Creating Unicode applications with PHP

By Cal Henderson

2010-09-14: I recently found this article, which was written some time around September 1st 2004 (6 years ago). It has since been much improved upon by my book, so this is really only of historical interest.

What is unicode? Unicode is a format for representing all the different characters you might want to on a computer. It includes standard latin characters, symbols, chinese, japanese, korean and vietnamese characters and a whole bunch beside. You're probably more familiar with ISO-8859-1, better known as Latin 1. ISO-8859-1 defines 255 'code points', so each code point can be stored in a single byte. This is great, until your alphabet is greater than 255 characters. Unicode allows a staggering number of characters (65535 for UCS2, 4,294,967,295 for UCS4) and several ways to represent them. The one we'll be dealing with is UTF-8 (Unicode Transformation Format, 8 bit). UTF-8 encodes characters using a variable number of bytes. Characters below 128 (7-bit ASCII) are sent as is, while the higher characters are used to encode the higher codepoints. A description of UTF-8 can be found on the Unicode website

Why bother? Lots of people in the world don't speak english, and many more prefer not to. If you can support any character set, then many more people can use your application.

How can we use it? The first step to making your site Unicode friendly is to start seving UTF-8 content. You'll need two things for this. Firstly, add a meta tag to your html:

```
<meta http-equiv=&quot;Content-Type&quot; content=&quot;text/html; charset=utf-8&quot;>
```

And secondly, send out a HTTP header with the same information before you send out each page:

```
header(&quot;Content-Type: text/html; charset=utf-8&quot;);
```

You'll notice here that you've had to also specify the mime-type of your content. Using text/html is a good bet, even if you're using XHTML, since old browsers (and the GoogleBot) wont break. You're welcome to decide on your mime-type programatically, by looking at the requesting user-agent, but it's best to keep it consistent across the two statements.

Now that you've added these declarations, you need to ensure that any static content within the templates is written in UTF-8. You wont need to change anything unless you're using 8-bit characters, such as letters with accents and some symbols. To write these in UTF-8, you'll need a unicode compatible text editor. Notepad on Windows XP can do this, if you make sure you save your files un UTF-8 mode.

You're now serving unicode content.

But what about getting some user input in unicode too? Well, this is much easier than you might imagine. Any forms submitted will use the encoding type of the page, so your php code will recieve data into $_POST, already un UTF-8. Now, the data you have is no longer a simple string - it's best to think of it as an array of bytes. We can store this in a file or database, without the filesystem or database having to know anything about the format. We can print the data straight out onto any of our pages since it's in the correct format already.

When escaping unicode data, make sure to always use HtmlSpecialChars() and never HtmlEntities(). HtmlEntities() escapes many 8-bit characters into their entity forms (such as accented letters), but any occurance of these bytes in our UTF-8 string are not supposed to be the characters themselves, but are part of the bytes belonging to another character. eek!

HtmlSpecialChars() only escapes < > & ", which are all 7-bit, so are represented as normal in UTF-8.

Sending UTF-8 email Since you have all this lovely unicode data, you don't want to have to throw it away when sending emails, so you'll need to send UTF-8 emails. Email body parts have mime-types, just like web pages, and you can set that with a custom header:

```
Content-type: text/plain; charset=utf-8
```

Unfortunately, the mime-type only applies to the body of the email, not the subject line. To allow unicode characters in the subject line, we need to use quoted-printable escaping (see RFC 1521 for a full explanation). Unfortunately, PHP doesn't have a quoted-printable escaping function. Fortunately, writing one is easy:

```
function escape_quoted_printable($data, $charset='utf-8'){
$data = preg_replace('/([^a-z ])/ie',
 'sprintf(&amp;quot;=%02X&amp;quot;,ord(StripSlashes(&amp;quot;\\1&amp;quot;)))',
 $data);
$data = str_replace(' ', '_', $data);
return = &amp;quot;=?$charset?Q?$data?=&amp;quot;;
}
```

Which will return a nicely formatted string. So let's bundle this all up into a function, also escaping to and from addresses, to allow unicode names.

```
function mail_utf8($to, $subject, $message, $from=''){

# escape fields

$subject = escape_quoted_printable($subject);
$to = escape_quoted_printable($to);
$from = escape_quoted_printable($from);

# prepare headers

$message_headers = &amp;quot;MIME-Version: 1.0\r\n&amp;quot;;
if ($from){
 $message_headers .= &amp;quot;From: $from\r\n&amp;quot;;
}
$message_headers .= &amp;quot;Content-Type: text/plain; charset=utf-8\r\n&amp;quot;;

mail($to, $subject, $message, $message_headers);
}
```

This function has a very similar prototype to mail() and can be used as an almost drop-in replacement, but simplifies the specifying of a from address.

(end of article)