We need to save the `Culture` selected in a session or a cookie variable because if the user moves to some other page in the same application, the thread's culture information would be lost as the new `Page` class will instantiate from the beginning (HTTP is stateless!). Cookies can be used if you do not want to lose the current thread's `Culture` on the user's session expiry.

Once we have pulled out all localizable static content from the web application and moved it into the resource files, and set the `Culture` and `UICulture` based on the user's choice, we are ready with our globalization framework. Now, the only thing left is the addition of resource-specific data in the resource files. For each culture, we need to have a separate (and appropriately named) resource file. This process is localization.

In the `web.config` file, we have used the following properties:

```
<globalization responseEncoding"=utf-8" requestEncoding="utf-8"
                    fileEncoding="utf-8" />
```

Note the encoding attributes—`UTF-8` (8-bit Unicode Transformation Format) is used as it is variable-length character encoding and can represent languages such as Greek, Arabic, and so on, in addition to being ASCII compatible. For more information on `UTF-8` encoding, refer to link: `http://en.wikipedia.org/wiki/UTF-8`.

Also, an important point to be noted here is that although we can have the resource files in raw XML format on the deployment server (so that the user can edit them without re-compiling the entire site), the application will re-start if we make any modification to the resource files. This can hamper the performance of the deployed application.

## dir Attribute for Language Direction

Sometimes we may also need to set the direction of the localized text (which is set using the `dir` attribute of the `<html>` or the `<body>` tag). This is necessary because some languages are read from **right-to-left** (**RTL**) as is the case in Arabic for example, instead of the standard **left-to-right** (**LTR**) style used in Hindi and English. This can be achieved quite easily by setting the `dir` attribute to the appropriate value from the `.resx` file.

First, create a "direction" (you can use any name) field in all your resource files, setting its property to RTL or LTR based on the individual resource files. For Arabic, the value of this field would be RTL, and for Hindi it would be LTR. Then, set the same in the `dir` attribute of the `<body>` tag as:

```
<body runat="server" dir="<%$ Resources: TestSiteResources, Direction
%>">
```

This will set the correct text direction, as the value will come from the resource file based on the current thread's culture.

## Editing Resource Files after publishing in ASP.NET 3.5

An important point to note when globalizing your ASP.NET 3.5 web applications is the dynamic updating of the `.resx` files once published on the remote server. We would like to have this flexibility so that the users can modify the values in the XML resource files (`.resx` files) themselves once the application is deployed on a server, without using VS and republishing the files. The ability to do this depends on the project model we have followed for our web application in VS.

If we are using the **WebSite Project model** (which is the default in VS), then only the resource files under the `App_LocalResources` will get published as raw `.resx` files on the server, as they are not compiled. These resource files can be edited on the server (as they are compiled during runtime). Files under the `App_GlobalResources` folder are compiled into individual resource-specific DLLs and published on the server. So you cannot edit the resource files that are under `App_GlobalResources` once they have been published using the WebSite project model, as it has a default pre-compiled web deployment model. If you need to add new locale resources, then you can either do so in VS, and recompile and republish the application, or you can first manually generate the new resource file using the tool `resgen.exe` and then compile it to a satellite assembly using the **Assembly Linker tool.**

If we are using a **Web Application Project (WAP)** model, then the files under both the `App_GlobalResources` and `App_LocalResources` folders will get published as raw `.resx` files that are editable. So using WAP gives you this slight flexibility, which the default WebSite model does not provide.

Whenever we change any resource file (local or global) under the `/bin` folder of the deployed web application, an application restart will occur, which may cause the loss of data (such as values stored in session variables, and so on). This happens because ASP.NET caches the resource file contents in memory, and if the content has changed, the in-memory cache needs to be re-loaded from the updated content.