Development-Localization and Internationalization

Unicode

See also: UTF-8 Encoding Tips

To Enable Unicode for your WO app, add the following to your application constructor:

```
WOMessage.setDefaultEncoding("UTF8");
```

This tells all WOResponse and WORequest to use UTF8 (Unicode).

Then you just need to tell the browser. Make all your .wo pages include this meta tag in their HTML:

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<!-- etc... -->
```

Jesse Barnum

Great tip - here is a simple method call you can stick in your Application object to automatically achieve the results outlined above:

```
private boolean enableUTFEncoding = false;

public void enableUTFEncoding() {
  enableUTFEncoding = true;
  WOMessage.setDefaultEncoding(_NSUtilities.UTF8StringEncoding);
}

public WOResponse dispatchRequest(WORequest theRequest) {
  WOResponse result = super.dispatchRequest(theRequest);
  if( enableUTFEncoding &&
"text/html".equals(result.headerForKey("content-type")) ) {
    result.setHeader("text/html; charset=UTF-8; encoding=UTF-8",
"content-type");
  }
  return result;
}
```

Helmut Schottmüller

Unfortunately it's not so easy if you want to use file upload fields and UTF-8 encoding in the same form. Adding a file upload component means that you have to set the form's enctype to "multipart/form-data". To force a multipart form to use UTF-8 encoding usually needs an enctype of "multipart/form-data; charset=UTF-8" but WO is not able to identify such a form as multipart form. You will get a "java.lang.IllegalArgumentException: This form is missing a 'enctype=multipart/form-data' attribute. It is required for WOFileUpload to work." error when you open the form in the browser.

To make sure that UTF-8 is supported in multipart forms as well, you have to add the following code to your Application object:

```
public WORequest createRequest(String aMethod, String aURL, String
anHTTPVersion,
    NSDictionary someHeaders, NSData aContent, NSDictionary someInfo) {
  WORequest newRequest = super.createRequest(aMethod, aURL, anHTTPVersion,
      someHeaders, aContent, someInfo);
  newRequest.setDefaultFormValueEncoding(_NSUtilities.UTF8StringEncoding);
  return newRequest;
}
```

To make WOFileUpload components working I also had to add the launch parameter -WOUseLegacyMultipartParser true to my application. This launch parameter forces the parsing of all form values, the first time WORequest.formValues is called. See the apple developer documentation for additional information. Without -WOUseLegacyMultipartParser true I had serious problems in my applications using a WOFileUpload component because the bindings *data* and *filePath* have been emptied after a form POST.

With Jesse's code and this extension, you will be able to handle UTF-8 character data correctly in your WO application.

If you use localized strings in your UTF-8 application you may also check out Project Wonder's ERXLocalizer class.

Project Localization tips

The following are some tips and suggestions for localizing a project in WOLips using Project Wonder.

Eclipse Default Encoding

I prefer to keep my entire project in UTF-8 format for consistency. You can set that in your Eclipse General->Workspace preferences. The exception, of course will be your Localizable.strings files. Those have to be in UTF-16 format. I generally get warnings about the WOO file for my initial Main component whenever I create a new project, but if you right-click your Main.wo you'll see "Properties" at the very bottom of the contextual menu. Open that and flip your encoding between project default and UTF-8, save it, then open it back up and return it to the project default and the problem should go away. This is also how you set your Localized.strings file to UTF-16 even if the rest of your project is not UTF-16.

Properties file

Let's say your project will be available in English and Japanese. You'll want to include the following in your Project->Resources->Properties file:

```
# Localization
er.extensions.ERXLocalizer.defaultLanguage=English
er.extensions.ERXLocalizer.fileNamesToWatch=("Localizable.strings","Valida
tionTemplate.strings")
er.extensions.ERXLocalizer.availableLanguages=(English,Japanese)
er.extensions.ERXLocalizer.frameworkSearchPath=(app,ERDirectToWeb,EREXtens
ions)

# Project Encoding
er.extensions.ERXApplication.DefaultEncoding=UTF-8
```

Note that if you need to customize the locale for a language, such as Canadian French, you can do so with this property:

```
er.extensions.ERXLocalizer.French_CA.locale = fr_ca
```

The other changes are then in the er.extensions.ERXLocalizer.availableLanguages and ERXLanguages:
In the above case after adding canadian french these would change in:

```
er.extensions.ERXLocalizer.availableLanguages=(English,Japanese,French_CA)
```

Localized formatters use this property:

```
er.extensions.ERXLocalizer.useLocalizedFormatters=false
```

Localized strings and components

For each language available, you will need a corresponding Localizable.strings file. This file should be located in Projects->Resources->"Lang".lproj directory. In these directories, you'll store localized resources such as Localizable.strings files and localized components. So, continuing with the above example, you should create two new Localizable.strings files in the following places in your project directory:

> *Project->Resources->English.lproj->Localizable.strings*
> *Project->Resources->Japanese.lproj->Localizable.strings*
> *Project->Resources->French_CA.lproj->Localizable.strings*

As mentioned earlier, it's recommended that these be in UTF-16 format. You can do that by right clicking on the file in WOLips and selecting "Properties." In the resources panel, change from the project default encoding to UTF-16.

If you have any components that need localizing, then you should relocate that component from your Project->Components folder into the appropriate Lang.lproj folder. Then make a copy of the component into the remaining lproj directories and you can begin the process of localizing the component. You do not need more than one copy of the associated API or java file. You only need duplicates of the WO. So, as an example, if you wanted to localize

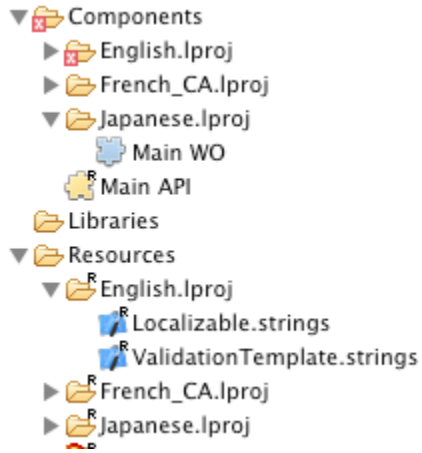
> *Project->Components->Main WO*

You would right-click->Refactor->Move it to

> *Project->Resources->English.lproj->Main WO*

and then right-click->Copy it from English.lproj and right-click->Paste it into Japanese.lproj. At this point, when you open the component in WOLips, there will be a tab at the bottom of the component editor view that allows you to switch back and forth between different localized versions of that component.

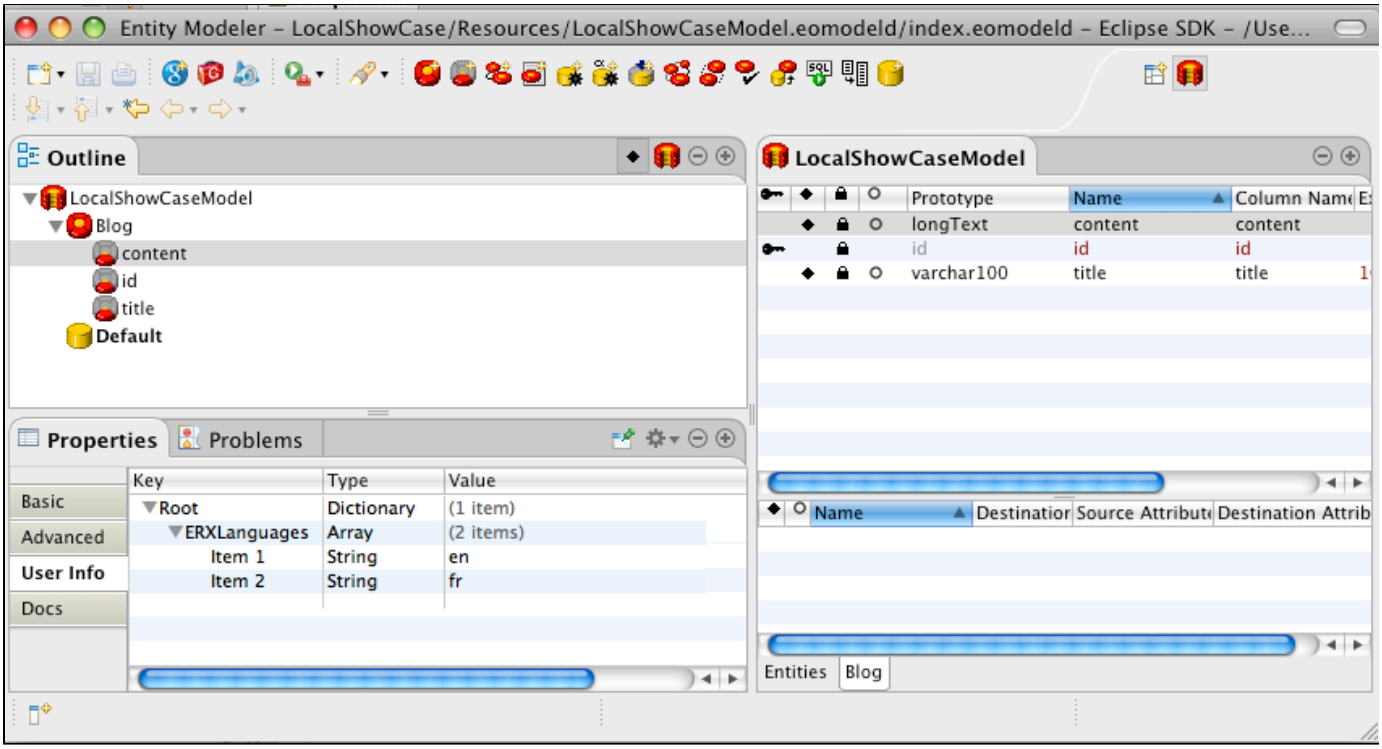
Your layout would end up something like this:



Localized EOAttributes

In Wonder, it is also possible to localize attributes. Let's say we have an entity *Blog* with an attribute *content* that we want to localize. This will be realized by not creating a column *content* in the database but a column for each specified language we want i.e. *content_en*, *content_fr*, ...

To tell EOF that we want a specific attribute localized you have to add a key ERXLanguages to its user info.



In this example we set the type to *Array* and add an item for each needed language setting its value to the language code. This must be done for each attribute in our model we want to localize. If you have many localized attributes that have the very same list of languages and you will likely be changing that list in the future you can define your language list either for a whole model or for all models instead. The first way to specify a per model language list is to put an ERXLanguages key into the user info of the model. For a global setting put that key into your property file:

```
ERXLanguages = (en,jp,fr_ca)
```

All attributes that should use those global settings must have a key *ERXLanguages* in their user info with a type **different** to *Array*. Its value can be anything as only the presence of the key is of importance. The order that the languages are applied to an attribute is:

- attribute user info with an array for ERXLanguages
- if type of found user info is not an array then get array from key ERXLanguages from user info of the model
- if user info of the model has no key ERXLanguages look for a property ERXLanguages in your property files
- if no property ERXLanguages is found ignore localization

Direct Actions

If you are defaulting to direct actions, you may not have a session. If you do not have a session, the server will return the default language specified in the Properties mentioned above. If you're using direct actions and you don't like that behavior, you can stick this in your direct action class:

```
@Override
 public WOActionResults performActionNamed(String actionName) {
  if(!context().hasSession()) {
   ERXLocalizer localizer =

ERXLocalizer.localizerForLanguages(context().request().browserLanguages());
   ERXLocalizer.setCurrentLocalizer(localizer);
  }
  return super.performActionNamed(actionName);
 }
```

That should give the user their browser's default language setting instead of your server's default language setting until a session is created.

LocalizerTest.zip is an example application demonstrating the sessionless use of the localizer with localized strings and localized components, storing the language state in a cookie.

Database setup

Outside of this, if you are using a database, you'll need to make sure that is encoded properly as well. I'm using MySQL, so I have in my EOModel:

jdbc:mysql://localhost/mydatabase?capitalizeTypenames=true&zeroDateTimeBehavior=convertToNull&characterEncoding=UTF-8

The database itself is set to default to "UTF8" encoding. (No hyphen in UTF8 for MySQL) You can set that in the "Options" pane of MySQL Administrator.app under the "Advanced" popup menu item in the "Def. char set" field. Of course, you'll need to use the correct database types too, meaning don't use a blob for text storage. Use varchar and longtext (varcharLarge is the name of the Wonder prototype) instead.

Localization presentation from WOWODC West 2009

Guido Neitzer did a localization presentation at WOWODC West 2009 that give a good overview of how to localize your apps. The presentation is available here.