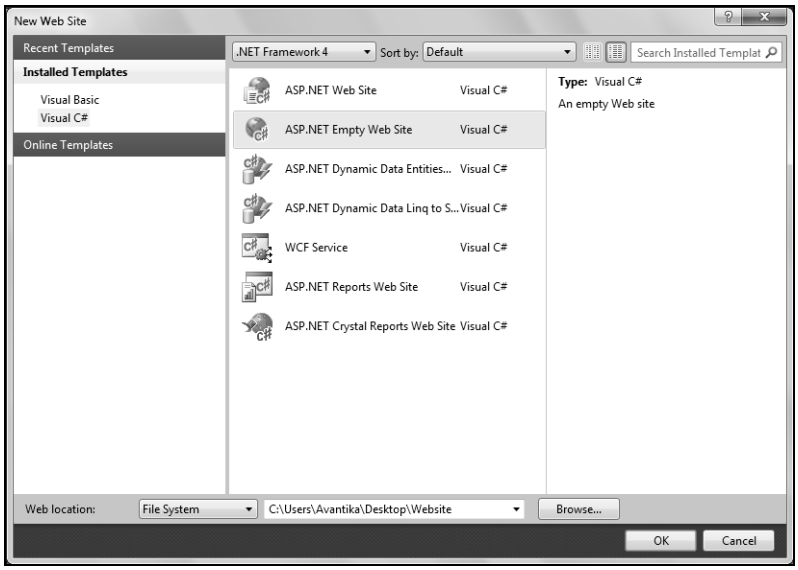

Figure 9.2: Displaying the New Web Site Dialog Box

2. Select the ASP.NET Empty Web Site template, enter the location and name of the website beside the Location field, and click the OK button.

This creates the website and adds the default files in Solution Explorer, as shown in Figure 9.3:

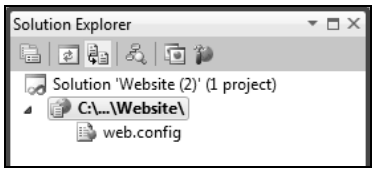

Figure 9.3: Showing the Default Files in Solution Explorer

3. Right-click the name of the website (`Website`) and select the Add ASP.NET `Folder`→`App_LocalResources` option from the context menu to add the `App_LocalResources` folder to the website.

Global Resources

Unlike local resources that are generated automatically, you need to add the global resources to a Web application. To add global resources, you can create a resource file with the `.resx` extension in the `App_GlobalResources` folder. The `App_GlobalResources` folder resides in the root directory of an ASP.NET Web application and contains the global resources for the Web application. In addition, the `App_GlobalResources` folder also provides a simple way to programmatically access global resources in a Web application.

The process of adding the global resources in a website is similar to the process of adding the local resources. The only difference is that you need to select the Add ASP.NET `Folder`→`App_GlobalResources` option from the context menu to add the `App_GlobalResources` folder.

Using Localization Expression

The localization expressions are added for controls, such as labels and buttons, on the Web page of a Web application. The localization expressions enable you to access the local and global resources of the control programmatically to generate the localized content. When the localization expressions are added to a Web application, it results in modification of the control declaration on the Web page. The declarative statement `meta:resourcekey`, which is added to the control declaration, indicates that the ASP.NET parser should generate code to retrieve property values from either the local or the global resources that are created. The localization expressions generated for resources are either implicit localization expressions or explicit localization expressions.

Implicit Expression

Implicit localization expressions are automatically generated when resources are generated using implicit localization. While using implicit localization, the properties of the controls created on a Web page are read from a resource file. You create a resource file that contains localized values for properties, such as text, of the controls. At runtime, .NET Framework examines the controls created on the Web page to verify if the controls are using implicit localization. The values of the controls that are using implicit localization are then substituted in the resource files.

To perform implicit localization, you need to create controls on a Web page and then generate a resource file for customizing the application for different languages. Let's create a new ASP.NET Empty WebSite, named `ImplicitLocalizationCS (also available on the CD)` to create controls on a Web page, and follow these steps:

1. Double-click the `Default.aspx` page in Solution Explorer to open the page.
2. Switch to the Design view of the `Default.aspx` page and drop three Label controls (label1, label2 and label3) and a Calendar control (calendar1) from the Toolbox onto the Design view of the file to create a UI of the Web application. You can also write the code, as shown in Listing 9.2, in the `Default.aspx` page to add controls:

Listing 9.2: Code for the `Default.aspx` Page

```
<%@ Page Title="Home Page" Language="C#" MasterPageFile="~/Site.master"
   AutoEventWireup="true"
     CodeBehind="Default.aspx.cs" Inherits="ImplicitLocalizationCS._Default"
   Culture="auto" UICulture="auto"%>
<asp:Content ID="HeaderContent" runat="server" ContentPlaceHolderID="HeadContent">
</asp:Content>
<asp:Content ID="BodyContent" runat="server" ContentPlaceHolderID="MainContent">
  <title>Implicit Localization</title>
    <h2>
        C# 2010 black book</h2>
    <p>
         
        <div>
        <asp:Calendar ID="Calendar1" runat="server" BackColor="White"
```

```
                    BorderColor="White"
                  BorderWidth="1px" Font-Names="Verdana" Font-Size="9pt" ForeColor="Black"
                    Height="190px"
                  NextPrevFormat="FullMonth" Width="254px">
                  <SelectedDayStyle BackColor="#333399" ForeColor="White" />
                  <TodayDayStyle BackColor="#CCCCCC" />
                  <OtherMonthDayStyle ForeColor="#999999" />
                  <NextPrevStyle Font-Bold="True" Font-Size="8pt" ForeColor="#333333"
                    VerticalAlign="Bottom" />
                  <DayHeaderStyle Font-Bold="True" Font-Size="8pt" />
                  <TitleStyle BackColor="White" BorderColor="Black" BorderWidth="4px"
                  Font-Bold="True"
                  Font-Size="12pt" ForeColor="#333399" />
                  </asp:Calendar>
            </div>
            <div>
            <br />
                  <br />
                                     
                        
                  <table style="width: 383px; height: 193px; background-color: #99ffff">
                  <tr>
                         <td colspan="2" style="width: 100px; background-color: white">
                                          
                              
                            <span style="text-decoration:
                           underline"><strong>I<span>mplicitLocalization</span>
                         </strong></span>
                         </td>
                  </tr>
                  <tr>
                         <td style="width: 75px; height: 18px; background-color: white">
                                   <strong>English</strong>
                         </td>
                         <td style="width: 100px; height: 18px; background-color: white">
                                      
                         <asp:Label ID="Label1" runat="server" Font-Bold="True"
                           meta:resourcekey="Label1Resource1"
                         Text="English"></asp:Label>
                         </td>
                  </tr>
                  <tr>
                         <td style="width: 75px; background-color: white">
                         <span style="background-color:
                           White"><strong>French</strong>  </span>
                         </td>
                         <td style="width: 100px; background-color: White">
                         <strong>             
                                 </strong><asp:Label
                         ID="Label2" runat="server" Font-Bold="True"
                           meta:resourcekey="Label2Resource1"
                         Text="French"></asp:Label>
                         </td>
                  </tr>
                  <tr>
                         <td style="width: 75px; background-color: white">
                                  <strong>German</strong>
                         </td>
                         <td style="width: 100px; background-color: white">
                                      
                         <asp:Label ID="Label3" runat="server" Font-Bold="True"
                           meta:resourcekey="Label3Resource1"
                         Text="German"></asp:Label>
                         </td>
                  </tr>
                  </table>
            </div>
      </p>
</asp:Content>
```

3. Generate a local resource file for the `Default.aspx` page. To do so, click the Design view of the `Default.aspx` page. Then, select the `Tools`→`Generate Local Resource` option from the menu bar.

A new folder `App_LocalResources` is created in Solution Explorer. Within the `App_LocalResources` folder, a file named `Default.aspx.resx` is created. In addition, a new `meta:resourcekey` attribute is added to all the control declarations and in the Page directive of the source code of the `Default.aspx` page. After creating the resource file, you need to use the Resource editor to place the localized text within the resource file.

4. Double-click the `Default.aspx.resx` resource file to open the Resource editor for the file. The Resource editor consists of the Name, Value, and Comment columns.
5. Change the property in the Value column as English, French, and German for the `Label1Resource1.Text`, `Label2Resource1.Text`, and `Label3Resource1.Text` fields, respectively, as shown in Figure 9.4:

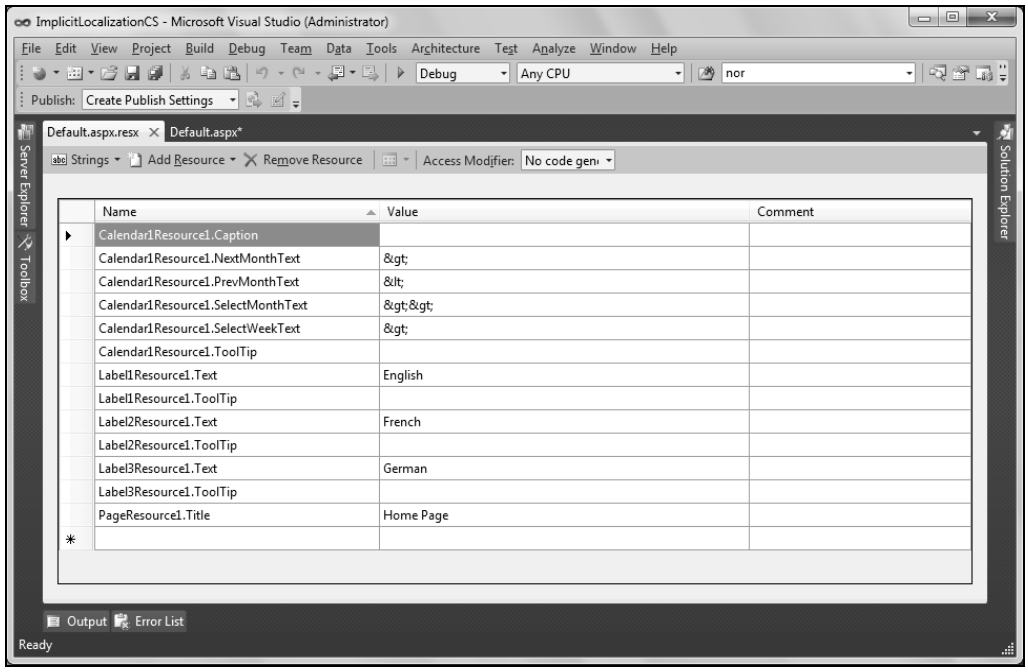

Figure 9.4: Displaying the Resource Editor of the Default.aspx.resx File

6. Press the `F5` key to run the Web application. The names of the labels in the Web application are displayed as edited in the `Default.aspx.resx` file.

   You need to create a unique resource file for each combination of culture and language that you need to include in a Web application.
7. Right-click the `Default.aspx.resx` resource file in Solution Explorer and select the Copy option from the context menu to copy the resource file, to create a French language resource file for the Web application.
8. Right-click the `App_LocalResources` folder and select the Paste option from the context menu to paste the file in the `App_LocalResources` folder. A resource file named `Copy of Default.aspx.resx` file is created in the `App_LocalResources` folder.
9. Right-click the `copy of Default.aspx.resx` resource file and select the Rename option to rename the resource file as `Default.aspx.fr.resx`.
10. Double-click the `Default.aspx.fr.resx` resource file to open it, and then set the text property in the Value column as Anglais, Français, and Allemand for the `Label1Resource1.Text`, `Label2Resource1.Text` and `Label3Resource1.Text` controls respectively, as shown in Figure 9.5:

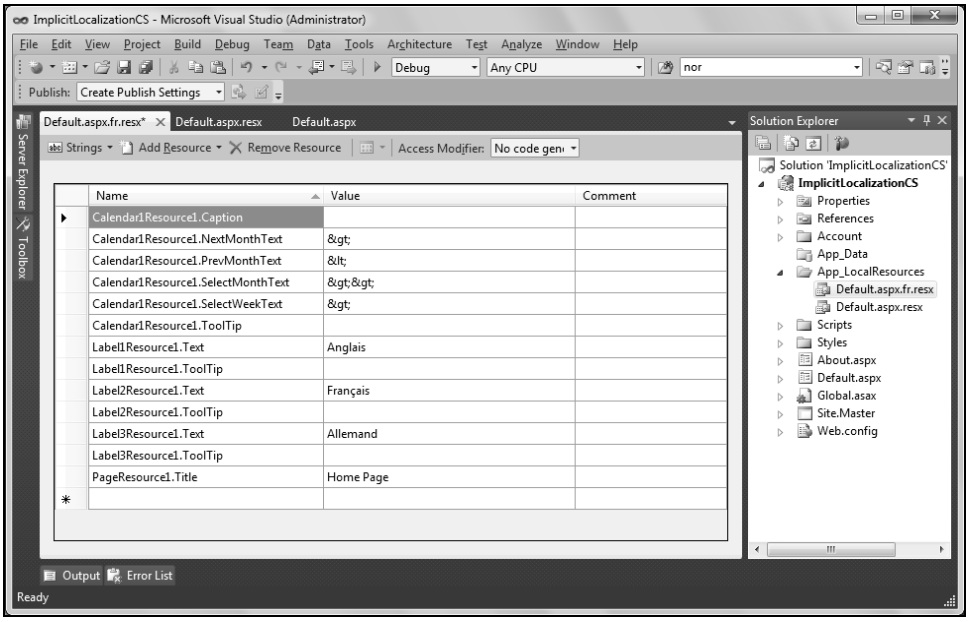

Figure 9.5: Displaying the Resource Editor for the Default.aspx.fr.resx File

11. Create and add a German language resource file (`Default.aspx.de.resx`) to the Web application. Figure 9.6 shows the Resource editor for German language:

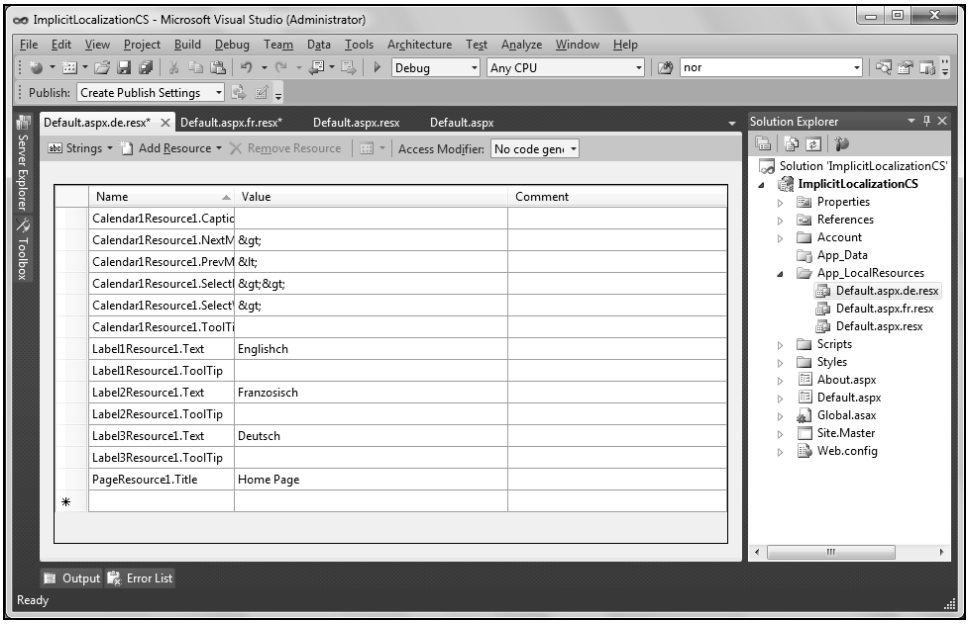

Figure 9.6: Displaying the Resource Editor for the Default.aspx.de.resx File

12. Modify the browser settings so that ASP.NET uses the resource file and includes the language and culture specified in the browser settings in the Web application. To change the language settings of the Microsoft Internet Explorer (IE) browser, select `Tools`→`Internet Options` on the menu bar of IE to open the Internet Options dialog box, as shown in Figure 9.7:

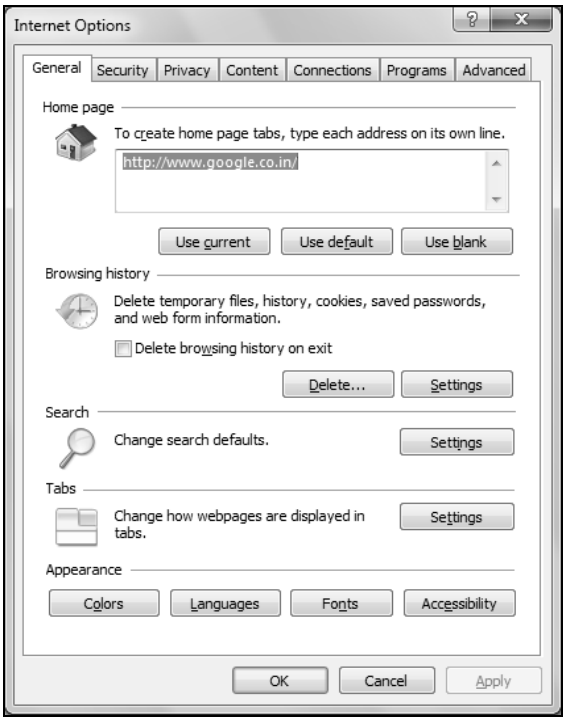

Figure 9.7: Showing the Internet Options Dialog Box

13. Click the Languages button on the General tab page to open the Language Preference dialog box, as shown in Figure 9.8:

Figure 9.8: Showing the Language Preference Dialog Box

14. Click the Add button on the dialog box to open the Add Language dialog box. Select the `French (France)[fr-FR]` option in the Language list box, as shown in Figure 9.9:

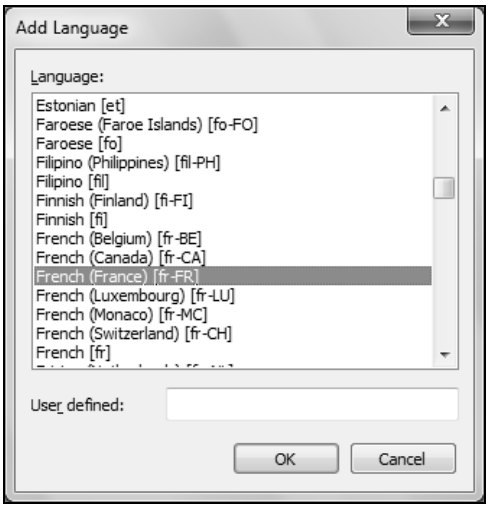

Figure 9.9: Showing the Add Language Dialog Box

15. Click the OK button to close the Add Language dialog box. Similarly, you can select `German (Germany)[de-DE]` language from the Add Language dialog box to add the German language. Select the `French [France][fr-FR]` option from the Language list box of the Language Preference dialog box.
16. Click the Move Up button to move the selected language up in the order. The `French [France][fr-FR]` option is moved up in the Language box, as shown in Figure 9.10:

Figure 9.10: Showing the Language Preference Dialog Box

17. Click the OK button to close the Language Preference dialog box, and then close and apply the changes made in the Internet Options dialog box.
18. Run the Web application by pressing the `F5` key to display the culture settings in the application by using implicit localization.

The output of the preceding application, when the Language settings in the browser indicate the preferred language as French, is shown in Figure 9.11:

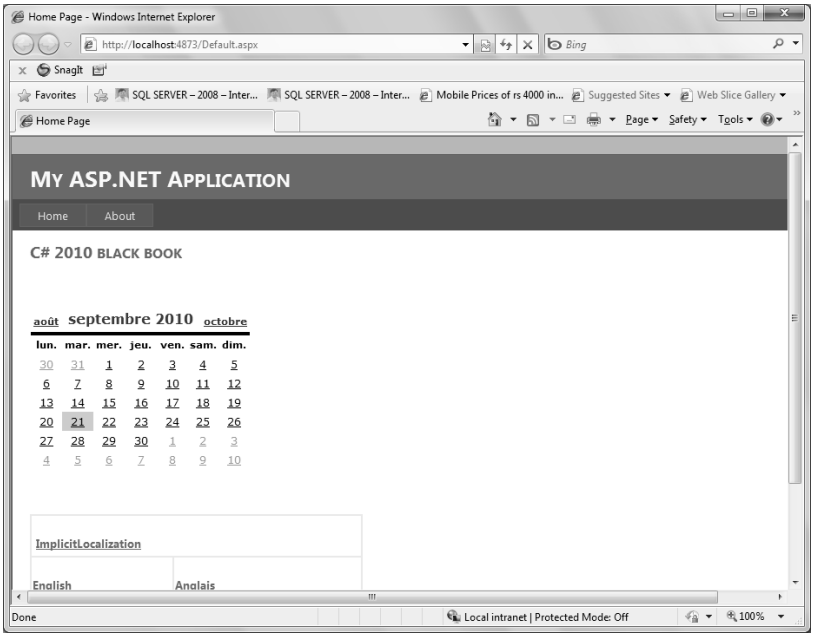

Figure 9.11: Displaying the Output of the Web Application with French as the Preferred Language

The output of the preceding application, when the Language settings in the browser indicate the preferred language as German, is shown in Figure 9.12:

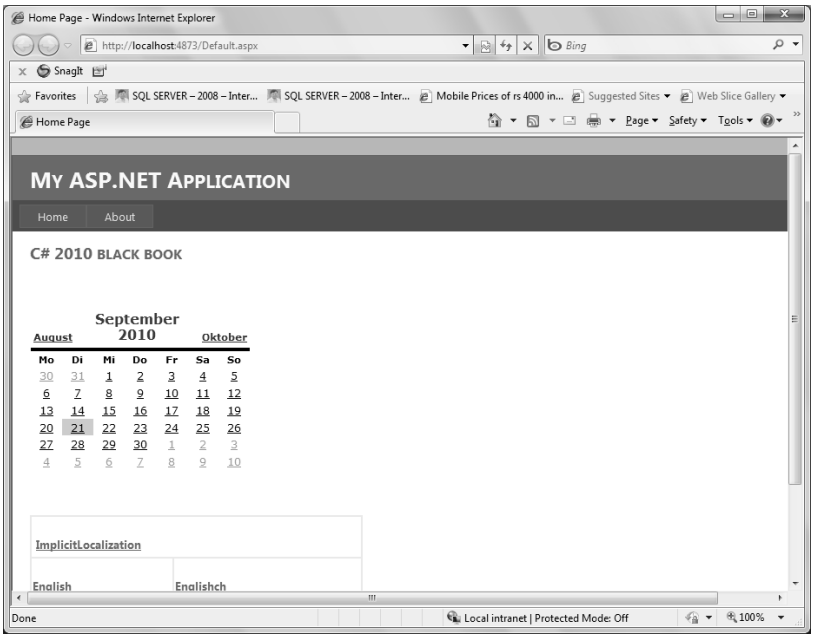

Figure 9.12: Displaying the Output of the Web Application with German as the Preferred Language

The output of the preceding application, when the Language settings in the browser indicate the preferred language as English, is shown in Figure 9.13:

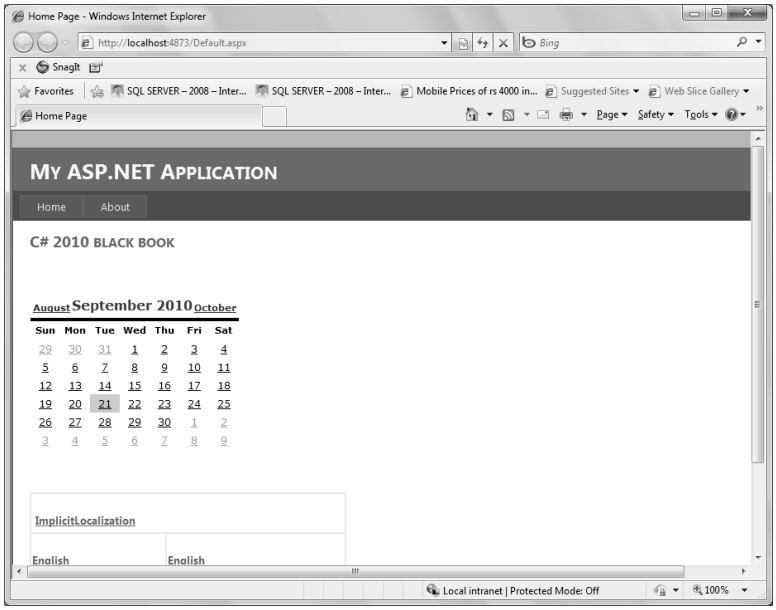

Figure 9.13: Displaying the Output of the Web Application with English as the Preferred Language

You can view the code of the resource files in XML(Text) Editor , by right-clicking the name of the file in the Solution Explorer, selecting the `Open With` option. The complete code of the resource file (`Default.aspx.resx`) is shown in Listing 9.3:

Listing 9.3: Complete Code for the `Default.aspx.resx` File

```
<?xml version="1.0" encoding="utf-8"?>
<root>
  <!--
    Microsoft ResX Schema
   Version 2.0
   The primary goals of this format is to allow a simple XML format
    that is mostly human readable. The generation and parsing of the
    various data types are done through the TypeConverter classes
    associated with the data types.
    Example:
    ... ado.net/XML headers & schema ...
    <resheader name="resmimetype">text/microsoft-resx</resheader>
    <resheader name="version">2.0</resheader>
    <resheader name="reader">System.Resources.ResXResourceReader, System.Windows.Forms,
  ...</resheader>
    <resheader name="writer">System.Resources.ResXResourceWriter, System.Windows.Forms,
  ...</resheader>
    <data name="Name1"><value>this is my long string</value><comment>this is a
  comment</comment></data>
    <data name="Color1" type="System.Drawing.Color, System.Drawing">Blue</data>
    <data name="Bitmap1" mimetype="application/x-microsoft.net.object.binary.base64">
        <value>[base64 mime encoded serialized .NET Framework object]</value>
    </data>
    <data name="Icon1" type="System.Drawing.Icon, System.Drawing"
  mimetype="application/x-microsoft.net.object.bytearray.base64">
        <value>[base64 mime encoded string representing a byte array form of the .NET
         Framework object]</value>
        <comment>This is a comment</comment>
    </data>
    There are any number of "resheader" rows that contain simple
    name/value pairs.
    Each data row contains a name, and value. The row also contains a
    type or mimetype. Type corresponds to a .NET class that support
    text/value conversion through the TypeConverter architecture.
    Classes that don't support this are serialized and stored with the
    mimetype set.
```

```
    The mimetype is used for serialized objects, and tells the
    ResXResourceReader how to depersist the object. This is currently not
    extensible. For a given mimetype the value must be set accordingly:
    Note - application/x-microsoft.net.object.binary.base64 is the format
    that the ResXResourceWriter will generate, however the reader can
    read any of the formats listed below.
    mimetype: application/x-microsoft.net.object.binary.base64
    value   : The object must be serialized with
            : System.Runtime.Serialization.Formatters.Binary.BinaryFormatter
            : and then encoded with base64 encoding.
    mimetype: application/x-microsoft.net.object.soap.base64
    value   : The object must be serialized with
            : System.Runtime.Serialization.Formatters.Soap.SoapFormatter
            : and then encoded with base64 encoding.
    mimetype: application/x-microsoft.net.object.bytearray.base64
    value   : The object must be serialized into a byte array
            : using a System.ComponentModel.TypeConverter
            : and then encoded with base64 encoding.
    -->
  <xsd:schema id="root" xmlns="" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
    <xsd:import namespace="http://www.w3.org/XML/1998/namespace" />
    <xsd:element name="root" msdata:IsDataSet="true">
      <xsd:complexType>
        <xsd:choice maxOccurs="unbounded">
          <xsd:element name="metadata">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="value" type="xsd:string" minOccurs="0" />
              </xsd:sequence>
              <xsd:attribute name="name" use="required" type="xsd:string" />
              <xsd:attribute name="type" type="xsd:string" />
              <xsd:attribute name="mimetype" type="xsd:string" />
              <xsd:attribute ref="xml:space" />
            </xsd:complexType>
          </xsd:element>
          <xsd:element name="assembly">
            <xsd:complexType>
              <xsd:attribute name="alias" type="xsd:string" />
              <xsd:attribute name="name" type="xsd:string" />
            </xsd:complexType>
          </xsd:element>
          <xsd:element name="data">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="value" type="xsd:string" minOccurs="0"
                 msdata:Ordinal="1" />
                <xsd:element name="comment" type="xsd:string" minOccurs="0"
                 msdata:Ordinal="2" />
              </xsd:sequence>
              <xsd:attribute name="name" type="xsd:string" use="required"
                 msdata:Ordinal="1" />
              <xsd:attribute name="type" type="xsd:string" msdata:Ordinal="3" />
              <xsd:attribute name="mimetype" type="xsd:string" msdata:Ordinal="4" />
              <xsd:attribute ref="xml:space" />
            </xsd:complexType>
          </xsd:element>
          <xsd:element name="resheader">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="value" type="xsd:string" minOccurs="0"
                 msdata:Ordinal="1" />
              </xsd:sequence>
              <xsd:attribute name="name" type="xsd:string" use="required" />
            </xsd:complexType>
          </xsd:element>
        </xsd:choice>
      </xsd:complexType>
    </xsd:element>
  </xsd:schema>
  <resheader name="resmimetype">
    <value>text/microsoft-resx</value>
  </resheader>
  <resheader name="version">
```

```
      <value>2.0</value>
    </resheader>
    <resheader name="reader">
      <value>System.Resources.ResXResourceReader, System.Windows.Forms, Version=4.0.0.0,
     Culture=neutral, PublicKeyToken=b77a5c561934e089</value>
    </resheader>
    <resheader name="writer">
      <value>System.Resources.ResXResourceWriter, System.Windows.Forms, Version=4.0.0.0,
     Culture=neutral, PublicKeyToken=b77a5c561934e089</value>
    </resheader>
    <data name="Calendar1Resource1.Caption" xml:space="preserve">
      <value />
    </data>
    <data name="Calendar1Resource1.NextMonthText" xml:space="preserve">
      <value>&amp;gt;</value>
    </data>
    <data name="Calendar1Resource1.PrevMonthText" xml:space="preserve">
      <value>&amp;lt;</value>
    </data>
    <data name="Calendar1Resource1.SelectMonthText" xml:space="preserve">
      <value>&amp;gt;&amp;gt;</value>
    </data>
    <data name="Calendar1Resource1.SelectWeekText" xml:space="preserve">
      <value>&amp;gt;</value>
    </data>
    <data name="Calendar1Resource1.ToolTip" xml:space="preserve">
      <value />
    </data>
    <data name="Label1Resource1.Text" xml:space="preserve">
      <value>English</value>
    </data>
    <data name="Label1Resource1.ToolTip" xml:space="preserve">
      <value />
    </data>
    <data name="Label2Resource1.Text" xml:space="preserve">
      <value>French</value>
    </data>
    <data name="Label2Resource1.ToolTip" xml:space="preserve">
      <value />
    </data>
    <data name="Label3Resource1.Text" xml:space="preserve">
      <value>German</value>
    </data>
    <data name="Label3Resource1.ToolTip" xml:space="preserve">
      <value />
    </data>
    <data name="PageResource1.Title" xml:space="preserve">
      <value>Home Page</value>
    </data>
  </root>
```

The complete code of another resource file (`Default.aspx.fr.resx`) is shown in Listing 9.4:

Listing 9.4: Complete Code for the `Default.aspx.fr.resx` File

```
<?xml version="1.0" encoding="utf-8"?>
<root>
  <!--
    Microsoft ResX Schema
    Version 2.0
    The primary goals of this format is to allow a simple XML format
    that is mostly human readable. The generation and parsing of the
    various data types are done through the TypeConverter classes
    associated with the data types.
    Example:
    ... ado.net/XML headers & schema ...
    <resheader name="resmimetype">text/microsoft-resx</resheader>
    <resheader name="version">2.0</resheader>
    <resheader name="reader">System.Resources.ResXResourceReader, System.Windows.Forms,
   ...</resheader>
    <resheader name="writer">System.Resources.ResXResourceWriter, System.Windows.Forms,
   ...</resheader>
    <data name="Name1"><value>this is my long string</value><comment>this is a
  comment</comment></data>
    <data name="Color1" type="System.Drawing.Color, System.Drawing">Blue</data>
```

```
    <data name="Bitmap1" mimetype="application/x-microsoft.net.object.binary.base64">
        <value>[base64 mime encoded serialized .NET Framework object]</value>
    </data>
    <data name="Icon1" type="System.Drawing.Icon, System.Drawing"
  mimetype="application/x-microsoft.net.object.bytearray.base64">
        <value>[base64 mime encoded string representing a byte array form of the .NET
  Framework object]</value>
        <comment>This is a comment</comment>
    </data>
    There are any number of "resheader" rows that contain simple
    name/value pairs.
    Each data row contains a name, and value. The row also contains a
    type or mimetype. Type corresponds to a .NET class that support
    text/value conversion through the TypeConverter architecture.
    Classes that don't support this are serialized and stored with the
    mimetype set.
    The mimetype is used for serialized objects, and tells the
    ResXResourceReader how to depersist the object. This is currently not
    extensible. For a given mimetype the value must be set accordingly:
    Note - application/x-microsoft.net.object.binary.base64 is the format
    that the ResXResourceWriter will generate, however the reader can
    read any of the formats listed below.
    mimetype: application/x-microsoft.net.object.binary.base64
    value   : The object must be serialized with
            : System.Runtime.Serialization.Formatters.Binary.BinaryFormatter
            : and then encoded with base64 encoding.
    mimetype: application/x-microsoft.net.object.soap.base64
    value   : The object must be serialized with
            : System.Runtime.Serialization.Formatters.Soap.SoapFormatter
            : and then encoded with base64 encoding.
    mimetype: application/x-microsoft.net.object.bytearray.base64
    value   : The object must be serialized into a byte array
            : using a System.ComponentModel.TypeConverter
            : and then encoded with base64 encoding.    -->
  <xsd:schema id="root" xmlns="" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
    <xsd:import namespace="http://www.w3.org/XML/1998/namespace" />
    <xsd:element name="root" msdata:IsDataSet="true">
      <xsd:complexType>
        <xsd:choice maxOccurs="unbounded">
          <xsd:element name="metadata">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="value" type="xsd:string" minOccurs="0" />
              </xsd:sequence>
              <xsd:attribute name="name" use="required" type="xsd:string" />
              <xsd:attribute name="type" type="xsd:string" />
              <xsd:attribute name="mimetype" type="xsd:string" />
              <xsd:attribute ref="xml:space" />
            </xsd:complexType>
          </xsd:element>
          <xsd:element name="assembly">
            <xsd:complexType>
              <xsd:attribute name="alias" type="xsd:string" />
              <xsd:attribute name="name" type="xsd:string" />
            </xsd:complexType>
          </xsd:element>
          <xsd:element name="data">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="value" type="xsd:string" minOccurs="0"
                 msdata:Ordinal="1" />
                <xsd:element name="comment" type="xsd:string" minOccurs="0"
                 msdata:Ordinal="2" />
              </xsd:sequence>
              <xsd:attribute name="name" type="xsd:string" use="required"
                 msdata:Ordinal="1" />
              <xsd:attribute name="type" type="xsd:string" msdata:Ordinal="3" />
              <xsd:attribute name="mimetype" type="xsd:string" msdata:Ordinal="4" />
              <xsd:attribute ref="xml:space" />
            </xsd:complexType>
          </xsd:element>
          <xsd:element name="resheader">
            <xsd:complexType>
```

```
                <xsd:sequence>
                  <xsd:element name="value" type="xsd:string" minOccurs="0"
                   msdata:Ordinal="1" />
                </xsd:sequence>
                <xsd:attribute name="name" type="xsd:string" use="required" />
              </xsd:complexType>
            </xsd:element>
          </xsd:choice>
        </xsd:complexType>
      </xsd:element>
    </xsd:schema>
    <resheader name="resmimetype">
      <value>text/microsoft-resx</value>
    </resheader>
    <resheader name="version">
      <value>2.0</value>
    </resheader>
    <resheader name="reader">
      <value>System.Resources.ResXResourceReader, System.Windows.Forms, Version=4.0.0.0,
     Culture=neutral, PublicKeyToken=b77a5c561934e089</value>
    </resheader>
    <resheader name="writer">
      <value>System.Resources.ResXResourceWriter, System.Windows.Forms, Version=4.0.0.0,
     Culture=neutral, PublicKeyToken=b77a5c561934e089</value>
    </resheader>
    <data name="Calendar1Resource1.Caption" xml:space="preserve">
      <value />
    </data>
    <data name="Calendar1Resource1.NextMonthText" xml:space="preserve">
      <value>&amp;gt;</value>
    </data>
    <data name="Calendar1Resource1.PrevMonthText" xml:space="preserve">
      <value>&amp;lt;</value>
    </data>
    <data name="Calendar1Resource1.SelectMonthText" xml:space="preserve">
      <value>&amp;gt;&amp;gt;</value>
    </data>
    <data name="Calendar1Resource1.SelectWeekText" xml:space="preserve">
      <value>&amp;gt;</value>
    </data>
    <data name="Calendar1Resource1.ToolTip" xml:space="preserve">
      <value />
    </data>
    <data name="Label1Resource1.Text" xml:space="preserve">
      <value>Anglais</value>
    </data>
    <data name="Label1Resource1.ToolTip" xml:space="preserve">
      <value />
    </data>
    <data name="Label2Resource1.Text" xml:space="preserve">
      <value>Français</value>
    </data>
    <data name="Label2Resource1.ToolTip" xml:space="preserve">
      <value />
    </data>
    <data name="Label3Resource1.Text" xml:space="preserve">
      <value>Allemand</value>
    </data>
    <data name="Label3Resource1.ToolTip" xml:space="preserve">
      <value />
    </data>
    <data name="PageResource1.Title" xml:space="preserve">
      <value>Home Page</value>
    </data>
  </root>
```

The complete code of another resource file (`Default.aspx.de.resx`) is shown in Listing 9.5:

Listing 9.5: Complete Code for the `Default.aspx.de.resx` File

```
<?xml version="1.0" encoding="utf-8"?>
<root>
  <!--
    Microsoft ResX Schema
    Version 2.0
```

```
    The primary goals of this format is to allow a simple XML format
    that is mostly human readable. The generation and parsing of the
    various data types are done through the TypeConverter classes
    associated with the data types.
    Example:
    ... ado.net/XML headers & schema ...
    <resheader name="resmimetype">text/microsoft-resx</resheader>
    <resheader name="version">2.0</resheader>
    <resheader name="reader">System.Resources.ResXResourceReader, System.Windows.Forms,
  ...</resheader>
    <resheader name="writer">System.Resources.ResXResourceWriter, System.Windows.Forms,
  ...</resheader>
    <data name="Name1"><value>this is my long string</value><comment>this is a
  comment</comment></data>
    <data name="Color1" type="System.Drawing.Color, System.Drawing">Blue</data>
    <data name="Bitmap1" mimetype="application/x-microsoft.net.object.binary.base64">
        <value>[base64 mime encoded serialized .NET Framework object]</value>
    </data>
    <data name="Icon1" type="System.Drawing.Icon, System.Drawing"
  mimetype="application/x-microsoft.net.object.bytearray.base64">
        <value>[base64 mime encoded string representing a byte array form of the .NET
  Framework object]</value>
        <comment>This is a comment</comment>
    </data>
    There are any number of "resheader" rows that contain simple
    name/value pairs.
    Each data row contains a name, and value. The row also contains a
    type or mimetype. Type corresponds to a .NET class that support
    text/value conversion through the TypeConverter architecture.
    Classes that don't support this are serialized and stored with the
    mimetype set.
    The mimetype is used for serialized objects, and tells the
    ResXResourceReader how to depersist the object. This is currently not
    extensible. For a given mimetype the value must be set accordingly:
    Note - application/x-microsoft.net.object.binary.base64 is the format
    that the ResXResourceWriter will generate, however the reader can
    read any of the formats listed below.
    mimetype: application/x-microsoft.net.object.binary.base64
    value   : The object must be serialized with
            : System.Runtime.Serialization.Formatters.Binary.BinaryFormatter
            : and then encoded with base64 encoding.
    mimetype: application/x-microsoft.net.object.soap.base64
    value   : The object must be serialized with
            : System.Runtime.Serialization.Formatters.Soap.SoapFormatter
            : and then encoded with base64 encoding.
    mimetype: application/x-microsoft.net.object.bytearray.base64
    value   : The object must be serialized into a byte array
            : using a System.ComponentModel.TypeConverter
            : and then encoded with base64 encoding.
    -->
  <xsd:schema id="root" xmlns="" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
    <xsd:import namespace="http://www.w3.org/XML/1998/namespace" />
    <xsd:element name="root" msdata:IsDataSet="true">
      <xsd:complexType>
        <xsd:choice maxOccurs="unbounded">
          <xsd:element name="metadata">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="value" type="xsd:string" minOccurs="0" />
              </xsd:sequence>
              <xsd:attribute name="name" use="required" type="xsd:string" />
              <xsd:attribute name="type" type="xsd:string" />
              <xsd:attribute name="mimetype" type="xsd:string" />
              <xsd:attribute ref="xml:space" />
            </xsd:complexType>
          </xsd:element>
          <xsd:element name="assembly">
            <xsd:complexType>
              <xsd:attribute name="alias" type="xsd:string" />
```

```
                <xsd:attribute name="name" type="xsd:string" />
              </xsd:complexType>
            </xsd:element>
            <xsd:element name="data">
              <xsd:complexType>
                <xsd:sequence>
                  <xsd:element name="value" type="xsd:string" minOccurs="0"
                   msdata:Ordinal="1" />
                  <xsd:element name="comment" type="xsd:string" minOccurs="0"
                   msdata:Ordinal="2" />
                </xsd:sequence>
                <xsd:attribute name="name" type="xsd:string" use="required"
                   msdata:Ordinal="1" />
                <xsd:attribute name="type" type="xsd:string" msdata:Ordinal="3" />
                <xsd:attribute name="mimetype" type="xsd:string" msdata:Ordinal="4" />
                <xsd:attribute ref="xml:space" />
              </xsd:complexType>
            </xsd:element>
            <xsd:element name="resheader">
              <xsd:complexType>
                <xsd:sequence>
                  <xsd:element name="value" type="xsd:string" minOccurs="0"
                   msdata:Ordinal="1" />
                </xsd:sequence>
                <xsd:attribute name="name" type="xsd:string" use="required" />
              </xsd:complexType>
            </xsd:element>
          </xsd:choice>
        </xsd:complexType>
      </xsd:element>
    </xsd:schema>
    <resheader name="resmimetype">
      <value>text/microsoft-resx</value>
    </resheader>
    <resheader name="version">
      <value>2.0</value>
    </resheader>
    <resheader name="reader">
      <value>System.Resources.ResXResourceReader, System.Windows.Forms, Version=4.0.0.0,
     Culture=neutral, PublicKeyToken=b77a5c561934e089</value>
    </resheader>
    <resheader name="writer">
      <value>System.Resources.ResXResourceWriter, System.Windows.Forms, Version=4.0.0.0,
     Culture=neutral, PublicKeyToken=b77a5c561934e089</value>
    </resheader>
    <data name="Calendar1Resource1.Caption" xml:space="preserve">
      <value />
    </data>
    <data name="Calendar1Resource1.NextMonthText" xml:space="preserve">
      <value>&amp;gt;</value>
    </data>
    <data name="Calendar1Resource1.PrevMonthText" xml:space="preserve">
      <value>&amp;lt;</value>
    </data>
    <data name="Calendar1Resource1.SelectMonthText" xml:space="preserve">
      <value>&amp;gt;&amp;gt;</value>
    </data>
    <data name="Calendar1Resource1.SelectWeekText" xml:space="preserve">
      <value>&amp;gt;</value>
    </data>
    <data name="Calendar1Resource1.ToolTip" xml:space="preserve">
      <value />
    </data>
    <data name="Label1Resource1.Text" xml:space="preserve">
      <value>Englishch</value>
    </data>
    <data name="Label1Resource1.ToolTip" xml:space="preserve">
      <value />
    </data>
    <data name="Label2Resource1.Text" xml:space="preserve">
```

```
    <value>Franzosisch</value>
  </data>
  <data name="Label2Resource1.ToolTip" xml:space="preserve">
    <value />
  </data>
  <data name="Label3Resource1.Text" xml:space="preserve">
    <value>Deutsch</value>
  </data>
  <data name="Label3Resource1.ToolTip" xml:space="preserve">
    <value />
  </data>
  <data name="PageResource1.Title" xml:space="preserve">
    <value>Home Page</value>
  </data>
</root>
```

Listings 9.3, 9.4, and 9.5 contain the complete code of all the resource files of the `ImplicitLocalizationCS` application, which can be used in C# language.

Explicit Expression

Explicit localization expressions are automatically generated when resources are created using explicit localization. You need to use explicit localization in the ASP.NET Web pages to localize large text messages, in addition to the controls and labels that are localized.

Let's create an ASP.NET Empty Website for explicit localization, `ExplicitLocalizationCS`. This application is also available on the CD-ROM. After creating the application, follow these steps:

1. Right-click the root directory of the Web application in Solution Explorer and select the `Add ASP.NET Folder→App_GlobalResources` option from the context menu to add the `App_GlobalResources` folder in Solution Explorer.
2. Right-click the `App_GlobalResources` folder and select the Add New Item option from the context menu to open the Add New Item dialog box, as shown in Figure 9.14:

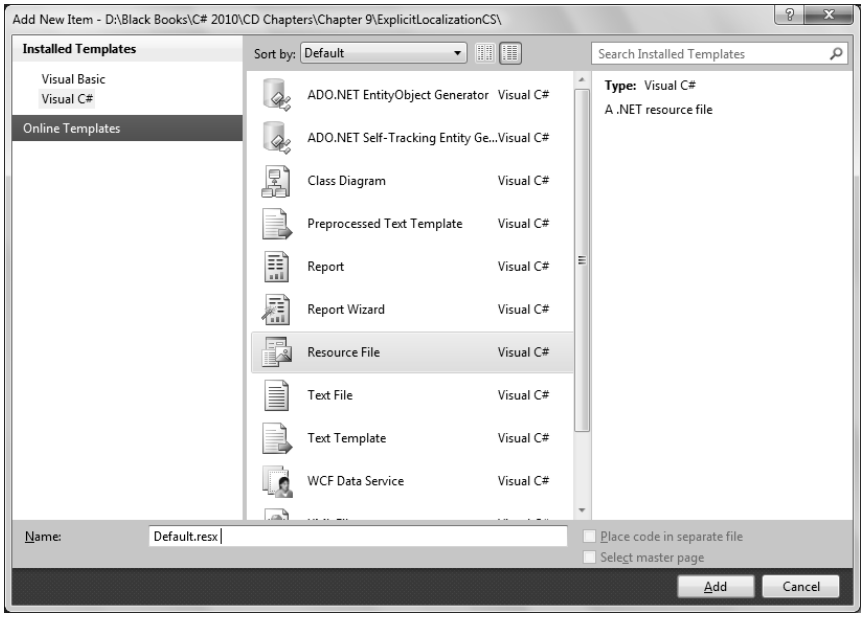

Figure 9.14: Adding a Resource File

3. Select Resource File in the Templates pane and type `Default.resx` in the Name field. Then, click the Add button to add the resource file in the `App_GlobalResources` folder.

4. Create two other resource files in the `App_GlobalResources` folder to include the resource files for the French and German languages. Name the resource files as `Default.fr.resx` and `Default.de.resx`.
5. Double-click the `Default.fr.resx` file in Solution Explorer to open it. Type `msg` in the Name column and `Français` in the Value column, as shown in Figure 9.15:

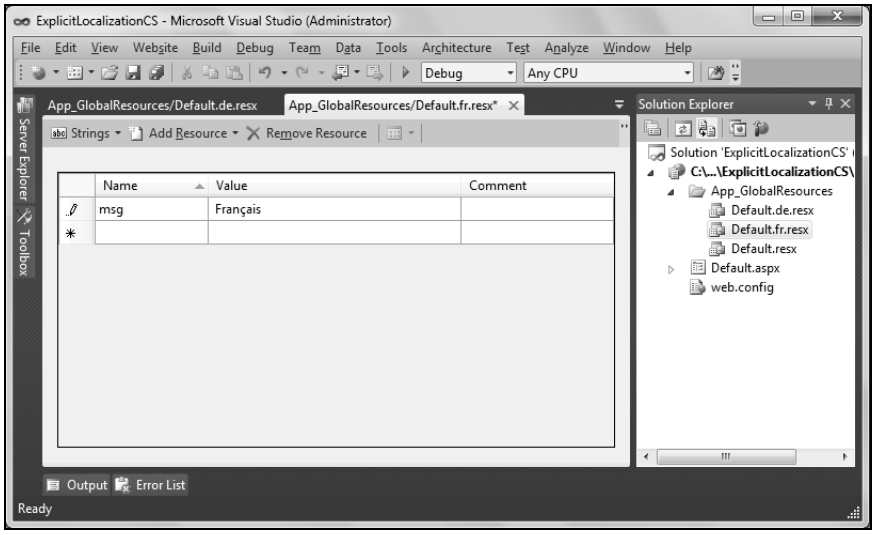

Figure 9.15: Showing the Resource Editor for the Default.fr.resx File

6. Double-click the `Default.de.resx` file in Solution Explorer to open the file. Type `msg` in the first row of the Name column and `Deutsch` in the first row of the Value column, as shown in Figure 9.16:

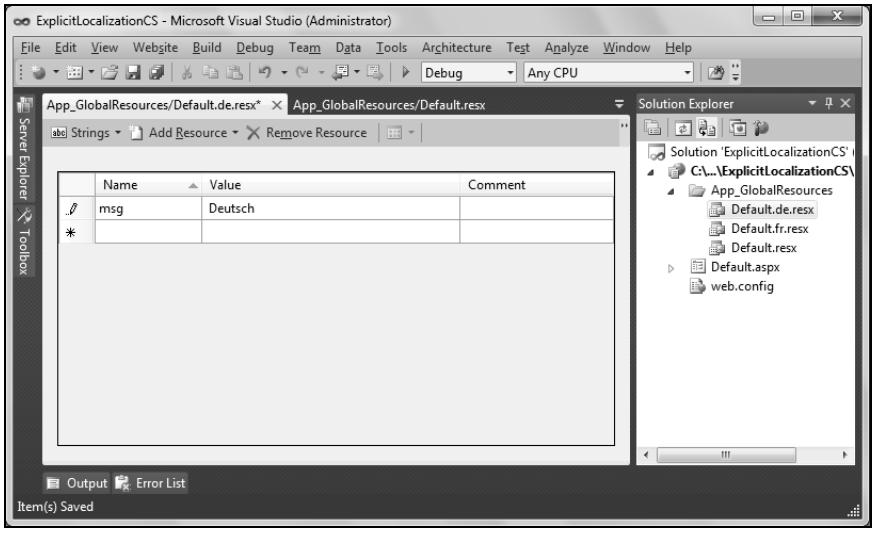

Figure 9.16: Showing the Resource Editor for the Default.de.resx File

7. Double-click the `Default.resx` file in Solution Explorer to open the file. Type `msg` in the first row of the Name column and `English` in the first row of the Value column.
8. Add a `Label` control on the `Default.aspx` page to display the text of the language, whose resource file is loaded in the application.

9. Open the properties of the Label control and click the ellipsis button in the Expressions property in the Properties window, which displays the Label1 Expressions dialog box (the name of this dialog box may vary on the basis of the name control). The Text option is selected by default in the Bindable properties section of the Label1 Expressions dialog box.
10. Select Resources from the drop-down list in the Expression type section. Set `ClassKey` to Default and `ResourceKey` to `msg` in the `ClassKey` and `ResourceKey` Expression properties, respectively, as shown in Figure 9.17:

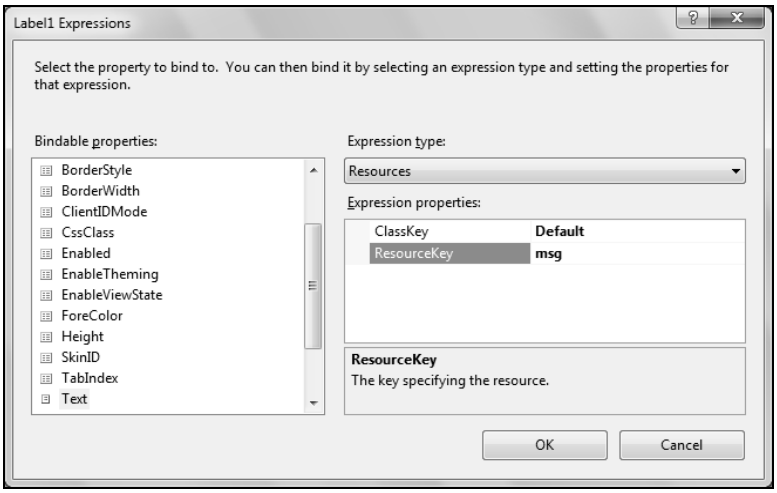

Figure 9.17: Specifying Values in the Label1 Expressions Dialog Box

11. Click the OK button to close the Label1 Expressions dialog box.

After creating the UI for a Web application using explicit localization, you need to change the browser settings. The browser settings are changed so that the desired language is displayed when the Web application is executed. The output of the Web application, when the browser settings are changed to English, is shown in Figure 9.18:

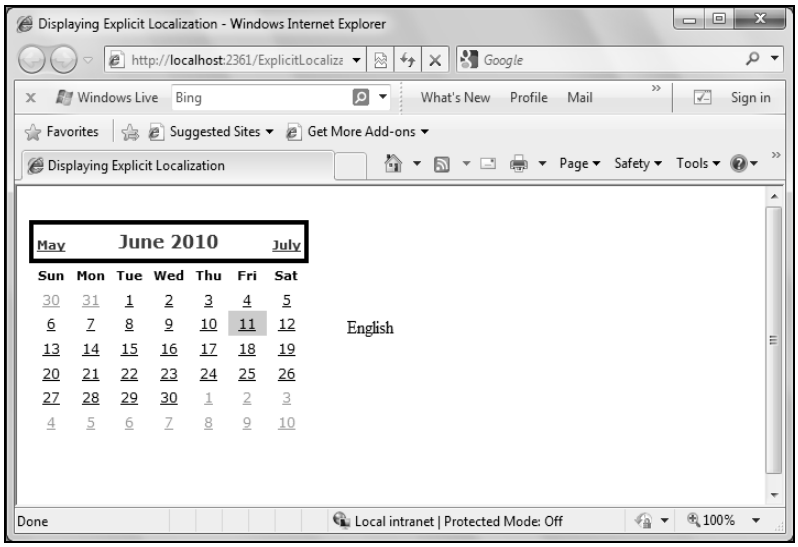

Figure 9.18: Displaying the Output of the Web Application with English as the Preferred Language

The output of the Web application, when the browser settings are changed to French, is shown in Figure 9.19:

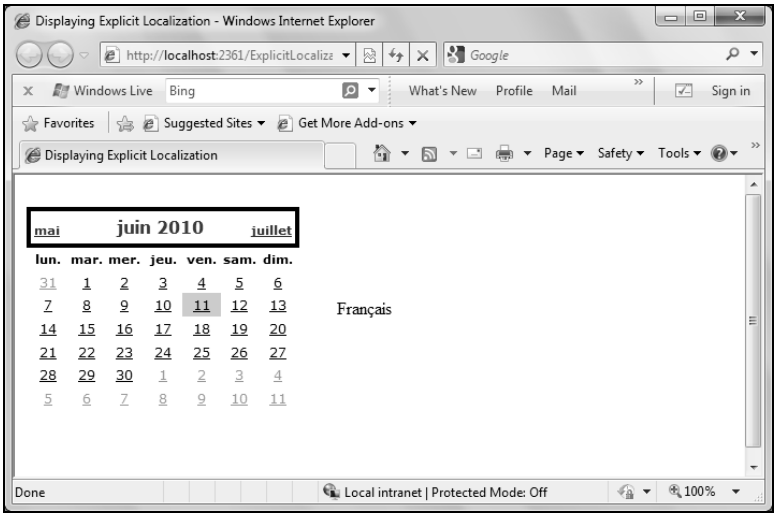

Figure 9.19: Displaying the Output of the Web Application with French as the Preferred Language

The output of the Web application, when the browser settings are changed to German, is shown in Figure 9.20:

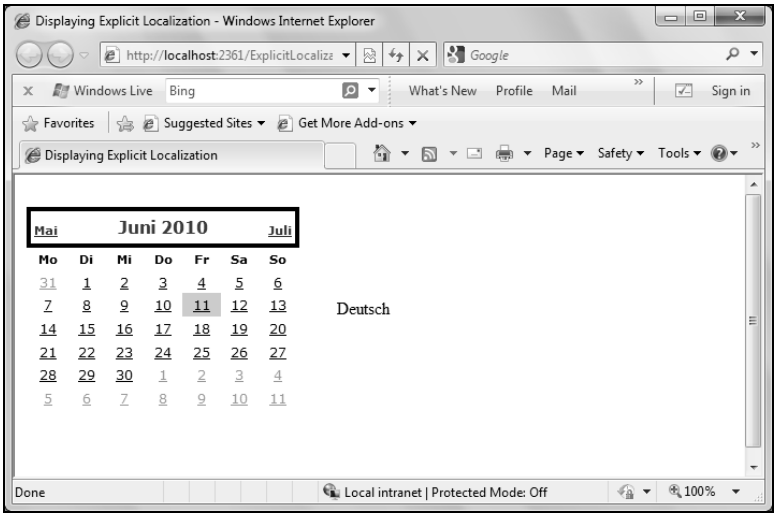

Figure 9.20: Displaying the Output of the Web Application with German as the Preferred Language

The complete code for the `Default.aspx` page of the `ExplicitLocalizationCS` application is shown in Listing 9.6:

Listing 9.6: Complete Code for the `Default.aspx` Page

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
  Inherits="_Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Displaying Explicit Localization</title>
    <style type="text/css">
        .style1
        {
```

```
            width: 100%;
            height: 191px;
        }
        .style2
        {
            width: 285px;
        }
    </style>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <br />
        <table class="style1">
            <tr>
                <td class="style2">
                   <asp:Calendar ID="Calendar1" runat="server" BackColor="White"
                   BorderColor="White"
                  BorderWidth="1px" Font-Names="Verdana" Font-Size="9pt" ForeColor="Black"
                    Height="190px"
                  NextPrevFormat="FullMonth" Width="254px">
                  <SelectedDayStyle BackColor="#333399" ForeColor="White" />
                  <TodayDayStyle BackColor="#CCCCCC" />
                  <OtherMonthDayStyle ForeColor="#999999" />
                  <NextPrevStyle Font-Bold="True" Font-Size="8pt" ForeColor="#333333"
                    VerticalAlign="Bottom" />
                  <DayHeaderStyle Font-Bold="True" Font-Size="8pt" />
                  <TitleStyle BackColor="White" BorderColor="Black" BorderWidth="4px"
                  Font-Bold="True"
                  Font-Size="12pt" ForeColor="#333399" />
                  </asp:Calendar>
                 </td>
                 <td>
                  <asp:Label ID="Label1" runat="server" Text="<%$ Resources:Default, msg
                  %>"></asp:Label>
                 </td>
            </tr>
            <tr>
                <td class="style2">
                     </td>
                <td>
                     </td>
            </tr>
        </table>
        <br />
        <br />
    </div>
    </form>
</body>
</html>
```

The complete code of the resource file (`Default.de.resx`) is shown in Listing 9.7:

Listing 9.7: Complete Code for the `Default.de.resx` File

```
<?xml version="1.0" encoding="utf-8"?>
<root>
  <!--
    Microsoft ResX Schema
    Version 2.0
    The primary goals of this format is to allow a simple XML format
    that is mostly human readable. The generation and parsing of the
    various data types are done through the TypeConverter classes
    associated with the data types.

    Example:

    ... ado.net/XML headers & schema ...
```

```
    <resheader name="resmimetype">text/microsoft-resx</resheader>
    <resheader name="version">2.0</resheader>
    <resheader name="reader">System.Resources.ResXResourceReader, System.Windows.Forms,
  ...</resheader>

    <resheader name="writer">System.Resources.ResXResourceWriter, System.Windows.Forms,
  ...</resheader>
    <data name="Name1"><value>this is my long string</value><comment>this is a
  comment</comment></data>
    <data name="Color1" type="System.Drawing.Color, System.Drawing">Blue</data>
    <data name="Bitmap1" mimetype="application/x-microsoft.net.object.binary.base64">
        <value>[base64 mime encoded serialized .NET Framework object]</value>
    </data>
    <data name="Icon1" type="System.Drawing.Icon, System.Drawing"
  mimetype="application/x-microsoft.net.object.bytearray.base64">
        <value>[base64 mime encoded string representing a byte array form of the .NET
  Framework object]</value>
        <comment>This is a comment</comment>
    </data>

    There are any number of "resheader" rows that contain simple
    name/value pairs.

    Each data row contains a name, and value. The row also contains a
    type or mimetype. Type corresponds to a .NET class that support
    text/value conversion through the TypeConverter architecture.
    Classes that don't support this are serialized and stored with the
    mimetype set.

    The mimetype is used for serialized objects, and tells the
    ResXResourceReader how to depersist the object. This is currently not
    extensible. For a given mimetype the value must be set accordingly:

    Note - application/x-microsoft.net.object.binary.base64 is the format
    that the ResXResourceWriter will generate, however the reader can
    read any of the formats listed below.

    mimetype: application/x-microsoft.net.object.binary.base64
    value   : The object must be serialized with
            : System.Runtime.Serialization.Formatters.Binary.BinaryFormatter
            : and then encoded with base64 encoding.

    mimetype: application/x-microsoft.net.object.soap.base64
    value   : The object must be serialized with
            : System.Runtime.Serialization.Formatters.Soap.SoapFormatter
            : and then encoded with base64 encoding.

    mimetype: application/x-microsoft.net.object.bytearray.base64
    value   : The object must be serialized into a byte array
            : using a System.ComponentModel.TypeConverter
            : and then encoded with base64 encoding.
    -->
  <xsd:schema id="root" xmlns="" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
    <xsd:import namespace="http://www.w3.org/XML/1998/namespace" />
    <xsd:element name="root" msdata:IsDataSet="true">
      <xsd:complexType>
        <xsd:choice maxOccurs="unbounded">
          <xsd:element name="metadata">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="value" type="xsd:string" minOccurs="0" />
              </xsd:sequence>
              <xsd:attribute name="name" use="required" type="xsd:string" />
              <xsd:attribute name="type" type="xsd:string" />
              <xsd:attribute name="mimetype" type="xsd:string" />
```

```
                <xsd:attribute ref="xml:space" />
              </xsd:complexType>
            </xsd:element>
            <xsd:element name="assembly">
              <xsd:complexType>
                <xsd:attribute name="alias" type="xsd:string" />
                <xsd:attribute name="name" type="xsd:string" />
              </xsd:complexType>
            </xsd:element>
            <xsd:element name="data">
              <xsd:complexType>
                <xsd:sequence>
                  <xsd:element name="value" type="xsd:string" minOccurs="0"
                   msdata:Ordinal="1" />
                  <xsd:element name="comment" type="xsd:string" minOccurs="0"
                   msdata:Ordinal="2" />
                </xsd:sequence>
                <xsd:attribute name="name" type="xsd:string" use="required"
                   msdata:Ordinal="1" />
                <xsd:attribute name="type" type="xsd:string" msdata:Ordinal="3" />
                <xsd:attribute name="mimetype" type="xsd:string" msdata:Ordinal="4" />
                <xsd:attribute ref="xml:space" />
              </xsd:complexType>
            </xsd:element>
            <xsd:element name="resheader">
              <xsd:complexType>
                <xsd:sequence>
                  <xsd:element name="value" type="xsd:string" minOccurs="0"
                   msdata:Ordinal="1" />
                </xsd:sequence>
                <xsd:attribute name="name" type="xsd:string" use="required" />
              </xsd:complexType>
            </xsd:element>
          </xsd:choice>
        </xsd:complexType>
      </xsd:element>
    </xsd:schema>
    <resheader name="resmimetype">
      <value>text/microsoft-resx</value>
    </resheader>
    <resheader name="version">
      <value>2.0</value>
    </resheader>
    <resheader name="reader">
      <value>System.Resources.ResXResourceReader, System.Windows.Forms, Version=4.0.0.0,
     Culture=neutral, PublicKeyToken=b77a5c561934e089</value>
    </resheader>
    <resheader name="writer">
      <value>System.Resources.ResXResourceWriter, System.Windows.Forms, Version=4.0.0.0,
     Culture=neutral, PublicKeyToken=b77a5c561934e089</value>
    </resheader>
    <data name="msg" xml:space="preserve">
      <value>Deutsch</value>
    </data>
  </root>
```

The complete code of another resource file (`Default.fr.resx`) is shown in Listing 9.8:

Listing 9.8: Complete Code for the `Default.fr.resx` File

```
<?xml version="1.0" encoding="utf-8"?>
<root>
  <!--
    Microsoft ResX Schema

    Version 2.0

    The primary goals of this format is to allow a simple XML format
```

```
    that is mostly human readable. The generation and parsing of the
    various data types are done through the TypeConverter classes
    associated with the data types.

    Example:
        ... ado.net/XML headers & schema ...
    <resheader name="resmimetype">text/microsoft-resx</resheader>
    <resheader name="version">2.0</resheader>
    <resheader name="reader">System.Resources.ResXResourceReader, System.Windows.Forms,
...</resheader>
    <resheader name="writer">System.Resources.ResXResourceWriter, System.Windows.Forms,
...</resheader>
    <data name="Name1"><value>this is my long string</value><comment>this is a
comment</comment></data>
    <data name="Color1" type="System.Drawing.Color, System.Drawing">Blue</data>
    <data name="Bitmap1" mimetype="application/x-microsoft.net.object.binary.base64">
        <value>[base64 mime encoded serialized .NET Framework object]</value>
    </data>
    <data name="Icon1" type="System.Drawing.Icon, System.Drawing"
mimetype="application/x-microsoft.net.object.bytearray.base64">
        <value>[base64 mime encoded string representing a byte array form of the .NET
Framework object]</value>
        <comment>This is a comment</comment>
    </data>

    There are any number of "resheader" rows that contain simple
    name/value pairs.

    Each data row contains a name, and value. The row also contains a
    type or mimetype. Type corresponds to a .NET class that support
    text/value conversion through the TypeConverter architecture.
    Classes that don't support this are serialized and stored with the
    mimetype set.

    The mimetype is used for serialized objects, and tells the
    ResXResourceReader how to depersist the object. This is currently not
    extensible. For a given mimetype the value must be set accordingly:

    Note - application/x-microsoft.net.object.binary.base64 is the format
    that the ResXResourceWriter will generate, however the reader can
    read any of the formats listed below.

    mimetype: application/x-microsoft.net.object.binary.base64
    value   : The object must be serialized with
            : System.Runtime.Serialization.Formatters.Binary.BinaryFormatter
            : and then encoded with base64 encoding.

    mimetype: application/x-microsoft.net.object.soap.base64
    value   : The object must be serialized with
            : System.Runtime.Serialization.Formatters.Soap.SoapFormatter
            : and then encoded with base64 encoding.

    mimetype: application/x-microsoft.net.object.bytearray.base64
    value   : The object must be serialized into a byte array
            : using a System.ComponentModel.TypeConverter
            : and then encoded with base64 encoding.
    -->
  <xsd:schema id="root" xmlns="" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
    <xsd:import namespace="http://www.w3.org/XML/1998/namespace" />
    <xsd:element name="root" msdata:IsDataSet="true">
      <xsd:complexType>
        <xsd:choice maxOccurs="unbounded">
          <xsd:element name="metadata">
            <xsd:complexType>
              <xsd:sequence>
```

```
                    <xsd:element name="value" type="xsd:string" minOccurs="0" />
                  </xsd:sequence>
                  <xsd:attribute name="name" use="required" type="xsd:string" />
                  <xsd:attribute name="type" type="xsd:string" />
                  <xsd:attribute name="mimetype" type="xsd:string" />
                  <xsd:attribute ref="xml:space" />
                </xsd:complexType>
              </xsd:element>
              <xsd:element name="assembly">
                <xsd:complexType>
                  <xsd:attribute name="alias" type="xsd:string" />
                  <xsd:attribute name="name" type="xsd:string" />
                </xsd:complexType>
              </xsd:element>
              <xsd:element name="data">
                <xsd:complexType>
                  <xsd:sequence>
                    <xsd:element name="value" type="xsd:string" minOccurs="0"
                     msdata:Ordinal="1" />
                    <xsd:element name="comment" type="xsd:string" minOccurs="0"
                     msdata:Ordinal="2" />
                  </xsd:sequence>
                  <xsd:attribute name="name" type="xsd:string" use="required"
                     msdata:Ordinal="1" />
                  <xsd:attribute name="type" type="xsd:string" msdata:Ordinal="3" />
                  <xsd:attribute name="mimetype" type="xsd:string" msdata:Ordinal="4" />
                  <xsd:attribute ref="xml:space" />
                </xsd:complexType>
              </xsd:element>
              <xsd:element name="resheader">
                <xsd:complexType>
                  <xsd:sequence>
                    <xsd:element name="value" type="xsd:string" minOccurs="0"
                     msdata:Ordinal="1" />
                  </xsd:sequence>
                  <xsd:attribute name="name" type="xsd:string" use="required" />
                </xsd:complexType>
              </xsd:element>
            </xsd:choice>
          </xsd:complexType>
        </xsd:element>
      </xsd:schema>
      <resheader name="resmimetype">
        <value>text/microsoft-resx</value>
      </resheader>
      <resheader name="version">
        <value>2.0</value>
      </resheader>
      <resheader name="reader">
        <value>System.Resources.ResXResourceReader, System.Windows.Forms, Version=4.0.0.0,
       Culture=neutral, PublicKeyToken=b77a5c561934e089</value>
      </resheader>
      <resheader name="writer">
        <value>System.Resources.ResXResourceWriter, System.Windows.Forms, Version=4.0.0.0,
       Culture=neutral, PublicKeyToken=b77a5c561934e089</value>
      </resheader>
      <data name="msg" xml:space="preserve">
        <value>Français</value>
      </data>

    </root>
```

The complete code of another resource file (`Default.resx`) is shown in Listing 9.9:

Listing 9.9: Complete Code for the `Default.resx` File

```
<?xml version="1.0" encoding="utf-8"?>
<root>
  <!--
    Microsoft ResX Schema

    Version 2.0

    The primary goals of this format is to allow a simple XML format
    that is mostly human readable. The generation and parsing of the
    various data types are done through the TypeConverter classes
    associated with the data types.

    Example:

    ... ado.net/XML headers & schema ...
    <resheader name="resmimetype">text/microsoft-resx</resheader>
    <resheader name="version">2.0</resheader>
    <resheader name="reader">System.Resources.ResXResourceReader, System.Windows.Forms,
...</resheader>
    <resheader name="writer">System.Resources.ResXResourceWriter, System.Windows.Forms,
...</resheader>
    <data name="Name1"><value>this is my long string</value><comment>this is a
comment</comment></data>
    <data name="Color1" type="System.Drawing.Color, System.Drawing">Blue</data>
    <data name="Bitmap1" mimetype="application/x-microsoft.net.object.binary.base64">
        <value>[base64 mime encoded serialized .NET Framework object]</value>
    </data>
    <data name="Icon1" type="System.Drawing.Icon, System.Drawing"
mimetype="application/x-microsoft.net.object.bytearray.base64">
        <value>[base64 mime encoded string representing a byte array form of the .NET
Framework object]</value>
        <comment>This is a comment</comment>
    </data>

    There are any number of "resheader" rows that contain simple
    name/value pairs.

    Each data row contains a name, and value. The row also contains a
    type or mimetype. Type corresponds to a .NET class that support
    text/value conversion through the TypeConverter architecture.
    Classes that don't support this are serialized and stored with the
    mimetype set.

    The mimetype is used for serialized objects, and tells the
    ResXResourceReader how to depersist the object. This is currently not
    extensible. For a given mimetype the value must be set accordingly:

    Note - application/x-microsoft.net.object.binary.base64 is the format
    that the ResXResourceWriter will generate, however the reader can
    read any of the formats listed below.

    mimetype: application/x-microsoft.net.object.binary.base64
    value   : The object must be serialized with
            : System.Runtime.Serialization.Formatters.Binary.BinaryFormatter
            : and then encoded with base64 encoding.

    mimetype: application/x-microsoft.net.object.soap.base64
    value   : The object must be serialized with
            : System.Runtime.Serialization.Formatters.Soap.SoapFormatter
            : and then encoded with base64 encoding.

    mimetype: application/x-microsoft.net.object.bytearray.base64
    value   : The object must be serialized into a byte array
            : using a System.ComponentModel.TypeConverter
            : and then encoded with base64 encoding.
```

```
    -->
  <xsd:schema id="root" xmlns="" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
    <xsd:import namespace="http://www.w3.org/XML/1998/namespace" />
    <xsd:element name="root" msdata:IsDataSet="true">
      <xsd:complexType>
        <xsd:choice maxOccurs="unbounded">
          <xsd:element name="metadata">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="value" type="xsd:string" minOccurs="0" />
              </xsd:sequence>
              <xsd:attribute name="name" use="required" type="xsd:string" />
              <xsd:attribute name="type" type="xsd:string" />
              <xsd:attribute name="mimetype" type="xsd:string" />
              <xsd:attribute ref="xml:space" />
            </xsd:complexType>
          </xsd:element>
          <xsd:element name="assembly">
            <xsd:complexType>
              <xsd:attribute name="alias" type="xsd:string" />
              <xsd:attribute name="name" type="xsd:string" />
            </xsd:complexType>
          </xsd:element>
          <xsd:element name="data">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="value" type="xsd:string" minOccurs="0"
                 msdata:Ordinal="1" />
                <xsd:element name="comment" type="xsd:string" minOccurs="0"
                 msdata:Ordinal="2" />
              </xsd:sequence>
              <xsd:attribute name="name" type="xsd:string" use="required"
                 msdata:Ordinal="1" />
              <xsd:attribute name="type" type="xsd:string" msdata:Ordinal="3" />
              <xsd:attribute name="mimetype" type="xsd:string" msdata:Ordinal="4" />
              <xsd:attribute ref="xml:space" />
            </xsd:complexType>
          </xsd:element>
          <xsd:element name="resheader">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="value" type="xsd:string" minOccurs="0"
                 msdata:Ordinal="1" />
              </xsd:sequence>
              <xsd:attribute name="name" type="xsd:string" use="required" />
            </xsd:complexType>
          </xsd:element>
        </xsd:choice>
      </xsd:complexType>
    </xsd:element>
  </xsd:schema>
  <resheader name="resmimetype">
    <value>text/microsoft-resx</value>
  </resheader>
  <resheader name="version">
    <value>2.0</value>
  </resheader>
  <resheader name="reader">
    <value>System.Resources.ResXResourceReader, System.Windows.Forms, Version=4.0.0.0,
   Culture=neutral, PublicKeyToken=b77a5c561934e089</value>
  </resheader>
  <resheader name="writer">
    <value>System.Resources.ResXResourceWriter, System.Windows.Forms, Version=4.0.0.0,
   Culture=neutral, PublicKeyToken=b77a5c561934e089</value>
  </resheader>
```

```
    <data name="msg" xml:space="preserve">
      <value>English</value>
    </data>
  </root>
```

Listings 9.7, 9.8, and 9.9 contain the complete code of all the resource files, namely Default.de.resx, Default.fr.resx, and Default.resx, respectively, of the `ExplicitLocalizationCS` application, which can be used in C# language.

Implementing Localization

You can generate resources for Server control properties by using implicit and explicit localization expressions. The localization expressions to the @ Page directive and other sections of HTML can be applied to identify the areas that need to be localized before generating the resources.

To implement localization in a Web Application, you need to localize the following elements:

- HTML controls
- Static content

HTML Controls

HTML controls cannot be localized using implicit or explicit expressions, unless they are running on the server. To run the HTML controls on the server, you need to include the `runat="server"` attribute in the `@ Page` directive. After the HTML controls are marked as Server controls, local resources are automatically generated for the localizable properties, such as the Text and ToolTip properties of the controls. HTML Server controls can also be bound to implicit or explicit expressions; the latter is generated by using the Expressions dialog box as described earlier in the explicit localization section. You can also bind the HTML elements, such as page titles and style sheet links of the page directive, to the resources.

Static Content

You can use the localization expressions to localize the properties of controls used on a Web page. However, sometimes the Web application also includes static content, such as copyright information or disclaimer information. ASP.NET Framework allows you to localize the static content. To localize a static content in a Web application, the `<Localize runat="server">` tag is used.

For example, consider that you need to add the disclaimer information in a Web application. The following code snippet displays the code to be added in the `Default.aspx` page of the Web application to localize static content:

```
<asp:Localize runat="server" meta:resourcekey="Information"> Disclaimer: This
  sample application is not intended to provide accurate or
up-to-date information.</asp:Localize>
```

In the preceding code snippet, a local resource value is generated for the `Text` property of the `Localize` control. You can also set the text direction for the static content used in a Web application for a specific local culture of a region, where the content is read from right to left or vice versa. To do so, create a new Web application in Visual Studio 2010 named `StaticContentCultureCS` (also available in the CD-ROM), and follow these steps:

1. Add the `App_GlobalResources` folder in the `StaticContentCultureCS` Web application.
2. Add a new item `Default.resx` in the `App_GlobalResources` folder of Solution Explorer and set the Name field as msg and TextDirection and their corresponding value in the Value field as You can change the direction of the static text and (Left-to-Right) LTR respectively, as shown in Figure 9.21:

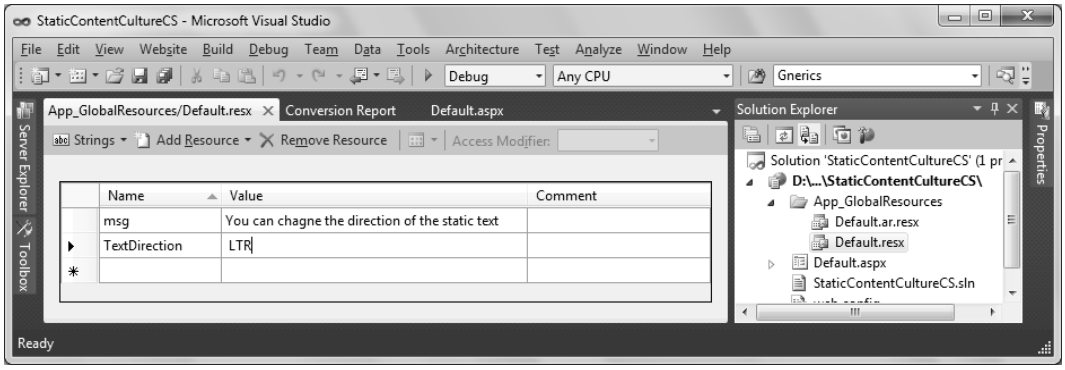

Figure 9.21: Displaying the Resource Editor for the Default.resx File

3. Add another new item `Default.ar.resx` for the Arabic language and set the values for the Name and Value fields in the Resource editor, as shown in Figure 9.22:

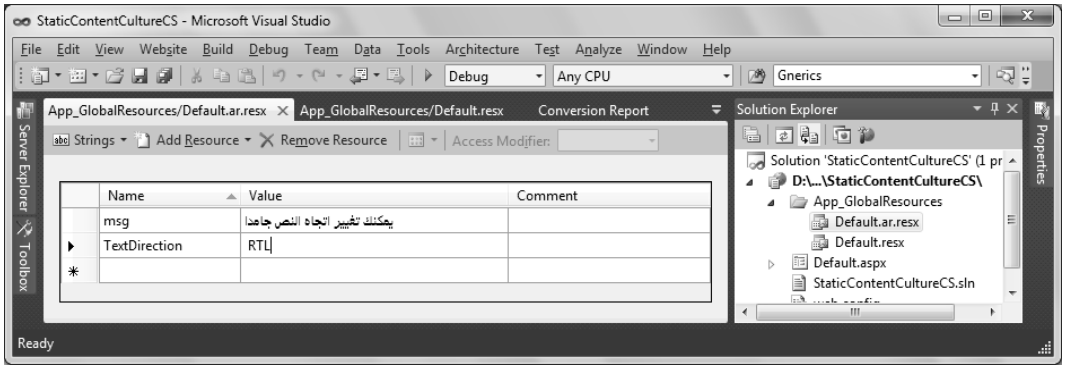

Figure 9.22: Displaying the Resource Editor for the Default.ar.resx File

4. Add a Label control in the Design view of the `Default.aspx` file and modify the code, as shown in Listing 9.10, in the `Default.aspx` page of the Web application to set the direction property of the static content:

Listing 9.10: Code for the `Default.aspx` Page

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
  Inherits="_Default"  Culture="auto" UICulture="auto"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html id="Html1" runat="server" dir="<%$ Resources:default,TextDirection %>"
  xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>StaticContentCulture Example</title>
</head>
<body id="Body1" runat="server" dir="<%$ Resources:default,TextDirection %>">
    <form id="form1" runat="server">
    <div>
    <strong>C# 2010 BLACK BOOK</strong><br />
        <br />
    <asp:Label ID="Label1" runat="server" Text="<%$ Resources:default, msg
  %>"></asp:Label>
    </div>
    </form>
</body>
</html>
```

The complete code for the resource file, `Default.ar.resx`, is shown in Listing 9.11:

Listing 9.11: Code for the `Default.ar.resx` File

```
<?xml version="1.0" encoding="utf-8"?>
<root>
  <!--
    Microsoft ResX Schema
    Version 2.0
    The primary goals of this format is to allow a simple XML format
    that is mostly human readable. The generation and parsing of the
    various data types are done through the TypeConverter classes
    associated with the data types.
    Example:
    ... ado.net/XML headers & schema ...
    <resheader name="resmimetype">text/microsoft-resx</resheader>
    <resheader name="version">2.0</resheader>
    <resheader name="reader">System.Resources.ResXResourceReader, System.Windows.Forms,
  ...</resheader>
    <resheader name="writer">System.Resources.ResXResourceWriter, System.Windows.Forms,
  ...</resheader>
    <data name="Name1"><value>this is my long string</value><comment>this is a
  comment</comment></data>
    <data name="Color1" type="System.Drawing.Color, System.Drawing">Blue</data>
    <data name="Bitmap1" mimetype="application/x-microsoft.net.object.binary.base64">
        <value>[base64 mime encoded serialized .NET Framework object]</value>
    </data>
    <data name="Icon1" type="System.Drawing.Icon, System.Drawing"
  mimetype="application/x-microsoft.net.object.bytearray.base64">
        <value>[base64 mime encoded string representing a byte array form of the .NET
  Framework object]</value>
        <comment>This is a comment</comment>
    </data>
    There are any number of "resheader" rows that contain simple
    name/value pairs.
    Each data row contains a name, and value. The row also contains a
    type or mimetype. Type corresponds to a .NET class that support
    text/value conversion through the TypeConverter architecture.
    Classes that don't support this are serialized and stored with the
    mimetype set.
    The mimetype is used for serialized objects, and tells the
    ResXResourceReader how to depersist the object. This is currently not
    extensible. For a given mimetype the value must be set accordingly:
    Note - application/x-microsoft.net.object.binary.base64 is the format
    that the ResXResourceWriter will generate, however the reader can
    read any of the formats listed below.
    mimetype: application/x-microsoft.net.object.binary.base64
    value   : The object must be serialized with
            : System.Runtime.Serialization.Formatters.Binary.BinaryFormatter
            : and then encoded with base64 encoding.
    mimetype: application/x-microsoft.net.object.soap.base64
    value   : The object must be serialized with
            : System.Runtime.Serialization.Formatters.Soap.SoapFormatter
            : and then encoded with base64 encoding.
    mimetype: application/x-microsoft.net.object.bytearray.base64
    value   : The object must be serialized into a byte array
            : using a System.ComponentModel.TypeConverter
            : and then encoded with base64 encoding.
    -->
  <xsd:schema id="root" xmlns="" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
    <xsd:import namespace="http://www.w3.org/XML/1998/namespace" />
    <xsd:element name="root" msdata:IsDataSet="true">
      <xsd:complexType>
        <xsd:choice maxOccurs="unbounded">
          <xsd:element name="metadata">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="value" type="xsd:string" minOccurs="0" />
```

```
                </xsd:sequence>
                <xsd:attribute name="name" use="required" type="xsd:string" />
                <xsd:attribute name="type" type="xsd:string" />
                <xsd:attribute name="mimetype" type="xsd:string" />
                <xsd:attribute ref="xml:space" />
              </xsd:complexType>
            </xsd:element>
            <xsd:element name="assembly">
              <xsd:complexType>
                <xsd:attribute name="alias" type="xsd:string" />
                <xsd:attribute name="name" type="xsd:string" />
              </xsd:complexType>
            </xsd:element>
            <xsd:element name="data">
              <xsd:complexType>
                <xsd:sequence>
                  <xsd:element name="value" type="xsd:string" minOccurs="0"
                   msdata:Ordinal="1" />
                  <xsd:element name="comment" type="xsd:string" minOccurs="0"
                   msdata:Ordinal="2" />
                </xsd:sequence>
                <xsd:attribute name="name" type="xsd:string" use="required"
                   msdata:Ordinal="1" />
                <xsd:attribute name="type" type="xsd:string" msdata:Ordinal="3" />
                <xsd:attribute name="mimetype" type="xsd:string" msdata:Ordinal="4" />
                <xsd:attribute ref="xml:space" />
              </xsd:complexType>
            </xsd:element>
            <xsd:element name="resheader">
              <xsd:complexType>
                <xsd:sequence>
                  <xsd:element name="value" type="xsd:string" minOccurs="0"
                   msdata:Ordinal="1" />
                </xsd:sequence>
                <xsd:attribute name="name" type="xsd:string" use="required" />
              </xsd:complexType>
            </xsd:element>
          </xsd:choice>
        </xsd:complexType>
      </xsd:element>
    </xsd:schema>
    <resheader name="resmimetype">
      <value>text/microsoft-resx</value>
    </resheader>
    <resheader name="version">
      <value>2.0</value>
    </resheader>
    <resheader name="reader">
      <value>System.Resources.ResXResourceReader, System.Windows.Forms, Version=2.0.0.0,
     Culture=neutral, PublicKeyToken=b77a5c561934e089</value>
    </resheader>
    <resheader name="writer">
      <value>System.Resources.ResXResourceWriter, System.Windows.Forms, Version=2.0.0.0,
     Culture=neutral, PublicKeyToken=b77a5c561934e089</value>
    </resheader>
    <data name="msg" xml:space="preserve">
      <value> يمكنك تغيير هذا النص المضاف </value>
    </data>
    <data name="TextDirection" xml:space="preserve">
      <value>RTL</value>
    </data>
  </root>
```

The complete code for the resource file, `Default.resx`, is shown in Listing 9.12:

Listing 9.12: Code for the `Default.resx` File

```
<?xml version="1.0" encoding="utf-8"?>
<root>
  <!--
    Microsoft ResX Schema
    Version 2.0
    The primary goals of this format is to allow a simple XML format
    that is mostly human readable. The generation and parsing of the
    various data types are done through the TypeConverter classes
    associated with the data types.
    Example:
    ... ado.net/XML headers & schema ...
    <resheader name="resmimetype">text/microsoft-resx</resheader>
    <resheader name="version">2.0</resheader>
    <resheader name="reader">System.Resources.ResXResourceReader, System.Windows.Forms,
  ...</resheader>
    <resheader name="writer">System.Resources.ResXResourceWriter, System.Windows.Forms,
  ...</resheader>
    <data name="Name1"><value>this is my long string</value><comment>this is a
  comment</comment></data>
    <data name="Color1" type="System.Drawing.Color, System.Drawing">Blue</data>
    <data name="Bitmap1" mimetype="application/x-microsoft.net.object.binary.base64">
        <value>[base64 mime encoded serialized .NET Framework object]</value>
    </data>
    <data name="Icon1" type="System.Drawing.Icon, System.Drawing"
  mimetype="application/x-microsoft.net.object.bytearray.base64">
        <value>[base64 mime encoded string representing a byte array form of the .NET
  Framework object]</value>
        <comment>This is a comment</comment>
    </data>
    There are any number of "resheader" rows that contain simple
    name/value pairs.
    Each data row contains a name, and value. The row also contains a
    type or mimetype. Type corresponds to a .NET class that support
    text/value conversion through the TypeConverter architecture.
    Classes that don't support this are serialized and stored with the
    mimetype set.
    The mimetype is used for serialized objects, and tells the
    ResXResourceReader how to depersist the object. This is currently not
    extensible. For a given mimetype the value must be set accordingly:
    Note - application/x-microsoft.net.object.binary.base64 is the format
    that the ResXResourceWriter will generate, however the reader can
    read any of the formats listed below.
    mimetype: application/x-microsoft.net.object.binary.base64
    value   : The object must be serialized with
            : System.Runtime.Serialization.Formatters.Binary.BinaryFormatter
            : and then encoded with base64 encoding.
    mimetype: application/x-microsoft.net.object.soap.base64
    value   : The object must be serialized with
            : System.Runtime.Serialization.Formatters.Soap.SoapFormatter
            : and then encoded with base64 encoding.
    mimetype: application/x-microsoft.net.object.bytearray.base64
    value   : The object must be serialized into a byte array
            : using a System.ComponentModel.TypeConverter
            : and then encoded with base64 encoding.
    -->
  <xsd:schema id="root" xmlns="" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
    <xsd:import namespace="http://www.w3.org/XML/1998/namespace" />
    <xsd:element name="root" msdata:IsDataSet="true">
      <xsd:complexType>
        <xsd:choice maxOccurs="unbounded">
          <xsd:element name="metadata">
            <xsd:complexType>
              <xsd:sequence>
                <xsd:element name="value" type="xsd:string" minOccurs="0" />
```

```
                </xsd:sequence>
                <xsd:attribute name="name" use="required" type="xsd:string" />
                <xsd:attribute name="type" type="xsd:string" />
                <xsd:attribute name="mimetype" type="xsd:string" />
                <xsd:attribute ref="xml:space" />
              </xsd:complexType>
            </xsd:element>
            <xsd:element name="assembly">
              <xsd:complexType>
                <xsd:attribute name="alias" type="xsd:string" />
                <xsd:attribute name="name" type="xsd:string" />
              </xsd:complexType>
            </xsd:element>
            <xsd:element name="data">
              <xsd:complexType>
                <xsd:sequence>
                  <xsd:element name="value" type="xsd:string" minOccurs="0"
                   msdata:Ordinal="1" />
                  <xsd:element name="comment" type="xsd:string" minOccurs="0"
                   msdata:Ordinal="2" />
                </xsd:sequence>
                <xsd:attribute name="name" type="xsd:string" use="required"
                   msdata:Ordinal="1" />
                <xsd:attribute name="type" type="xsd:string" msdata:Ordinal="3" />
                <xsd:attribute name="mimetype" type="xsd:string" msdata:Ordinal="4" />
                <xsd:attribute ref="xml:space" />
              </xsd:complexType>
            </xsd:element>
            <xsd:element name="resheader">
              <xsd:complexType>
                <xsd:sequence>
                  <xsd:element name="value" type="xsd:string" minOccurs="0"
                   msdata:Ordinal="1" />
                </xsd:sequence>
                <xsd:attribute name="name" type="xsd:string" use="required" />
              </xsd:complexType>
            </xsd:element>
          </xsd:choice>
        </xsd:complexType>
      </xsd:element>
    </xsd:schema>
    <resheader name="resmimetype">
      <value>text/microsoft-resx</value>
    </resheader>
    <resheader name="version">
      <value>2.0</value>
    </resheader>
    <resheader name="reader">
      <value>System.Resources.ResXResourceReader, System.Windows.Forms, Version=2.0.0.0,
     Culture=neutral, PublicKeyToken=b77a5c561934e089</value>
    </resheader>
    <resheader name="writer">
      <value>System.Resources.ResXResourceWriter, System.Windows.Forms, Version=2.0.0.0,
     Culture=neutral, PublicKeyToken=b77a5c561934e089</value>
    </resheader>
    <data name="msg" xml:space="preserve">
      <value>You can chagne the direction of the static text</value>
    </data>
    <data name="TextDirection" xml:space="preserve">
      <value>LTR</value>
    </data>
  </root>
```

5. Run the Web application by pressing the F5 key. The output of the `StaticContentCultureCS` Web application, when the preferred language in the browser settings is changed to English, is shown in Figure 9.23:

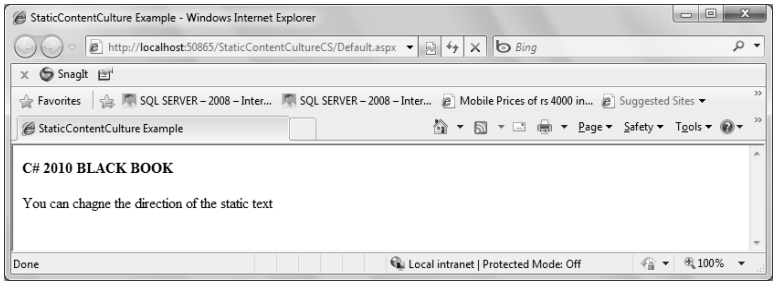

Figure 9.23: Showing the Output in English Language

The output of the Web application, when the preferred language in browser settings is changed to Arabic, is shown in Figure 9.24:

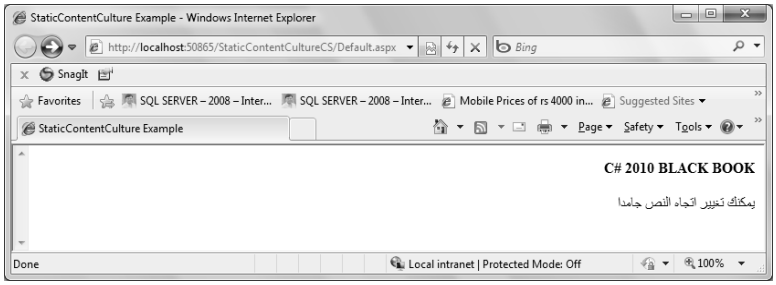

Figure 9.24: Showing the Output in Arabic Language

Localizing Script Files

You can localize your script files with the help of the `ScriptManager` control. The `ScriptManager` control enables certain automatic behaviors for localized applications, which include the following:

- Automatically locating script files based on settings and naming conventions
- Enabling the definition of cultures, including custom cultures.
- Caching scripts to efficiently manage many requests.

Now, let's create a Web application named `ScriptLocalizationCS` (also available in the CD-ROM) and follow these steps to localize the script files:

1. Add a Class Library project named `LocalizingResources` by right-clicking the name of the Solution (`ScriptLocalizationCS`) in Solution Explorer, and select `Add→New Project`, as shown in Figure 9.25:

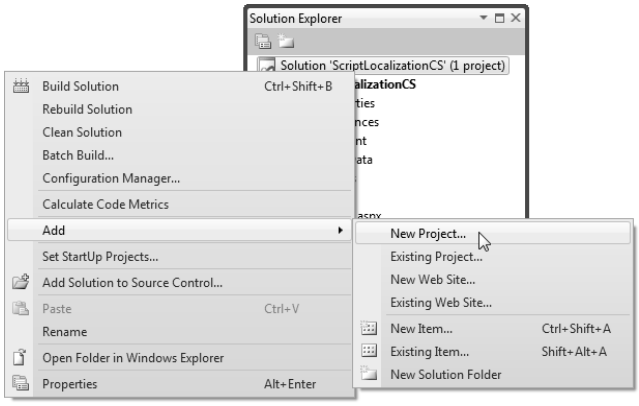

Figure 9.25: Adding New Project in the ScriptLocalizationCS Application

The New Project option opens the Add New Project dialog box (Figure 9.26).

2. Select the Class Library template and change the name of the Class Library project as `LocalizingResources`, as shown in Figure 9.26:

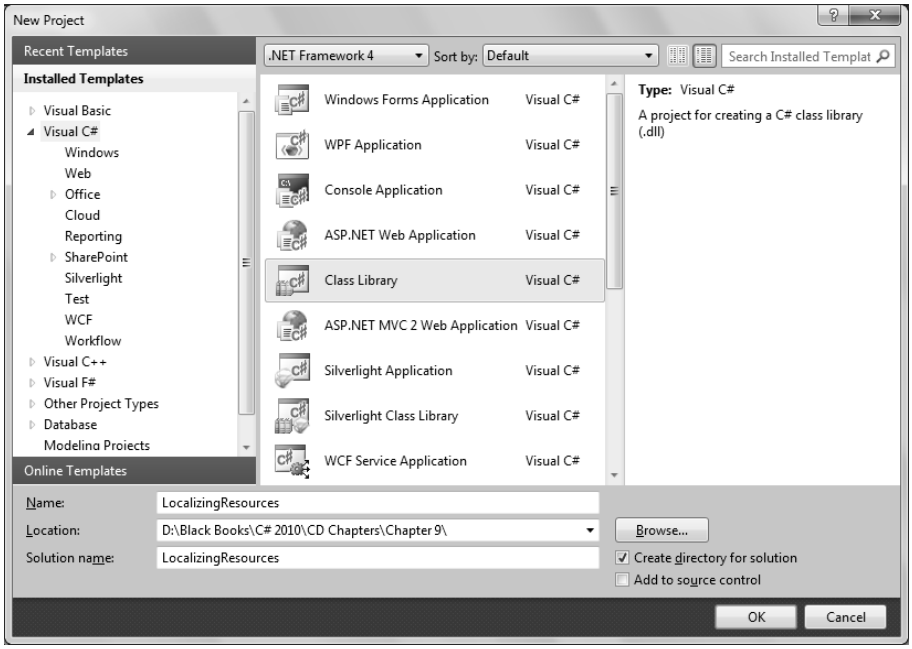

Figure 9.26: Displaying the Add New Project Dialog Box

3. Click the OK button. It adds the `LocalizingResources` application to Solution Explorer.
4. Add the Jscript file named `CreateScript.js` and change the value of the `Build Action` property of the script file to `Embedded Resource`.
5. Add the following code in the `CreateScript.js` file:

```
function CreateScript(fileName)
{
  alert (Message.FileCreated.replace(/FILENAME/, fileName));
}
```

6. Add two resource files in the `LocalizingResources` application as `CreateResource.resx` for English language and `CreateResource.ru.resx` for Russian language and add the following resource string to the `CreateResource.resx` file:

   - **FileCreated**—Indicates that the file FILENAME has been created

   Also add the following resource string to the `CreateResource.ru.resx` file:

   - **FileCreated**— Указывает, что файл FILENAME была создана

7. Add the following lines of code in the `AssemblyInfo` file of the `LocalizingResources` application:

```
 [assembly: System.Web.UI.WebResource("LocalizingResources.CreateScript.js",
   "text/javascript")]
[assembly: System.Web.UI.ScriptResource("LocalizingResources.CreateScript.js",
   "LocalizingResources.CreateResource", "Message")]
```

8. Add references of the `System.Web` and `System.Web.Extensions` assemblies in the `LocalizingResources` project. You also need to add the reference of the `LocalizingResources` assembly in the `ScriptLocalizationCS` project. You can see the `LocalizingResources` assembly in the Projects tab of the Add Reference dialog box, as shown in Figure 9.27:

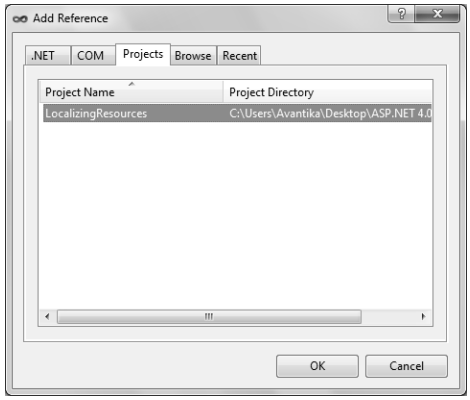

Figure 9.27: Displaying the Add Reference Dialog Box

9. Open the `Default.aspx` page of the `ScriptLocalizationCS` application and modify the code shown in Listing 9.13:

Listing 9.13: Code for the `Default.aspx` Page

```
<%@ Page Title="Home Page" Language="C#" MasterPageFile="~/Site.master"
  AutoEventWireup="true"
    CodeBehind="Default.aspx.cs" Inherits="ScriptLocalizationCS._Default" Culture="auto"
  UICulture="auto" %>
<asp:Content ID="HeaderContent" runat="server" ContentPlaceHolderID="HeadContent">
</asp:Content>
<asp:Content ID="BodyContent" runat="server" ContentPlaceHolderID="MainContent">
    <h2>
        C# 2010 black book</h2>
    <br />
    <asp:ScriptManager ID= "SM1" runat="server" EnableScriptLocalization="true">
    <Scripts>
    <asp:ScriptReference Assembly="LocalizingResources"
  Name="LocalizingResources.CreateScript.js" />
    </Scripts>
    </asp:ScriptManager>
    <br />
    <br />
    <asp:Button ID="button" runat="server"
        OnClientClick="CreateScript('Hello World.txt');" Text="Create" Width="92px"/>
</asp:Content>
```

10. Press the F5 key to run the Web application. Notice that the Create button is displayed in the browser (Figure 9.28).
11. Click the Create button. A message box appears, as shown in Figure 9.28:

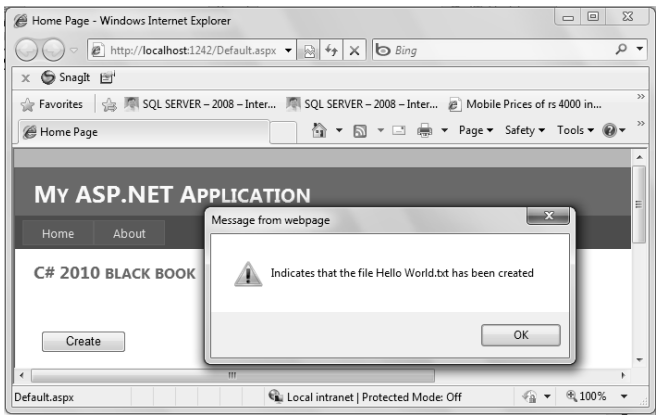

Figure 9.28: Displaying the Confirmation Message Box

In Figure 9.28, the preferred language is English.

12. Change the preferred language to Russian. The output of the `ScriptLocalizationCS` application after changing the preferred language is shown in Figure 9.29:

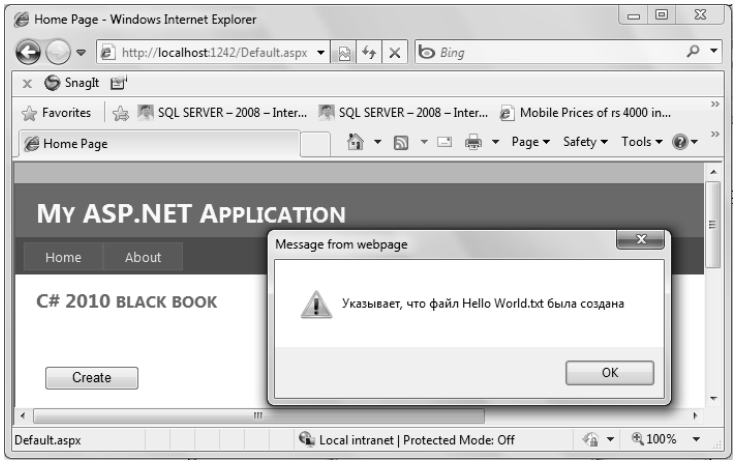

Figure 9.29: Displaying the Confirmation Message Box after Selecting Russian as the Preferred Language

Figure 9.29 displays a message box with a message in Russian language.

Summary

This chapter has discussed the concept of globalization in ASP.NET, which includes i**nternationalization** and localization of Web applications. You also learned to create local and global resources. Next, you learned about implicit and explicit expressions and how to implement localization in Web applications. Towards the end, you have learned how to localize the script files in a Web application.

The practical implementation of globalization and localization is discussed in the Immediate Solutions section of this chapter.

In the next chapter, you learn to Cryptography in .NET 4.0.