ApacheCon US
March 22, 2001
Singapore

Tutorial:
Multilingual sites with mod_perl and Template Toolkit

by **Stas Bekman**
http://stason.org/
<stas@stason.org>
eXtropia.com, Senior Engineer

This document is originally written in **POD**, converted to **HTML**, **PostScript** and **PDF** by `Pod::HtmlPsPdf` Perl module.

1 Multi-lingual Sites with Perl

1.1 Agenda

- To dynamic or not to dynamic? (i.e. static :)
- User language preferences negotiation
- Storing the language preferences once we know them.
- The major parts the output is generated from.
- The difference between site browsing and site searching.
- How the language specific data is to be stored in the database.
- One template per language, or one template for all languages

- Abiding to user’s native date and times presentation conventions
- Helping clients to do the right thing with the output.

1.2 Dynamic versus Static Pages

- Do your choice of the service: Dynamic or Static
- If static, how many pages (per language)?
- If just a few, a dynamic site might be an overkill
- If more than a few, static is an overkill, manual maintenance is:
- - time consuming (always)
- - error-prone
- You can always easily turn a dynamic site into a static site

- The opposite is not simple
- Template Toolkit generates static snaphots with `tpage` anc `ttree` utilities
- From now on we assume that you want to develop a dynamic site.

1.2.1 User Language Detection

- What language should be used when presenting the content?

Algorithm:

- Divide users into 2 groups:
- #1 those who visit the site for the first time
- #2 those who have visited the site previously.

- Group #2: cookies or log-in let you know user’s language preference
- Otherwise everybody falls into group #1
- For log-in technique, you still need to know the language (chicken-n-egg paradox)
- If you figure out that all users understand a common language you don’t need a multi-lingual site.

Deducing the language preferences from the user’s country:

- Running reverse DNS lookup on his IP address
- Looking up the TLD of the resulting computer name
- If TLD == .fr, the user is from France, probably speaks French.
- Now consider that some Singaporian traveler connects from Paris
- Moreover, many hosts have broken reverse DNS mapping
- Don’t attempt deducing the country from IP
- Many users connect trough proxies outside their countries (AOL!)

- So we are back to the group #1, where we know nothing abou
them
- Provide a way to choose a language
- Present a page with links to all available languages
- Language name should be written in that language

Doing intelligent guess:

- Look-up: the `Accept-Language` header sent by most browsers
- Localized versions of modern browsers set up the preferred language at install time
- A knowledgeable user may adjust her language preferences, e.g.:

```
German
English-US
French
```

- A broser sends:

```
Accept-Language: de,en-us;q=0.7,fr;q=0.3
```

- American English: 70% and French: 30%.
- Parsing the `Accept-Language` header
- - manually
- - the `Apache::Language` module under mod_perl
- - the all-purpose `HTTP::Negotiate` module

It’s still a guess!!!

- Don’t lock user out of making another language choice based or this guess.
- The preference might be set incorrectly! (Singaporian in Paris)
- Still use this information, be user-friendly

- After a user has made her selection
- - we can proceed with the content generation phase
- - or deliver a static content using the chosen language

1.3 Generating the Content.

- Content generation recipe is based on 2 basic ingredients:
- - The *invariant* data: like table and page headers (always the same).
- - The *variant* data: unknown a priori, and depends on user input.
- Single language service: the 2 elements are strightforward
- - Templates/hard-code for invariant data
- - Database or else for variant data

- More complicated for multi-lingual service

1.4 Fetching Dynamic Data

- I divide the dynamic data requests into two groups:
- #1 Those that require user input
- #2 And those that don't
- A site search feature falls into into the first category
- Whereas browsing the site belongs to the second one.

1.4.1 Searching

- Let’s take a French user searching for a movie as an example
- French and other languages include accented characters
- An accented character usually uses some non-accented characte
  as a base.
- For example characters: *â*, *à*, *á* are all based on character *a*.
- The problems is that some users use these characters and some
  don’t for various reasons.
- The service is still supposed to interpret the input correctly

- Solution: need to store 2 versions of the text
- #1 search index free of accented characters
- #2 original text - needed to present the correct output

- From now I’ll use ISO-8859-1 character set as an example
- It is used by many Western European languages.
- In your handouts you will find two functions:
- - `iso_8859_1_lc()` which turns any input using ISO-8858-1 into a lowercase, accent-free version.
- - `iso_8859_1_uc()` is similar but upper cases the input string.

```
print $charsets{'iso-8859-1'}{lc}->('Bienvenüe');
```

- prints: **bienvenue**

- These functions are used twice:
- - when creating the search index
- - when accepting the search string, before the actual search is performed.
- Usually using the lower case for searching is the accepted technique.

- These functions are also used for sorting:

```
sub cmp_nocase{
    return My::Language::lc($_[0], $_[1])
              cmp
           My::Language::lc($_[0], $_[2]);
}
```

- which then can be used as:

```
my @correctly_sorted =
    sort { My::Language::cmp_nocase($lang,$a,$b) } @data;
```

1.4.2 Browsing

- When browsing, preset data inputs are used
- e.g. when the results of search are presented
- (Don’t trust the user, not to change things)
- Now you can use the real text in links, but it should be encoded
- or the browsers might interpret the request incorrectly
- - use URI::Escape::uri_escape()
- - use Apache::Util::escape_uri() (much faster under mod_perl)

- The text in the page should be encoded as well
- - use HTML::Entities::encode()
- - use Apache::Util::escape_html() (mod_perl, again faster)

1.4.3 Data Retrieval

- Database design is not trivial
- Avoid creating language specific fields in every table that includes multi-lingual data. e.g.:

```
table movies:
-------------
title_fr
title_en
title_es
description_fr
description_en
description_es
....
```

- Better place all the language specific data into one table:

```
id
lang
real_text
search_text
```

- *id* is needed to map every record into the table the data belongs t
  and usually more complex, e.g.:

```
orig_table
orig_column
orig_id
```

- The concatenation of these three fields gives a unique mapping
- Searching only in specific tables:

```
SELECT * FROM lang where orig_table='movies'
AND orig_column='description' AND lang='fr'
AND search_text LIKE '%foo%'
```

- Retrieving all the language fields tied to some record we can do:

```
SELECT * FROM lang where orig_table='movies'
AND orig_id=123456
```

- Also consider having a table for each language, if it creates too much overhead for db.

1.5 Invariant Data

- Data which doesn't change:
- - is either hard-coded in the code
- - or, better, placed in a template.

- Single language service search template

```
<!-- results set -->
[% IF input %]
    [% IF total_hits %]
        A total of [% total_hits %] movies was found.
   [% ELSE %]
        No Results.
   [% END %]
[% END %]
```

```
<!-- input form -->
<form>
<input type="text"   name="input"  value="[% input %]" size="32">
<input type="submit" name="search" value="Search">
</form>
```

- A corresponding code to produce the output

```
use Template;
my $r = shift;
$r->send_http_header('text/html');

my $t = Template->new(INCLUDE_PATH => '/templates/path');

$t->process('search.ttml',
            { input      => 'foo',
              total_hits => 15,
            },
            $r
           ) or die $t->error();
```

- For multi-lingual site: should we have
- #1 a page per language,
- #2 one page for all languages?
- Option #1 leads to many pages -- hard to keep in sync
- Option #2 makes it easy to add languages and modify -- all strings are stored in the same place.

- How do we know to take one language out of many:
- Using XML tags, which are parsed by TT:
- - <text> for the text sections
- - two letter code tags for language specific sections.

- The search input template becomes:

```
<form>
<input type="text"   name="input"  value="[% input %]" size="32">
<input type="submit" name="search"
       value="<text>
                 <en>Search</en>
                 <fr>Chercher</fr>
                 <it>Cercare</it>
              </text>">
</form>
```

- and the search results output template now looks like this:

```
[% IF input %]
    <text>
         <en>Search Results</en>
         <fr>Résultats de la recherche</fr>
         <it>Risultati della ricerca</it>
    </text>
    [% IF total_hits %]
        <text>
            <en>A total of [% total_hits %] movies was found.</en>
            <fr>[% total_hits %]
                [% IF total_hits > 1 %]
                   films trouvés
                [% ELSE %]
                   film trouvé
                [% END %].</fr>
            <it>[% total_hits %]
                [% IF total_hits > 1 %]
                   movies trovati
                [% ELSE %]
                   movie trovato
                [% END %].</it>
        </text>
   [% ELSE %]
        <text>
            <en>No Results.</en>
            <fr>Aucun résultat.</fr>
            <it>Nessun risultato.</it>
        </text>
   [% END %]
[% END %]
```

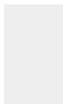
- Hard to edit in WYSIWYG editors
- Professional HTML designers use plain text editors

- Your handouts include the code to overload the default TT parser
- and the actual mod_perl script to produce the output
- Don’t forget to pass the *lang* variable to the template object

1.6 Handling Dates and Time Presentation

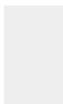
- Date and time formats can be stored in the code
- Beware of different countries using different conventions

- An American user expects:



Thursday March 22, 2001 2:25pm

- whereas a French user parses correctly this:



Jeudi 22 Mars 2001 14h25

- Let’s assume that these conventions are tied to languages, rathe than countries.

- In reality you want to ask users about their format preference

- This is a subset of formats for French users
- The full set for four languages is in your handouts.

```
fr => { name    => 'français',
        charset => 'iso-8859-1',
        months  => [qw(Janvier Février Mars Avril Mai
                       Juin Juillet Août Septembre Octobre
                       Novembre Décembre)],
        days    => [qw(Dimanche Lundi Mardi Mercredi
                       Jeudi Vendredi Samedi)],
        date    => {short  => '%d/%m/%Y',
                    medium => '%e %B %Y',
                    long   => '%A %e %B %Y',
                   },
        time    => {short  => '%Hh%M',
                   },
      },
```

- These are handy compile time constants, used in the code.

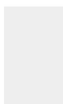
```
use enum qw(YEAR MONTH DAY);
use enum qw(HOUR MINUTE);
```

- Your handouts include the date and time formatting macros based on `strftime(3)`

- Finally there are two functions that format the date and time according to a language. (See your handouts)

1.7 Generating Correct Charset Headers

- HTTP response need to declare the charset of its data.
- Then the browser will know how to render the text correctly.
- Two techniques to accomplish that:
- #1 Sending HTTP header



```
Content-Type: text/html; charset=ISO-8859-1
```

- #2 Embedding `META HTTP-EQUIV` tag:

```
<HEAD>
    <META HTTP-EQUIV="Content-Type"
          CONTENT="text/html; charset=ISO-8859-1">
</HEAD>
```

- With mod_perl you can do #1 with:

```
my $r = shift;
$r->send_http_header('text/html; charset=ISO-8859-1');
```

- #2 is less preferred, because it requires that ASCII characters stand for themselves until after the <META> tag and often causes an annoying redraw with Netscape.

- Of course users can asjust the encoding, if they know how
- Keep the usability of your service high, sparse a hassle to you users

1.8 Conclusions

- We have discussed the following multi-lingual site developmen issues:
- - It’s almost always better to develop a dynamic site rather than a static one.
- - Language selection is done by either asking the user about it and looking at the `Accept-Language` header.
- - Memory of the user preference is best done via cookies, or by making the user log in.

- - The generated output is comprised from semi-static template tex and dynamic database content.
- - Site browsing is different from site searching in the terms o multi-lingual input processing.
- - The language specific data can be database design
- - Multilingual variants of text can coexist in the same template.
- - The presentation of dates and time can be adjusted to the use preferences.
- - One has to tell client browsers to render the output using a correct language encoding.

- For more information about discussed topics please refer to the *Resources* section.
- In addition a simple introduction to mod_perl follows in you handouts.

1.9 Resources

- mod_perl home page: http://perl.apache.org/ mod_perl guide http://perl.apache.org/guide/
- Template Toolkit home page: http://www.template-toolkit.org/
- Internationalization / Localization: Charset parameter: http://www.w3.org/International/O-HTTP-charset.html
- ‘‘A tutorial on character code issues’’ by Jukka Korpela: http://mirror.subotnik.net/jkorpela/chars.html
- ‘‘Localizing Your Perl Programs’’ by Sean Burke and Jordan Lachler. (The Perl Journal issue 13 spring 99.)

- CPAN modules mentioned in this talk:

  HTTP::Negotiate: http://search.cpan.org/search?dist=libwww-perl

  HTML::Entities: http://search.cpan.org/search?dist=libwww-perl

  Apache::Util: http://search.cpan.org/search?dist=mod_perl

  Apache::Language:
  http://search.cpan.org/search?dist=Apache-Language

;o

2 mod_perl: Getting Started Fast

2.1 mod_perl in Four Slides

- Installation
- Configuration
- The “mod_perl rules” Apache::Registry Scripts
- The “mod_perl rules” Apache Perl Module

2.2 What is mod_perl?

Solves numerous mod_cgi shortcomings:

- Embedded Perl Interpreter -- no loading overhead
- Code compiled only once per process life -- no compilatior overhead
- No forking per request -- process reuse
- Response processing is now reduced to running your code.
- Response times improve by a factor of 10 to 100

- A bigger size, but just a few processes can handle a much bigge
  load
- mod_cgi compatibility preserved (Apache::Registry and Apache::PerlRun modules)
- Persistent database connections

Extended mod_cgi’s functionality:

- A complete Perl API added to the Apache core
- Handling of all phases of request processing in Perl.
- Writing complete Apache modules in Perl
- Complete server configuration in Perl.
- Numerous 3rd party modules are available

Logistics:

- Developed by Doug MacEachern
- Licensed under the Apache Software License.
- Home page http://perl.apache.org
- Mailing list: send an empty email to modperl-subscribe@apache.org
- January 2001 -- 2 Million mod_perl hosts (according to http://perl.apache.org/netcraft/)

2.3 Installation

```
% lwp-download \
  http://www.apache.org/dist/apache_x.x.x.tar.gz
% lwp-download \
  http://perl.apache.org/dist/mod_perl-x.xx.tar.gz
% tar xzvf apache_x.x.x.tar.gz
% tar xzvf mod_perl-x.xx.tar.gz
% cd mod_perl-x.xx
% perl Makefile.PL APACHE_SRC=../apache_x.x.x/src \
  DO_HTTPD=1 USE_APACI=1 EVERYTHING=1
% make && make test  && make install
% cd ../apache_x.x.x && make install
```

- That’s all!

2.4 Configuration

- Add the following to the configuration file *httpd.conf*:

```
  # for Apache::Registry mode
Alias /perl/ /home/httpd/perl/

PerlModule Apache::Registry
<Location /perl>
  SetHandler perl-script
  PerlHandler Apache::Registry
  Options ExecCGI
  allow from all
  PerlSendHeader On
</Location>
```

2.5 The "mod_perl rules" Apache::Registry Scripts

- You can write plain perl/CGI scripts just as under mod_cgi:

```
mod_perl_rules1.pl
------------------
print "Content-type: text/plain\r\n\r\n";
print "mod_perl rules!\n";
```

- Of course you can write them in the Apache Perl API:

```
mod_perl_rules2.pl
------------------
my $r = shift;
$r->send_http_header('text/plain');
$r->print("mod_perl rules!\n");
```

- Save both files under the */home/httpd/perl* directory
- Make them executable and readable by server,
- and issue these requests using your favorite browser:

```
http://localhost/perl/mod_perl_rules1.pl
http://localhost/perl/mod_perl_rules2.pl
```

- In both cases you will see on the following response:

```
mod_perl rules!
```

2.6 The "mod_perl rules" Apache Perl Module

- To create an Apache Perl module, all you have to do is to wrap the code into a `handler` subroutine:

```
ModPerl/Rules.pm
----------------
package ModPerl::Rules;
use Apache::Constants;

sub handler{
  my $r = shift;
  $r->send_http_header('text/plain');
  print "mod_perl rules!\n";
  return OK;
}
1;
```

- Create a directory called *ModPerl* under one of the directories ir
  `@INC`
- and put *Rules.pm* into it.
- Then add the following snippet to *httpd.conf*:

```
PerlModule ModPerl::Rules
<Location /mod_perl_rules>
  SetHandler perl-script
  PerlHandler ModPerl::Rules
</Location>
```

- Now you can issue a request to:

```
http://localhost/mod_perl_rules
```

- and just as with our *mod_perl_rules.pl* scripts you will see:

```
mod_perl rules!
```

- as the response.

2.7 Is That All I Need To Know About mod_perl?

- Definitely not!
- These slides are intended to show you that you can install and start using a mod_perl server within 30 minutes of downloading the sources.
- There is much more to mod_perl than this.
- Fortunately, there are many resources and lots of help freely available to you.

;o