Session: 12

Globalization

Objectives

- Define and describe how to implement internationalization
- Define and describe how to implement localization

Implementing Internationalization

- At recent times, most of the organizations do business in the global marketplace.
- So, these types of organizations must design applications to accommodate users from a wide variety of cultures.
- Therefore, you need to enable your Web applications to represent information in the users' native language and formats.
- This functionality can be provided by implementing internationalization.
- The ASP.NET MVC Framework allows you to develop multicultural applications.
- You can develop an ASP.NET MVC application to retrieve information about a particular culture and region to which a user belongs.
- Based on this information, you can format and present your application data in the desired language.

Culture Code 1-2

- To implement internationalization in an application, you need to understand about the culture code that the .NET Framework uses.
- The .NET Framework uses the following syntax to represent a culture code:

```
<languagecode>-<country/regioncode>
```

where,

  - **<languagecode>**: Is a lowercase two-letter code for a language.
  - **<country/regioncode>**: Is an uppercase two-letter code for a country or a region.

- In an ASP.NET MVC application, you can access all the supported culture of the .NET Framework using the CultureInfo class of the System.Globalization namespace.

Culture Code 2-2

- Following code snippet shows using the CultureInfo class:

Snippet

```
ArrayList cultureNameList = new ArrayList();
CultureInfo[] cInfo =
CultureInfo.GetCultures(CultureTypes.AllCultures);
    foreach (CultureInfo cultures in cInfo)
      {
         cultureNameList.Add(String.Format("Culture Name {0} :
         Display Name {1}",cultures.Name,cultures.DisplayName));
      }
```

- This code:
  - First, calls the CultureInfo.GetCultures() method passing a CultureTypes.AllCultures enumeration member.
  - Next, it accesses the CultureInfo.Name and CultureInfo.DisplayName properties of all the supported cultures and adds the information as a formatted string to an ArrayList.

Setting Cultures 1-4

- Before you configure internationalization in an ASP.NET MVC application, you need to understand about the following properties of the Page class:
  - The Culture property is a string that determines the results of culture-dependent functions, such as formatting of date and time and currency.
  - The UICulture property is a string that determines the resource file that contains locale specific content for a page.
- While configuring internationalization in an application, you might often need to consider assigning different values for these properties.
- In an ASP.NET MVC, you can set and access the current culture and UI culture of a page from the request processing thread by using the following properties:
  - Thread.CurrentThread.CurrentCulture
  - Thread.CurrentThread.CurrentUICulture

Setting Cultures 2-4

- The .NET Framework examines both these properties before rendering culture-dependent functions.
- Following code snippet shows explicitly setting the CurrentCulture property of the thread executing the Index() action method of the HomeController:

Snippet

```
public class HomeController : Controller  {
  public ActionResult Index()  {
    Thread.CurrentThread.CurrentCulture = new
System.Globalization.CultureInfo("fr-FR");
        ViewBag.CultureName =
System.Globalization.CultureInfo.CurrentCulture.Name;
        ViewBag.Message = DateTime.Now.ToLongDateString();
         return View();
        }
  }
```

- This code:
  - Sets the CurrentCulture property to a CultureInfo object initialized with the fr-FR culture name.
  - Then, adds the name of the current culture and the current date to a ViewBag object before it returns the view.

Setting Cultures 3-4

- In the corresponding Index.cshtml view, you can access and display the ViewBag messages.
- Following code snippet shows how to access and display the ViewBag messages:

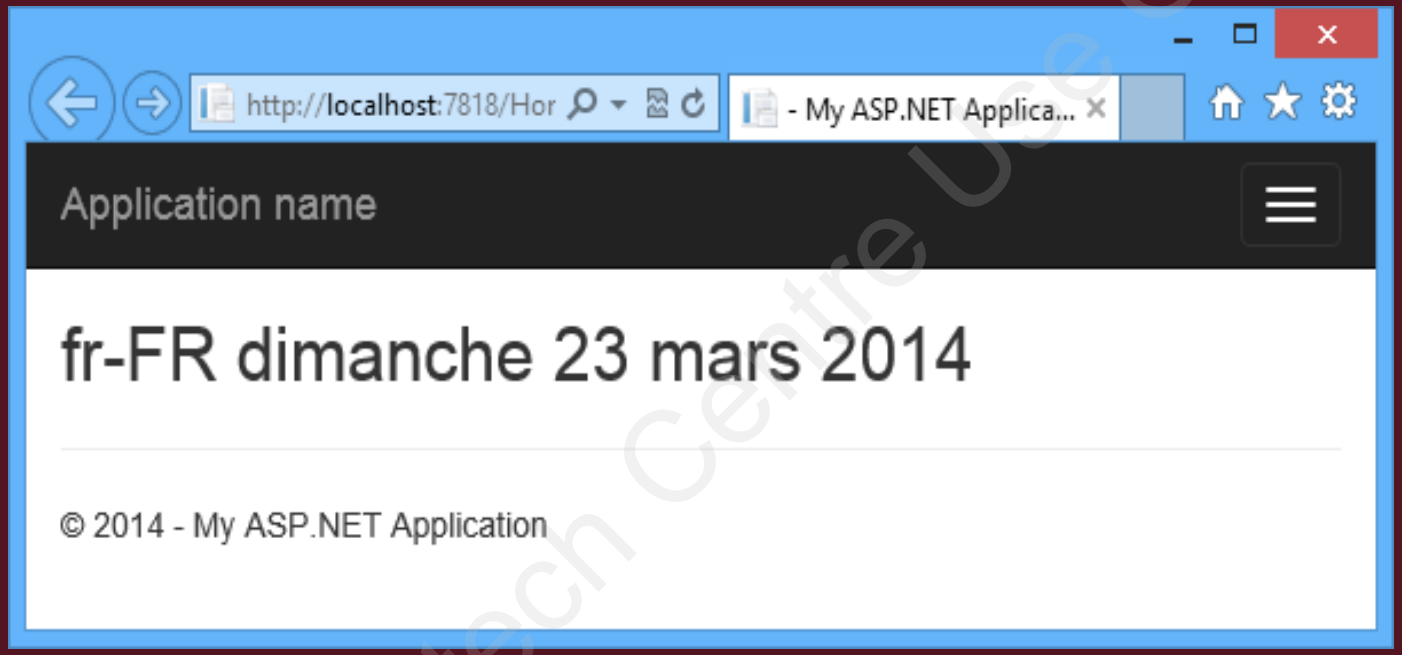
Snippet

```
<h2>@ViewBag.CultureName @ViewBag.Message</h2>
```

- When the Index.cshtml view is accessed in a browser, it displays the fr-FR as the culture name and the date in French.

Setting Cultures 4-4

- Following figure shows the Index.cshtml view in French:



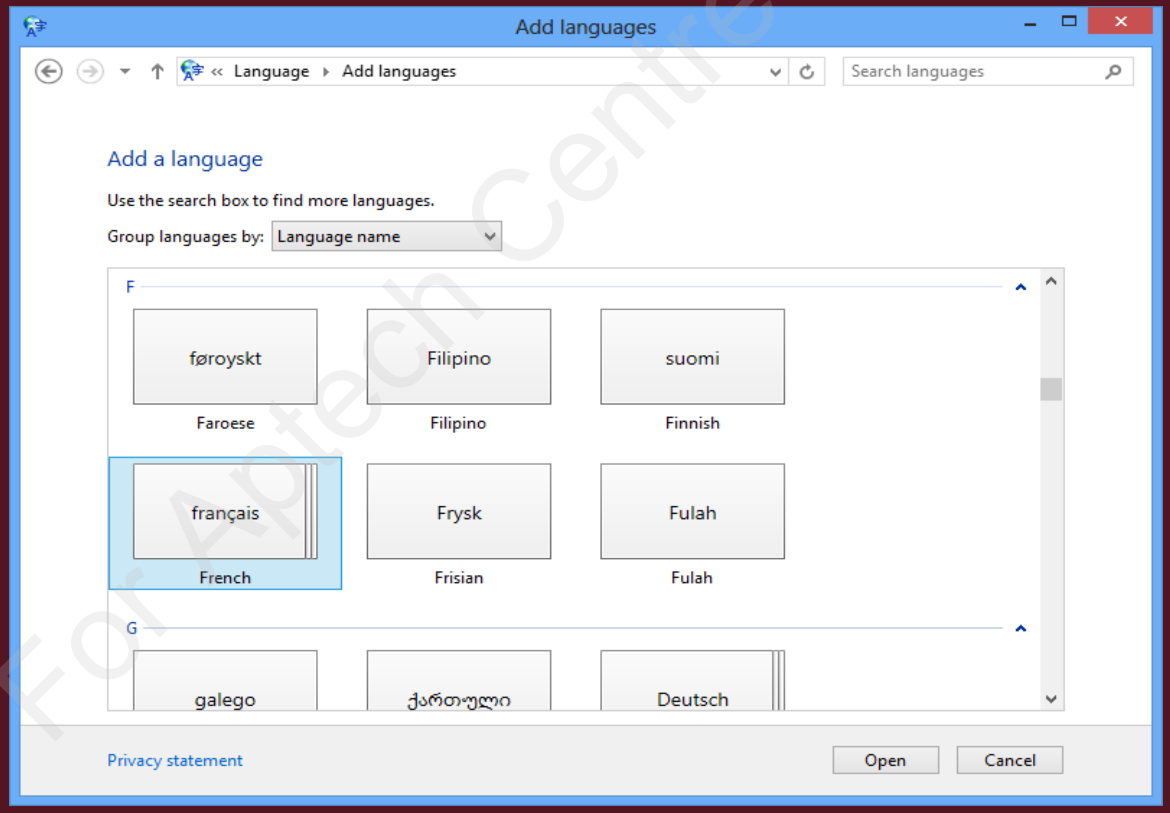
Using the Web.config File 1-6

- In an ASP.NET MVC application, you can use the Web.config file to use the language that is set as the default language in the browser that sends requests to the application.
- For that, you need to know how to set the preferred language of the browser.
- Then, you need to know how to configure the Web.config file to use the preferred language.
- To set the preferred language in a browser, you need to perform the following steps:
  - Open **Internet Explorer (IE)**.
  - Press the **Alt+X** keys to open the Tools menu.
  - Click **Internet Options** in the Tools menu. The **Internet Options** dialog box is displayed.
  - Click **Languages** in the Internet Options dialog box. The **Language Preference** dialog box is displayed.

Using the Web.config File 2-6

- Click **Set Language Preferences** in the Language Preference dialog box. The **Add languages** window is displayed.
- Locate and select French in the Add Languages window.

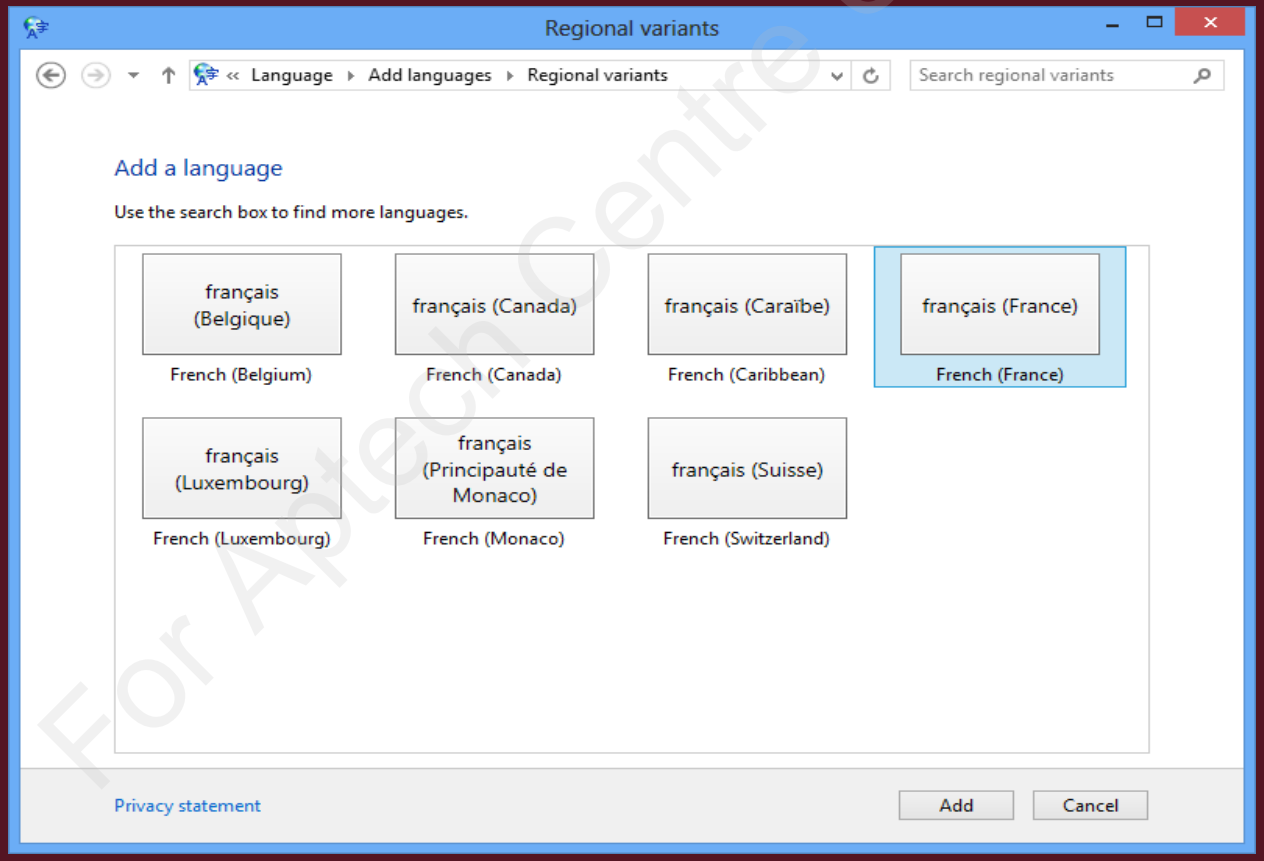
- Following figure shows selecting French in the Add languages window:



Using the Web.config File 3-6

- Click the **Open** button. The **Regional variants** dialog box is displayed.
- Select French (France) in the Regional variants dialog box.

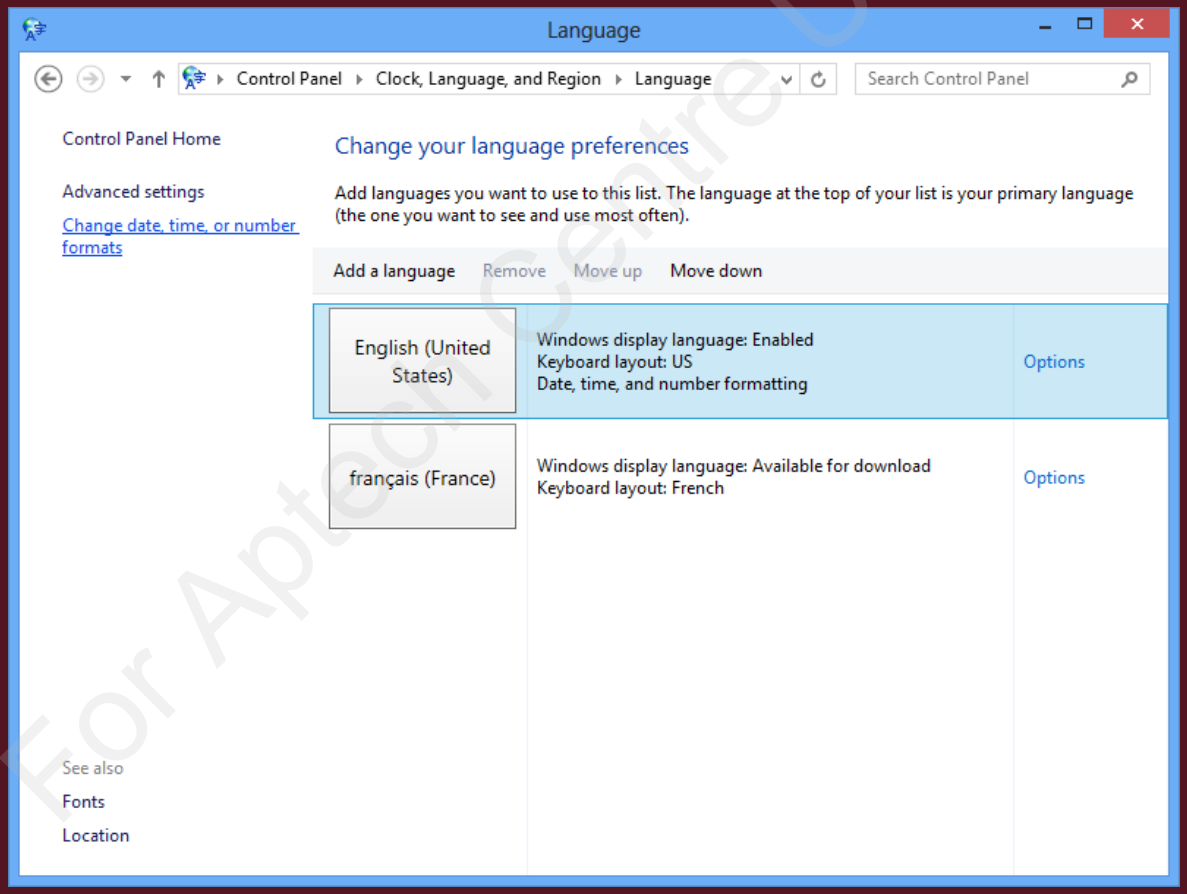
- Following figure shows selecting French (France) in the Regional variants window:



Using the Web.config File 4-6

- Click the **Add** button. The **Language** window displays the newly added language.

- Following figure shows the newly added francais (France) language:



Using the Web.config File 5-6

- Select **francais (France)** and click the Move up button to make French (France) as the first preferred language of the browser.
- The Language window displays francais (France) at the top of the language list.
- Close the Language window.
- Close the Internet Options dialog box.
- Click **OK** in the Language Preferences dialog box.
- Click **OK** in the Internet Options dialog box

- Once you set the preferred language of the browser, you can configure the Web.config file to use the browser preferred language by adding the <globalization> element under the <system.web> element.

Using the Web.config File 6-6

- Following code snippet shows using the <globalization> element:

Snippet

```
<system.web>
    <globalization culture="auto" uiCulture="auto"
       enableClientBasedCulture="true"/>
. . . .
. . . .
<system.web>
```

- In this code, the true value of the enableClientBasedCulture attribute instructs ASP.NET to set the UICulture and Culture properties for a Web page, based on the preferred languages set in the browser sending the request.

Implementing Localization

- You can implement localization in your application to display content in languages that a user prefers.
- For that, you need to separate the language-related and the culture-related content from the application code.
- The two approaches to implement localization are as follows:
  - Using Resource files
  - Using Separate views

Resource Files 1-7

- When you use resource files, you must first create a base resource file for the application, for example, MyResources.resx.
- Next, you must create separate resource files for each culture that your application supports.
- These resource files will have the name, as shown in the following syntax:

```
<base_resource-filename>.<language_code>-<country_code>.resx
```

where,

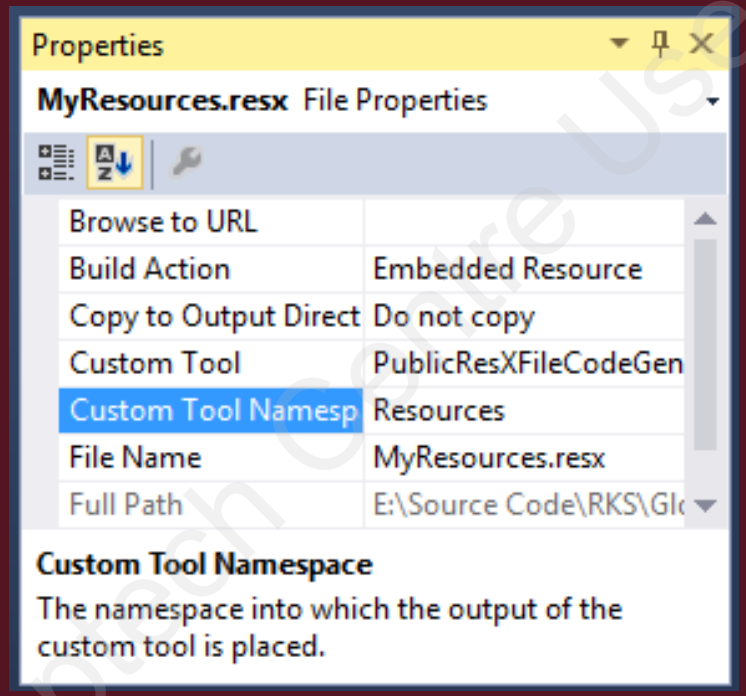
  - **base_resource-filename**: Is the name of the base resource file, such as MyResources.
  - **language_code**: Is the language code of the resource file, such as 'fr' for the French language.
  - **country_code**: Is the country code of the resource file, such as 'FR' for France.

Resource Files 2-7

- In Visual Studio 2013, you can create resource files by performing the following tasks:
  - Create a **Resources** folder in your application.
  - Right-click the **Resources** folder and select **Add→New Item**.
  - Select **General** from the Installed templates and select **Resources File**.
  - Replace the name given in the Name text box with **MyResources.resx**.
  - Click the **Add** button. The Resource Editor displays the MyResources.resx file.
  - Select **Public** from the Access Modifier drop-down list.
  - Select **MyResources.resx** file in the Solution Explorer window. The Properties window displays the properties of the MyResources.resx file.
  - Type **Resources** for the Custom Tool Namespace property.

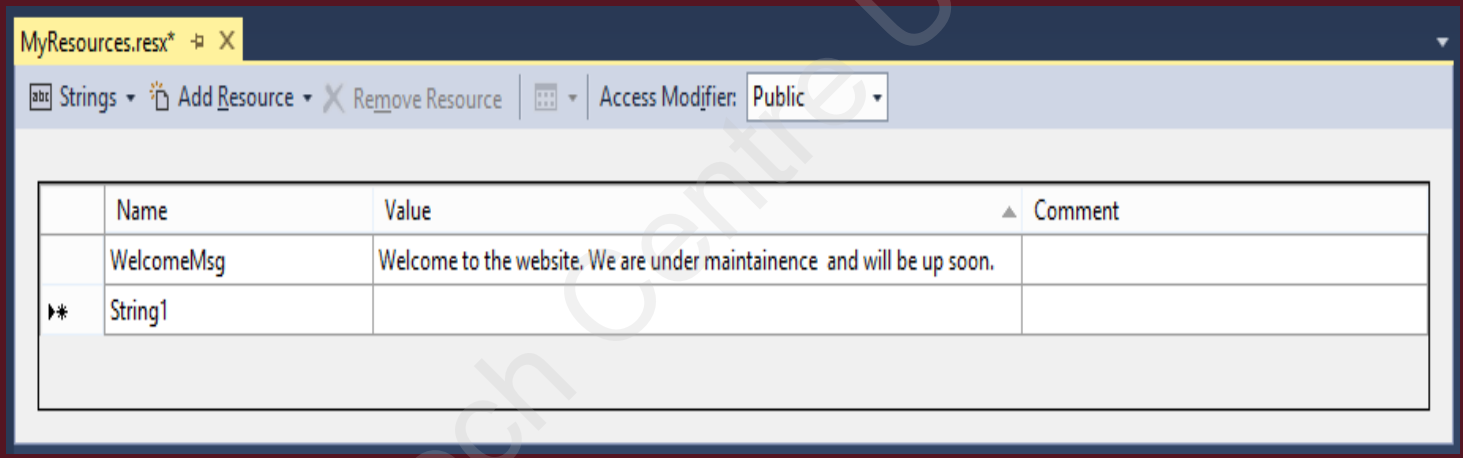
Resource Files 3-7

- Following figure shows specifying values for the Custom Tool Namespace property:



Resource Files 4-7

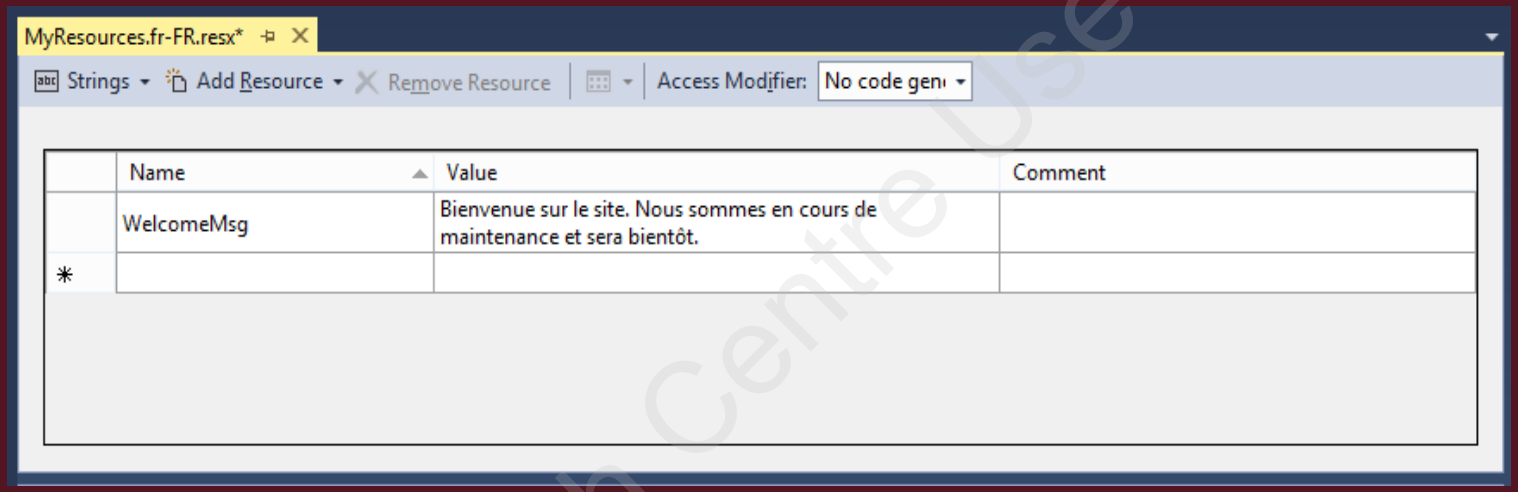
- Add the Name-Value pairs in the Resource Editor.
- Following figure shows specifying Name-Value pairs in the Resource Editor:



- Repeat steps to create a MyResources.fr-FR.resx file in the Resources folder. Then, add the Name-Value pairs of the MyResources.fr-FR.resx file in the Name and Value columns of the Resource Editor.

Resource Files 5-7

- Following figure shows specifying the Name-Value pairs of the MyResources.fr-FR.resx file:



- In the Index.cshtml page of the HomeController, you can access the value having the WelcomeMsg key that you have specified in the resource files.

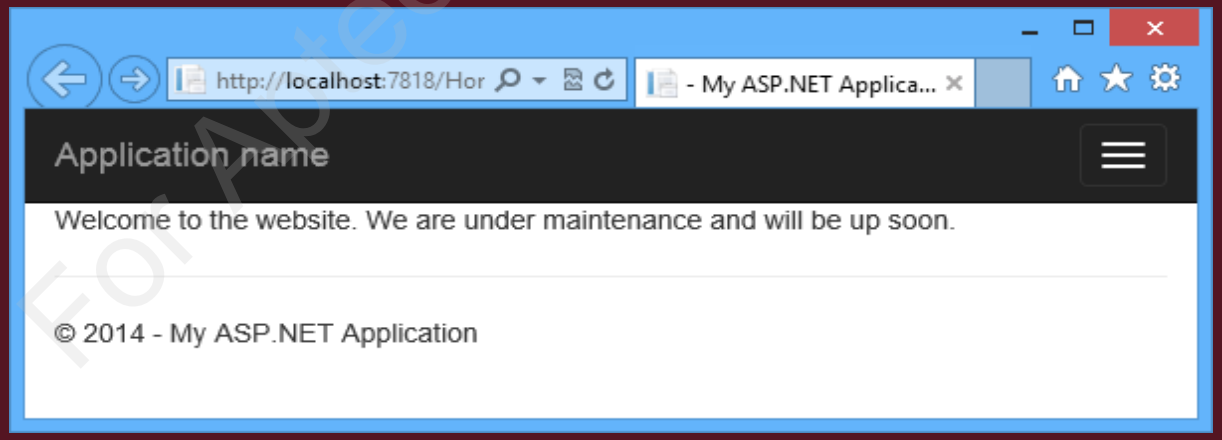
Resource Files 6-7

- Following code snippet shows how to access the value having the WelcomeMsg key:

Snippet

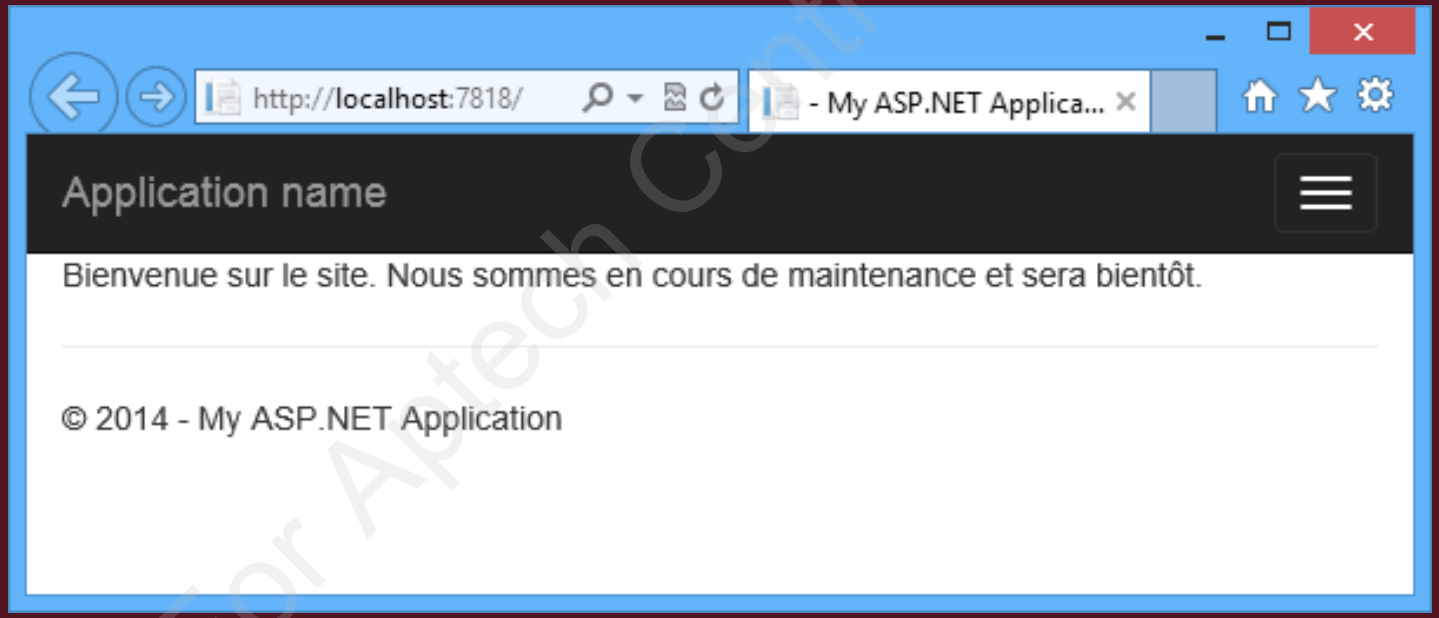
```
<p>@Resources.MyResources.WelcomeMsg</p>
```

- A browser whose preferred language is not set to fr-FR will render the view using content from the default resource file, named MyResources.resx file.
- Following figure shows the view using the content from the default resource file:



Resource Files 7-7

- A browser whose preferred language is set to fr-FR will render the view using content from the resource file, named MyResources.fr-FR.resx file.
- Following figure shows the view using the content from MyResources.fr-FR.resx file:



Using Separate Views 1-2

- In an application, you can use separate views each for the supported culture and hard code the localized text in them.
- This will not only result in clean and readable views, but also enable you to control how to position localized text elements in the view.
- You can use the approach of using separate views by creating different culture-based views for a particular view that needs to be localized.
- After creating the views, you can make conditional checks in the controller action to obtain the browser preferred language and accordingly return the corresponding view.

Using Separate Views 2-2

- Following code snippet shows the Index() action method of a HomeController, that returns different views based on the preferred language of the browser:

Snippet

```
public ActionResult Index()
    {
       if(HttpContext.Request.UserLanguages[0]=="fr-FR")
       return View("Index.fr-FR");
       else if (HttpContext.Request.UserLanguages[0] == "en-US")
           return View("Index.en-US");
       else
           return View();
   }
```

- This code uses an if-else statement to check the browser preferred language and accordingly returns a corresponding view. If the application does not have a view for a specific language, the action returns the default view.

Summary

- Globalization is a process of designing and developing applications that can be used for multiple cultures.
- In an ASP.NET MVC application, you can access all the supported culture of the .NET Framework using the CultureInfo class.
- In an ASP.NET MVC application, you can set and access the current culture and UI culture of a page from the request processing thread.
- In an ASP.NET MVC application, you can use the Web.config file to use the language that is set as the default language in the browser that sends requests to the application.
- You can implement localization in your application to display content in languages that a user prefers.
- You can use two approaches to implement localization in your application, such as using resource files and using separate views.
- You can use separate views by creating different culture-based views for a particular view that needs to be localized.