Tikiwiki, PHP, UTF-8 character encoding and MySQL

Table of contents

- Tikiwiki, PHP, UTF-8 character encoding and MySQL
  - What is UTF-8 and why should one use it ?
  - Is Tikiwiki fully UTF-8 ?
  - Why everything seems to work and what's wrong ?
  - What kind of problems may I encounter if I use Latin1 tables for UTF8 data ?
  - Ok, so... how does all this stuff work ?
    - The web browser
    - PHP in association with a web server
    - The database server (MySQL in this article)
    - Interaction between the three components
  - What about MySQL collations ?
  - So... I'm using Tiki4, with the new PDO abstraction layer, and more and am sure that everything is fine... am I right ?
  - Related links
  - Alias

What is UTF-8 and why should one use it ?

In short, UTF-8 is a character encoding that uses 1 to 3 bytes for each character.
It is one of the existing character encodings of the UCS (Universal Character Set), that contains nearly a hundred thousand abstract characters (including ASCII characters).

UTF-8 greatly simplifies the task of internationalization by replacing multiple alternative encodings (such as ISO8859-15 Latin-9, which encodes those English, French, German, Spanish and Portuguese characters not available in ASCII).

To learn more about UTF-8, you should read this :

- The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets (No Excuses!),
- Wikipedia UTF-8 article,
- Wikipedia Universal Character Set article
- http://www.phpwact.org/php/i18n/charsets

Is Tikiwiki fully UTF-8 ?

In most cases, it seems to be... but the answer is : No ;)
(... and it also depends on which version we are talking, since Tiki 3.x LTS is able to work nicely with UTF-8 databases/content - supposing your MySQL installation is fine too and your DB really created using UTF-8 charset encoding - but there was a regression that makes Tiki4 / Tiki5 broken)

Nowadays, a typical MySQL installation will - most often - create, by default, Latin1 and not UTF8 databases. And since Tikiwiki doesn't specify the database, table, or field encodings in its installation scripts, the data sources are typically created in Latin1. So... data is stored in Latin1 tables and not in UTF8 ones.

... And, with Tiki4 and - for the moment - the future Tiki5, it's even worse because your data will be corrupted even if your MySQL DB is well configured to be an UTF-8 database (see below for details).

Why everything seems to work and what's wrong ?

Because it's not completely impossible to have UTF8 data in Latin1 tables !

The most frequent mistake is to think that specifying encoding at the web server level (through HTTP headers) or in the HTML code is enough. This is not so easy to understand, especially because this could work (or let you think it works) in many situations.

In fact you should also use an UTF8 database with UTF8 content. Many people are missing this part of the problem or don't even know that their databases are not correctly configured to handle UTF8 data.

What kind of problems may I encounter if I use Latin1 tables for UTF8 data ?

As said before, some UTF8 characters use 3 bytes. Since your database server will not be aware of the real character encoding, you may have this kind of problems :

- Truncated data.

Example : If you try to store an 8 character string that contains 5 of those 3byte-UTF8 character, your data will require 18 bytes of space and not 8 bytes. So, if you try to store this string in a database field defined to be a string with a maximum lenght of 8 characters, your string will be truncated to 8 bytes.

- Wrong results from some database functions.

Example : If you use functions that will count the number of characters in the string or that will return a substring, you may respectively have a total of characters greater than the what it should be, and a substring smaller than the one expected.

Ok, so... how does all this stuff work ?

There are three major components you should consider when trying to understand how encoding works for an application like tikiwiki :

- The web browser
- PHP in association with a web server
- The database server

The web browser

Browsers are able to auto-detect character encoding if nothing is specified. This is absolutely not a good thing. Consider an empty page with just a form (FORM tag) to send data to your web application. The browser can't detect, for example, UTF8 if there is only ASCII chars in your page (remember that UTF8 characters that are already available as ASCII chars are stored in only one byte, as if UTF8 was not used).

This bad feature exists only because web browsers are designed to work with most content, including those created by people that forgot to specify the encoding or even don't really know what it is.

In the case of an (X)HTML or XML page generated by PHP, there is multiple way to specify the encoding to the browser :

- asking the web server to send an appropriate HTTP header to the browser, which can be done either by using the php "header" native function in the application code (1), or by setting up the web server to send this header (2) :

(1) header('Content-Type: text/html; charset=utf-8');
(2) directive for Apache (to be placed, for example, in httpd.conf or .htaccess files) : AddDefaultCharset UTF-8

- using XML or HTML tags :

(3) HTML tag :

(4) XML tag :
There is not only one method because :

- (1) is useful if you are neither generating HTML or XML code (e.g. plain text output), nor able to modify the web server configuration (e.g. not allowed by the server admin),
- (2) is useful if you are neither generating HTML or XML code, nor using a programming language such as PHP,
- (3) is useful if you are not using PHP (e.g. static files) and have no way to modify the web server configuration,
- (4) is useful if you want to be XML or XHTML compliant (see XHTML 1.0 specification)

Since Tikiwiki should be at least XHTML 1.0 compliant, both (3) and (4) should be used. It is a good idea to use (1) and (2) too, because we have static and dynamic non-(X)HTML and non-XML pages. Note that HTTP headers are used in priority if they are different from HTML meta tag.

Now that the browser knows the encoding of the web server output and input, let's talk about the two other components...

PHP in association with a web server

PHP doesn't handle UTF-8 natively (until PHP6), this is why you should use php mbstring extensions. There is no need to use mb* functions if you enable the mbstring functions overloading (see "Function Overloading Feature" in php mbstring extensions documentation).

(There is also a possibility to set default_charset = "utf8" in the php.ini configuration file. Comments in this file explains that : "As of 4.0b4, PHP always outputs a character encoding by default in the Content-type: header. To disable sending of the charset, simply set it to be empty.")

Configuration example :

default_charset = UTF-8
mbstring.language = Neutral
mbstring.internal_encoding = UTF-8
mbstring.encoding_translation = On
mbstring.http_input = UTF-8
mbstring.http_output = UTF-8
mbstring.detect_order = auto
mbstring.substitute_character = none

mbstring.func_overload = 0

If you don't have access to the server's php.ini configuration file, you can use this syntax in .htaccess files :

php_value mbstring.language "Neutral"
php_value mbstring.internal_encoding "UTF-8"

...

(Note that tikiwiki source code also need an encoding. Most of the developpers didn't have to worry about that since only the translations files contains non-english words (i.e. ASCII characters). The good news is that tikiwiki translation files (lang/*/language.php) are already in UTF-8 ;))

The database server (MySQL in this article)

First of all, only MySQL at version 4.1 or more supports Unicode (see MySQL documentation).

MySQL is configured to communicate with "clients". PHP (through it's mysql extensions) is one of them. An encoding is used for the data exchange between MySQL and it's clients. MySQL is able to convert it's data on-the-fly to or from another encoding, this is why this encoding is not absolutely the same as the database's one.

There is two ways to specify this encoding :

- (1) by forcing MySQL to always use the same encoding for all clients,
- (2) by setting up the client to send this information

(1) is done by adding the following lines in your MySQL configuration file (/etc/mysql/my.cnf on Debian GNU/Linux) :

[client]
default-character-set = utf8
[mysqldump]

default-character-set = utf8
(2) this should be done by sending this query to the database :

SET NAMES 'UTF8';
Unfortunately, this is not completely functionnal in practice and this is a known problem 😟

In fact, for the solution (1), PHP mysql extension doesn't read the /etc/mysql/my.cnf file and don't use the specified encoding. In order to handle this, you will need to use the mysqli (MySQL Improved) extension for PHP. By chance, this one is already useable with Tikiwiki, because :

- The API of mysqli is compatible with mysql's one,
- Tikiwiki use ADOdb as the database abstraction layer, which is also able to use mysqli

This new extension has been written to be used with MySQL 4.1 or later, in order to handle the new functionnalities of this MySQL version, including the UTF8 stuff. (For more information about MySQLi, check PHP documentation or this MySQLi overview)

To switch to mysqli extension, you need to :

- check it is installed on your server (on Debian, the php5-mysql package include it),
- modify your tikiwiki db/local.php file in order to set "$db_tiki='mysqli';" instead of "$db_tiki='mysql';"

(There is also a known MySQL problem with a hardcoded limit of 1000 bytes for the database keys. Considering that keys have a fixed size depending on the maximum size of fields they are based on, and that MySQL calculate this maximum size using the worst case (3 bytes for all characters))

(For latin1 to UTF8 data migration, you can find articles via google ;))

Another important point concerning MySQL is the encoding really used to store data. There is no need to explicitely specify the tables or fields character set, and Tikiwiki doesn't specify anything. You only need to specify UTF8 as the whole database encoding. This is done with a query like this :

CREATE DATABASE `tikiwiki` CHARACTER SET utf8;

(There is no need to specify collation, the default one will be based on the character set)

Just for information, if you want to change your MySQL default character set (that will be used when creating a database without specifying character set), add this to you my.cnf configuration file :

[mysqld] default-character-set = utf8

Interaction between the three components

More precisely, when somebody ask Tikiwiki a web page through it's browser, the following steps are done :

- Step 1: the browser send an HTTP request (i.e. GET or POST) to ask the page at the web server hosting Tikiwiki,
- Step 2: the server ask PHP to generate the page,
- Step 3: PHP interprets the source code of the page and request (if needed) the data to MySQL using mysql*() PHP functions to send the SQL query,
- Step 4: those mysql*() functions will then call specific libraries to get data from MySQL,
- Step 5: MySQL executes the request and send the data to PHP through the same libraries,
- Step 6: PHP generate the page (often HTML code),
- Step 7: the server send the page to the browser,
- Step 8: the browser display the page.

What about MySQL collations ?

Collations are only used to know how to sort data. That's all

So... I'm using Tiki4, with the new PDO abstraction layer, and more and am sure that everything is fine... am I right ?

Sadly, again, no.

There is currently a bad regression that is still not fixed in Tiki4 and Tiki5 (this does not affect you if you choosed to use the AdoDB abstraction layer).

ML: I disagree with this statement. Tiki4 with PDO handles the data the same way that Tiki3 and before does. All my upgrades went smoothly.

This is already bad because many characters that exists in UTF-8 encoding does not exist in Latin1 (I'm not speaking about accentued characters like "é", which can be converted to their latin1 equivalents. But japanse kanji, for example, does not exist in latin1 encoding. As an additionnal note, you can often use this string to make some UTF-8 tests : Iñtërnâtiônàlizætiøn)...

... but, with the new "PDO" DB abstraction layer of Tiki4 (and for the moment also with the future Tiki5), the situation is even worse. You won't see any problem with those characters because Tiki will encode them into UTF-8. The problem is that you finally get, in terms of raw database data, either UTF-8 content inside latin1 DB or double encoded UTF-8 content inside UTF-8 DB... really bad... but, as a workaround, you can still configure your Tiki to use the old AdoDB abstraction layer.

ML: again: I disagree that it's "worse" or "really bad" . I have site in many languages. The UTF-8 on Latin-1 tables permits me to have a site in Arabic even though my DB is not UTF-8. Not ideal, and should be improved but it works and it's not a catastrophe. And it can be a great workaround on some shared hosting contexts where full utf-8 is not available.

There is currently discussions about this on the devel list and this problem should be, hopefully, fixed before 5.0 final release.

Related links

- http://irc.tiki.org/irclogger_log/tikiwiki?date=2010-07-17,Sat&sel=40#l36

Alias

- utf8
- Collation
- Collations
- charset
- charsets