Now, before we move ahead, let me throw some light on culture and locale.

Culture and Locale

Languages also depend upon the geographical location. For example, French is spoken in France as well as Canada (besides many other countries). But linguistically speaking, Canadian French is quite different from French spoken in France. Similarly, there are linguistic differences between US English and British English. Therefore, the language needs to be associated with the particular region where it is spoken, and this is done by using **Locale** (which is the combination of language and location).

For example, `fr` is the code for French language. `fr-FR` means French language in France. So, `fr` specifies only the language, whereas `fr-FR` is the locale. Similarly, `fr-CA` defines another locale implying French language and culture in Canada. If we use only `fr`, it implies a neutral culture (that is, location neutral).

How do we Define or Change the Current Culture?

There are two properties of the `CultureInfo` class in the .NET FCL (Framework Class Library) that we can set using the overloaded constructor of the class, and then use the class to change the culture of the currently executing thread:

- `UICulture`: This property gets or sets the user interface culture for the currently-executing thread. It helps the runtime to load the resource strings from a specific resource file (which we will see later). This property can take neutral cultures as well as locales. For example:

  ```
  Thread.CurrentThread.CurrentUICulture = new CultureInfo("fr");
  ```

  Or,

  ```
  Thread.CurrentThread.CurrentUICulture = new CultureInfo("fr-CA");
  ```

- `Culture`: This property gets or sets the region-specific culture and formats of currency, dates etc. This needs a language as well as location (locale). For example:

  ```
  //correct as we have given locale
  Thread.CurrentThread.CurrentCulture = new CultureInfo("fr-A");
  //wrong, will not work
  Thread.CurrentThread.CurrentCulture = new CultureInfo("fr");
  ```

Sometimes we need a culture that does not belong to any language or locale, and which is not a variant of any region/language. For this, we have the `CultureInfo.InvariantCulture` property. This is used during the internal system processes that need to be culture independent, or to store data that does not need to be displayed directly to the end user.

Both `UICulture` and `Culture` properties can be defined in the `Web.Config` file under the `<GLOBALIZATION>` property. They can also be specified at the page level. But we don't want to hard-code these values, and would like to set them dynamically instead. As seen above, we could also get or set these values from the code using the `Thread.CurrentThread.CurrentCulture` and `Thread.CurrentThread.CurrentUICulture` properties. So, we will use these properties in our application.

Switching Locale

Coming back to our application, we need a way to switch the locale. There are two approaches to this:

- **Use the browser settings:** In IE, the user can change the culture by going to **Internet Options | General | Languages**. For this to work, we need to set both the `Culture` and the `UICulture` to `auto` and `enableClientBasedCulture = true` as:

  ```
  <GLOBALIZATION culture="auto" uiculture="auto"
   enableClientBasedCulture=""true"" />
  ```

- **User specified settings:** We can provide an option for the user to specify and change the culture and the language at runtime. This is the recommended approach, as sometimes the browser itself may not have the user-specific culture set (for example, a French tourist might be surfing the net in India)., Further, changing language settings via the browser may also sometimes be blocked.

Following the second recommended approach, I have created a section on the top (inside a `Panel` control) in the master page where I have a drop-down with these language options, which lets the users choose a particular locale. For illustration purposes, I have only four language options to choose from: Hindi, US English, British English, and French.

For my application to be globalized, it is required that whenever the user selects a locale from the language, the following should happen:

- All content should be localized. This means that all strings and text should be displayed in the chosen language and locale.
- Each control's caption or content should also show the text in the local language.
- Date and currency formatting should be done according to the chosen locale.
- All messages displayed to the user should be in the local language.

To achieve these goals, the first thing you need to do is to take out content from the code and put it in separate resource files, which are simple XML files with a `.resx` extension in .NET.