Translating Blesta

Table of Contents

Did you know?

A language translation utility is provided at http://translate.blesta.com.This project is intended to facilitate the crowd sourced translation of Blesta into many languages and participation is highly encouraged.

Language in Blesta is handled by a collection of language files, heretoafter referred to as a "language pack." Language packs are primarily located in the language directory **/installdir/language/**, but additional files may be located in various other locations including Modules, Payment Gateways, Plugins, and some components or helpers. The default language in Blesta is English, US located in **/installdir/language/en_us/**.

Why not place all language in a single directory?

Allowing Modules, Gateways, and Plugins to specify their own language packs allows for a simplified installation process and helps ensure users will successfully install third-party add-ons.

Language Packs

Language packs must be named according to ISO 639-1 and 3166-1 standards. For example, English, US is stored in language pack **en_us**. In addition to a collection of language files, each language pack must contain a file of the same name as its directory with a single line defining the name of the language in its native format.

/installdir/language/en_us/en_us

```
English, US
```

Language Direction

Language direction may be defined in the app_controller.php language file. This identifies which direction text should read (ltr for left-to-right, or rtl for right-to-left).

/installdir/language/en_us/app_controller.php

```
$lang['AppController.lang.dir'] = "ltr";
```

If not defined language will default to the "ltr" (left-to-right) direction.

Anatomy of a Language File

Each language file should correspond to the class name in which the language is used. Take for example **/installdir/language/en_us/admin_clients.php**. This file contains all language associated with the AdminClients controller and related views.

A language file may contain one or more language entries. Entries are represented by array key/value pairs that correspond to an index and definition, respectively.

/installdir/language/en_us/admin_clients.php

```
$lang['AdminClients.index.boxtitle_browseclients'] = "Browse Clients";
```

Index Formats

Indexes are primarily comprised of three dot-segments. The two main formats are:

1. [ClassName].[methodName].[entry_name]
2. [ClassName].[!error/!success/!notice].[entry_name]

The exclamation point identifies an entry as not belonging to any particular method. This convention makes it easy to identify error, success, and notice messages that may be used by various methods.

Variable Substitution

Some definitions contain variable substitutions, which can be identified by their syntax: **%N$s**, where N is a number (1, 2, ...). The $ in the variable name is escaped by a backslash (\) as the $ character has special meaning in double-quoted ('') strings.

/installdir/language/en_us/admin_clients.php

```
$lang['AdminClients.!success.makepayment_processed'] = "The payment was successfully processed for %1\$s.
Transaction Number: %2\$s"; // %1$s is the payment amount, %2$s is the transaction number
```

Variables can appear in any order

The definition above could easily have been written as "The payment was successfully processed! Transaction Number: %2\$s, Amount: %1\$s".

Using Definitions

Loading Language Files

Language files may be loaded from anywhere in Blesta. By default the Language library looks for language files in **/installdir/language/**, but you can override this value.

From somewhere in a controller or model

```
Language::loadLang("lang_file"); // loads from /installdir/language/[language_pack]/lang_file.php
Language::loadLang("lang_file", null, PLUGINDIR . "my_plugin" . DS . "language" . DS); // loads from /installdir
/plugins/my_plugin/language/[language_pack]/lang_file.php
```

Displaying Definitions

Controllers and Models must invoke the Language library directly to access language definitions.

From somewhere in a controller or model

```
echo Language::_("ControllerName.methodName.entry_name");
```

Views have access to a wrapper method and should utilize the following syntax, instead:

From somewhere in a view

```
<?php $this->_("ControllerName.methodName.entry_name");?>
```

To set variables for substitution, add them beginning as the 3rd parameters to Language::_() or $this->_(). For example:

From somewhere in a view

```
<?php $this->_("ControllerName.methodName.entry_name", false, $param1, $param2, ..., $paramN);?>
```