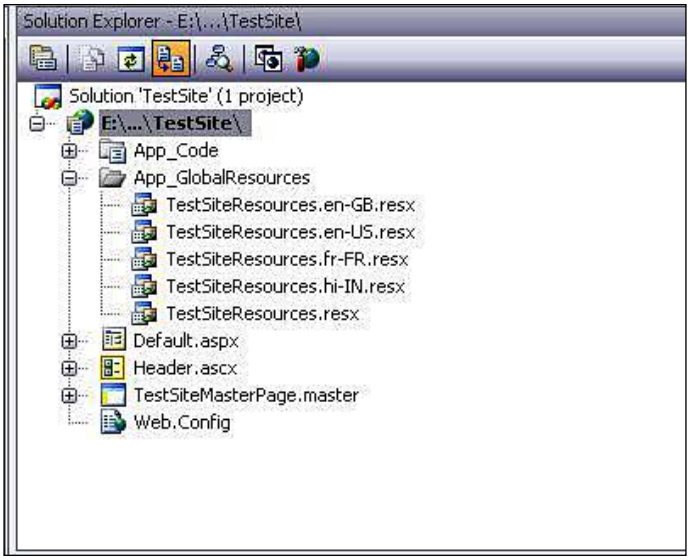
- **Programmatically accessing strongly-typed resource classes**: This can be achieved by using the following code:

```
lblWelcome.Text = Resources.TestSiteResources.Welcome;
```

  This will work, but it needs to be coded for every control on the page. Therefore, it is better to use the second method (Explicit localization) for all of the controls, and use this method to access resource strings for other content, if needed. Also, note that controls such as the `Calendar` control have in-built localization. As soon as the `UICulture` and `Culture` of the current thread changes, it shows localized content by itself, thanks to ASP.NET!

Incorporating Globalization

In our sample website, after creating the resource files and creating some localized data, we first start using the explicit localization to set the text of the controls such as the `Labels` in our web site so that they get their values from the resource files. Because in our example there are four languages, let us create four resource files, along with a fifth fallback resource file (with no locale name). Here is a screenshot of these files:



Note that the resource files have the locale as their middle names, so we need to set the `UICulture` to the same named locale so that ASP.NET can access these resource files.

But the problem is, how do we change the culture dynamically on the postback event? Fortunately, ASP.NET provides a method in the `Page` class to override `InitializeCulture()`. This method executes very early in the page life cycle (much before any control is generated), and here we can set the `UICulture` and `Culture` of the current thread.

Because this method is in the `Page` class and we do not want to repeat the same code for each web page, let us create a `BasePage` class, and all of the ASPX pages in our application will be derived from this `BasePage` class. Before moving ahead with the code for this method, let us understand how we can set the culture and maintain its state throughout postbacks.

Setting the Culture of the Thread Based on User Selection

Going back to the UI design we created earlier, we had a `MasterPage` and a `Header` user control in it (inside a `ContentPlaceHolder`). We had a default page associated with that `MasterPage`. The entire site had to be localized dynamically. So in the header control, there was a drop-down from where the user could select a language and culture. In the `BasePage`'s `InitilializeCulture` method, we have to get the value of the item the user has selected from the drop-down. But, we have a problem here.

The `InitializeCulture()` method is fired very early in the page life cycle. At that stage, none of the controls on the page have been initialized. So we cannot access any control directly like this:

```
String userLanguage = ddlLanguage.SelectedValue;
```

The only way to access the user selection from the drop-down control is to use the `Form` collection (from the `Response` object). Here is the code to do this:

```
public const string LanguageDropDownID =
        "ctl00$cphHeader$Header1$ddlLanguage";
public const string PostBackEventTarget = "__EVENTTARGET";
```

Here, we are using the "`parentControl$ChildControl`" syntax to access the control from the `Form` collection. You can access any nested control generated by ASP.NET by using this convention. Using this value of the selected item from the `Form` collection, we can set the culture in a `switch case` statement, as follows:

```
protected override void InitializeCulture()
{
    if(Request[PostBackEventTarget] != null)
    {
        string controlID = Request[PostBackEventTarget];
```