

# DOMINA Ƨ|ng the World

OsCOM 4

September 30th, 2004. Zürich, Switzerland

Derick Rethans <[derick@php.net](mailto:derick@php.net)>

<http://pres.derickrethans.nl/wereldveroverend-oscom4>

- ISO-8859: Group of often used character sets
- latin-X: Group of character sets based on the "latin" set
- Contains most of European's characters
- Adds 96 characters to ASCII (A0 - FF)

ISO	Latin	Usage	Example
8859	1	West European languages	é ë ç à
	2	Central and East European languages	š č ř
	3	Southeast European and miscellaneous languages	
	4	Scandinavian/Baltic languages	
	5	Latin/Cyrillic	Б Ж П Д
	6	Latin/Arabic	ت ج ق
	7	Latin/Greek	α ζ π ω
	8	Latin/Hebrew	כ צ ט ז
	9	Latin-1 modification for Turkish	Ş ş İ ÿ
	10	Lappish/Nordic/Eskimo languages	
	11	Latin/Thai	ท อ น ฃ
	13	Baltic Rim languages	
	14	Celtic	
	15	West European languages	€
	16	Romanian	Đ Ț Ț đ

- Each set has only 256 positions
- Impossible to convert everything
- Conversion will result in broken text
- <http://www.eki.ee/letter/>

ISO 8859-1		ISO-8859-2		Description
Hex	Char	Hex	Char	
E0	à	--	-	LATIN SMALL LETTER A WITH GRAVE
E1	á	E1	á	LATIN SMALL LETTER A WITH ACUTE
E2	â	E2	â	LATIN SMALL LETTER A WITH CIRCUMFLEX
E3	ã	--	-	LATIN SMALL LETTER A WITH TILDE
E4	ä	E4	ä	LATIN SMALL LETTER A WITH DIAERESIS
E5	å	--	-	LATIN SMALL LETTER A WITH RING ABOVE
E6	æ	--	-	LATIN SMALL LETTER AE
E7	ç	E7	ç	LATIN SMALL LETTER C WITH CEDILLA
E8	è	--	-	LATIN SMALL LETTER E WITH GRAVE
E9	é	E9	é	LATIN SMALL LETTER E WITH ACUTE
EA	ê	--	-	LATIN SMALL LETTER E WITH CIRCUMFLEX
EB	ë	EB	ë	LATIN SMALL LETTER E WITH DIAERESIS
EC	ì	--	-	LATIN SMALL LETTER I WITH GRAVE
ED	í	ED	í	LATIN SMALL LETTER I WITH ACUTE
EE	î	EE	î	LATIN SMALL LETTER I WITH CIRCUMFLEX
EF	ï	--	-	LATIN SMALL LETTER I WITH DIAERESIS

- They need more than 256 positions
- Multi-byte
- One character is often one word
- Problems with NULL characters and special characters
- addslashes() mis-recognises chinese-big5 words like "0xA5 0x5C" (¥\, 功, E5 8A 9F)

侂存内洋国

- UCS uses 31 bits for character storage
- Contains all known characters and symbols
- First 128 bytes are the same as ASCII
- First 256 bytes are the same as ISO-8859-1
- Unicode 3.0 describes the BMP (16 bits)
- Unicode 3.1 describes other planes (21 bits)
- Characters are ordered in language/script blocks: Basic latin, Cyrillic, Hebrew, Arabic, Gujarati, Runic, CJK etc.
- Encoding in numerous encodings: UCS-2, UCS-4, UTF8, UTF16 etc.

A Ъ ث ن ا † 媛

- All UCS characters  $> 0x7f$  (127) are stored as multi-byte characters  $\geq 0x80$
- No problems with control characters
- The first byte of a multi-byte character is always in the range  $0xc0 - 0xfd$  and describes how long this byte-sequence is.
- European languages usually use up to two bytes per character
- Characters of other languages usually use up to three bytes per character
- The longest UTF8 encoded character is 6 bytes

### strlen() no longer "works":

```
<?php
$str = "Vær så god!";
echo "The string has length: ", strlen($str);
?>
```

### output:

The string has length: 13

### substr() may mangle text:

```
<?php
$str = "Vær så god!";
echo "The first 7 characters are: ", substr($str, 0, 7);
?>
```

### output:

The first 7 characters are: Vær s♦

### wordwrap() wraps too early:

```
<?php
$str = "Vær så god!";
echo wordwrap($str, 6, '|');
?>
```

### output:

Vær|så|god!

### iconv\_strlen():

```
<?php
$str = "Vær så god!";
echo "The string has length: ", iconv_strlen($str, 'utf8');
?>
```

### output:

The string has length: 11

### iconv\_substr():

```
<?php
iconv_set_encoding('internal_encoding', 'utf8');
$str = "Vær så god!";
echo "The first 7 characters are: ", iconv_substr($str, 0, 7);
?>
```

### output:

The first 7 characters are: Vær så

## preg\_match():

```
<?php
$str = "1978,Dérïck,Rethans";
if (preg_match('@,.{6}),'@', $str, $result)) {
    echo 'We matched: ', $result[1], "<br/>\n";
} else {
    echo 'There was no match!', "<br/>\n";
}
?>
```

## output:

```
There was no match!
```

## preg\_match() in UTF8 mode:

```
<?php
$str = "1978,Dérïck,Rethans";
if (preg_match('@,.{6}'),'@u', $str, $result)) {
    echo 'We matched: ', $result[1], "<br/>\n";
} else {
    echo 'There was no match!', "<br/>\n";
}
?>
```

## output:

```
We matched: Dérïck
```

### iconv():

```
<?php
$str = "1978,Dérïck,Rethans";
echo iconv('iso-8859-1', 'utf8', $str);
?>
```

### output:

```
1978,DÃ©rick,Rethans
```

### iconv():

```
<?php
$str = "© 2004, by Derick Rethans";
echo iconv('iso-8859-1', 'iso-8859-2', $str);
?>
```

### output:

```
Notice: iconv() [function.iconv]: Detected an illegal character in input string in
/home/httpd/pres2/display.php(461) : eval()'d code on line 3
```

### iconv():

```
<?php
$str = "© 2004, by Derick Rethans";
echo iconv('iso-8859-1', 'iso-8859-2//ignore', $str);
?>
```

### output:

```
Notice: iconv() [function.iconv]: Detected an illegal character in input string in
/home/httpd/pres2/display.php(461) : eval()'d code on line 3
2004, by Derick Rethans
```

### ob\_iconv\_handler():

```
<?php
// Output text as-is
echo 'Nittenhundreogåttiåtte', "<br/>\n";

// Set the internal and output encodings
iconv_set_encoding("internal_encoding", "iso-8859-1");
iconv_set_encoding("output_encoding", "utf-8");

// Start the output buffer handler and echo the string again
ob_start("ob_iconv_handler");
echo 'Nittenhundreogåttiåtte', "<br/>\n";

// Flush and stop the output buffer handler
ob_end_flush();
?>
```

### output:

```
Nittenhundreogåttiåtte
NittenhundreogÅ¥ttiÅ¥tte
```

- Most sites don't set any Content-Type header
- Browsers and/or digital environments will move more and more to UTF-8 as default encoding

Result:



```
<?php
    header("Content-Type: text/html; charset=iso-8859-1");
?>
Vær så god
```

Very easy to do, and the result is much better:

Vær så god



## Locales

*language and cultural rules*

A locale is a set of language and cultural rules. These cover aspects such as language for messages, different character sets, lexicographic conventions, etc.

Locale types:

- **LC\_COLLATE**: string collation
- **LC\_CTYPE**: character properties, case sensitivity
- **LC\_MONETARY**: monetary formatting
- **LC\_NUMBER**: number formatting
- **LC\_TIME**: date and time formatting

```
<?php
  $b = $a = array("øks", "ærlig", "åtte", "øyeblikk");

  setlocale(LC_COLLATE, 'C');
  sort($a);
  var_dump($a);

  setlocale(LC_COLLATE, 'no_NO.UTF8');
  sort($b, SORT_LOCALE_STRING);
  var_dump($b);

?>
```

output:

```
array
  0 => 'åtte'
  1 => 'ærlig'
  2 => 'øks'
  3 => 'øyeblikk'

array
  0 => 'ærlig'
  1 => 'øks'
  2 => 'øyeblikk'
  3 => 'åtte'
```

Mathematical order: å (229), æ (230), ø (248)

Natural order: æ, ø, å

## LC\_CTYPE

*language oriented case changes*

```
<?php
  $a = "øyeblikk";

  setlocale(LC_CTYPE, 'C');
  echo strtoupper($a), "<br/>\n";

  setlocale(LC_CTYPE, 'no_NO');
  echo strtoupper($a), "<br/>\n";

  setlocale(LC_CTYPE, 'tr_TR');
  echo strtoupper($a), "<br/>\n";
?>
```

### output:

```
øYEBLIKK
ØYEBLIKK
ØYEBLYKK
```

```
<pre><?php
    $locales = array(
        'Arabic (Egypt)' => 'ar_EG.UTF-8', 'American' => 'en_US',
        'Dutch'          => 'nl_NL',      'Hebrew'   => 'iw_IL',
        'Hebrew (UTF-8)' => 'iw_IL.UTF-8', 'Japanese' => 'ja_JP.UTF-8',
        'Norwegian'     => 'no_NO.UTF-8', 'Turkish'  => 'tr_TR.UTF-8'
    );

    foreach ($locales as $country => $locale) {
        setlocale(LC_TIME, $locale);
        echo '<b>', $country, "</b>\t", strftime('%c'),
            "\n";
    }
?></pre>
```

## output:

```
Arabic (Egypt) 16 2004 , سبت CEST 11:03:46 م
American      Thu 16 Sep 2004 11:03:46 PM CEST
Dutch         do 16 sep 2004 23:03:46 CEST
Hebrew        CEST 23:03:46 2004 ♦♦♦ 16 ♦'
Hebrew (UTF-8) CEST 23:03:46 2004 ה 16 ספט'
Japanese      2004年09月16日 23004203分46秒
Norwegian     tor 16-09-2004 23:03:46 CEST
Turkish       Prş 16 Eyl 2004 23:03:46 CEST
```

```
<pre><?php
    $locales = array(
        'Arabic (Egypt)' => 'ar_EG.UTF-8', 'American' => 'en_US',
        'Dutch'          => 'nl_NL',      'Hebrew'   => 'iw_IL',
        'Hebrew (UTF-8)' => 'iw_IL.UTF-8', 'Japanese' => 'ja_JP.UTF-8',
        'Norwegian'     => 'no_NO.UTF-8', 'Turkish'  => 'tr_TR.UTF-8'
    );

    foreach ($locales as $country => $locale) {
        setlocale(LC_NUMERIC, $locale);
        $ldata = localeconv();
        echo "<b>$country</b>\t",
            number_format("31415.92654", 2, $ldata['decimal_point'],
                $ldata['thousands_sep']), "\n";
    }
?></pre>
```

output:

<b>Arabic (Egypt)</b>	31,415.93
<b>American</b>	31,415.93
<b>Dutch</b>	31415.93
<b>Hebrew</b>	31,415.93
<b>Hebrew (UTF-8)</b>	31,415.93
<b>Japanese</b>	31,415.93
<b>Norwegian</b>	31415.93
<b>Turkish</b>	31415.93

## LC\_MONETARY

*formatting amounts of money*

```
<pre><?php
$locales = array(
    'Egypt'    => 'ar_EG.UTF-8', 'Finland' => 'fi_FI.UTF-8',
    'Germany' => 'de_DE.UTF-8', 'Israel'  => 'iw_IL.UTF-8',
    'Japan'    => 'ja_JP.UTF-8', 'Netherlands' => 'nl_NL.UTF-8',
    'Norway'   => 'no_NO.UTF-8',
);

foreach ($locales as $country => $locale) {
    setlocale(LC_MONETARY, $locale);
    $nr = 31415.92654;
    echo "<b>$country</b>\t",
        money_format("[%i]", $nr), " \t", money_format("[%n]", $nr), "\n";
}
?></pre>
```

**output:**

<b>Egypt</b>	[EGP 31,415.927]	[31,415.927 .م.ج]
<b>Finland</b>	[31 415,93EUR]	[31 415,93€]
<b>Germany</b>	[31.415,93 EUR]	[31.415,93 €]
<b>Israel</b>	[ILS 31,415.93]	[31,415.93 ₪]
<b>Japan</b>	[JPY 31,416]	[¥31,416]
<b>Netherlands</b>	[EUR 31 415,93]	[€ 31 415,93]
<b>Norway</b>	[NOK31.415,93]	[kr31.415,93]

string **setlocale**(const *category*, string *locale*, ...)  
string **setlocale**(const *category*, array *locales*)

Localenames:

- Are OS dependent (deu\_deu vs. de\_DE)
- Are not always available

Example:

```
<?php
    $locale = setlocale(LC_ALL,
        "nl_NL.UTF-8@euro", "nl_NL.UTF-8", "dutch", "nld_nld");
    echo "New locale: ", $locale, "<br />\n";

    $locale = setlocale(LC_ALL, array("nld_nld", "nederlands"));
    echo "New locale: ", $locale, "<br />\n";
?>
```

output:

```
New locale: nl_NL.UTF-8@euro
New locale:
```

A few things are important:

- Non-technical translators should have an easy job
- Easy discovery of translatable text
- There needs to be a fall back to the default language
- It needs to be fast

## Method 1: Constants

*Using define() to define translation strings*

### How it works:

- Each string is a constant
- Different include files for each language

### Definition file:

```
trans/nl.php:
<?php
    define("CLICK_HERE", 'Klik <a href="%1">hier</a> om');
    define("LOGIN", 'in te loggen');
    define("LOGOUT", 'uit te loggen');
?>
```

### Usage:

```
<?php
    include 'trans/nl.php';
    printf(CLICK_HERE . ' ' . LOGIN, 'login.php');
?>
```

## Method 1: Constants

*Using `define()` to define translation strings*

### Pro:

- It is very simple

### Con:

- Translators need to fiddle with PHP code
- Translation files need to be manually created
- Fall back is hardly possible
- `define()` is not fast

## Method 2: Phrase dictionary

*Using an array to define translation strings*

### Pro:

- It is very simple
- Fall back mechanism in place

### Con:

- Translators need to fiddle with PHP code
- Translation files need to be manually created
- Arrays can use quite some memory
- Two definition files are loaded

## Method 3: Gettext

*Using gettext to manage translations*

### How it works:

- Extract phrases: `xgettext gettext.php`
- Create directories:  
`mkdir -p n\LC_MESSAGES`
- Copy file: `cp messages.po n\LC_MESSAGES`
- Translate:  
`vi n\LC_MESSAGES/messages.po`
- Compile:  
`msgfmt n\LC_MESSAGES/messages.po \`  
`-o n\LC_MESSAGES/messages.mo`

### Usage:

```
<?php
bindtextdomain('messages', './');
setlocale(LC_MESSAGES, 'n\');
printf(_("Click <a href='%s'>here</a>") . " " .
        _("to login"), 'login.php');
?>
```

## Method 3: Gettext

*Using gettext to manage translations*

### Translation file:

```
#: gettext.php:4
msgid "Click <a href='%s'>here</a>"
msgstr "Klik <a href='%s'>hier</a>"

#: gettext.php:4
msgid "to login"
msgstr "om in te loggen"
```

### Pro:

- Fall back mechanism in place
- Translation files can automatically be generated
- It's very efficient

### Con:

- Setting it up is not an easy task
- Translators still not have a very easy job

## Method 4: The Qt way

*Using XML based translation files*

### How it works:

- Extract phrases:  
lupdate linguist.php -ts en.ts nl.ts
- Translate: linguist nl.ts

### Usage:

```
<?php
// Parse XML file "nl.ts" into $dictionary

function tr($phrase)
{
    return $GLOBALS['dictionary'][$phrase];
}

printf(tr("Click <a href='%s'>here</a>") . " " .
       tr("to login"), 'login.php');
?>
```

Qt Linguist by Trolltech - /home/httpd/ez\_35/share/translations/dut-NL/translation.ts

File Edit Translation Validation Phrases View Help

Done	Context	Items
?	design/stan...	0/2
✓	design/stan...	7/7
?	design/stan...	0/6
✓	design/stan...	17/17
?	design/stan...	53/55
✓	design/stan...	12/12
✓	design/stan...	25/25
✓	design/stan...	14/14
✓	design/stan...	1/1
✓	design/stan...	6/6
✓	design/stan...	28/28
?	design/stan...	147/173
✓	design/stan...	1/1
✓	design/stan...	23/23
?	design/stan...	253/294
✓	design/stan...	5/5
?	design/stan...	0/23

Done	Source text	Translation
✓	ez.no: Orderconfirmation %1	ez.no: bestellingsbevestiging %1
✓	Remove items	Verwijder producten
?	Customer list	Klanten lijst
?	Number of orders	Aantal orders

**Source text**

Number of orders

**Translation**

Aantal orders

**Phrases and guesses:**

Source phrase	Translation	Definition
Number of phrases	Aantal termen	Guess (Ctrl+1)
Name of operator	Naam van de operator	Guess (Ctrl+2)
Member of groups	Lid van groepen	Guess (Ctrl+3)

1992/2540 MOD

## Method 4: The Qt way

*Using XML based translation files*

### Pro:

- Fall back mechanism in place
- Translation files can automatically be generated and updated
- Translators have a very easy job

### Con:

- The XML needs to be parsed

# DOMINA Ƨ|ng the World

This presentation:

<http://pres.derickrethans.nl/wereldveroverend-oscom4>

Questions?: [derick@php.net](mailto:derick@php.net)

Characters in the ASCII range are stored "as-is"

Characters in the range 0x0080 to 0x07ff:

```

byte 1: 110x xxxx
byte 2: 10xx xxxx

ë (0x00eb)   110x xxxx  10xx xxxx
1110 1011 -> ---+ ++11  --10 1011
              =====
              1100 0011  1010 1011
                   0xc3   0xab
    
```

Characters in the range 0x8000 to 0xffff:

```

byte 1: 1110 xxxx
byte 2: 10xx xxxx
byte 3: 10xx xxxx

功 (0x529f)   1110 xxxx  10xx xxxx  10xx xxxx
0101 0010  1001 1111 -> ---+ 0101  --00 1010  --01 1111
              =====
              1110 0101  1000 1010  1001 1111
                   0xe5   0x8a   0x9f
    
```

## ob\_iconv\_handler():

```
<?php
// Output text as-is
echo 'Nittenhundreogåttiätte', "<br/>\n";

// Set the output and internal encodings
iconv_set_encoding("output_encoding", "utf-8");
iconv_set_encoding("internal_encoding", "iso-8859-1");

// Start the output buffer handler and echo the string again
ob_start("ob_iconv_handler");
echo 'Nittenhundreogåttiätte', "<br/>\n";
ob_end_flush();

// Change the internal encoding
iconv_set_encoding("internal_encoding", "iso-8859-2");

// Start the output buffer handler and echo the string again
ob_start("ob_iconv_handler");
echo 'Nittenhundreogåttiätte', "<br/>\n";
ob_end_flush();
?>
```

## output:

```
Nittenhundreogåttiätte
NittenhundreogÅ#ttiÅ#tte
NittenhundreogÅ@ttiÅ@tte
```