www.phparch.com

February 2014
VOLUME 13 - ISSUE 2

php[architect]

Databases

Introduction to Neo4j for PHP - Part 2

SQL to NoSQL Migration

I18n by Translation Through Proxy

ALSO INSIDE

Editorial:
Masters of Data

Education Station:
Generate Test Data with Ease Using Faker

Confident Coder:
Attention to Detail

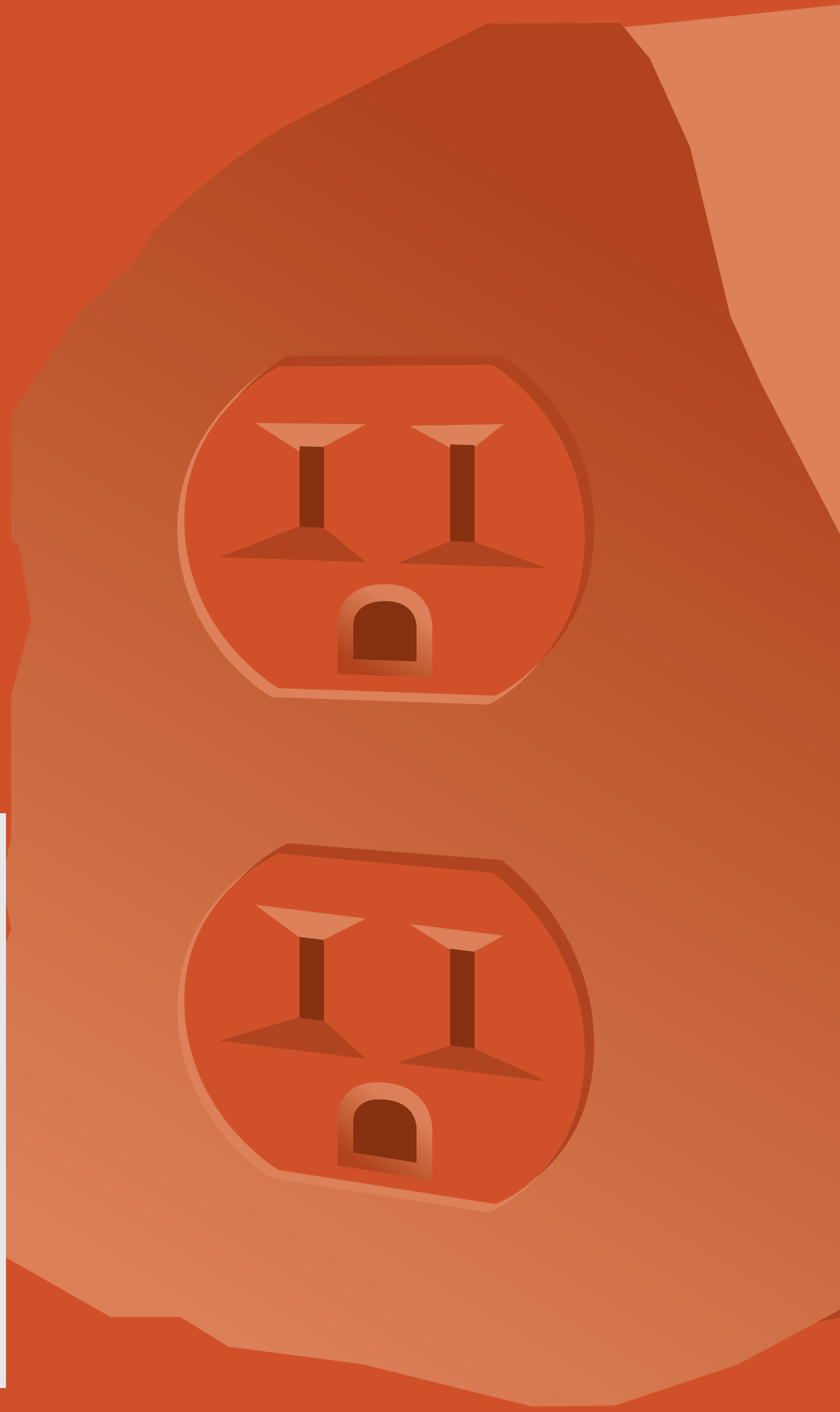
Laravel Tips:
Rapid Coding and Prototyping Through Generators

finally{}:
On Elegance

I18n by Translation Proxy

Steve Hannah

Using a translation proxy can significantly reduce development and maintenance costs for multilingual applications compared to conventional internationalization approaches. We will discuss some of the advantages of the translation proxy approach, and then demonstrate a typical internationalization process using SWeTE, an open source translation proxy written in PHP.



DisplayInfo()

Requirements:

- PHP: 5.3+
- SWeTE: 0.3 or higher

Other Software:

- MySQL 5+
- Apache 1.3+ with mod_rewrite installed

Related URLs:

- SWeTE Website – http://swete.weblite.ca
- SWeTE GitHub Repository – http://github.com/shannah/swete
- Open Cart Website - http://www.opencart.com/

I18n is the pseudonym for internationalization. Its derivation is based on the fact that there are 18 letters between the 'i' and the 'n' in "internationalization". I18n is the process of preparing an application for multiple markets, and it almost always involves adding support for multiple languages.

In PHP, the conventional process of i18n involves factoring out all static strings from the PHP code and templates into language files. On each HTTP request, the application would then determine the appropriate language based the environment (e.g. a cookie, a URL convention, or a user preference) and load the corresponding language file.

Translating dynamic content is more difficult as it often requires more substantial changes to the application structure and flow. For example, a shopping cart application that supports multiple languages will need to have a database structure that accommodates the translations of product descriptions. Also, the logic that interacts with the data source will need to be aware of this structure so that it can load the correct product translations at runtime and update the correct translations when product descriptions are modified.

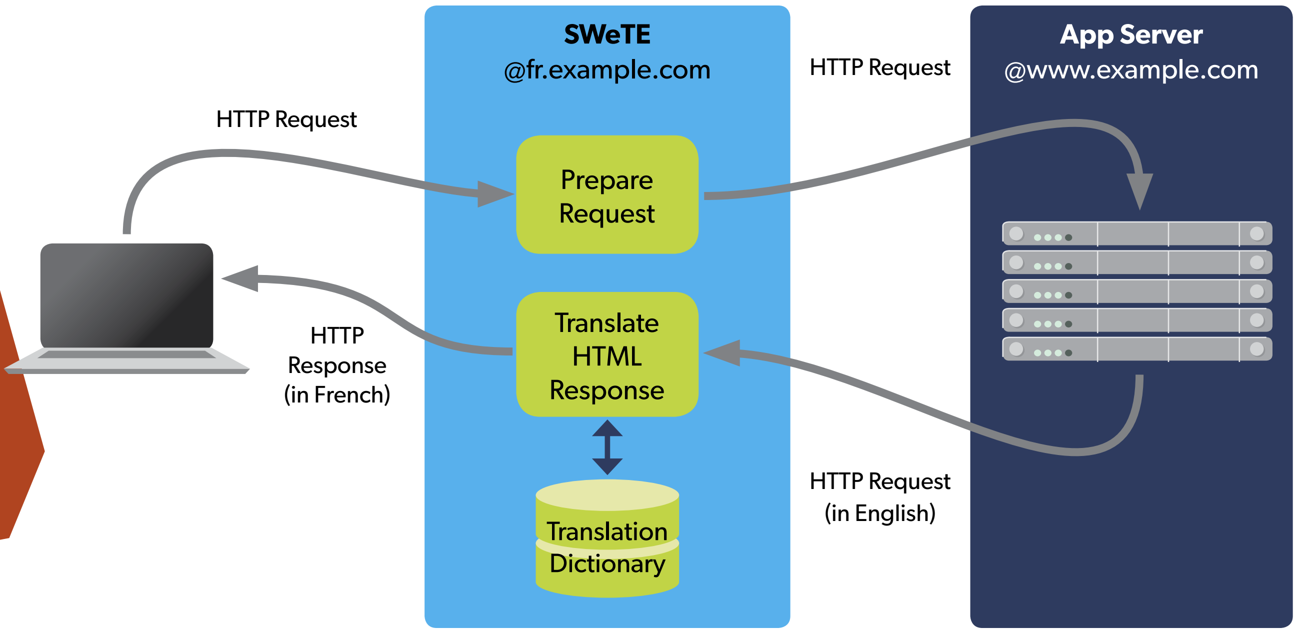
The more complex the application, the more difficult it will be to internationalize it. In some cases, it may actually be easier to build the application from scratch than it would be to refactor the existing app to support multiple languages.

In this article, I discuss an alternative approach to i18n using **SWeTE** (http://swete.weblite.ca), an open source translation proxy written in PHP. The SWeTE approach to i18n is radically different than conventional strategies as it allows you to keep your application (almost) completely insulated from the process. Your existing, monolingual application can remain unchanged with no knowledge that it is supporting multiple markets in multiple languages.

What is a Translation Proxy?

A translation proxy is simply a reverse HTTP proxy with a built-in translation dictionary and mechanisms to perform inline i18n. As shown in Figure 1, it sits transparently between the client and the application server, modifying the HTTP requests and responses as necessary to "fool" the client into believing that the application speaks their language. The HTML responses are parsed into individual strings and compared to a translation dictionary (which can contain either human translations, machine translations, or a combination of both). Strings are replaced by their corresponding translations and embedded into the response before it is passed back to the client.

FIGURE 1



Advantages of a Translation Proxy

By isolating internationalization into a layer of its own, the translation proxy reaps substantial rewards in terms of complexity, maintainability, and features. Adding support for multiple languages entails quite a number of additional feature requirements for an application. At a minimum, internationalization implies at least the following additional requirements:

1. A mechanism to select the appropriate language for the client.
2. A mechanism to translate content from one language to another, e.g. a translation form or additional fields to edit content in the alternate languages.

These two requirements can be surprisingly broad and deep in scope. If your application only translates static text, then the mechanism for translation might be as simple as sending your language files to a translator to be translated. However, if there is dynamic data that is stored inside a database, then you'll probably need to add sections to the user interface to manage these translations. If your application deals with large amounts to data, then you'll also need to make sure that you have a way of keeping track of translation status for all content so that you know what still needs to be translated or reviewed. If you are dealing with external translators, you'll likely also need to develop a workflow to export the translations in a format that the translator can work with and import the translations back into your application when they are complete.

As you think more about the internationalization issue, you will realize that it comes with an enormous amount of baggage - so much so that its feature requirements can encompass an application of its own. This is where a translation proxy begins to make a lot of sense. Rather than weigh down your application with generic translation features, you can offload them to a system that specializes in translation and leave your application to focus on what it does best.

SWeTE, for example, provides a rich interface for managing your application's translations. It provides translation forms and import/export capability so that you can easily interface with your translator's preferred CAT tools. It also makes it easy to monitor the state of various translations so that you always know if there is content that requires your attention.

SWeTE By Example

Much can be said about the theory behind reverse proxies, translation dictionaries, and the like, but I'm going to skip all that and cut straight to a real-world example of using SWeTE to perform i18n on a PHP application. As a demonstration, I'm going to take the open source shopping cart **Open Cart** (http://www.opencart.com) and use SWeTE to set up a French proxy for it. The English shopping cart will be located at:

http://opencartdemo.weblite.ca

The French version will be accessible at:

http://fr.opencartdemo.weblite.ca

Note, that I could have decided to set up the French site in a subdirectory like http://opencartdemo.weblite.ca/fr/ instead (or at any URL for that matter), but I personally find it easier to use subdomains for the separate sites.

I'll skip the details of how to install Open Cart. For this demo, I simply cloned the Git repository (https://github.com/opencart/opencart) and followed the installation instructions. The default install includes some dummy products and categories, so I didn't bother entering any custom data.

Installing SWeTE

SWeTE should run on any server that has PHP and MySQL installed. If running on Apache, mod_rewrite needs to be installed. If running on IIS, the URL rewrite module needs to be installed. This article assumes that you are running on Apache. Examples involving mod_rewrite directives will need to be changed to the equivalent URL rewrite directives if you are running on IIS.

SWeTE is really just a normal PHP/MySQL application, so installation is not much different than for any other PHP/MySQL application. For this article, I'll be working directly off of the Github repository (http://github.com/shannah/swete) so that I can work with the latest changes. You can do this also, or you can download the latest release from the SWeTE website (http://swete.weblite.ca).

Step 1: Clone the GIT Repository:

```
$ git clone https://github.com/shannah/swete.git
Initialized empty Git repository in
   /swete-data-recovery/home/phparch/swete/.git/
remote: Counting objects: 891, done.
remote: Compressing objects: 100% (477/477), done.
remote: Total 891 (delta 447), reused 821 (delta 377)
Receiving objects: 100% (891/891), 4.64 MiB | 7.30 MiB/s,
   done.
Resolving deltas: 100% (447/447), done.
```

Step 2: Run the ANT Build Script.

SWeTE contains an ANT build script which automates most of the rest of the application setup. It requires you to have ANT installed on your system.

```
$ cd swete
$ ant … [ Lots of output from running git and checking out submodules etc..]
```

Step 3: Create an Empty MySQL Database and User

Next, we use code similar to that in Listing 1 to create an empty MySQL database and a user.

Step 4: Set Up the Configuration

Rename the `swete-admin/conf.db.ini.sample` file to `swete-admin/conf.db.ini`. Then, edit the contents to connect to the database we just created.

```
$ cd swete-admin
$ mv conf.db.ini.sample conf.db.ini
$ vi conf.db.ini
```

We change the contents of `conf.db.ini` to be:

```
[_database]
   host=localhost
   name="opencart_fr"
   user="opencart_fr"
   password="password"
```

LISTING 1

```
$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 1631248
Server version: 5.1.52 Source distribution

Copyright (c) 2000, 2010, Oracle and/or its affiliates. All
rights reserved. This software comes with ABSOLUTELY NO
WARRANTY. This is free software, and you are welcome to
modify and redistribute it under the GPL v2 license

Type 'help;' or '\h' for help. Type '\c' to clear the
   current input statement.

mysql> create database opencart_fr;
Query OK, 1 row affected (0.00 sec)

mysql> create user opencart_fr@localhost identified by
   'password';
Query OK, 1 row affected (0.00 sec)

mysql> grant all privileges on opencart_fr.* to
   opencart_fr@localhost;
Query OK, 1 row affected (0.00 sec)

mysql> flush privileges;
Query OK, 0 rows affected (0.00 sec)

mysql> exit;
```

Step 5: (Optional) Set Up DNS and Virtual Host

For our example, we're setting up a subdomain for the French proxy so we need to set up DNS and add a VirtualHost entry in the Apache config file to point to the SWeTE installation. This step is optional. The VirtualHost section of the apache config file looks like:

```
<VirtualHost *:80>
   ServerAdmin info@example.com
   DocumentRoot /home/phparch/swete
   ServerName fr.opencartdemo.weblite.ca
   ServerAlias fr.opencartdemo.weblite.ca
   ErrorLog logs/fr.opencartdemo.weblite.ca-error_log
   CustomLog logs/fr.opencartdemo.weblite.ca-access_log common
</VirtualHost>
```

After changing the Apache config file, we need to restart the web server for the changes to take effect.

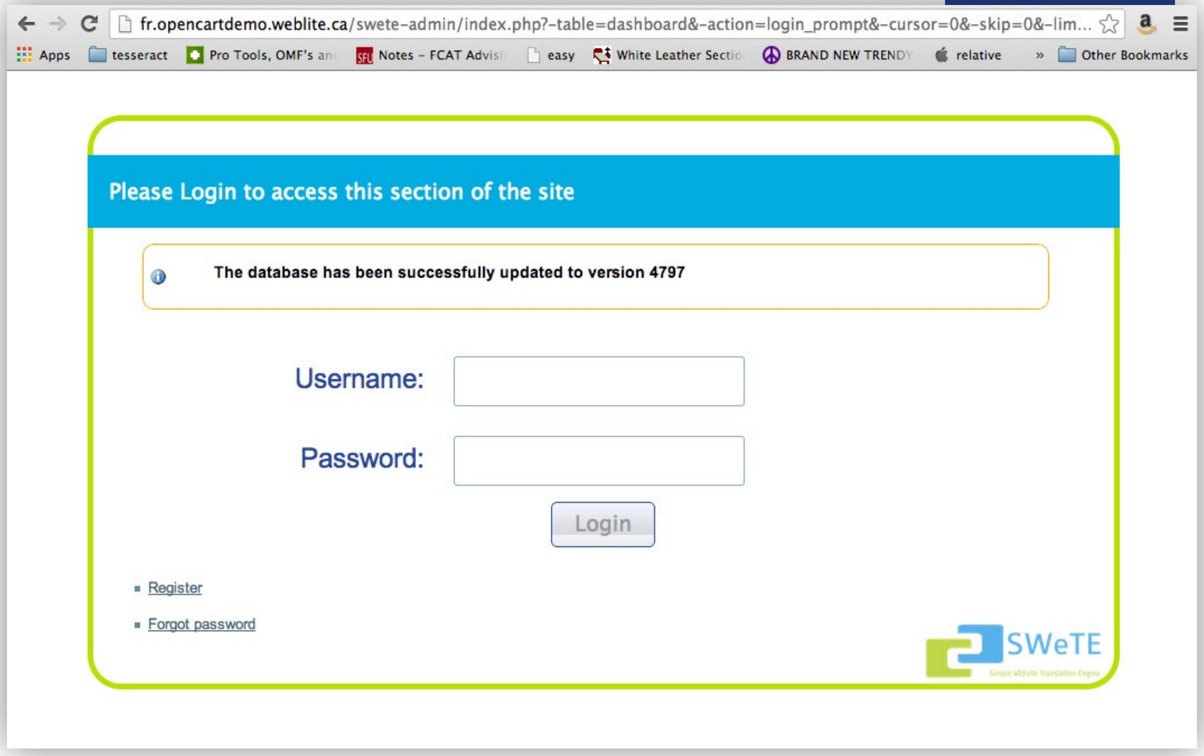
Step 6: Logging into SWeTE for the First Time

Once all of this configuration is done, open a web browser and point it to the `swete-admin` directory of the SWeTE installation:

http://fr.opencartdemo.weblite.ca/swete-admin/

This page takes a few seconds to load the first time because SWeTE needs to run its install scripts to populate the database. Once this is complete, you should see a login page as shown in Figure 2.

FIGURE 2



fr.opencartdemo.weblite.ca/swete-admin/index.php?-table=dashboard&-action=login_prompt&-cursor=0&-skip=0&-lim...

Please Login to access this section of the site

The database has been successfully updated to version 4797

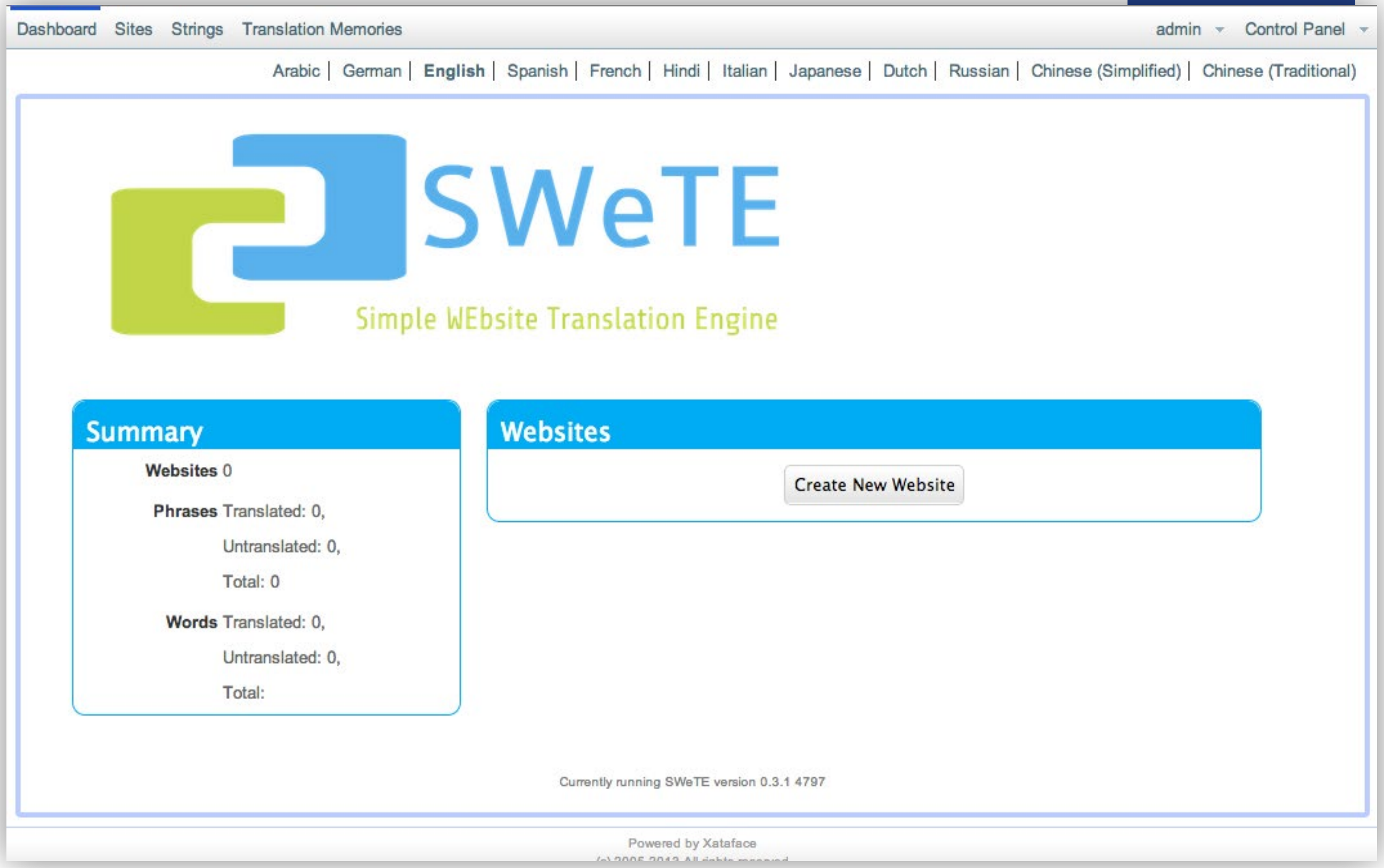
The default login username and password are:

username: admin
password: password

After logging in to SWeTE, you will be directed to the dashboard as shown in Figure 3. Before proceeding to set up the Open Cart demo proxy, let's take a moment to get acquainted with the user interface. There are some main tabs at the top of the page:

- **Dashboard** - The page we are currently on.
- **Sites** - Contains the set of websites that are currently set up to be proxied. This is where you can add, remove, and edit the configuration of websites that are being hosted by this SWeTE instance.
- **Strings** - Contains the set of strings in the translation dictionary. This is where you can view, search, translate, edit, import, and export strings and translations for your websites.

FIGURE 3


Dashboard
Sites
Strings
Translation Memories
admin
Control Panel
Arabic | German | English | Spanish | French | Hindi | Italian | Japanese | Dutch | Russian | Chinese (Simplified) | Chinese (Traditional)
SWeTE
Simple WEbsite Translation Engine
Websites
Create New Website
Powered by Xataface

Summary

Websites 0

Phrases Translated: 0,
Untranslated: 0,
Total: 0

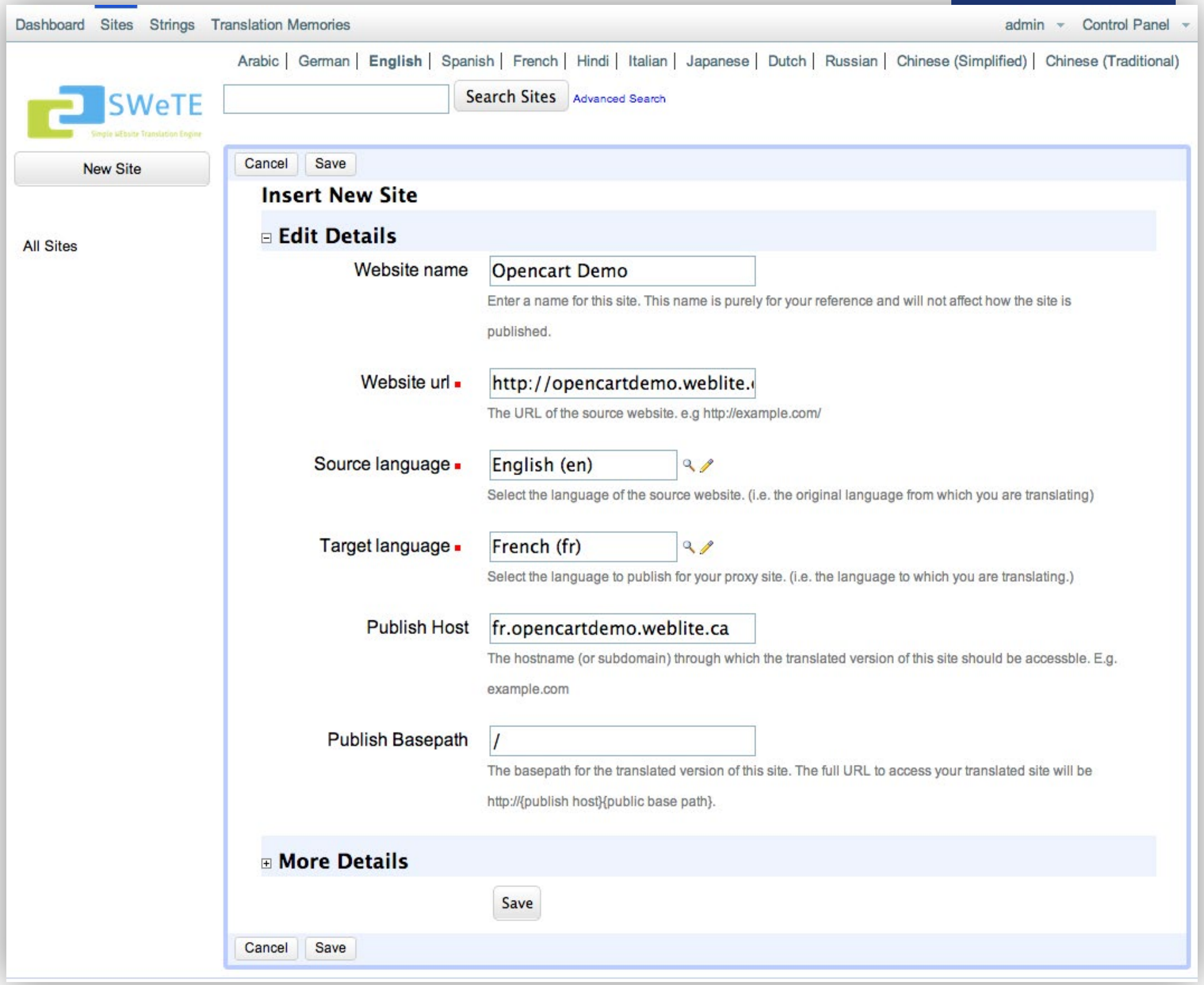
Words Translated: 0,
Untranslated: 0,
Total:

Currently running SWeTE version 0.3.1 4797

Adding the Open Cart Website

Now, let's proceed to create a proxy site for our shopping cart. On the dashboard, click the "New Website" button. This should take you to the "New Website" form. We'll fill in the fields as shown in Figure 4.

FIGURE 4


Dashboard
Sites
Strings
Translation Memories
admin
Control Panel
Arabic | German | English | Spanish | French | Hindi | Italian | Japanese | Dutch | Russian | Chinese (Simplified) | Chinese (Traditional)
SWeTE
Search Sites
Advanced Search
New Site
All Sites
Cancel
Save

Insert New Site

Edit Details

Website name: Opencart Demo
Enter a name for this site. This name is purely for your reference and will not affect how the site is published.

Website url: http://opencartdemo.weblite.
The URL of the source website. e.g http://example.com/

Source language: English (en)
Select the language of the source website. (i.e. the original language from which you are translating)

Target language: French (fr)
Select the language to publish for your proxy site. (i.e. the language to which you are translating.)

Publish Host: fr.opencartdemo.weblite.ca
The hostname (or subdomain) through which the translated version of this site should be accessble. E.g. example.com

Publish Basepath: /
The basepath for the translated version of this site. The full URL to access your translated site will be http://{publish host}{public base path}.

More Details

The important fields are:

- **Website URL** - This should be the full URL (including "http://") of the English shopping cart. In my case, I have it installed at http://opencartdemo.weblite.ca/.
- **Source Language** - This is the language of the shopping cart.
- **Target Language** - This is the language to which we are translating the site using SWeTE.
- **Publish Host** - This is the hostname (i.e. domain or subdomain) to access the proxy site through. In our case, we set up the virtual host "fr.opencartdemo.weblite.ca"" for this purpose.
- **Publish Basepath** - In our case, we will leave this as the root (/) of the site. It is possible to set up a site to run in a subdirectory also.

With these fields filled in, click "Save". Now if you return to the Dashboard, (by clicking the "Dashboard" link in the upper left), you should see the Open Cart Demo website listed in the "Websites" section.

Testing the French Proxy

Notice that there are two links to the right of the Open Cart Demo listing: “English” and “French”. Right click on the “English” link and select “Open in new Window” (or the equivalent for your browser). This should take you to the source shopping cart (at http://opencartdemo.weblite.ca). If you do the same (open in new window) for the “French” link, you will be taken to the URL where the French SWeTE proxy is hosted (http://fr.opencartdemo.weblite.ca). If everything went well, you should see the exact same content as on the English site.

On my demo site, I ran into a glitch where the homepage wasn’t proxied properly. This is a small glitch in SWeTE with how it handles requests for the root directory, without specifying a subpath. In order to resolve this issue, I added a mod_rewrite rule in the SWeTE `.htaccess` file to rewrite requests for the root directory so that they go to `index.php`. If we were proxying a different kind of application, like a .Net application, we’d probably change this do something like `default.aspx`.

After making the change, my `.htaccess` file looks like :

```
RewriteEngine On
RewriteRule ^$ index.php
RewriteRule ^swete-admin/ - [L,NC]
RewriteRule . swete-admin/index.php?-action=swete_handle_request [L]
```

(All I did was add the rule `RewriteRule ^$ index.php`).

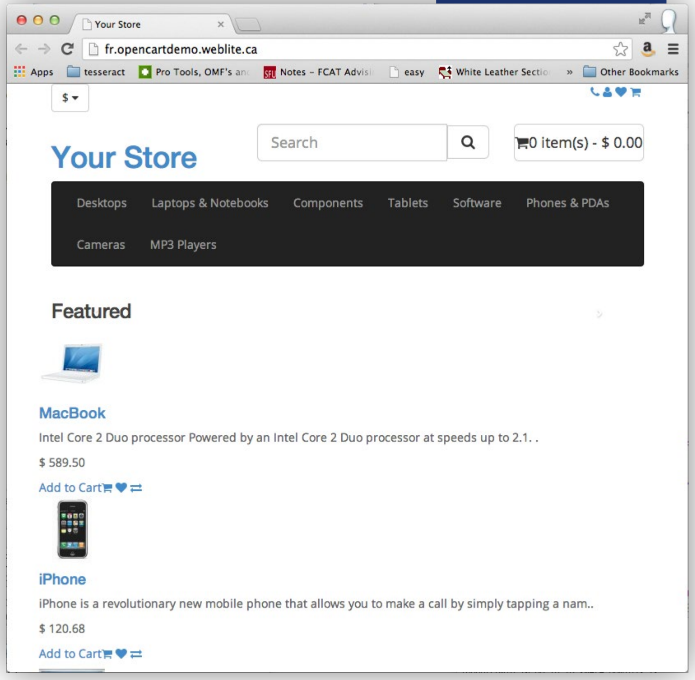
The other two rules in this `.htaccess` file hint at how SWeTE works under the hood.

Now let’s try to reload our proxy site from http://fr.opencartdemo.weblite.ca). You should see the shopping cart (shown in Figure 5) exactly as it appeared when loading the application directly.

At this point, SWeTE is running simply as a reverse proxy. This can have benefits in itself (e.g. for performance caching and load-balancing), but our primary objective in using SWeTE is to translate the site.

One cool aspect of a SWeTE site is that it should act as a fully functional proxy of the source application. We should be able to access not only the public portions of the application through the SWeTE proxy, but also the administrative interface with its full functionality. To demonstrate, let’s visit the Open Cart administration section through the French proxy (located at http://fr.opencartdemo.weblite.ca/admin). This is not to be confused with the SWeTE admin console located at http://fr.opencartdemo.weblite.ca/swete-admin/. As expected, it allows me to log in just as if I were using the English application.

FIGURE 5



In my tests, I did run across one small bug when accessing the admin page through the Proxy. I ran into a JavaScript error that caused the charts not to show up properly. Upon inspection, I found that the JavaScript contained some embedded HTML strings that caused the SWeTE HTML parser to choke (it uses the `DOMDocument` HTML parser). In order to resolve this issue, I needed to add a directive to SWeTE inside the `.htaccess` file so that it encodes JavaScript in a way that won't interfere with the HTML parser. After doing this, my `.htaccess` file looks like:

```
RewriteEngine On
RewriteRule .* - [E=ENCODE_SCRIPTS:1]
RewriteRule ^$ index.php
RewriteRule ^swete-admin/ - [L,NC]
RewriteRule
    . swete-admin/index.php?-action=swete_handle_request [L]
```

What this essentially does is set the `ENCODE_SCRIPTS` environment variable which SWeTE responds to. SWeTE supports a handful of these environment variables for customizing such things as caching rules and parsing rules. The full list of environment variables can be found in the SWeTE manual.

Capturing Strings

The translation process in SWeTE has two parts:

1. Capture the strings on the site and load them into the translation dictionary.
2. Translate the strings in the translation dictionary by using the web-based translation form, exporting them (to CSV, XLIFF, or XLS) and sending them to a translator for translation, or sending them to Google translate for an automatic machine translation.

The string capturing process is kind of like holding up a fishing net to catch strings of an HTML response as they pass through the proxy. If string capturing is enabled, then SWeTE will make note of any strings that it encounters that haven't yet been added to the translation dictionary. This happens while the page response is being processed so it is seamless. In other words, if you want to import the strings for a particular webpage into the translation dictionary, all you have to do is enable string capturing and load your page of interest in the web browser.

Let's walk through how we set this up.

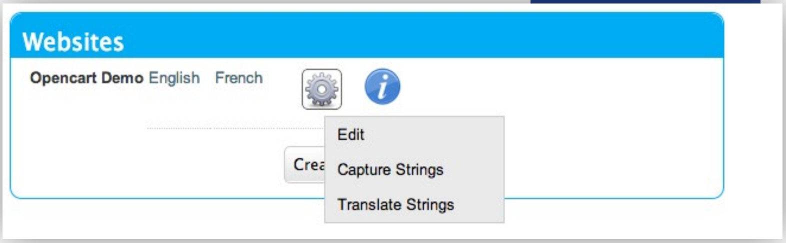
Step 1. Enable String Capturing.

From the SWeTE Dashboard, click on the "menu" icon to the right of the "Open Cart Demo" website listing and select "Edit Site" (as shown in Figure 6). This will bring you to the "Edit Website" form for the website.

Click the `[+]` icon next to the "More Details" heading and check the box next to "Log Translation Misses".

Click save when done.

FIGURE 6



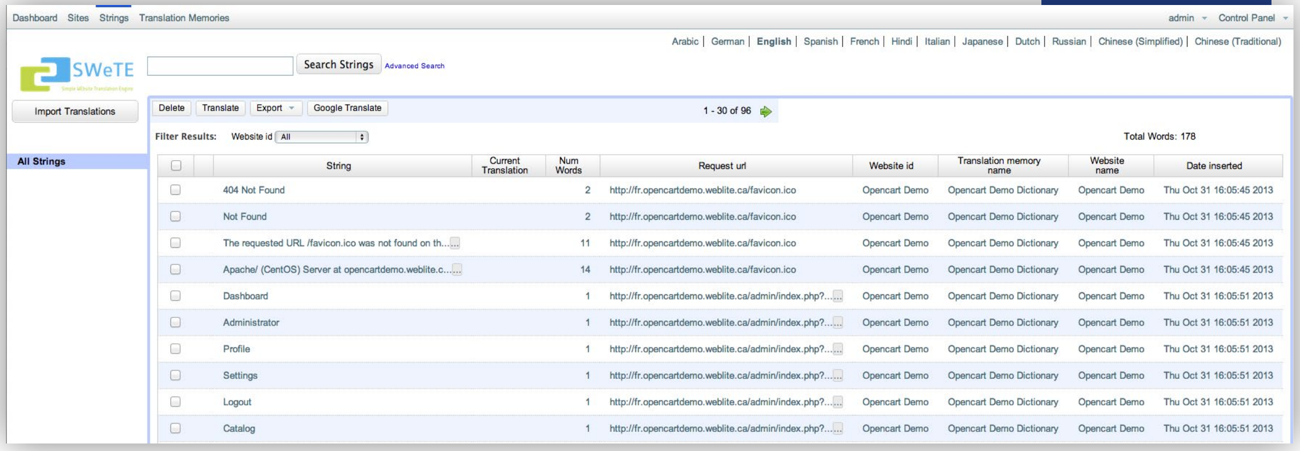
Step 2. Refresh the Open Cart Admin Page.

In a separate window, open the Open Cart admin page to its dashboard. By hitting "Refresh" in the browser, we are able to cause SWeTE to capture all of the strings on this page.

Step 3. Check the "Strings" Section of the SWeTE Admin

All of the strings on the admin page should now appear in the Strings section of the SWeTE admin page, as shown in Figure 7.

FIGURE 7



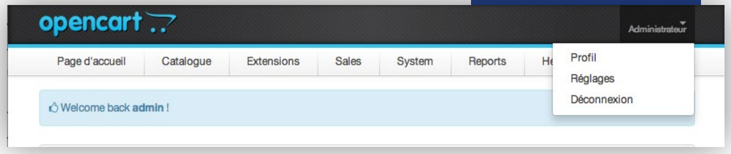
String	Current Translation	Num Words	Request url	Website id	Translation memory name	Website name	Date inserted
404 Not Found		2	http://fr.opencartdemo.weblite.ca/favicon.ico	Opencart Demo	Opencart Demo Dictionary	Opencart Demo	Thu Oct 31 16:05:45 2013
Not Found		2	http://fr.opencartdemo.weblite.ca/favicon.ico	Opencart Demo	Opencart Demo Dictionary	Opencart Demo	Thu Oct 31 16:05:45 2013
The requested URL /favicon.ico was not found on th...		11	http://fr.opencartdemo.weblite.ca/favicon.ico	Opencart Demo	Opencart Demo Dictionary	Opencart Demo	Thu Oct 31 16:05:45 2013
Apache/ (CentOS) Server at opencartdemo.weblite.c...		14	http://fr.opencartdemo.weblite.ca/favicon.ico	Opencart Demo	Opencart Demo Dictionary	Opencart Demo	Thu Oct 31 16:05:45 2013
Dashboard		1	http://fr.opencartdemo.weblite.ca/admin/index.php?...	Opencart Demo	Opencart Demo Dictionary	Opencart Demo	Thu Oct 31 16:05:51 2013
Administrator		1	http://fr.opencartdemo.weblite.ca/admin/index.php?...	Opencart Demo	Opencart Demo Dictionary	Opencart Demo	Thu Oct 31 16:05:51 2013
Profile		1	http://fr.opencartdemo.weblite.ca/admin/index.php?...	Opencart Demo	Opencart Demo Dictionary	Opencart Demo	Thu Oct 31 16:05:51 2013
Settings		1	http://fr.opencartdemo.weblite.ca/admin/index.php?...	Opencart Demo	Opencart Demo Dictionary	Opencart Demo	Thu Oct 31 16:05:51 2013
Logout		1	http://fr.opencartdemo.weblite.ca/admin/index.php?...	Opencart Demo	Opencart Demo Dictionary	Opencart Demo	Thu Oct 31 16:05:51 2013
Catalog		1	http://fr.opencartdemo.weblite.ca/admin/index.php?...	Opencart Demo	Opencart Demo Dictionary	Opencart Demo	Thu Oct 31 16:05:51 2013

Translating Strings

Now that we've loaded some strings into the dictionary, we can translate them by checking the box next to them and clicking the "Translate" button at the top or bottom of the table. This brings us to a translation form (shown in Figure 8) that lists the English string above the editable text box where we can enter a French translation.

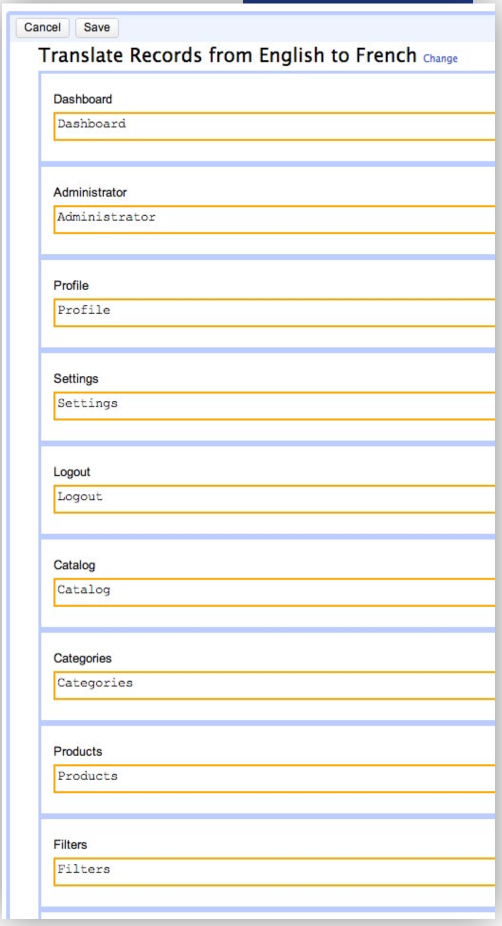
After entering some translations, we can return to the Open Cart admin page on the French proxy site and hit refresh. This results in those strings appearing in French (see Figure 9).

FIGURE 9



For the rest of the strings, I'm going to cheat a little bit and just use Google to translate them all. The "Site Edit Form" in SWeTE allows you to specify your Google translation API key. You will then be able to send batches of strings directly to Google for translation. Sending a batch of strings to Google is accomplished by checking the box beside the strings we want to translate and clicking the "Google Translate" button along the top bar (in the Strings section).

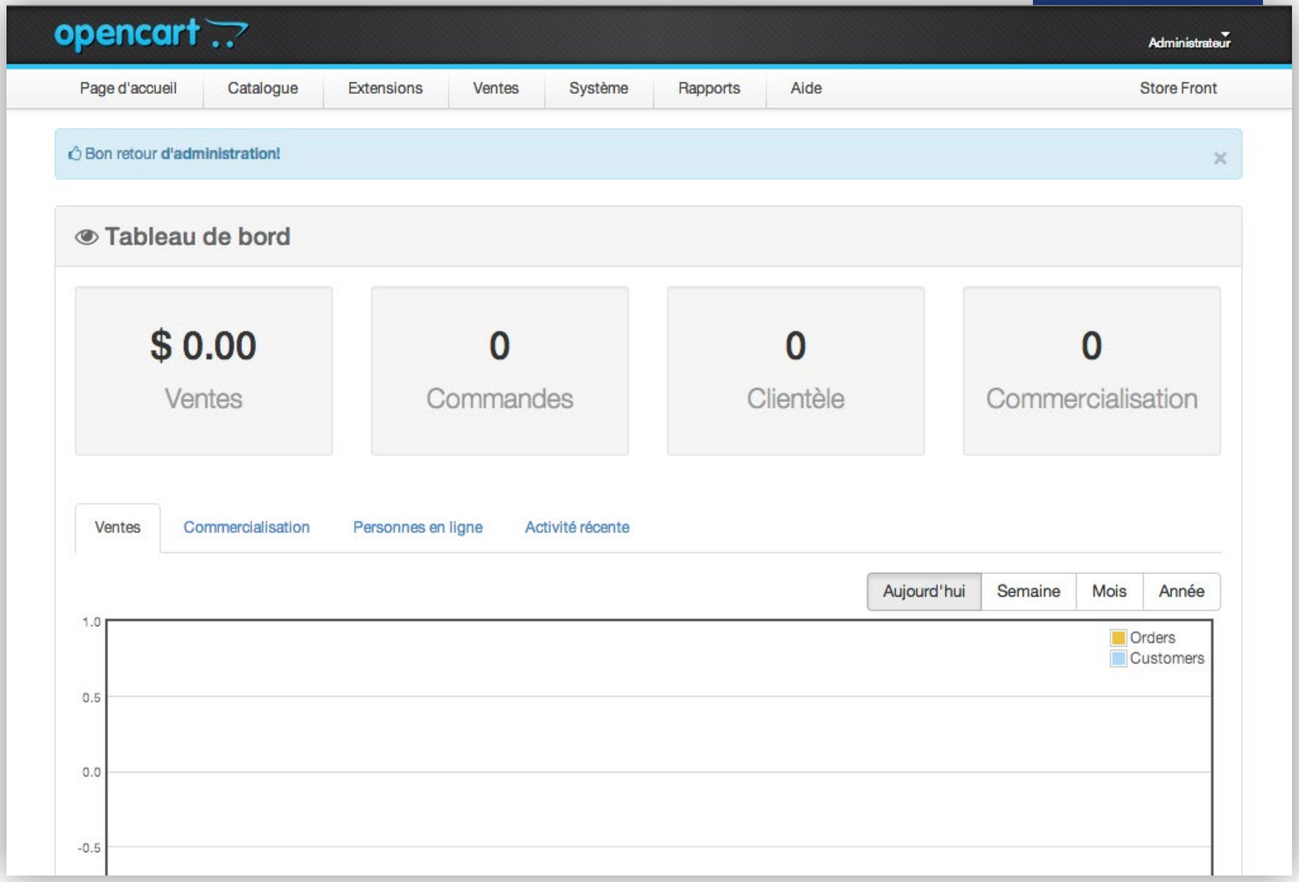
FIGURE 8



Dashboard	Dashboard
Administrator	Administrator
Profile	Profile
Settings	Settings
Logout	Logout
Catalog	Catalog
Categories	Categories
Products	Products
Filters	Filters

After loading a few more pages and translating them using Google, we are able to navigate our French shopping cart in 100% French (shown in Figure 10).

FIGURE 10



Bon retour d'administration!

Caveats

At this point, you might be tempted to say that we're done. By traversing the website with string capturing enabled, we've been able to capture all of the strings in the site and translate them. What is left to do? In my experience, this initial process of installing SWeTE, capturing the strings, and translating the strings will usually get us about 99% of the way the way there. Unfortunately, that last 1% is just as important as the first 99%.

What is the Remaining 1%?

The actual content of the 1% will depends on the application. The types of things that SWeTE doesn't handle out of the box are:

- **Translating Non-HTML Content** - If the application loads strings inside JavaScript source files or JSON via Ajax, you may need to implement some custom post-processing for SWeTE to handle it.
- **Custom Styles** - You may need to provide custom styles for the translated version to satisfy the requirements of a different audience. SWeTE enables you to create your own custom styles to target the translated site, but it won't design these rules for you.
- **Translation of Images and Videos** - If your application uses images with embedded text, SWeTE won't translate these by default. SWeTE allows you to specify alternate images and videos to be used for different languages, but you would need to create those resources first.

- **Formatted Strings** - Most applications will include some strings that are created at runtime using a string template and placeholders that are replaced with variables from the environment. "Welcome back, admin!" is a string from the Open Cart admin page that is customized for each user on the site. If the site has a million users, then SWeTE would need a million separate entries in the translation dictionary - one for each variation. SWeTE supports markup to specify that certain parts of a string are variables or placeholders, but you'll need to either add this markup in the source application's HTML or implement a SWeTE preprocessing filter to add this markup at runtime.
- **Content that Doesn't Pass Through HTTP** - Some applications send email messages in response to certain actions. These don't pass through the SWeTE proxy like the rest of the application content, so they need to be translated separately. SWeTE does pass an HTTP header to the application to indicate the language in which the user is accessing the application, and this information can be used by the application to modify how the email is sent. It also can be accessed as a web service so it is possible for the web application to pass the email contents through SWeTE via this web service to obtain a translation before sending the email. In any case, this is one part that will require extra care above and beyond the initial setup and translation of the SWeTE site.

Implementing a Site Delegate Class

When you get down to the last 1%, you may find it helpful to implement a PHP delegate class for your website in SWeTE. A delegate class allows you to define PHP methods that are called at various stages of the HTTP request/response loop to assist in processing.

Delegate classes should be placed in a PHP file at `swete-admin/sites/<site id>/Delegate.php`, and the class name should be: `sites_<site id>_Delegate`.

In the case of my demo site, the site ID is 428, so my delegate class is located in the file: `swete-admin/sites/428/Delegate.php`, and the contents should be:

```
class sites_428_Delegate {
}
```

SWeTE supports several hooks for this delegate class. Three of the most commonly implemented are:

- `fixHtml($html)` - This method is called before SWeTE parses the HTML content of an HTTP response from the application. It receives a string with HTML content and should return a string with either the same content it received (i.e. if no changes were necessary to "fix" the HTML) or a modified version of it. This method is useful for fixing pages that contain invalid HTML that causes problems for the HTML parser. It can also be helpful for making changes to the content using regular expressions.
- `preprocess($dom)` - This method receives a DOMDocument object which is a parsed representation of the HTML page that is to be translated. You can modify this object as desired to make changes to it before it is passed to the document translator. This can be useful for adding SWeTE markup such as HTML attributes to signify that a section should or should not be translated or to specify that certain sections should be treated as variable placeholders.
- `preprocessHeader(&$headers)` - This hook receives the headers that have been returned from the application as part of the HTTP response. It provides you with an opportunity to modify, add, or remove headers as you would want them to be returned to the client. This is, for example, useful for changing the destination of a redirect header.

You can see a default implementation of the site delegate class in Listing 2.

LISTING 2

```
class sites_428_Delegate {
   public function fixHtml($html){
      return $html;
   }

   public function preprocessHeaders(&$headers){

   }

   public function preprocess($dom){

   }
}
```

Dealing with Personalized Strings

As was mentioned previously, the Open Cart admin page contains the string "Welcome back, admin!", which will be slightly different for each logged in user. This will result in a separate string loaded into the translation dictionary for each user in the system, which is not ideal. SWeTE offers a solution to this problem by allowing you to specify that certain HTML tags should be treated as variables or placeholders. If you wrap a string in a span tag as follows:

```
<span data-swete-translate="0">some value </span>
```

SWeTE will ignore its contents for translation. Hence, we can take a string like:

```
Welcome back <strong>admin</strong>
```

change it to:

```
Welcome back <span data-swete-translate="0" style="font-weight:bold">admin</span>
```

and SWeTE will then recognize that "admin" is just a variable placeholder. Then, any string in the form of:

```
Welcome back <span data-swete-translate="0" … >xxx</span>
```

can use a single dictionary entry for its translation.

There are two ways to fix this with the Open Cart application:

1. Change the HTML that Open Cart itself outputs to wrap all variables like this in span tags with the `data-swete-translate="0"` attribute.
2. Add this tag at runtime using one of SWeTE's preprocessing hooks.

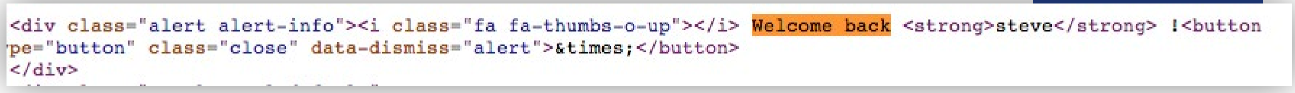
For this example, we're going to try option 2. That way, we don't need to make any changes to the Open Cart source at all. This can be done by adding a regular expression pattern replacement in the site delegate's `fixHtml()` method as follows:

```
public function fixHtml($html){
   $html = preg_replace(
      '#Welcome back <strong>([^<]+)</strong>#',
      'Welcome back <span data-swete-translate="0" style="font-weight:bold">
          $1</span>',
      $html
   );
   return $html;
}
```

Since this regular expression will be applied to every page, any page that contains that string will be modified such that the name of the logged in user is marked as a variable.

Let's run a quick test to make sure that this worked. If we now reload the Open Cart Dashboard in the French Proxy and look at the HTML source for the page, we can see that our change was made (Figure 11).

FIGURE 11



```
<div class="alert alert-info"><i class="fa fa-thumbs-o-up"></i> Welcome back <strong>steve</strong> !<button
pe="button" class="close" data-dismiss="alert">&times;</button>
</div>
```

If we check out the "Strings" tab in the SWeTE admin console and perform a find for "Welcome", we see that there is a string that just says "Welcome back ". Clicking on this string reveals that there is actually some hidden markup:

```
<g id="1"></g> Welcome back <v id="2"></v> !
```

This markup is a glimpse of SWeTE's normalization function which it performs on strings before they are passed to the translation dictionary. The `<g id="1"></g>` tag represents any HTML tag with a closing tag. This appears because there is actually an `<i></i>` HTML inline tag just preceding the "Welcome back" text in the application output. The `<v id="2"></v>` is a variable marker, meaning that it represents any HTML tag and contents that contain the `data-swete-translate="0"` attribute.

If we translate this string and reload the Open Cart dashboard through the proxy, we should see that this is translated no matter which user we log in with. (Shown in Figure 12 and Figure 13, logged in as "steve" and "admin", respectively).

FIGURE 12

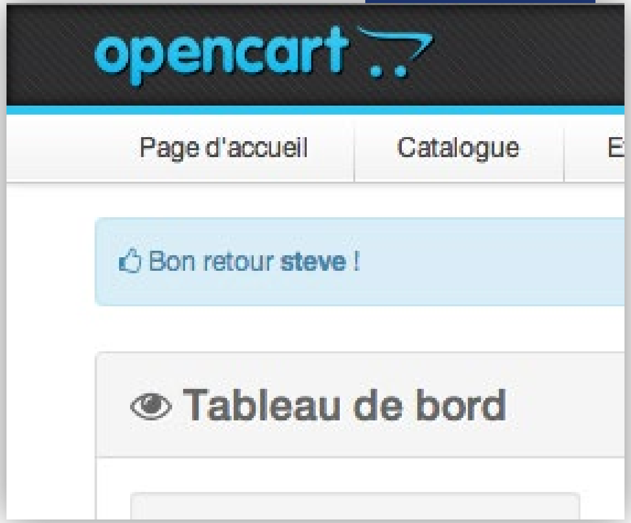
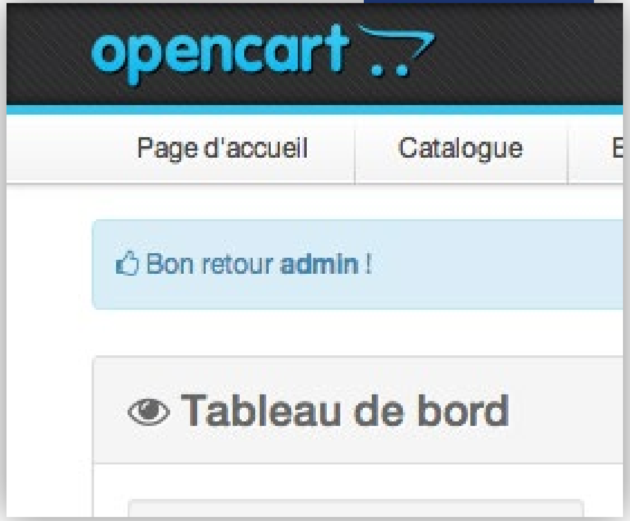


FIGURE 13



Summary

Conventional strategies for i18n in PHP present enormous challenges in terms of both development and maintenance. Translation proxies allow you factor out most (or all) i18n logic into a separate layer which is largely decoupled from the rest of the application. This can lead to substantial savings in both development and maintenance resources.

In this article, we saw how to use SWeTE (a PHP translation proxy) to create a French version for an English shopping cart. The same process can be followed to localize any web application, though each application will present its own challenges. SWeTE provides both a gentle learning curve (you should be able to set up a translation proxy for your application in under 20 minutes) and a great degree of flexibility. Most applications can be internationalized with very little to no customization. For those applications that require more attention, SWeTE provides several avenues for customization including:

1. Environment variables
2. Hints and directives embedded in HTML markup
3. Custom PHP event handlers

This article dealt with a basic case, but expanding these examples to a more complex case should be fairly straight forward.

As with any open source project, SWeTE will improve as adoption increases, but it already includes most of the features necessary for maintaining a multilingual application. If you want to find out more, try it out, or contribute, you can visit the SWeTE website at http://swete.weblite.ca.

STEVE HANNAH is the creator and primary developer for SWeTE. In addition to his work developing, maintaining, and contributing to open source software projects, he blogs periodically about software issues on his blog http://sjhannah.com.

@shannah78