1 Apache::Language - Perl transparent language support for Apache modules and mod_perl scripts

1 Apache::Language - Perl transparent language support for Apache modules and mod_perl scripts

1.1 SYNOPSIS

```
In YourModule.pm:

sub handler {
my $r = shift;
use Apache::Language;
my $lang = Apache::Language->new($extra_args);
#$lang is now a hash ref that will automacigally pick the right language

print $lang->{'Error01'} if exists $lang->{'Error01'};

foreach ( keys %$lang ){
  print "$_ is " . $lang->{$_};
  }

[...]
}
```

1.2 DESCRIPTION

The goal of this module is to provide a simple way for mod_perl module writers to include support for multiple language requests.

This is version 0.03, and it's a complete rewrite from the ground-up of the previous release. It's still backward-compatible with the other releases, but now it's much more advanced.

An Apache::Language object acts like a language-aware hash. It stores key/language/values triplets. Using the Accept-Language: field sent by the web-client, it can pick the best fit language for that specific client. It's usage is transparent and should prove to be quite convenient (and hopefully, efficient).

The method used to store/fetch information is now completely modular and will allow easy creation of new storage methods thru a simple API (see the API section).

1.2.1 BASIC USAGE EXAMPLE

This section will describe the easiest way to start using Apache::Language.

Apache::Language is used to create a hash that will contain key/language/value triplets. Say you are building a module that prints a few error messages, but since your users speak 3 different languages, you'd like your module to be nice to them and print the messages in their native language.

For this approach to work, a few things are needed. First, the client software (Netscape/IE/lynx, etc.) should send an Accept-Language: header telling the webserver what languages it's user understands. This might sound simple, but as a web-surfer, did you set that language setting in your browser? Most likely, you didn't. So the first step is to correctly configure your browser and try to make sure your users/visitors will have done the same, or they might not get what you want them to read.

Secondly, you must store the messages you want to display in each avaliable languages. For this example, we will use the default LanguageHandler Apache::Language::PlainFile that ships with this distribution. To do that, suppose your module is named Apache::MyModule and it's stored in a file named MyModule.pm. You will only need to edit a file named MyModule.dic in the same place your module is stored. The format of that file is : (for more information see *Apache::Language::PlainFile*(3)).

```
error01:en

Error Message 01

error01:fr

Message d'erreur 01

error02:en

Error Message 02
```

Once that file contains your error messages, you're all set. Just add this to the top of your module:

```
use Apache::Language
my $lang = Apache::Language::new($extra_args)
```

Then $lang will be a language-enabled hash reference that you can use like this:

```
print $lang->{error01}
```

That line will print your error message 01 in the best language for the client that your module served. Of course, there are a lot more things you can do with Apache::Language. All these features are explained below.

1.3 DETAILLED USAGE

The key to using Apache::Language is to understand the the data retrieving/storing is now done in a manner similar to mod_perl content handlers. That's why I called them Language Handlers. A language handler is a perl module that is specialized in storing/fetching key/language/values triplets.

Apache::Language simply manages those modules and makes interaction with other modules painless. For every request, Apache::Language maintains a stack of the LanguageHandlers configured for that request's location. Asking each of them in order to try finding a value for the requested key that would satisfy the client acceptable languages.

That way you can have a number of language handlers, one looking up in a system-wide dictionnary, one looking up in a module-specific dictionnary and one looking up a RDBM. Each of them would be queried until an acceptable response is found. This makes maintaining language definitions pretty versatile and configurable.

With this package is shipped 2 basic LanguageHandlers: *Apache::Language::PlainFile* and *Apache::Language::DBI*. Check out their respective man pages for more specific information. And more of those LanguageHandlers will be shipped in future versions and if you need a specific language handling

module, simply write your own. Read the API section for more information about writing your own LanguageHandlers and check the 2 mentionned above, they are nice examples.

1.4 CONFIGURATION DIRECTIVES

There are a some configurations directives you can use to customize how Apache::Language handles requests.

- **LanguageHandler**

  This is the main directive. It sets the language handlers for a specific location. It's usage is like Index-Options. Meaning it supports +/- notations.

  With a list of handlers without any +/- signs it simply sets the handler stack for the current location to those.

  With a list of handlers all with +/- signs, it merges them with the parent configuration:

```
LanguageHandler Apache::Language::handler1 Apache::Language::handler2
<Location /test/>
LanguageHandler +Apache::Language::handler3 -Apache::Language::handler2
</Location>
```

  Will result in a handler stack of [Apache::Language::Handler1 , Apache::Language::Handler3] for requests in the /test/ directory

  2 things to note though:

  The default handler (Apache::Language::PlainFile) will always be the last handler in the stack.

  If a module isn't found, it will be tried again with Apache::Language:: prepended to it, so you can write LanguageHandler handler2 instead of LanguageHandler Apache::Language::handler2

- **LanguageForced**

  This directive is given a list of languages that should exclusively be tried to please the client. This a way to mark a certain <Location> as specifically danish, for example. If no definition is found for that language, the standard 'sorry, no language definition found' message is printed.

- **LanguageDefault**

  Provided with a list of languages, this directives sets the default languages it should try to send if none of the languages requested by the client can be found.

- **DefaultLanguage**

  This is a not an Apache::Language directive, but in the absence of a LanguageDefault directive, that setting will be used to set the default languages

- **LanguageDebug**

  This will set the level of debugging information sent to the apachelog. Values ranges from 0-4. 4 being **very** noisy and 0 very quiet. 0 will still warn you about fatal errors. A setting of -1 simply doesn't log anything. Nothing.

  There are also a few other debugging options avaliable

  LanguageDebug Prefix=(LanguageDebug Postfix=) LanguageDebug NotFoundPrefix=[[[LanguageDebug NotFoundPostfix=]]]

  They will prefix/postfix language tokens with the specified delimiters with special symbols, usefull for debugging

1.5 STATUS Handler

If you have the Apache::Status module loaded before Apache::Language, a custom status module will be installed. Usually accessed thru /perl-status, it will give you an Apache::Language menu with quite a few bits of interesting information about Apache::Language status.

1.6 API

The Apache::Language API is pretty simple and might evolve in future versions. The first argument to all function calls will be the class name of the handler.

The API was developped to provide for: 1. abstraction of the data storage/retrieval and 2. an easy way for other developpers to write new containers without having to modify the whole package.

If you want to write your own Language::Handler, read on. You might also find more interesting information by looking at the couple of LanguageHandlers distributed as part of the Apache::Language package. I also think that new LanguageHandlers should be named under the Apache::Language::* namaspace for simplicity. As of this writing I am have quite a few other LanguageHandlers on the way, so if you are planning on writing a new one, get in touch with me before, maybe I already have some work done I'd be happy to send you.

There is one thing that you must understand before writing your own LanguageHandler. It's how Apache::Language caches information about LanguageHandlers. For performance reasons, and since most mod_perl scripts/modules will want to have their own language definitions, information is cached on a per-Module/script basis. This means that suppose the module Apache::MyHandler uses Apache::Language, all LanguageHandlers serving requests for Apache::MyHandler will be given access to a cache ($data) that's specific to Apache::MyHandler and to another cache that's specific to Apache::MyHandler and the current LanguageHandler. In the case that a certain handler doesn't want this degree of encapsulation (i.e. a dictionnary of common used error messages, system wide) it's free to maintain it's own cache (in it's namespace) and disregard the Module & Handler specific cache. That cache is merely provided so LanguageHandler writers won't have to deal with per Module cache themselves. But you can do without it.

This is what a small dump of the cache might look like : (For the visual types)

#Cached information for the mod_perl Handler Apache::MyHandler1 $VAR1 = 'Apache::MyHandler1'; $VAR2 = bless({ #current LanguageHandler stack 'Handlers' => ['Apache::Language::Handler1', 'Apache::Language::Handler2'], 'Request' => [Apache $r object] #Filename of the Calling package 'Filename' => '/usr/lib/perl5/Apache/MyHandler1.pm', #Language requested by the client 'Lang' => ['en', 'fr-ca', 'fr'], #Package name of the Calling package 'Package' => 'Apache::MyHandler1', #Specific data for LanguageHandler Apache::Language::Handler1 'Apache::Language::Handler1' => { #Is the LanguageHandler a listable one? 'listable' => 1, 'DATA' => { #this is the $cfg object passed to the handler #it's completely private to the Language::Handler 'dbname' => 'DBI::Pg', 'query' => 'select value from language where key=? and lang=?' }, #Is the LanguageHandler a storable one? 'storable' => 1 }, #Specific data for LanguageHandler Apache::Language::Handler2 'Apache::Language::Handler2' => { #Is the LanguageHandler a listable one? 'listable' => 0, 'DATA' => { #this is the $cfg object passed to the handler #it's completely private to the Language::Handler 'option1' => 'read/write', 'debug' => 1 }, #Is the LanguageHandler a storable one? 'storable' => 1 }, 'Config' => [Opaque structure for directory configuration]

```
}, 'Apache::Language' );
```

#Cached information for the mod_perl Handler Apache::MyHandler2 $VAR3 = 'Apache::MyHandler2'; $VAR4 = bless({ #current LanguageHandler stack 'Handlers' => ['Apache::Language::Handler2'], 'Request' => [Apache $r object] #Filename of the Calling package 'Filename' => '/usr/lib/perl5/Apache/MyHandler1.pm', #Language requested by the client 'Lang' => ['en', 'fr-ca', 'fr'], #Package name of the Calling package 'Package' => 'Apache::MyHandler1',

```
  #Specific data for LanguageHandler Apache::Language::Handler2
  'Apache::Language::Handler2' =E<gt> {
       #Is the LanguageHandler a listable one?
       'listable' =E<gt> 0,
       'DATA' =E<gt> {  #this is the $cfg object passed to the handler
                    #it's completely private to the Language::Handler
                   'dbname' =E<gt> 'DBI::mysql',
                   'query' =E<gt> 'select value from language2 where key=? and lang=?'
                 },
       #Is the LanguageHandler a storable one?
       'storable' =E<gt> 1
       },
  'Config' =E<gt>  [Opaque structure for directory configuration]
}, 'Apache::Language' );
```

1.6.1 API data structures

All functions calls to LanguageHandlers are passed these 2 data structures as the second and third arguments. The first argument always being the class name of the LanguageHandler.

$data

This is the data associated with the particular http request. It's also unique to the currently served Module, so it can be used to store some information, but since Apache::Language stores a few things of itself there, be carefull to use keys unique to your module (i.e. $data->{"Apache::Language::YourHandler::key"} =

"private data") It's an Apache::Language object with the following calls avaliable:

- **lang()**

  returns a preference-sorted list of the language the clients requested.

- **handlers()**

  returns the current LanguageHandler stack (list)

- **request()**

  returns the current request (Apache::Request)

- **package()**

  returns the name of the package handling the request

- **filename()**

  returns the filename of the package handling the request

- **extra_args()**

  returns a reference to a list of what was passed to the new constructor

$cfg

This is a private storage space for the current Handler. Handlers can store whatever they want there. That data will be preserved for this Handler/(Calling module/package) pair. Look in Apache::Language::Plain-File for a good exapmle of using that.

1.6.2 Required API functions

- **initialize ($data, $cfg)**

  This is called for the first request so a LanguageHandler can do it's initialization. This will also be called each time a call to the modified() function informs Apache::Language that something has changed.

  It has the opportnity to examine the requesting conditions and if it chooses not to be involved in the handling of that specific requesting Module, it should return anything but L_OK. That way it will be removed from the LanguageHandler stack for that Module and therefore will not be called for that Module anymore

- **modified ($data, $cfg)**

  This is called for each request to know if the datasource changed and should be re-initialized. It should return true if something is modified and re-initialization is needed.

- **fetch ($data, $cfg, $key, $lang)**

  This function can be called in 2 ways. First if there $lang is defined, the LanguageHandler is requested to produce that exact language version for the key $key. This happends if a user requested a specific language with $lang->{key/lang} or to find an entry for the default language if everything else failed to produce a value. It should return undef if unknown.

  Second, if $lang is undefined, this means that the Handler should try to return the best possible match for the current request. $data->lang will return the list of language the client understands, sorted preferred first. But since language-tags allow for variants and dialects, it's now always easy for a module to determine what language it should pick if it knows 'en-us' and 'en' and the clients understands 'en-ca' and 'de'.

- **Everything else**

  If a call is made to an unknown routine, Apache::Language will try to call it on each Handler in the current handler stack until someone can deal with that function call. The call is made with ($data, $cfg, @args) as arguments. @args being the arguments passed to that function by the calling user. Remember that to get at the new() arguments, you just have to do a @args = @{$data->extra_args()};

  To simplyfy some things, there is the helper function $data->best_lang described below.

1.6.3 Helper API Functions

- **$data->best_lang(@avaliable_language)**

  This is the generic best-language picking routine. This function given the list of known languages to the handler, returns the best language based on the client's request.

  For example, a simple handler could handle queries like this:

```
sub fetch {
   my ($class, $data, $cfg, $key, $lang) = @_;
   #the language information is kept in the $cfg hash

   #if a specific language is asked for:
   return $cfg-E<gt>{$key}{$lang} if $lang;

   #we must choose the right language ourself.
   #(keys % {$cfg-E<gt>{$key}}) produces the list of languages in wich $key is known
   my $variant = $data-E<gt>best_lang(keys % {$cfg-E<gt>{$key}});

   #$variant now holds the best pick for us, or undef if nothing matches..
   return $cfg-E<gt>{$key}{$variant} if $variant;

   #give up
   return undef;
}
```

1.6.4 Optionnal API Functions (store)

store ($data, $cfg ,$key, $lang, $value)

If this function is present, the handler will be makred as 'storable' and any assignments to the language hash will call this function. Storing in a language hash should be done by providing a key/language pair as key to the hash (ie. $lang->{key01/en-us} = "string"). This should return L_OK on success and L_ERROR in case of an error.

1.6.5 Optionnal API Functions (firstkey/nextkey)

This are functions necessary to make the Language hash listable (i.e. **keys** and **each**). The tricky thing is that since LanguageHandlers are stackable and often each have a change at the request, listing a specific language hash is ambiguious. The way it's handled is straightforward. The first listable module that returns something else than undef on the call to firstkey is the chosen one. It will be used to list the hash until it's nextkey function returns undef, and all subsequent list request will be handled by that module.

It should be noted that I don't really know what might happend in the case of recursive/re-entrant lists, so if you don't want to spend time fixing it yourself, be carefull about that.

These functions follow exactly the same rules as the generic TIEHASH mechanism. So, for more information see *perltie*

- **firstkey ($data, $cfg)**

  This will be called first when a list request is encountered. It should return the first key of the hash and set up things for the nextkey calls.

- **nextkey ($data, $cfg, $lastkey)**

  This will be called once the listing of the language hash has started. Make sure it will eventually return undef to mark the end of the list or else you will end up in an infinite loop.

1.7 TODO

- **Find and correct bugs.**
- **Find new features to add.**

1.8 SEE ALSO

perl(1), *Apache*(3), *Apache::Language::Constants*(3), and all *Apache::Language::**.

1.9 SUPPORT

Please send any questions or comments to the Apache modperl mailing list <modperl@apache.org> or to me at <gozer@cpan.org>

1.10 NOTES

This code was made possible by :

- Doug MacEachern <dougm@pobox.com> Creator of mod_perl. That should mean enough.
- Andreas Koenig <koenig@kulturbox.de> The one I got the idea from in the first place.
- The mod_perl mailing-list at <modperl@apache.org> for all your mod_perl related problems.

1.11 AUTHOR

Philippe M. Chiasson <gozer@cpan.org>

1.12 CVS The CVS repository of Apache::Language is avaliabe thru anoncvs at:

```
$> cvs -d :pserver:anoncvs@cvs.ectoplasm.org:/home/anoncvs login
password: anoncvs
$> cvs -d :pserver:anoncvs@cvs.ectoplasm.org:/home/anoncvs co Apache-Language
```

1.13 COPYRIGHT

Table of Contents:

1 Apache::Language - Perl transparent language support for Apache modules and mod_perl scripts 1
1.1 SYNOPSIS . . . 2
1.2 DESCRIPTION . . . 2
1.2.1 BASIC USAGE EXAMPLE . . . 2
1.3 DETAILLED USAGE . . . 3
1.4 CONFIGURATION DIRECTIVES . . . 4
1.5 STATUS Handler . . . 5
1.6 API . . . 5
1.6.1 API data structures . . . 6
1.6.2 Required API functions . . . 7
1.6.3 Helper API Functions . . . 8
1.6.4 Optionnal API Functions (store) . . . 9
1.6.5 Optionnal API Functions (firstkey/nextkey) . . . 9
1.7 TODO . . . 9
1.8 SEE ALSO . . . 9
1.9 SUPPORT . . . 10
1.10 NOTES . . . 10
1.11 AUTHOR . . . 10
1.12 CVS The CVS repository of Apache::Language is avaliabe thru anoncvs at: . . . 10
1.13 COPYRIGHT . . . 10