Firefox Extension Development Tutorial :: Localization

Table of contents

1. Overview

Firefox, and the various other Mozilla browsers, have a large and diverse community of users. Preparing your extension for localization is important because users prefer to use applications in their own language and culture.

An extension is constructed of one or more source files, which are either interface files (XUL) or script files (JS). These source files contain references to textual data such as labels and buttons. While it seems natural to place ordinary strings in these source files, it will make it harder for people who use different languages. In fact, nothing stands out more to a user then to have some parts of a program use a language different from his or her system settings. To be a professional extension developer, it is essential to store textual elements, called entities, in a language file.

2. Entities as String Constants

Many applications are built such that translating the interface into a different language is as simple as possible [1]. In general, building translatable software is as easy as placing all textual data in a resource file as string constants and then referencing those constants from within the source code. In XML, a constant is known as an Entity.

The XML syntax for declaring an entity is simple, but different than might be expected. The tag starts with “!ENTITY” followed by two parameters, the name of the tag and the string value associated with that tag. The XML parser will replace all occurrences of the tag with the associated string value.

```
<!ENTITY TagName "String Value">
```

Again, notice the syntax of the declaration. There is no trailing slash. In the context of an extension, the Entities are stored in a DTD file, which brings us to the next section.

3. Language Document Template Definition (DTD)

Consider the following Entities from the hpsched.dtd, located in the locale/en-US/ file:

```
<!ENTITY  newSchedule.button     "New Schedule">
<!ENTITY  editSchedule.button    "Edit Schedule">
<!ENTITY  deleteSchedule.button  "Delete Schedule">
```

For an extension, the locale folder contains one or more languages in various directories named after the ISO language codes such as en-US, en-UK, fr, es, etc. Say, for example, that

someone wanted to translate the Home Page Scheduler extension into Spanish, he would copy the hpsched.dtd file to locale/es/ and translate the string values, such as:

```
<!ENTITY  newSchedule.button     "Cree el Horario">
<!ENTITY  editSchedule.button    "Redacte el Horario">
<!ENTITY  deleteSchedule.button  "Borre el Horario">
```

By keeping the textual elements defined as constants within the DTD file, it is easy for translators to convert the application to support various languages without modifying code. As an extension programmer, you can focus on writing an awesome application and others can translate it to work in Catalan, Dutch, Japanese, and numerous other languages that you do not know. You may note that the translation in this example was conducted using machine translation from freetranslation.com. In general, it is a good idea to have someone who knows a particular language perform the translation. So by keeping the code and textual data separate, you can make the extension better for everyone.

4. Referencing an Entity in Your Application

Now that you are convinced of the importance of allowing easy localization [3] of your extension, it is time to see how to reference an entity from source files. To dereference an Entity in a source file, simply escape include the tag name like &TagName;. The ampersand (&) tells the parser that a tag name is to follow and the semicolon (;) ends the tag.

The following XUL code declares three buttons with labels defined in the DTD file for the system's language.

```
<button label="&newSchedule.button;" oncommand="newSch();"/>
<button label="&editSchedule.button;" oncommand="editSch();"/>
<button label="&deleteSchedule.button;" oncommand="delSch();"/>
```

So now, if someone were to be using the Home Page Scheduler extension in Firefox on a Catalan system, the ca/hpsched.dtd file would be used instead of the en-US file. Sadly, at this time our extension does not have a Catalan-language directory, which means a Catalan-speaking user's system would default to the Spanish (if it existed) or to the available English strings.

5. Supporting Multiple Locales

Mozilla maintains extension code as a package in the chrome registry. There are usually three different parts to a chrome package, although they are all optional. Each part is stored in a different directory. These three sets are the content, the skin and the locale. A particular package might provide one or more skins and locales, but a user can replace them with their

own. In addition, the package might include several different applications each accessible via different chrome URLs. The packaging system is flexible enough so that you can include whatever parts you need, and allow other parts, such as the text for different languages [4].

The locales directory can hold one or more language DTD files. Copying the initial language file into a new directory and then translating the String Values are the several Entities defined within the DTD file creates a new language. The browser will select the best match for the user system's language setting.

6. Further Reading

1. XUL Tutorial: Localization. Devmo 2005 [cited November 5, 2005].
2. ISO 639: Code for the representation of names of languages. 1990 [cited November 6, 2005].
3. How to localize Mozilla. 2005 [cited November 6, 2005].
4. XUL Structure. XUL Planet Tutorial 2005 [cited November 6, 2005]