Robert L. Bogue
MS MVP, MCSE, MCSA:Security
317-844-5310
Rob.Bogue@ThorProjects.com

SharePoint Localization Tips and Tricks

Living in the United States I sometimes forget that most of the world doesn't speak English. Sure I occasionally run into someone who speaks Spanish, Japanese, or Mandarin Chinese as a primary language but that's more of a rarity than an everyday occurrence. As a result if you ask me the number of times that I've needed to build localized versions of my software the answer would be --- not very often. However, just because it's not something I run into every day doesn't mean that I shouldn't be more cognizant and aware of the right practices to support localized software.

In this article, I'll round up the scattered pieces of information about localization in SharePoint, provide some tips and tricks and leave you with a working approach to localization. However, before we work on meeting the challenge, we should understand it.

The Low Down on Localization

Software localization is the process of making software works for people who live in a different culture than the software was originally developed in. Localization breaks down into two broad areas: language and formatting. We'll be focusing on the language aspects of the localization challenge in this article. The other area of localization, formatting is handled relatively well by Windows, the .NET framework, and SharePoint. The formatting of dates, currencies, etc., all tend to appear the way that they should appear with little or no developer intervention. Language is, however, more challenging as it's not a matter of moving around periods and commas. It's a matter of selecting the right word, having different abbreviations, and generally making the right words work.

Most of the time, we barely recognize that the software has language embedded in it. Labels are prime candidate for needing to be converted as are error messages. Anything that you might display in the body of the web part qualifies as text that may need to be localized. Similarly in a SharePoint world, there are properties of the web part which need to be made language specific. The property's friendly name, category, and description are all candidates for localization. The challenge is that much of this text is hardcoded into controls through attributes and constants. The localization technique must make it easy to take working code and convert the strings which are already hard coded into a language specific string.

.NET and the web part infrastructure does provide facilities which make localization easier. In fact all of the fundamentals are provided, it's up to you to string them together and make them work. One of the fundamental components necessary to localize software is the Locale Id (lcid). This is an internationally standardized ID to indicate the language used by a particular location. The beauty of this is that you can read from code the lcid from the running thread very easily (System.Threading.Thread.CurrentThread.CurrentUICulture.LCID).

Once you know what language you're working with you can apply a technique for loading the appropriate text string for the locale the software is being used in by taking in a key and fetching the locale. In the SharePoint web part class there is a virtual method called LoadResource which takes in a string-based key and returns a string value. By overriding this method you can control how SharePoint loads the localized strings for the application.

Typically, .NET leverages resource DLLs which are simply compiled versions of resources. The resources in this case are strings. Because we're most frequently concerned with strings while localizing, it's possible to take a different approach than resource files to get our localization information. We can load and use XML files directly. Before we get to the code to do this, we need to figure out how to get these files.

Creating the non-Resource File

Sometimes the simplest solutions are best. While resource files are more compact and offer more flexibility, in a simple application like a web part - particularly one where you want to encourage others to write language files - an XML file is a good place to start.

Bil Simser proposed a very simple XML file format which I like quite a bit, it looks like this:

```
<?xml version="1.0" encoding="utf-8" ?>
<strings>
  <string id="SomeKeyName">SomeValue</string>
</strings>
```

The simplicity of this file format allows you to quickly add new entries without worrying about XML namespaces or extra processing. A suggested naming the files with the language ID and an extension of .XML makes it easy to find the files after they've been deployed. Naming the language files with their locale id with a .XML extension is exactly what the code shown below will expect. The Locale ID for US English is 1033. So a US English resource file would be 1033.xml.

Deploying Resources

When you deploy a web part with the STSADM tool you create a web part CAB file. In that CAB file you include a manifest.xml which tells SharePoint what to do with the files that are contained inside it. Basically you end up with an Assemblies section and a DwpFiles section. In the Assemblies section you describe the assemblies that must be installed and the supporting files that those assemblies need.

The tag contains one or more tags which are the assemblies (DLLs) to be installed. Within that tag you can specify class resources in a tag in addition to the SafeControls entries that you want to add. Class Resources, each in its own tag require only a FileName. When the tags are provided these files are copied out of the CAB file and into a special resources folder on the web server during the installation of the web part. The specific location of the resources folder will vary slightly based on the installation so there's no one, easy way to know precisely where the files will be placed.

The good news is that you can find the path that the class resources are deployed in by looking at the Web Parts' ClassResourcePath property. It will point you to where the class resources were copied. That makes the process of actually finding the files once they're deployed very easy.

A word of caution, the tag won't deploy any file you want. There's a relatively small list of file types that it will support. It appears the list includes (but isn't necessarily limited to): .js, .gif, .jpg, .txt, .htm, and .xml. There are definitely types that it will not deploy correctly. I tried .lng and although no errors were raised, the files were not deployed.

Making Text Local

With the files created and deployed, it's time to create a way to read the resource files and return an appropriate string. In order to read the file there are three pieces of information you need, the class resource path, which you can get from the web part object itself, the locale id (lcid), and the id of the string to return. The first step is to get the resource file. This requires the class resource path and the lcid. A method to get the language resource file appears in Listing 1 as GetLanguageXmlResource(). The GetLanguageXmlResource() leverages a method called GetFromWeb() which appears in a web.cs as shown in Listing 2.

Listing 1: Localization.cs - Localization Support functions

Listing 2: Web.cs - Web support functions.

The GetLangaugeXmlResource checks the lcid to make sure it's valid and then tries reading the file for the lcid. It does this by forming the full name of the file with GenerateLangaugeFileName() and then calling Web.GetFromWeb() to actually fetch the resource from the server via a web call. A web call is necessary since translating the path to a physical file won't work from within a sub-site in SharePoint because the file doesn't really exist in the sub-site location. SharePoint is remapping the URL.

The second method that is necessary is a method to load the resource. This appears in Listing 1 as LoadResource(). The LoadResource() method takes the ID to retrieve and the XmlDocument returned from the GetLanguageXmlResource() call. It uses an xPath query to return the string that matches the id provided and returns its value - or returns null if the item isn't found.

Between these files you have a foundation for localization. You can locate and load the appropriate resource file and you can extract entries from it. All you need to do now is start to connect these methods with your web part. This is done by adding a private variable for multiple thread synchronization, a protected variable to hold the XmlDocument of the resource file, and an override for the LoadResource() method. Listing 3 shows LocalizeWebPart.cs - a demonstration web part for attaching localization.

Listing 3 - LocalizeWebPart.cs - A test localization web part.

The LoadResource first locks an object to prevent multiple threads from entering the same region of code at the same time. In this region it checks to see if the XmlDocument for the resource file is loaded, if not it loads it. After the document has been loaded the static LoadResource() method from Localization.cs as shown in Listing 1, is called. This extracts the string from the file and returns it.

At this point, any text that you want to have localized can be run through LoadResource() of the local object. You'll set this in the CreateChildControls() override method as it uses LoadResource() to transform the entry for localString that the user provided into whatever the corresponding string is in the resource file. This allows you to make any of your text localize - however, what about the ToolParts? That's the next step.

ToolParts Tango

Typically when you define a property that you want to appear in the tool pane (where properties for a web part are configured) you add a FriendlyName, Category, and Description attribute above the property. However, in doing that you've committed yourself to the text that exists in the code - and you don't want the code to change to match every locale. However, there is a mechanism for specifying that you want to load the strings for friendly name, category, and description from a resource file. That mechanism is the [Resources()] attribute. It takes three parameters for name, category, and description respectfully. You provide the keys to use to calls to LoadResources() and the Web Part infrastructure makes the calls for you automatically. In listing 3 you see this in practice for the Lcid property and the LocalString property.

How these properties appear on the tool pane will be driven by the information returned from LoadResource() overridden method that was defined above. In our case if we create entries in the XML file for a locale. Our LoadResource file will load the string from the XML entry in the file.

Lingual Limbo

You can download the source code for this article, compile it and run it. When you do when you change the lcid to a different locale the text display will change to reflect the new locale. It takes an extra refresh (or apply) after the update of the locale id in order for the changes to be seen because the change to the locale actually happens after the resource file is loaded. In order to keep the sample simple the code to cause an extra post back was omitted.

Included in the download are files for US English (1033) and Finnish (1035). Changing the locale to 1035 will cause Finnish strings to be displayed.

Conclusion and Thanks

Now you have all of the pieces to doing localization for your SharePoint web parts. From how to create the resource files to how to deploy the resource files to how to implement tool part support, you have got everything you need to a web part that can be localized.

Thanks are due to Bil Simser, Maurice Prather, Todd Bleeker, and the other MVPs who expressed their thoughts on this topic to me or have expressed their perspectives on the problem via blog entries. In addition, special thanks are due to Anna-Liisa Kaila-Walsh and Mike Walsh who provided the Finnish translation for this sample which is included in the sample application.

About the Author

Robert Bogue, MCSE (NT4/W2K), MCSA:Security, A+, Network+, Server+, I-Net+, IT Project+, E-Biz+, CDIA+ is the president of Thor Projects LLC. He has contributed to more than 100 book projects and numerous other publishing projects. He was recently honored to become a Microsoft MVP for Microsoft Office SharePoint Server. Before that Robert was a Microsoft Commerce Server MVP and before that Microsoft Windows Servers-Networking MVP. Robert blogs at http://www.thorprojects.com/blog. You can reach Robert at Rob.Bogue@thorprojects.com.