Deke Smith
Adobe for Enterprise, Not-Quite-Yet Enterprise and Not-At-All Enterprise



# Internationalization within Sling (and CQ)

Sling, and as an extension, CQ, both use a model for internationalization that is very similar to the one used by Java and Flex. Java and Flex store translations as key/value pairs within properties files. Within Sling, translations are stored within the repository as key/value pairs. Once defined, the developer is able to use the key/value pairs stored in the repository to populate strings used in a graphic interface such as a Web page.

CQ has additional tools for internationalization than those provided by Sling. *The CQ additions are not part of the scope of this article.*

## Locales

Before using internationalization within Sling the developer must be knowledgeable about the concept of locale. Locales within Java can be defined by three different properties: the language, the region and the variant. The primary defining property of a locale is a human language. Regions are used because each language can have several dialects and usage of the language varies greatly based on the region it is spoken and written. For that reason, a region code can be associated with a language. In that way, Canadian French may have its own definitions and translations and European French a different set. Additional information about the locale can be defined in the variant property. This property can contain information about a specific locale that is not covered by the other two properties. Examples of variant information for a locale is type of the language used, the target operating system or the text encoding.

Sling uses a standard representation of locale. The language is represented as the ISO 639-1 code for the language in lower case. This is usually a two letter code. English, for example, would be represented as *en*. If needed, a region code can be specified. This is done by adding an underscore plus the ISO 3166-1 region code in upper case to the language code. So for English as spoken and written in the USA, the full locale code would be *en_US*. Very rarely would there be a variant defined in a locale for Sling.

Sling uses this standard because browsers use this standard for locale. Within Sling, the typical usage of internationalization would be to return the content of a Web page using the default locale of the Web browser. The default locale or locales are passed to Sling as part of the request. The site developer has the choice of respecting the locale of the Web browser and sending back content appropriate for that locale.

## Key/value pairs

What do key/value pairs define? I call them translations but technical they are translation segments. Translation implies taking a word or phrase and finding the best match in another language. Translation segment is what you get after someone has done the translation for a specific use and the results are saved in a look-up table. The value of translation segment has a very specific meaning. When the phrase, OK, is used on a button the translation segment in another locale for that phrase is not necessarily a direct translation of the word, OK, but what the OK button is called in the target locale. Translation segment is very context driven. Because of that it can contain style information or substitutions indicated for data set at run time. It can even be a pattern for how information, such as a date or number, is rendered in a locale.

Within Sling, key/value pairs are organized by locale. If a site supports two languages, matching key/value pairs should exist for both. The locales CQ defines key/value pairs for can be seen at */libs/wcm/core/i18n*.

Those keen of eye may notice that the locale code, *en*, is not there. In Java, if the value of a key is requested for a locale and that key does not exist, an exception is thrown. Not so in Sling. If a key is not defined in a locale Sling does not throw an exception. Instead, the key itself is returned. The keys for the key/value pairs in */libs/wcm/core/i18n* are the translations in English. Since the locale for English does not exist the key is returned when the value of a key for the locale, *en*, is requested.

In CQ, the default for a key is the English phrase. The key and the English phrase are the same. Using the English phrase as the key for the key/value pair is an architectural decision and is not the only way to build translation memory in CQ.

I prefer to not use defaults for the phrases as keys. One thing that is constant in development is change. And it is likely that the text for the user interface will change. If it does, the key must be changed in all instances in the code and for each locale. Where multiple teams edit each of these items this can get to be a logistical challenge. I would use, *hello_world*, as a key versus the default, English, value for the phrase, *Hello World!* Doing things in that way allows the translators and the developers to work relatively independently of one another once the key/value pairs are defined.

Using the English phrase as the key does have its advantages and this is the reason this is the convention CQ uses. The advantage to using a default phrase as key is that the keys themselves will show up in the interface when the translation does not yet exist in a locale. Having non-phrase keys within a user interface does look like a mistake. When the phrase in English is the key and the translation does not exist in the target locale then the English phrase is shown.

Which convention to use is a matter of preference and the specific use case.

## Translations in the repository

As can be seen for the CQ translations at */libs/wcm/core/i18n*, the key/value pairs are stored by locale. The locale node must use the mixin, *mix:language*, and have the property, *jcr:language*, defined with a locale. In addition, the locale node may have the property, *sling:basename*. The *sling:basename* property is either a *String* or *String array* of names that can be used a labels or tags to filter translations with. The name of the locale node is not significant and can be anything. Be nice to others who have to edit what you have done, though. Name the locale node to match the locale within *jcr:language* or name of the language. For example, name a locale node for English either *English* or *en*. The primaryType of the locale node is not significant for translations.

Within the locale node are key/value child nodes, each one containing the information for a single key/value pair. These child nodes must either have the primaryType of *sling:MessageEntry* or they must have *sling:Message* as a mixin. The key/value node must contain a property, *sling:message*, that is the value for the key. If the property, *sling:key*, exists then that value is used as the key. If the *sling:key* property does not exist then the name of the node is used as the key. Following the Be Nice Rule, name the key/value node the name of the key. If you do that, the *sling:key* property is redundant.

Here is an example of key/value pairs for both English and French:

```
/apps/myapp/i18n
             +-- en (nt:folder, mix:language)
             |    +-- jcr:language = "en"
             |    +-- hello_world (sling:MessageEntry)
             |    |    +-- sling:key = "hello_world"
             |    |    +-- sling:message = "Hello world!"
             |    +-- goodbye (sling:MessageEntry)
             |         +-- sling:key - "goodbye"
             |         +-- sling:message = "Goodbye!"
             +-- fr (nt:folder, mix:language)
                  +-- jcr:language = "fr"
                  +-- hello_world (nt:unstructured,sling:Message)
                  |    +-- sling:key = "hello_world"
                  |    +-- sling:message = "Bonjour tout le monde!"
                  +-- goodbye (nt:unstructured,sling:Message)
                       +-- sling:key - "goodbye"
                       +-- sling:message = "Au revoir!"
-
```

## Using internationalization within JSP

Within this blog post I will only give an example of using key/value pairs for locales within JSP. They can be used elsewhere in Sling, as well.

When the <*sling:defineObjects* /> tag is used within a JSP page, the *slingRequest* value is created. The *slingRequest* implements the **SlingHttpServletRequest** interface and has a couple of very useful methods for internationalization: *getLocale(), getLocales(), getResourceBundle(Locale),* and *getResourceBundle(String, Locale)*.

The *getLocale()* method gets the default **Locale** of the request. The *getLocales()* method gets all of the methods for a request. Typically the value for these come from the browser.

The method, *getResourceBundle(Locale)*, gets a **ResourceBundle** instance that has all of the found key/value pairs for the locale. The method, *getResourceBundle(String, Locale)*, gets a **ResourceBundle** with all of the key/value pairs for a base name and a locale. The **ResourceBundle** interface has a method, *getString(String)*, in which the argument is the key and the value of the key is returned. If the **ResourceBundle** does not have that key, the value returned is the key itself. One thing to keep in mind is that a **ResourceBundle** is not the same thing as an OSGi bundle. They are two different concepts using the same name.

An example:

```
<%@ page contentType="text/html; charset=UTF-8" language="java" errorPage="" %><%
%><%@ page session="false" %><%
%><%@ page import="javax.jcr.*,
org.apache.sling.api.resource.Resource"
%><%
%><%@ taglib prefix="sling" uri="http://sling.apache.org/taglibs/sling/1.0" %><%
%><sling:defineObjects /><%
%><!DOCTYPE HTML>
<html>
<head>
<meta charset="UTF-8">
<title>Test</title>
```

```
</head>
<body>
<%= slingRequest
.getResourceBundle(slingRequest.getLocale())
.getString("hello_world") %>
</body>
</html>
/xml]
```

**See also:**

Apache Sling – Internationalization Support



**2 Responses to *Internationalization within Sling (and CQ)***

**Dan Klco** *says:*
November 6, 2012 at 2:36 pm

I'd mention that CQ also has a UI built in for managing and updating translations:
http://www.6dlabs.com/blog/dklco/2012-05-25/adobe-cq5-translator

**Deke Smith** *says:*
November 6, 2012 at 2:58 pm

That is a very good link. Yes, CQ has additional internationalization utilities added on top of what exists in Sling. There are some Java-based internationalization frameworks that can be used by the CQ developer as well. I wanted to take on Sling->CQ i18n piece by piece. Your link is a good description of how CQ builds on top of and extends the Sling i18n framework. Thanks.

There are some clever things that can be done with fragment bundles to add language support as needed. I will try to describe that as well later.

**By** Deke Smith

CQ (14) internationalization (1)

locale (1) Sling (3)

**Comments (2)**

**Created**
October 21, 2012