

## Musings on a Translation System

Translations are probably the last "big" must for Quantum Star SE - yes, I know the game barely exists (well, see the 0.6 pre-alpha on Sourceforge). Translations have been offered for QS in about 15 languages at this stage, but unfortunately without a prefixed list of templates, text and names it was impossible to accept at the time. Some time after the 0.18 release therefore I will add in a Translation feature, and we'll take it from there (it'll be restricted to the installer for testing).

There are as many translation schemes as developers...well, almost. I'm not entirely sure which scheme I'll use myself. There's the simple approach - define a list of constants for every english string, and have a set of include files where those constants have translated versions. Its simple, easy to mock up, but it may not be as flexible as one might think (how do you deal with dynamic data within sentences?).

One variant I've been thinking would use a third party set of classes. Using the liberal New BSD license, ezComponents carries a well formed translation package that hinges on full English strings and allows the integration of dynamic data (ship names for example). However running a translate command, using an array as the data source, can also be slow and lumbering. So I'm thinking of adapting this to cache the translated template. Might not be the ezComponent way, but I find it interesting.

How would this work? Well, I'm far from designing it - this entry is more brain dump than a viable plan.

Basically one would set up an easily editable XML file containing a list of English sentences, and terms used in the game. Now the idea is not to output a translated string - but to integrate it into a secondary cached template. One might have:

Welcome! You are named {User.Name}.

translated to (forgive my poor french spelling):

Bienvenue! Tu t'appelles {User.Name}.

This would be sourced in an xml file of form:

```
<message>
<source>Welcome! You are named {User.Name}.</source>
<translation>Bienvenue! Tu t'appelles {User.Name}.</translation>
</message>
```

The origin template could look like:

```
<p>getTranslation('Welcome! You are named {User.Name}.'); ?></p>
```

Now we would fetch the translation (from the parsed XML) and integrate it into the origin template. This secondary form of the origin template would then be cached into a secondary template.


The secondary template would itself be the actual template to receive a final parsing by Smarty. Which would parse the {User.Name} reference to integrate the dynamic portion of the translated string.

Hey presto - the output is a translated page, which being cached does not require translation on the fly (which presumably would ease up on processing a bit). I am of course ignoring the possible need to translate the dynamic data on the fly...

To differentiate between cached translated templates, they would be prefixed with the relevant language string, e.g.

Origin: location\_display.tpl.html

Secondary: fr-frlocation\_display.tpl.html

fr-fr is an abbreviation for French (France). fr-ca would mean French (Canada) depending on the local variant or dialect. We should also have a default "fr" whenever a more specific regional alternative does not exist, i.e. if fr, fr-ca exist, and someone is looking for fr-bg (French-Belgium), we should default back to the root language (fr), which is probably likely going to equate to fr-fr or fr-ca (more likely native French). Sorry if that's all confusing... 

Key to understanding this complicated scheme (which may or may not be realistic) is that we are using a source template to generate Smarty templates specific to each language - the Smarty templates are generated once (before distribution probably - users probably won't need to do any XML parsing), and the Partholan View layer will use the user's language preference to select a language specific template for Smarty to parse.

Key is that all this happens from a single origin template - themers need only create one template using the generation scheme when designing (and make certain they utilise an XML translation file to facilitate the Smarty template generation stage for distribution!).

Another thing to keep in mind is that ideally QS should be running with text encoded in UTF-8. At the moment that extends to the HTML, CSS and PHP files themselves. I haven't taken a swing at database charsets just yet, and have not used the form accept-charset attribute to tell client browsers to sent data encoded as UTF-8. UTF-8 will ensure non-roman character translations are properly displayed (if the language is installed on the client browser).

The other side is internationalisation. We need to manage the local settings of users - their language, date preferences (and day/month names which need to be done manually), language headers in HTML etc., and numeric formatting... I'll take that separately from Translation however.

Another side is using Smarty - I'm not sure how easy this system could migrate to a pure PHP solution like Savant (we'd need to generate PHP, not simply markup). I'd like to migrate to Savant at some stage... Neater than Smarty IMO if a little less secure for using PHP direct.

Anyone with bright ideas or another view throw a comment.

Possible issues... Is this level of complexity worthwhile for the end result? Language specific templates without any on the fly translation work. Can it be pulled off? How to manage HTML headers on a per file basis (for informing the browser of language). Few other niggling issues - still brain dumping. Might draw up an actual design document in a week or so and get more formal feedback on the forums...

Posted by Pádraic Brady at 00:21