Albrecht Weinert

WP3 progress report

Integrating WordPress 3 into an existing website

Rev.: 20.04.2013

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

Labor für Medien und verteilte Anwendungen (MEVA-Lab) meva-lab.de
Fachbereich Informatik der Hochschule Bochum

Integrating WordPress 3 into an existing website

`V01.00, 19.02.2013:` unfinished draft
`V01.01, 15.04.2013:` first ready for public
`V01.02, --.--.2013:` -

Version: V1.01
Last modified by A. Weinert at 20.04.2013

Note on the 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/Wordpress3Integr.odt
Note on publications: See also	http://a-weinert.de/publication_en.html , http://blog.a-weinert.de/ and http://blog.a-weinert.de/wordpress3integr/
This document's URL:	http://a-weinert.de/pub/Wordpress3Integr.pdf. The version there might be newer if this is from elsewhere or on paper.

Table of content

1. Motivation

WordPress is often used as a Blog or small content management (CM) system just by downloading the software and putting it into operation, choosing one of dozens available designs respectively themes. This standard usage is kind of stand alone or from scratch. Server preconditions given, this approach works quite well and without greater problems or own development work in most cases.

This report is about integrating WordPress 3 (WP3) with existing web sites and designs. The task arises when

A) giving an existing (large, mainly static) site a blog or CM component afterwards

or if

B) a (technically) separate blog and one or some (static) sites shall have the same design and look and feel.

Paragraphs 1.1 and 1.2 (page 4) describe a motivation example or use case (for A)). They may be skipped to see the solution and not the rationale of the requirements.

1.1 Bringing Blog/CM to an existing site – use case

A medium sized (>2000 files) website maintained for a customer consisted on mainly static php, html and xml with client side transforms. The rendering was governed by a 6K css file and some JavaScript (JQuerry) to have drop down menus and optional image slide shows.

The content, balanced English and German plus a wee bit of French, was completely maintained by only few authors respectively administrators. Using only Western European languages all files and the website are 100% ISO-8859-1. Maintaining and editing work is mostly done within Eclipse using the editors for html, php, xml and so on. As used for content editing Eclipse was amended with good German spell checking.

All files are under under version control using Subversion (SVN). Common parts of the (mainly static) content, like parts of the html-header, the page header, navigation, footers and so on are evacuated to about 40 include files, mostly one for each language used. The spot to include is marked by a pair of html/xml comments. This system of common text replacement is elaborated in the sense of single point of truth respectively no repetition or the DRY principle.

Before deployment to the (Apache) web server

1) the SVN keyword tags (Id HeadURL Date Revision Author) are “beautified” so their values can be used as part of the web content and
2) said comment pairs (plus the text within) is replaced by the respective replacement text (include file content).

These two steps are done on the server by the Frame4J tools CVSkeys respectively FuR controlled by a SVN post commit hook.

For a person accustomed to Eclipse and SVN the pure technical part of any maintenance is hardly existing reducing the workload to content authoring and copy editing.

```
    <!-- headStart-common Start --><html><head><!-- headStart End -->
    <!-- headEnd-common Start --></head>
        <body><div><div><div><!-- headEnd End -->/>
```
Listing 1: Marking the place for replacement text by a pair of comments.

Note: The (also replaced) content within the Start/End comment pair mirrors the minimal expected structure elements to keep (xml, html, Eclipse) syntax checkers happy.

```
  ignoreCase=false
  ignoreWS=true
  keepBraces=true
.......
  old5 = <!-- headStart-common Start -->
  end5 = <!-- headStart End -->
  newFile5 = head-start.txt
  keepBraces5=false

  old6 = <!-- headEnd-common Start -->
  end6 = <!-- headEnd End -->
  newFile6 = head-to-body.txt
```
Listing 2: Excerpt of FuR's property file to control the (post commit) replacements.

1.2 The desire for an additional CM / Blog component

Nevertheless for a small part of the web-site like reports on conferences some more people wanted to write content and deliver images.

Some of those persons were not able or unwilling to handle the SVN machinery. Some also disliked to be confronted with some minimal set of HTML. The argument “that there are good integrated tools” or demonstrating the benefits of an adequately configured Eclipse are hardly accepted – even by people whose area of work suggests the necessary skills as given.

But alas, experience shows it's of absolute no avail to discuss the underlying attitude: refusing to learn the basics of a long living language and prevalent tools but attending long courses about the idiosyncrasies of one special Typo3 installation with sensual pleasure.

So the desire to have a browser access to content management arose for a web site existing since years.

1.3 Choosing WordPress 3

To implement this request WordPress was chosen. The reasons were:

- good (mostly) long term experiences with the product (2.8.x) as stand alone Blog
- sufficiently lightweight and (hopefully) modifiable to integrate as part of a website
- good browser based editing and media handling for laymen in most cases without any need for training – quite contrary to e.g. Typo3.
- existence of all infrastructure needed on the server in question:
  PHP, mySQL, Apache.
- a sufficient protection / rights management

2. Requirement

2.1 Basic

As said we want to

A) give a large existing web site an extra blog or CM component or

B) let technically separate blogs and sites having the same style and look and feel.

And we use WordPress3 in both cases as WP3 can do it well:

It comfortably brings the functionality wanted (see Paragraph 1.3).
It is – really – not just small, but it is no overkill.
And it can flexibly be adapted.

Nevertheless it was foreseen WordPress would be bitchy on some of the indispensable requirements.

2.2 Extra boundary conditions and requirements

Compared to a stand alone / from scratch WP installation we must consider:

a) WordPress is not to be **the** blog or **the** website – only a quite small part of an existing one – in use case A)

b) the design / the look and feel of the existing site has not to be touched or modified in direction of existing WP themes / layouts. Quite the contrary:
WP It must adapt to an existing style – in use cases A) and B).
This (and some other reasons) will lead to the creation of an own theme.

c) This theme's (php) page templates must integrate fully in above said development, version control and deployment scheme of the already existing site. They should use the same common building blocks, style sheets and tools.

d) Therefore and for other reasons WP generated web content must (use case A)) respectively should (use case B)) use the same coding – ISO-8859-1 in our examples – as the existing sites *).

e) The texts, contributors, commentators, authors and WP-adminitrators are confronted with, have to meet some minimal professional / business standards. **)

Note *): Not to use UTF-8 but ISO-8859-1 or similar if the content is restricted mainly to Western European languages perfectly makes sense. With text files and snippets in a fitting 8 bit encoding you won't even notice. Bringing UTF to such environment, you get a big bag of ugly effects, non-working German and French spell checkers being only one of them.

Note **): This concerns WP's translation facilities mainly for German language. WordPress is not multi-lingual – you just can set one Blog language by configuration. With some expenditure it can at least be educated to be bi-lingual in the sense of “English” and “not English”, see below. Luckily this is what is most often needed – de-DE (German) plus en-GB e.g.

3. Implementation – problems – solutions

To have WP3 we update the existing Blog(s) (use case B)) or install WP3 into the existing web space (use case A)) the steps are:

- installing WordPress 3.5.0 – updated later to .3.5.1 – in a new directory named /blog/ of the web-space,
- providing a mySQL schema,
- making the due modification to Apache's configuration and then
- doing the notorious "5 minutes" initialisation

That's as easy as with the versions before.

3.1 Debugging WP3.x

The routine is deranged by 3.5x seeming much more buggy than say 2.8x. Hence one has to keep "debug on" and fight with php until none of the (de) bug reports re-appear.

Observation: Googling the report texts seldom leads to a page helping the developer but to hundreds of company pages exposing these reports to their visitors.

The first step out of the jungle is to replace numerous deprecated functions and parameter setting. In most of those cases the replacement recipe is directly found within the deprecated code. (Here one doubts the 3.5.x developers ever having used debug= on.)

The next step is to fight the real (hard core) bugs. This mostly involves getting some precondition checks (`isset()`) in order. In the end debugged were: `wp-includes/comment.php`, `wp-includes/cron.php`, `wp-includes/functions.php`, `plugins\user-photo\user-photo.php`, `<xyzTheme>/contact.php` and some more.

3.2 Integrating it as part of an existing design

This integration (use case A)) or adaption (use case B)) is mostly style sheet work. One has to create a (combined) style sheet integrating the existing one and WP's extra needs.

And it is usually recommendable to write a new theme from scratch mimicing the existing look. This sounds harder than it was in the end. One only has to muster the discipline to do it consistently and to start it in spite of dozens of alluring themes available.

```
/* Theme:   Meva4we
Theme Name: meva4we
Theme URI: http://meva-lab.de/
Description: The 2013 theme for blog.a-weinert.de is fully integrated in the
       sites a-weinert.de and meva-lab.de. It hence uses ... all in one line.
Author: Albrecht Weinert
Author URI: http://a-weinert.de/
Version: 1.00
Tags: light, one-column, right-sidebar, flexible-width
Copyright 2013   Albrecht Weinert   a-weinert.de
V.$Revision: 304 $ ($Date: 2013-04-15 11:04:46 +0200 (Mo, 15 Apr 2013) $)
*/
@import url(http://a-weinert.de/inc/meva.css);
```

Listing 3: The (new) theme's style sheet.

After putting WP's extra needs into the site's existing style the theme's css is reduced to Listing 3. This .css file has to live in the theme's directory. It is, in the first place, a WP “file header”. WP file headers define a syntax within the first comment to give some (meta) information available to some WP API functions and in the administrative back end. For a theme this meta information has to be put in the theme's own style sheet. (Therefore if for nothing else it is required.)

Secondly the (empty) theme style sheet just references the common style sheet in the last line. This should be kept, even if our specialised theme (its header.php in most cases) references the common style sheet directly.

If there is more in the theme's .css (listing 3. page 6), i.e. if you find css rules, we do not have a common style for the site and the Blog yet.

By the way: Having static sites and WP Blogs using the same css (and all elements related there) also alows static pages to use some Blog style elements freely.
That may be quite attractive.

3.3 Making all components use ISO-8859-1

In former versions one could choose the “WordPress encoding” in the administrative backend or “dashboard”, perhaps wrongly suggesting it could painlessly changed on the fly. But doing so usually spelled disaster. That handle is gone in 3.5.x (and rightly so at WP's present state).

To get 8859-1 one has to do according settings in the empty database (mySQL schema) as well as in the configuration files before touching the initial (“five minutes”) configuration. Forgetting it (requirement d) above) one is in for a complete “throwaway and redo”.

Remark / Warning: So what remained in WP 3.x is WP's complete inflexibility concerning encoding – much worse than with languages. In our use case B) (i.e. making an existing Blog look like existing sites) one should refrain from changing the Blog's encoding even if it is UTF-8 and the site and tools are ISO8859-1. A workaround for common text blocks is the retreat to stupid US-ASCII as common denominator by using HTML entities.

Additionally many WP programmers, copywriters and translators simply assume UTF-8 being set for all (web) content in spite of possible alternative settings. These hard-coded texts are not re-encoded or modified automatically. Hence one is forced to search the UTF-8 doubles and replace them by either

1. ä ö ü Ä Ö Ü ß ê or
2. &auml; &ouml; &uuml; &Auml; &Ouml; &Uuml; &szlig; &ecirc; …

As English text is unconcerned, the process can mostly be automated.

Approach 1's advantage is readability and spell checkers working. Its minus is being unusable in case of stepping back to or alternate use in an UTF setting.

Approach 2's advantage clearly is being encoding independent, albeit unreadable and annoying spell checkers. Additionally the HTML entities are not accepted by some browsers in alt and title tags / attributes, so for image and tool tips one has to resort to 1. (The same applies if the text blocks are used as .xml.)

3.4 Making it behave

This (extra requirement e page 5) was the point most underestimated with regard to labour. In German one has to address persons, one is neither close friend nor near relative to, in third person plural instead of the obvious second person. It is “Sie sollten ...” instead of “Du sollst ...” (“Thou shalt ...”). That difference is almost completely gone in English (except John Steinbeck's Cannery Row) but still present in many other languages. As you might have suspected all WP's text translations to German treat you as pal – i.e. using the “Du”.

A consequent “Du” instead of “Sie” could perhaps be sold as cultural background of the underlying software. But yet WP's English addressing of the users has a tendency to be extra cool by using some ghetto like slang (“Slow down cowboy ..”). As the German open source scene is in many respects worse than all others you find wordings much less adequate.

As most of the texts may pop up to the user / customer it was tried to fit them all to interpersonal respect. WP's translation files to German add up to 1 MB text. So having done that to about 80% was an immense work. It was slightly complicated by the missing tools and documentation about WP's translation approach.

To give the answer: It is php's .mo/.po-approach using the full English clear text as key to the particular translation. And the tool needed is poedit(.exe) (V1.5.4 April 2013).

3.5 Writing an own specialised theme

As suggested a few times above we get our requirements best fulfilled by writing an own theme. There are some tutorials on that and it is easier than one may think if one understands the basics. A minimal theme is a new subdirectory in /conten/themes/ accordingly named just containing style.css (compare listing 3. page 6) and an index.php. A more realistic furnishing is listing 4.

```
21.03.2013  06:58             1.647   404.php
12.04.2013  17:27             3.309   archive.php
22.03.2013  16:09             5.032   comments.php
20.03.2013  18:02             3.300   footer.php
19.03.2013  13:13             6.934   functions.php
15.04.2013  11:07             4.629   header.php
22.03.2013  16:09             1.931   index.php
12.04.2013  17:27             1.816   page-archives.php
19.03.2013  16:09               947   page-links.php
19.03.2013  16:09             1.875   page.php
22.03.2013  16:09             2.633   search.php
19.03.2013  13:13             1.929   sidebar.php
22.03.2013  16:09             1.885   single.php
15.04.2013  11:07               564   style.css;
```

Listing 4: The (new) theme's files.

Of course, there are cons to writing an own and specialised theme:

- There are so many (hundreds) nice themes around; why re-invent the wheel?
- The specialised theme is not flexibly configurable, changing or adding something means programming (and not just ticking check-marks in the dashboard).

The pros are

- The specialised theme is not configurable but flexibly programmable to the exact requirements given.
- The (php) code involved in the end is smaller, better to understand and faster than a configurable army knife. That is not important in most cases – but with some providers it brings you within php CPU time limits.
- Extra functionality brought in later, like e.g. bilingualism, is done by relatively few extra lines of code instead of bringing in tons of plug-in code, besides the pains of choosing the right plug-in and the risk not to be underestimated.

4. 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 some people involved strive to affirm all unfair prejudices against open source – at least so it seems at some points. But … that's the point were the openness and flexibility comes in again: We can change and improve it to met our needs – or we can have other people do that work for us.

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