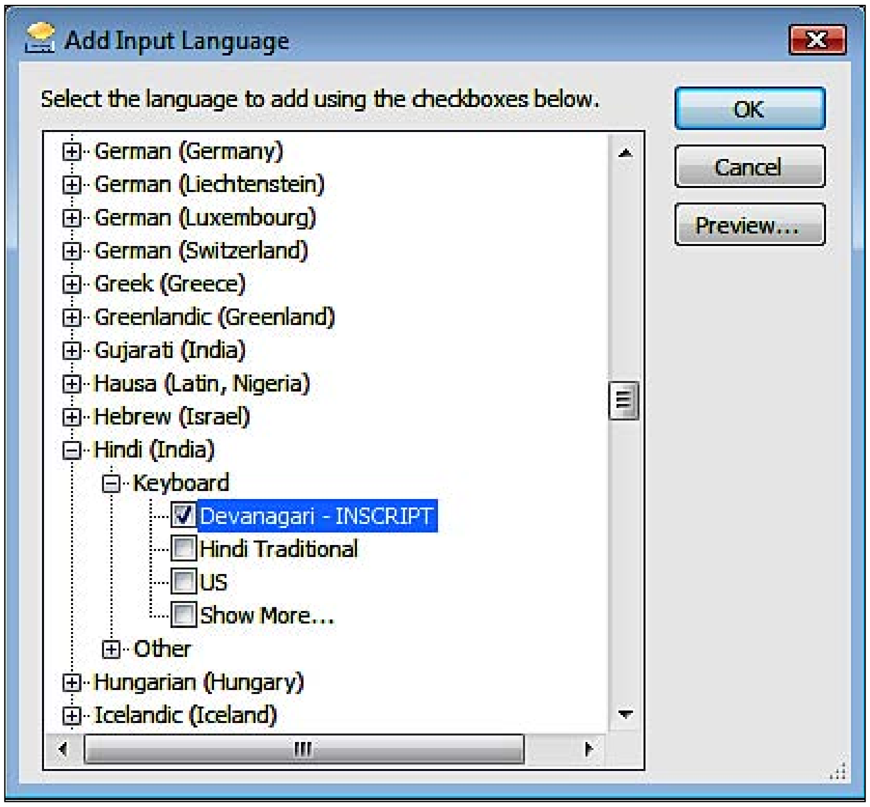
4. In the screenshot, the default language is English. To add additional languages, click the **Add** button and select the language you need (see the following screenshot):



5. This particular example will add the Hindi Language keyboard. Click the **OK** button to save your changes, and exit the **Control Panel**. You will see the language bar on the top right of your desktop, and you can click the language button and switch languages accordingly. You can use a Hindi keyboard to enter Hindi Unicode characters in the resource files.

In this way, you can switch to different languages and enter characters in your test web site to check the localization. There is an excellent article on how to use IME to enter Japanese, Korean, and Chinese language characters, available via the following link:

`http://www.microsoft.com/globaldev/handson/user/IME_Paper.mspx`

Using a Database for Localization

We have seen how to localize the text of the controls and the application content in the UI. But sometimes, the content is stored in a database, and this content also needs to be localized (for example, when using a Content Management System). Because this content is stored in a database, we cannot use resource files for this. Sometimes we may not be able to use resource files at all because we might want to avoid an application recompilation or restart when updating localized content (remember that updating a resource file under the `/bin` folder will cause an application recompilation or an application restart.).

Also, using a database gives us the flexibility to edit content easily as there are no resource files involved. We can create a simple edit form, so that content authors can edit and update the content without having any programming knowledge.

Custom Resource-Provider-Model in ASP.NET

When we use a database instead of `.resx` files to store the localized content in our application, we will need to make sure that we can access the resources using implicit as well as explicit localization, and get different localized content based on changing the current thread's locale. In short, we will need to create a similar framework to the one that ASP.NET provided us with when using `.resx` files.

Fortunately, in ASP.NET 3.5, this process is made easy by extending the **Resource-Provider-Model** and creating a custom database-based resource provider. Using the provider model will help us to leverage the ASP.NET infrastructure to manage localized resources from a database and use its API for performing the same tasks as for the XML based `.resx` files. Because the Resource-Provider-Model has a simple implementation approach, we will not study it in detail in this chapter. You can refer to the links given here for a detailed step-by-step example of how to extend the Resource-Provider-model to use a database instead of the `.resx` files:

`http://www.west-wind.com/presentations/wwDbResourceProvider/`