Multilingual Information Processing based on UTF-8 Character Encoding

Mario F. Gaul <mario.gaul@gmx.net>,
Timo Schmidt <timo@xomit.de>

October 18, 2002

Contents

Howto build a multilingual web-based application based on `UTF-8` character encoding.

1 Introduction

This article gives an overview of how to use the universal character encoding standard for representation of text (Unicode Standard) to build a multilingual web-based application. Generally it's about the visual presentation of information on the client side: a common web browser.

A basic knowledge of various procedures, techniques and protocols (like `HTTP` for example) is assumed, as the focus is on the practical implementation of a multilingual web-application.

For further informations about Unicode and UTF-8, please refer to the "Unicode and UTF-8 FAQ" [4] of Markus Kuhn, "Alan Wood's Unicode Resources" [4] and - last but not least — "The Unicode Standard Version 3.0" [1].

2 Internationalization

Although international employment was one of the key requirements when starting a software project for a german billing company, it soon became clear that there are a couple of practical reasons why a web-application should be capable of supporting different languages - irrespective of whether used only in one or 10 countries with different languages.

The most important one is the fact that the application can be centrally maintained: It's only interface to the user is the language in its written form. The Unicode Standard [1] provides the coding system which (theoretically) can represent over 65.000 characters, including the most common ones like `ISO-8859-1` (Latin-1), `ISO-20022-JP` or `ASCII`. Hence all you need is a standardized character encoding which allows the use of those characters in a web-application resp. their representation in a browser.

3 Language Catalogs

The use of a catalog containing all (character-based) information to be displayed in a specific language - that is to say available in Unicode - is the easiest way to make a web-application multilingual. Whenever you need a new language, you just add a new catalog. After all you can leave it up to the user which language he prefers to choose. But how does this work? A short introduction to the technical problems: The 16-bit-Unicode representation of a character is not 8-bit clean, so there has to be done some encoding before the characters can savely be transfered by `HTTP`. `UTF-8` encoding does the job - and preserves `ASCII`-compatibility which is important for the use of the language catalogs as we will see later on. As we are talking about a web-application, there might as well be some structural or functional elements, like a button (image) used in a form saying "Bildschirmarbeitsplatzbrillenlagerbestand" anzeigen". It is no big effort to also create graphics multilingual resp. according to the language chosen - just add a few graphic libraries to your system and you'll understand what the button says (provided that the interpreter who translated the words has ever heard of them...).

4 Pre-Conditions

To develop a multilinugal, web-based application, the following pre-conditions are required:

- Webserver (We all know the best one arround :-))

  To serve the web-based application, a webserver is required. This should not be a surprise, at least not at this conference. :-)

- PHP

  `PHP` [3] comes with all features to create and manipulate the desired `UTF-8` sequences. If the usage of `UTF-8`-labeled images is desired, PHP must support `GD`-, `PNG`- and FreeType-support (see below).

- A Unicode capable editor

  To create and edit a Unicode file, you will need a Unicode capable editor to create an Unicode language catalog. There are some quite good Unicode capable editors available for this job. One of them is "Yudit" [6] for the X-Window System which is supposed to run on all unices (even a Win32 binary is available).

- A Unicode to `UTF-8` converting library

  Once the Unicode files are created, a Unicode to `UTF-8` library [10] converts the Unicode language catalogs to `UTF-8` sequences to prepare them for the journey through the net.

- Properly set HTML header

  To ensure a correct representation on the client side, the HTML file must contain the following header to handle the `UTF-8` sequences properly:

```
header("Content-Type:text/html; charset=UTF-8")
```

Nearly all up-to-date browser are capable of handling Unicode and `UTF-8` and therefore fulfill the required pre-conditions to display the characters accurate.

- `UTF-8`-labeled images

  To create language specific buttons as `GUI`-elements, the following freely available graphic libraries and tools are required in addition the already mentioned pre-conditions:

  - GD [7]
    A graphic library to create `JPEG` and `PNG` images
  - `PNG` [8]
    Portable Network Graphics
  - FreeType [9]
    A TrueType Font Engine

5 Implementation

Our aim is to develop a web-based application which can be used worldwide. Choose your language!

5.1 Data storage

Data in this context is static text information which represents the language catalog. Data can be stored in files (e.g., XML) or in a database. It may depend on the scale of the project to decide which solution is best.

> **Tip:** If you decide to use a database to store the language catalogs, please keep in mind, that - depending on the choosen langugage - the data requires more space to be stored as `UTF-8` sequences compared to the native character set as `UTF-8` encoded sequences may use up to four bytes to represent one character!

5.2 Format of the language catalog

The following short example describes a catalog which is stored in a file. By means of a Unicode-capable editor (e.g., Yudit) with the desired font for the language installed we create the catalog formatted like this:

```
IDENTIFIER_1 VALUE_1
     .          .
     .          .
     .          .
IDENTIFIER_N VALUE_N
```

The identifiers, which function as keys during the subsequent procedures, should only contain `ASCII`-characters. The converter used later on will interpret a complete line as a key-value pair, i.e. linebreaks if desired

must be set explicitly in the value part (i.e. `VALUE_N`).

Let's see how we use this catalog in a web application. We assume PHP is at hand.

5.3 The Converter: Converting Unicode to UTF-8

Although there are existing libraries (e.g., `iconv`) which somehow do encodings on unices, we decided to write a new converter (to get more involved in the encoding process). The converter gets the Unicode catalog file as parameter and returns a PHP-file as result. It is written in C due to the possibilty of using such mechanisms like `make` or `autoconf`, which especially offer advantages when it comes to large scale projects. What it does: It verifies that the Unicode language catalog provided contains Unicode and reads it's content line-by-line. Each Unicode character is converted to it's 8-bit representation (this can be one or more 8-bit values, depending on the character). The resulting bytes are transformed according to the `UTF-8` Coding table, which can be found in "The Unicode Standard Version 3.0" [1].

After each line was processed, the encoded character strings are written to a PHP file. The key-value pairs in the Unicode encoded catalog already give a hint on how they are used by PHP — as variables resp. arrays.

Below is an example of what the catalog will look like after converting it to be PHP-conform, so that it can be included wherever needed:

```
<?php
$__TXT['IDENTIFIER_1'] = "VALUE_1\nLinebreak!";
$__TXT['IDENTIFIER_2'] = "VALUE_2";
             .           .      .
             .           .      .
             .           .      .
$__TXT['IDENTIFIER_N'] = "VALUE_N";
?>
```

Keep in mind that the values now are ready-to-use UTF-8 strings.

5.4 Handling user input

Since we're now able to display text defined within the catalogs, we're at the point to take care about the data entered by the user via HTML forms.

Defined by the right HTML header (please refer to section 4), we can expect all data entered by the user as UTF-8 encoded sequences. These sequences can be treated as normal strings. But keep in mind, that one character could be represented by more than one byte. To determine e.g. the string length, the PHP function strlen() will return a length of ten characters where possibly only six characters are. The following function will determine the right string length:

```
function utf8strlen($string)
{
    $length = 0 ;
    for ($i = 0; $i < strlen($string); ++$i)
    {
        $testbit = ord(substr($string, $i, $i + 1));
        if($testbit & 128)
        {
            if ($testbit & 192 && ~$testbit & 32)
                ++$i;
            elseif ($testbit & 222 && ~$testbit & 16)
                $i += 2;
            elseif ($testbit & 238 && ~$testbit & 8)
                $i += 3;
        }
        ++$length;
    }
    return $length;
}
```

6 An Example Application

Basically the application will work as follows. We create language specific catalog files with a Unicode capable editor, convert the files to `UTF-8` files and include them in the application.

Let me guide you now step by step through the complete process.

1. Let's assume, we're in the document root of the Apache Webserver. We open a file `main.uni` with the `Yudit` Unicode editor and enter the following:

   ```
   % yudit main.uni
   HEADLINE       Welcome to the Address DB
   INPUT_NAME     Name:
   INPUT_SURNAME  Surname:
   INPUT_ADDRESS  Address:
   INPUT_CITY     City
   INPUT_STATE    State
   INPUT_EMAIL    E-Mail:
   ```

   The words on the left side are the later used identifiers for the text located on the right side. Identifier and text are seperated by a tab (`\t`).

   Whether you choose such a simple structure as shown above or e.g. an XML file depends on your personal taste. And remember: The identifiers should be entirely written in `ASCII` characters as these identifier must be unique throughout our application and not converted to `UTF-8` sequences.

   Tip: To ensure, that the file really contain Unicode characters we can simply look at the hexadecimal representation

of the file and check the first two bytes. On a UNIX like operating system we might want to type

```
% hexdump main.uni | head -1
0000000 feff 0048 0045 0041 0044 004c 0049 004e
```

If the first bytes are `0xffef` or `0xfffe` everything is fine, otherwise we have to controll the editor settings.

With apropriate fonts we can insert text in any langauge and can use the characters of this certain language without restrictions. We're not stuck anymore on characters only supported by a certain language.

2. The Unicode to `UTF-8` converting library reads the Unicode file line-by-line, converts the Unicode sequences on the right side of the tab (`\t`) and converts them to the respective `UTF-8` representations. For this example, we're going to save the UTF-8 representations as an PHP array with the identifiers as indexes and write them to the file `main.utf8`:

```
% cat main.utf8
<?php
$_MAIN['HEADLINE']      = "Welcome to the Address DB";
$_MAIN['INPUT_NAME']    = 'Name:';
$_MAIN['INPUT_SURNAME'] = 'Surname:';
$_MAIN['INPUT_ADDRESS'] = 'Address:';
$_MAIN['INPUT_CITY']    = 'City:';
$_MAIN['INPUT_STATE']   = 'State:';
$_MAIN['INPUT_EMAIL']   = 'E-Mail:';
?>
```

3. Just like we've created Unicode language catalogs we're now going to create a Unicode image catalog.

   As the language catalog, the image catalog contains an identifier on the left side and text on the right side seperated by the tab charachter (`\t`). But this time, the identifier will be used as the later filename of the image to generate. So let's create the file `main_images.uni` with the following content:

```
% yudit main_images.uni
submit    Submit
cancel    Cancel
```

Processing the file `main_images.uni` will result in two `PNG` images saved as files `submit.png` and `cancel.png`.

4. After converting the Unicode file to `UTF-8` sequences saved in `PHP` syntax in the file `main.utf8` we can easily access the converted text. To include the `UTF-8` file in the file `index.php`, we type:

```
% vi index.php
<?php
/*
 * Include language catalog
 */
require_once('main.utf8');
?>
```

And the `UTF-8` sequences can be accessed by the given array `$_MAIN`:

```
<HTML>
<HEAD>
  <TITLE><?php print $_MAIN['HEADLINE']; ?></title>
  <META http-equiv="Content-Type"
        content="text/html; charset=UTF-8">
</HEAD>
<BODY>
  <H1><?php print $_MAIN['HEADLINE']; ?></H1>
  <form name="enter_address"
        action="index.php"
        method="post">
    <?php print $_MAIN['INPUT_NAME']; ?>
    <input type="text" name="name" size="20" />
    <br />
    <?php print $_MAIN['INPUT_SURNAME']; ?>
    <input type="text" name="surname" size="20" />
    <br />
    <?php print $_MAIN['INPUT_ADDRESS']; ?>
    <input type="text" name="address" size="20" />
    <br />
    <?php print $_MAIN['INPUT_CITY']; ?>
    <input type="text" name="city" size="20" />
    <br />
```

```
      <?php print $_MAIN['INPUT_STATE']; ?>
      <input type="text" name="state" size="20" />
      <br />
      <?php print $_MAIN['INPUT_EMAIL']; ?>
      <input type="text" name="email" size="20" />
      <br />
      <input type="image" src="submit.png" name="submit" />
      <input type="image" src="cancel.png" name="cancel" />
    </form>
  </BODY>
  <HTML>
```

At this point, we are ready to load the file `index.php` with a common browser. To change a language, just replace the file `main.utf8` with a `UTF-8` catalog in the language desired and generated as described in steps 1 to 3 and reload the file `index.php`.

Bibliography

[1] The Unicode Standard 3.0
The Unicode Consortium (`http://www.unicode.org/`)
Copyright © 1991 - 2002 Unicode Inc.

[2] Apache HTTP Server
The Apache Software Foundation (`http://www.apache.org/`)
Copyright © 1999 - 2002 by The Apache Software Foundation

[3] PHP
The PHP Project (`http://www.php.net/`)
Copyright © 2001 - 2002 by The PHP Group

[4] UTF-8 and Unicode FAQ for Unix/Linux
Markus Kuhn (`http://www.cl.cam.ac.uk/ mgk25/unicode.html`)

[5] Alan Wood's Unicode Resources
Alan Woods (`http://www.alanwood.net/unicode/`)
Copyright © 1999 - 2002 by Alan Wood

[6] Yudit
Yudit.org (`http://www.yudit.org/`)

[7] GD
Boutell.Com, Inc. (`http://www.boutell.com/gd/`)
Copyright © 1995 - 2002 by Boutell.Com, Inc.

[8] PNG
Greg Roelofs (`http://www.libpng.org/pub/png/libpng.html`)
Copyright © 2000 - 2002 by Greg Roelofs

[9] FreeType
The FreeType Project (`http://www.freetype.org/`)

[10] Unicode to UTF-8 converting library [1]
Timo Schmidt
Copyright © 2000 - 2002 by Timo Schmidt

[1] For further information about the library, please contact the author at <timo@xomit.de>