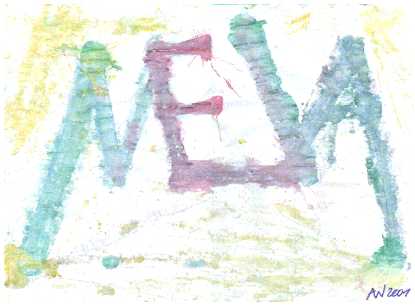
Albrecht Weinert

WP3 progress report

Making WordPress 3 bilingual



Rev.: 26.04.2013

Prof. Dr.-Ing. Albrecht Weinert a-weinert.de
weinert – automation weinert-automation.de

Labor für Medien und verteilte Anwendungen (MEVA-Lab) meva-lab.de
Laboratory for Media and versatile Applications

Fachbereich Informatik der Hochschule Bochum
Computer Science department – Bochum University of Applied Sciences

Making WordPress 3 bilingual

V01.00, 18.04.2013: unfinished draft
V01.01, 22.04.2013: first ready for public
V01.02, --.--.2013: -

Version: V1.01
Last modified by A. Weinert at 26.04.2013

Note on the document template:	There is one common numbering for figures, lists, tables etc. (if present)
Note on version control:	SVN URL is https://ai2t.de/svn/albrecht/pub/Wordpress3Biling.odt
Note on publications: See also	http://a-weinert.de/publication_en.html , http://blog.a-weinert.de/ and http://blog.a-weinert.de/wordpress3biling/
This document's URL:	http://a-weinert.de/pub/Wordpress3Biling.pdf. That might be newer if this is from elsewhere or on paper.

Table of content

1. Motivation

WordPress (WP) is often used as a Blog or small content management (CM) system be it stand alone or as integral part ([1]) of an existing website or as companion with existing sites using the same style and look. Some websites are multilingual for good reasons offering content in say English, German, French, Italian, Croatian or further languages.

There may be

- articles in different languages in the sense of partly publishing in English and, for example, mainly in German,
- the same article / content in different languages in the sense of offering a number of translations

and

- offering the site's technical requisites, like menu, navigation, pop-up help texts or titles etc., in different languages.

The latter may be offered – and often is welcome – even if an article is given in just one language. Also note, that the language of menus and other requisites cannot be directly tied to an article's language. WP's overview, archive, search results etc. may show (excepts of) many articles/pages and hence show more than one language.

1.1 Requirement and WordPress' deficiencies

Implementing above points in a website mainly involves the burden of having all texts in question translated to the respective target languages. Perhaps some polishing of used style sheets (.css), transformers, generators etc. may be involved. If implemented in websites the same multilingualism is expected from CMs and Blogs as a matter of course, especially when integrated to or looking alike a multilingual site.

But with CMs and Blogs that's usually harder than one would expect, as these approaches are “sold” to bring great flexibility compared to generating/delivering mainly static, html and perhaps xml with client side transforms.

Considering WordPress (3) in this respect one can say:

- Yes, it brings translations for menus, help texts and else offering ample translations to many languages.
- Yes, WP allows setting of the language.
- Yes, WP allows setting of the encoding.

But the latter two settings are Blog wide – and partly for installation time only. Flexible language changes on a dynamic page by page basis are not on WP's table. Plug-ins promising the required multilingualism are complex, resource hungry and may have side-effects.

That's not (only) the plug-in's fault but a common weakness of the WordPress architecture and specification. Content and settings are distributed over directories, files and databases. Some parts may be moved to other locations (at least so says the spec) shared with or ported to other blogs etc. Some plug-ins change the database structure or move common information to other tables etc. Adding complexity to a WP installation is an

opportunities to get into troubles and inconsistencies – or to explode the whole thing at the next tiny update.

```
#: wp-includes/media-template.php:99
msgid ""
"Your browser has some limitations uploading large files with the multi-file "
"uploader. Please use the <a href=\"%1$s\" target=\"%2$s\">browser uploader</"
"a> for files over 100MB."
msgstr ""
"Ihr Browser hat eine Uploadbegrenzung f&uuml;r gro&szlig;e Dateien im Multi-"
"Dateien-Uploader. Bitte nutzen Sie den <a href=\"%1$s\" target=\"%2$s"
"\">Browser Uploader</a> f&uuml;r Dateien gr&ouml;&szlig;er als 100MB."
```

Listing 1: Excerpt of (improved) ..blog../wp-content/languages/de_DE.po (size:262kByte)

Another point is performance. WP's translation mechanism is php's .mo/.po-approach. It uses the full English clear text as key (all three lines after msgid in listing 1) to the particular translation. This is nice to the programmer and allows some semi-automatic separation of programmer=first English text author and translators. But algorithmically at runtime this is horridly inefficient – alleviated only by caching a number of translations done for the one set Blog language. WordPress is monolingual by design. Changing the Blog language dynamically (as do some plug-ins) invalidates that cache.

The approach's main advantage is keeping the php program / page text mixture fluently readable, compare see listing 2.

So, bending a Blog/CM system that's designed to be monolingual to be polyglot is tricky and prone to undesirable side-effects.

```
<?php   _e('The English text as put by the programmer/page designer ');   ?>
```

Listing 2: Decorating a text for automatic translation at runtime

1.2 Reduced requirement

But often real multilingualism is not needed. In most cases bilingualism is wanted:

I. one language is the Blog's/site's native tongue –
German (de-DE) in our example. –

and

II. the other language is …
English (en-GB in Europe and en-US elsewhere)

English being the “natural” second language stems from being today's lingua franca or the Latin of the last century.

The language I – German in our example – may be French or Spanish as well. This is the Blog's set language. Language II is not flexible it must be English. And that is the reduced requirement because English is no language at all for WordPress in the sense of all being programmed and expressed so.

Hence we may now switch

I. translation to the (one) set Blog's language and

II. no translation at all.

That's not hard but still a bit tricky as the ubiquitous "_e()"-decoration (listing 2, page 4) always translates.

1.3 Pre-requisites

For the solution described in the next chapter we need

A) the translation of all (and that will be many) text building blocks to our language I (German e.g.)

and

B) an own theme specialised and tailored to our site's design and needs.

A) seems easy as the WordPress community delivers those translations for dozens of languages – but it may be the hardest part.

B) may seem hard at first glance – but it is not.

See [1] for the background.

2. Solution, implementation

2.1 Bilingual menus and else

Obviously menus and other requisites text blocks is the only thing WordPress may automatically translate at runtime. This concerns the backend (authoring, administration) too, but our focal point is the so called frontend – what customers, readers and commentators see.

As said, all we want do is switching the automatic translation, done by php constructs like listing 2, page 4 on and off. Regrettably no simple boolean on/off variable or parameter for these _e-constructs was to be found. Nearest comes an optional second string parameter naming the translation domain. Here 'default' chooses the standard translation to the one set Blog language, whereas an unknown domain (like '-') turns the translation off. Voila.

For that “domain” name we define a variable $trDom in every page template and inset of our spezialised theme. It is then used in every _e- and the like construct as second parameter or for language switches by just if-else; see listing 3.

```
 ........
 <h2 class="pagetitle"><?php  _e("Author's Archive ", $trDom) ?></h2>

 <?php /* If this is a paged archive */ } else if (isset($_GET['paged'])
  && !empty($_GET['paged'])) { ?>
  <h2 class="pagetitle"><?php  _e("Blog Archive ", $trDom) ?></h2>
<?php }
while (have_posts()) : the_post(); ?>
   .........

 <p align="center"><?php
if ($trDom === '-') { // English
  posts_nav_link(' - ','&#171; Previous','Next &#187;');
} else { // (English else) default: German
  posts_nav_link(' - ','&#171; Zur&uuml;ck','Weiter &#187;');
} // default: German
?>
```

Listing 3: Bilingual menus and else (two excerpts from archive.php)

2.2 The language choice

To switch this between translation to the one Blog language (de-DE for example) and no translation / no language meaning English we need

a) a handle / button / link for the reader to switch to the alternate choice

and

b) a sensible automatic choice of language if the reader didn't make a choice by a) (yet).

In the end this will set the page's/article's variable $trDom to either – or default (listing 3).

```php
/* Gets the request URI (for the 'where bar') as string
*
* If no paramter is given thats it.
*
* A parameter $trDom '-' says the page actually being displayed with English
* menus and else and appends a '[de]' link to switch to German (menu) display.
*
* A parameter $trDom 'default' says the page actually being displayed with
* default (i.e. German) menus and appends a '[en]' link to switch the menu
* display to English.
*
* Note: Neither of those links loads an alternate or translated page. It just
* reloads with or without a ?lang=en/de parameter contradicting the page's
* tagged language. That is evaluated by the function getTrDom().
*
* The second parameter if given will be appended as parameter to a [de/en]
* link if generated as further URL parameters. It must neither start
* by ? nor by &.
*/
function reqURIwhere($trDom = 'noLang', $linkAppend = '')  {
 $ret = " ";
 $reqUr = $_SERVER["REQUEST_URI"];
 $askPos = strrpos($reqUr, '?');
 if ($askPos === false) {
     /* no params appended */
 } else {
      $reqUr = substr($reqUr, 0, $askPos);
 }
 $ret = $ret . $reqUr;
 $prevApp = '?';

 if ($trDom === '-') { // page is displayed English make link [de]
   $ret =
      $ret . '   <a title="Diese Seite mit deutschen Men&uuml;s" href="';
   $ret = $ret .  home_url($reqUr);
   if (has_tag('english') && !is_home())  {
     $prevApp = '&';
     $ret = $ret . '?lang=de';
   }
   if ($linkAppend != '') {
     $ret = $ret . $prevApp . $linkAppend;
   }
   $ret = $ret . '">[de]</a>';
 } else if ($trDom === 'default') { // page is displayed German make link [en]
   $ret = $ret . '   <a title="This page with English menus" href="';
   $ret = $ret .  home_url($reqUr);
   if (!has_tag('english') || is_home())  {
     $prevApp = '&';
     $ret = $ret . '?lang=en';
   }
   if ($linkAppend != '') {
     $ret = $ret . $prevApp . $linkAppend;
   }
   $ret = $ret . '">[en]</a>';
 }
 return $ret;
}  // reqURIwhere($trDom = 'noLang', $linkAppend = '')
```

Listing 4: Link for user's menu language choice (excerpt from theme's functions.php)

To implement the optional user's choice – that is a) above – we introduce an URL parameter named lang, that may be used to reload the current page. Listing 4, page 7, is a function to generate the respective link text. It is used in all page's/article's headers.

The sensible automatic choice of the menu language – that is b) above – should be the (first) articles content language, i.e. having German menus with German text and English with English. That should be the standard consistent look as long as the user does not chose otherwise.

Also for this automatism we introduce a tag “english” which (surprise) is put to all arcticles/pages in English language. The function in Listing X page Y evaluates that to out $trDom value, given a user choice (by URL parameter lang) precedence.

```
/* Checks lang pararameter for en/de post tag for english
 *
 * returns '-' if english and 'default' else.
 * used as second (domain) parameter in __() and
 * the like turns translation off for English posts.
 */
 function getTrDom() {
   if (isset($_GET['lang'])) {
     $lang =  $_GET['lang'];
     if ($lang === 'en') return  '-';
     if ($lang === 'de') return 'default';
   }
   if (is_home()) return 'default';
   if (has_tag('english')) return  '-';
   return 'default';
 } // getTrDom()
```

Listing 5: Evaluating language for menus / requisites (excerpt from theme's functions.php)

2.3 Translated / multilingual content

In the case of an article existing in two (or more) languages the author has to put a link to the respective other language version at the very beginning; listing X shows both. The link is right aligned and the text is floating around by some css support.

```
<div class="imgonright"><a class="readmore"
href="http://blog.a-weinert.de/a-comprehensive-german-dictionary-for-eclipse/"
title="This artcle in English language">[en]</a></div>Wer nicht nur Englisch
schreibt sondern auch viele deutsche Texte in JavaDoc- oder Doxygen-Kommentaren
...
-------
<div class="imgonright"><a class="readmore"
href="http://blog.a-weinert.de/deutsches-worterbuch-fur-eclipse/"
title="Dieser Beitrag auf Deutsch">[de]</a></div>If your HTML, XML and JavaDoc
or Doxygen is to some extend German (and not just English) Eclipse's spell
checker gets more bothering than a help. Here ...
```

Listing 6: Link to switch to other language version of the article (excerpt from the articels)

3. Resume

Regarding the overall functionality and usability WordPress is a good product. It is well adaptable to the requirements reported here. Being open source and free of licence fees is a big advantage. Anyway WP is good enough to get a very broad user base and to even conquer some commercial application.

On the other hand it has its limitations – one of them is being mono-lingual by original design. Using an own theme specialized to the wanted blog design and look is recommendable for many reasons (see [1]). This gives a good base for the simple and efficient solution for bilingualism in the sense of “other Language and English” given here.

A Abbreviations

API application programmer's interface

CSS cascading style sheets

CM content management

DRY don't repeat yourself! as approach

HTML Hypertext Markup Language

JAXP Java API for XML Parsing

JS Java Script aka ECMA script

PHP PHP Hypertext Preprocessor; Server side web programming language

WP WordPress

XML eXtensible Markup Language

XSD XML schema definition

XSL eXtensible Stylesheet Language

XSLT XSL Transformation

L References

[1] Albrecht Weinert
Integrating WordPress 3 into an existing website
progress report, Februar 2013, (.pdf, blog)

[2, 3 Albrecht Weinert, Wordpreess
Blog's technical background, 2008 (blog en, blog de)