62. FPDF LIBRARY.

62.1. What is FPDF?

FPDF is a PHP class which allows to generate PDF files with pure PHP, that is to say without using the PDFlib library. F from FPDF stands for Free: you may use it for any kind of usage and modify it to suit your needs.

FPDF has other advantages: high level functions. Here is a list of its main features:

- Choice of measure unit, page format and margins.
- Page header and footer management.
- Automatic page break.
- Automatic line break and text justification.
- Image support (JPEG, PNG and GIF).
- Colors.
- Links.
- TrueType, Type1 and encoding support.
- Page compression.

FPDF requires no extension (except ***zlib*** to activate compression and ***GD*** for GIF support). It works with PHP 4 and PHP 5 (the latest version requires at least PHP 4.3.10).

What languages can I use? The class can produce documents in many languages other than the Western European ones: Central European, Cyrillic, Greek, Baltic and Thai, provided you own TrueType or Type1 fonts with the desired character set. Chinese, Japanese and Korean are supported too.

UTF-8 support is also available.

What about performance? Of course, ***the generation speed of the document is less than with PDFlib***. However, the performance penalty keeps very reasonable and suits in most cases, unless your documents are particularly complex or heavy.

62.2. Minimal Example.

Let's start with the classic example:

```
<?php
require('fpdf.php');

$pdf = new FPDF();
$pdf->AddPage();
$pdf->SetFont('Arial','B',16);
$pdf->Cell(40,10,'Hello World!');
$pdf->Output();
?>
```

After including the library file, we create an FPDF object. The FPDF() constructor is used here with the default values: pages are in A4 portrait and the unit of measure is millimeter. It could have been specified explicitly with:

```
$pdf = new FPDF('P','mm','A4');
```

It's possible to use landscape (**L**), other page sizes (such as **Letter** and **Legal**) and units (**pt**, **cm**, **in**).

There's no page at the moment, so we have to add one with *AddPage*(). The origin is at the upper-left corner and the current position is by default set at 1 cm from the borders; the margins can be changed with *SetMargins*().

Before we can print text, it's mandatory to select a font with *SetFont*(), otherwise the document would be invalid. We choose ***Arial bold 16***:

```
$pdf->SetFont('Arial','B',16);
```

We could have specified italics with **I**, underlined with **U** or a regular font with an empty string (or any combination). Note that the font size is given in points, not millimeters (or another user unit); it's the only exception. The other standard fonts are Times, Courier, Symbol and ZapfDingbats.

We can now print a cell with *Cell*(). A cell is a rectangular area, possibly framed, which contains a line of text. It is output at the current position. We specify its dimensions, its text (centered or aligned), if borders should be drawn, and where the current position moves after it (to the right, below or to the beginning of the next line). To add a frame, we would do this:

```
$pdf->Cell(40,10,'Hello World !',1);
```

To add a new cell next to it with centered text and go to the next line, we would do:

```
$pdf->Cell(60,10,'Powered by FPDF.',0,1,'C');
```

Remark: the line break can also be done with *Ln*(). This method additionally allows to specify the height of the break.

Finally, the document is closed and sent to the browser with *Output*(). We could have saved it to a file by passing the desired file name.

Caution: in case when the PDF is sent to the browser, nothing else must be output by the script, neither before nor after (no HTML, not even a space or a carriage return). If you send something before, you will get the error message: "***Some data has already been output, can't send PDF file***". If you send something after, the document might not display.

62.3. Header, footer, page break and image.

Here's a two page example with header, footer and logo:

```
<?php
require('fpdf.php');

class PDF extends FPDF
{
// Page header
function Header()
{
    // Logo
    $this->Image('logo.png',10,6,30);
    // Arial bold 15
    $this->SetFont('Arial','B',15);
    // Move to the right
    $this->Cell(80);
    // Title
    $this->Cell(30,10,'Title',1,0,'C');
    // Line break
    $this->Ln(20);
}

// Page footer
function Footer()
{
    // Position at 1.5 cm from bottom
    $this->SetY(-15);
    // Arial italic 8
    $this->SetFont('Arial','I',8);
    // Page number
    $this->Cell(0,10,'Page '.$this->PageNo().'/{nb}',0,0,'C');
}
}
```

```
// Instanciation of inherited class
$pdf = new PDF();
$pdf->AliasNbPages();
$pdf->AddPage();
$pdf->SetFont('Times','',12);
for($i=1;$i<=40;$i++)
    $pdf->Cell(0,10,'Printing line number '.$i,0,1);
$pdf->Output();
?>
```

This example makes use of the *Header*() and *Footer*() methods to process page headers and footers. They are called automatically. They already exist in the FPDF class but do nothing, therefore we have to extend the class and override them.

The logo is printed with the ***Image***() method by specifying its upper-left corner and its width. The height is calculated automatically to respect the image proportions.

To print the page number, a null value is passed as the cell width. It means that the cell should extend up to the right margin of the page; this is handy to center text. The current page number is returned by the ***PageNo***() method; as for the total number of pages, it's obtained via the special value ***{nb}*** which is substituted when the document is finished (provided you first called ***AliasNbPages***()).

Note the use of the ***SetY***() method which allows to set position at an absolute location in the page, starting from the top or the bottom.

Another interesting feature is used here: the automatic page breaking. As soon as a cell would cross a limit in the page (at 2 centimeters from the bottom by default), a break is issued and the font restored. Although the header and footer select their own font (*Arial*), the body continues with Times. This mechanism of automatic restoration also applies to colors and line width. The limit which triggers page breaks can be set with ***SetAutoPageBreak***().

62.4. Header, footer, page break and image.

Let's continue with an example which prints justified paragraphs. It also illustrates the use of colors.

```
<?php
require('fpdf.php');

class PDF extends FPDF
{
function Header()
{
    global $title;

    // Arial bold 15
    $this->SetFont('Arial','B',15);
    // Calculate width of title and position
    $w = $this->GetStringWidth($title)+6;
    $this->SetX((210-$w)/2);
    // Colors of frame, background and text
    $this->SetDrawColor(0,80,180);
    $this->SetFillColor(230,230,0);
    $this->SetTextColor(220,50,50);
    // Thickness of frame (1 mm)
    $this->SetLineWidth(1);
    // Title
    $this->Cell($w,9,$title,1,1,'C',true);
    // Line break
    $this->Ln(10);
}

function Footer()
{
    // Position at 1.5 cm from bottom
    $this->SetY(-15);
    // Arial italic 8
```

```
        $this->SetFont('Arial','I',8);
        // Text color in gray
        $this->SetTextColor(128);
        // Page number
        $this->Cell(0,10,'Page '.$this->PageNo(),0,0,'C');
    }

    function ChapterTitle($num, $label)
    {
        // Arial 12
        $this->SetFont('Arial','',12);
        // Background color
        $this->SetFillColor(200,220,255);
        // Title
        $this->Cell(0,6,"Chapter $num : $label",0,1,'L',true);
        // Line break
        $this->Ln(4);
    }

    function ChapterBody($file)
    {
        // Read text file
        $txt = file_get_contents($file);
        // Times 12
        $this->SetFont('Times','',12);
        // Output justified text
        $this->MultiCell(0,5,$txt);
        // Line break
        $this->Ln();
        // Mention in italics
        $this->SetFont('','I');
        $this->Cell(0,5,'(end of excerpt)');
    }

    function PrintChapter($num, $title, $file)
    {
        $this->AddPage();
        $this->ChapterTitle($num,$title);
        $this->ChapterBody($file);
    }
    }

    $pdf = new PDF();
    $title = '20000 Leagues Under the Seas';
    $pdf->SetTitle($title);
    $pdf->SetAuthor('Jules Verne');
    $pdf->PrintChapter(1,'A RUNAWAY REEF','20k_c1.txt');
    $pdf->PrintChapter(2,'THE PROS AND CONS','20k_c2.txt');
    $pdf->Output();
    ?>
```

The ***GetStringWidth***() method allows to determine the length of a string in the current font, which is used here to calculate the position and the width of the frame surrounding the title. Then colors are set (via *SetDrawColor(), SetFillColor()* and *SetTextColor()*) and the thickness of the line is set to 1 mm (instead of 0.2 by default) with *SetLineWidth*(). Finally, we output the cell (the last parameter true indicates that the background must be filled).

The method used to print the paragraphs is *MultiCell*(). Each time a line reaches the right extremity of the cell or a carriage return character is met, a line break is issued and a new cell automatically created under the current one. Text is justified by default.

Two document properties are defined: the title (*SetTitle*()) and the author (*SetAuthor*()). There are several ways to view them in Adobe Reader. The first one is to open the file directly with the reader, go to the **File** menu and choose the **Properties** option. The second one, also available from the plug-in, is to right-click and select **Document Properties**. The third method is to type the ***Ctrl+D*** key combination.

62.5. Multi-columns.

This example is a variant of the previous one showing how to lay the text across multiple columns.

```
<?php
require('fpdf.php');

class PDF extends FPDF
{
// Current column
var $col = 0;
// Ordinate of column start
var $y0;

function Header()
{
    // Page header
    global $title;

    $this->SetFont('Arial','B',15);
    $w = $this->GetStringWidth($title)+6;
    $this->SetX((210-$w)/2);
    $this->SetDrawColor(0,80,180);
    $this->SetFillColor(230,230,0);
    $this->SetTextColor(220,50,50);
    $this->SetLineWidth(1);
    $this->Cell($w,9,$title,1,1,'C',true);
    $this->Ln(10);
    // Save ordinate
    $this->y0 = $this->GetY();
}

function Footer()
{
    // Page footer
```

```
        $this->SetY(-15);
        $this->SetFont('Arial','I',8);
        $this->SetTextColor(128);
        $this->Cell(0,10,'Page '.$this->PageNo(),0,0,'C');
    }

    function SetCol($col)
    {
        // Set position at a given column
        $this->col = $col;
        $x = 10+$col*65;
        $this->SetLeftMargin($x);
        $this->SetX($x);
    }

    function AcceptPageBreak()
    {
        // Method accepting or not automatic page break
        if($this->col<2)
        {
            // Go to next column
            $this->SetCol($this->col+1);
            // Set ordinate to top
            $this->SetY($this->y0);
            // Keep on page
            return false;
        }
        else
        {
            // Go back to first column
            $this->SetCol(0);
            // Page break
            return true;
        }
    }
    function ChapterTitle($num, $label)
    {
        // Title
        $this->SetFont('Arial','',12);
        $this->SetFillColor(200,220,255);
        $this->Cell(0,6,"Chapter $num : $label",0,1,'L',true);
        $this->Ln(4);
        // Save ordinate
        $this->y0 = $this->GetY();
    }

    function ChapterBody($file)
    {
        // Read text file
        $txt = file_get_contents($file);
        // Font
        $this->SetFont('Times','',12);
        // Output text in a 6 cm width column
        $this->MultiCell(60,5,$txt);
        $this->Ln();
        // Mention
        $this->SetFont('','I');
        $this->Cell(0,5,'(end of excerpt)');
        // Go back to first column
        $this->SetCol(0);
    }

    function PrintChapter($num, $title, $file)
    {
        // Add chapter
        $this->AddPage();
        $this->ChapterTitle($num,$title);
        $this->ChapterBody($file);
    }
    }
```

```
      $pdf = new PDF();
      $title = '20000 Leagues Under the Seas';
      $pdf->SetTitle($title);
      $pdf->SetAuthor('Jules Verne');
      $pdf->PrintChapter(1,'A RUNAWAY REEF','20k_c1.txt');
      $pdf->PrintChapter(2,'THE PROS AND CONS','20k_c2.txt');
      $pdf->Output();
      ?>
```

The key method used is ***AcceptPageBreak***(). It allows to accept or not an automatic page break. By refusing it and altering the margin and current position, the desired column layout is achieved.

For the rest, not many changes; two properties have been added to the class to save the current column number and the position where columns begin, and the ***MultiCell***() call specifies a 6 centimeter width.

62.6. Tables.

This tutorial shows different ways to make tables.

```
<?php
require('fpdf.php');

class PDF extends FPDF
{
// Load data
function LoadData($file)
{
    // Read file lines
    $lines = file($file);
    $data = array();
    foreach($lines as $line)
        $data[] = explode(';',trim($line));
    return $data;
}

// Simple table
function BasicTable($header, $data)
{
    // Header
    foreach($header as $col)
        $this->Cell(40,7,$col,1);
    $this->Ln();
    // Data
    foreach($data as $row)
    {
        foreach($row as $col)
            $this->Cell(40,6,$col,1);
        $this->Ln();
    }
}
```

```
// Better table
function ImprovedTable($header, $data)
{
    // Column widths
    $w = array(40, 35, 40, 45);
    // Header
    for($i=0;$i<count($header);$i++)
        $this->Cell($w[$i],7,$header[$i],1,0,'C');
    $this->Ln();
    // Data
    foreach($data as $row)
    {
        $this->Cell($w[0],6,$row[0],'LR');
        $this->Cell($w[1],6,$row[1],'LR');
        $this->Cell($w[2],6,number_format($row[2]),'LR',0,'R');
        $this->Cell($w[3],6,number_format($row[3]),'LR',0,'R');
        $this->Ln();
    }
    // Closing line
    $this->Cell(array_sum($w),0,'','T');
}

// Colored table
function FancyTable($header, $data)
{
    // Colors, line width and bold font
    $this->SetFillColor(255,0,0);
    $this->SetTextColor(255);
    $this->SetDrawColor(128,0,0);
    $this->SetLineWidth(.3);
    $this->SetFont('','B');
    // Header
    $w = array(40, 35, 40, 45);
    for($i=0;$i<count($header);$i++)
        $this->Cell($w[$i],7,$header[$i],1,0,'C',true);
    $this->Ln();
    // Color and font restoration
    $this->SetFillColor(224,235,255);
    $this->SetTextColor(0);
    $this->SetFont('');
    // Data
    $fill = false;
    foreach($data as $row)
    {
        $this->Cell($w[0],6,$row[0],'LR',0,'L',$fill);
        $this->Cell($w[1],6,$row[1],'LR',0,'L',$fill);
            $this->Cell($w[2],6,number_format($row[2]),'LR',0,'R',
$fill);
            $this->Cell($w[3],6,number_format($row[3]),'LR',0,'R',
$fill);
        $this->Ln();
        $fill = !$fill;
    }
    // Closing line
    $this->Cell(array_sum($w),0,'','T');
}
}

$pdf = new PDF();
// Column headings
$header = array('Country', 'Capital', 'Area (sq km)', 'Pop.
(thousands)');
// Data loading
$data = $pdf->LoadData('countries.txt');
$pdf->SetFont('Arial','',14);
$pdf->AddPage();
$pdf->BasicTable($header,$data);
$pdf->AddPage();
$pdf->ImprovedTable($header,$data);
$pdf->AddPage();
$pdf->FancyTable($header,$data);
$pdf->Output();
?>
```

A table being just a collection of cells, it's natural to build one from them. The first example is achieved in the most basic way possible: simple framed cells, all of the same size and left aligned. The result is rudimentary but very quick to obtain.

The second table brings some improvements: each column has its own width, headings are centered, and numbers right aligned. Moreover, horizontal lines have been removed. This is done by means of the border parameter of the ***Cell***() method, which specifies which sides of the cell must be drawn. Here we want the left (L) and right (R) ones. It remains the problem of the horizontal line to finish the table. There are two possibilities: either check for the last line in the loop, in which case we use **LRB** for the border parameter; or, as done here, add the line once the loop is over.

The third table is similar to the second one but uses colors. Fill, text and line colors are simply specified. Alternate coloring for rows is obtained by using alternatively transparent and filled cells.

62.7. Links and flowing text.

This tutorial explains how to insert links (internal and external) and shows a new text writing mode. It also contains a basic HTML parser.

```
<?php
require('fpdf.php');

class PDF extends FPDF
{
var $B;
var $I;
var $U;
var $HREF;
```

```
function PDF($orientation='P', $unit='mm', $size='A4')
{
    // Call parent constructor
    $this->FPDF($orientation,$unit,$size);
    // Initialization
    $this->B = 0;
    $this->I = 0;
    $this->U = 0;
    $this->HREF = '';
}

function WriteHTML($html)
{
    // HTML parser
    $html = str_replace("\n",' ',$html);
                          $a      =      preg_split('/<(.*)>/U',
$html,-1,PREG_SPLIT_DELIM_CAPTURE);
    foreach($a as $i=>$e)
    {
        if($i%2==0)
        {
            // Text
            if($this->HREF)
                $this->PutLink($this->HREF,$e);
            else
                $this->Write(5,$e);
        }
        else
        {
            // Tag
            if($e[0]=='/')
                $this->CloseTag(strtoupper(substr($e,1)));
            else
            {
                // Extract attributes
                $a2 = explode(' ',$e);
                $tag = strtoupper(array_shift($a2));
```

```
                    $attr = array();
                    foreach($a2 as $v)
                    {
                         if(preg_match('/([^=]*)=["\']?([^"\']*)/',$v,
$a3))
                              $attr[strtoupper($a3[1])] = $a3[2];
                    }
                    $this->OpenTag($tag,$attr);
               }
          }
     }
}

function OpenTag($tag, $attr)
{
     // Opening tag
     if($tag=='B' || $tag=='I' || $tag=='U')
          $this->SetStyle($tag,true);
     if($tag=='A')
          $this->HREF = $attr['HREF'];
     if($tag=='BR')
          $this->Ln(5);
}

function CloseTag($tag)
{
     // Closing tag
     if($tag=='B' || $tag=='I' || $tag=='U')
          $this->SetStyle($tag,false);
     if($tag=='A')
          $this->HREF = '';
}

function SetStyle($tag, $enable)
{
     // Modify style and select corresponding font
     $this->$tag += ($enable ? 1 : -1);
```

```
        $style = '';
        foreach(array('B', 'I', 'U') as $s)
        {
            if($this->$s>0)
                $style .= $s;
        }
        $this->SetFont('',$style);
    }

    function PutLink($URL, $txt)
    {
        // Put a hyperlink
        $this->SetTextColor(0,0,255);
        $this->SetStyle('U',true);
        $this->Write(5,$txt,$URL);
        $this->SetStyle('U',false);
        $this->SetTextColor(0);
    }
    }

    $html = 'You can now easily print text mixing different styles:
    <b>bold</b>, <i>italic</i>,
    <u>underlined</u>,   or   <b><i><u>all   at   once</u></i></b>!
    <br><br>You can also insert links on
    text, such as <a href="http://www.fpdf.org">www.fpdf.org</a>, or
    on an image: click on the logo.';

    $pdf = new PDF();
    // First page
    $pdf->AddPage();
    $pdf->SetFont('Arial','',20);
    $pdf->Write(5,"To find out what's new in this tutorial, click ");
    $pdf->SetFont('','U');
    $link = $pdf->AddLink();
    $pdf->Write(5,'here',$link);
    $pdf->SetFont('');
    // Second page
    $pdf->AddPage();
```

```
        $pdf->SetLink($link);
        $pdf->Image('logo.png',10,12,30,0,'','http://www.fpdf.org');
        $pdf->SetLeftMargin(45);
        $pdf->SetFontSize(14);
        $pdf->WriteHTML($html);
        $pdf->Output();
        ?>
```

The new method to print text is ***Write***(). It's very close to ***MultiCell***(); the differences are:

- The end of line is at the right margin and the next line begins at the left one.
- The current position moves at the end of the text.

So it allows to write a chunk of text, alter the font style, then continue from the exact place we left it. On the other hand, you cannot justify it.

The method is used on the first page to put a link pointing to the second one. The beginning of the sentence is written in regular style, then we switch to underline and finish it. The link is created with ***AddLink***(), which returns a link identifier. The identifier is passed as third parameter of *Write*(). Once the second page is created, we use ***SetLink***() to make the link point to the beginning of the current page.

Then we put an image with an external link on it. An external link is just a URL. It's passed as last parameter of ***Image***().

Finally, the left margin is moved after the image with ***SetLeftMargin***() and some text in HTML format is output. A very simple HTML parser is used for this, based on regular expressions. Recognized tags are <b>, <i>, <u>, <a> and <br>; the others are ignored. The parser also makes use of the *Write*() method. An external link is put the same way as an internal one (third parameter of *Write*()). Note that *Cell*() also allows to put links.

62.8. Adding new fonts and encoding support.

This tutorial explains how to use *TrueType, OpenType* and *Type1* fonts so that you are not limited to the standard fonts any more. The other benefit is that you can choose the font encoding, which allows you to use other languages than the Western ones (the standard fonts having too few available characters).

Remark: for *OpenType*, only the format based on *TrueType* is supported (not the one based on *Type1*).

There are two ways to use a new font: embedding it in the PDF or not. When a font is not embedded, it is searched in the system. The advantage is that the PDF file is lighter; on the other hand, if it's not available, a substitution font is used. So it's preferable to ensure that the needed font is installed on the client systems. If the file is to be viewed by a large audience, it's highly recommended to embed.

Adding a new font requires two steps:

- Generation of the font definition file.
- Declaration of the font in the script.

For Type1, you need the corresponding AFM file. It's usually provided with the font.

Generation of the font definition file

The first step consists in generating a PHP file containing all the information needed by FPDF; in addition, the font file is compressed. To do this, a helper script is provided in the *makefont* directory of the package: ***makefont.php.*** It contains the following function:

MakeFont(string fontfile, [, string enc [, boolean embed]])

- fontfile : Path to the .ttf, .otf or .pfb file.
- Enc : Name of the encoding to use. Default value: cp1252.
- Embed : Whether to embed the font or not. Default value: true.

The first parameter is the name of the font file. The extension must be either .ttf, .otf or .pfb and determines the font type. If your Type1 font is in ASCII format (.pfa), you can convert it to binary (.pfb) with the help of t1utils.

For Type1 fonts, the corresponding .afm file must be present in the same directory.

The encoding defines the association between a code (from 0 to 255) and a character. The first 128 are always the same and correspond to ASCII; the following are variable. Encodings are stored in .map files. The available ones are:

- cp1250 (Central Europe)
- cp1251 (Cyrillic)
- cp1252 (Western Europe)
- cp1253 (Greek)
- cp1254 (Turkish)
- cp1255 (Hebrew)
- cp1257 (Baltic)
- cp1258 (Vietnamese)
- cp874 (Thai)
- ISO-8859-1 (Western Europe)
- ISO-8859-2 (Central Europe)
- ISO-8859-4 (Baltic)
- ISO-8859-5 (Cyrillic)
- ISO-8859-7 (Greek)
- ISO-8859-9 (Turkish)
- ISO-8859-11 (Thai)
- ISO-8859-15 (Western Europe)
- ISO-8859-16 (Central Europe)
- KOI8-R (Russian)
- KOI8-U (Ukrainian)

Of course, the font must contain the characters corresponding to the chosen encoding.

Remark: the standard fonts use cp1252.

After you have called the function (create a new file for this and include *makefont.php*), a .php file is created, with the same name as the font file. You may rename it if you wish. If the case of embedding, the font file is compressed and gives a second file with .z as extension (except if the compression function is not available, it requires Zlib). You may rename it too, but in this case you have to change the variable $file in the .php file accordingly.

Example:

```
<?php
require('makefont/makefont.php');

MakeFont('c:\\Windows\\Fonts\\comic.ttf','cp1252');
?>
```

which gives the files *comic.php* and *comic.z.*

Then copy the generated files to the font directory. If the font file could not be compressed, copy it directly instead of the .z version.

Another way to call *MakeFont*() is through the command line:

php makefont\makefont.php c:\Windows\Fonts\comic.ttf cp1252

Finally, for TrueType and OpenType fonts, you can also generate the files online instead of doing it manually.

Declaration of the font in the script

The second step is simple. You just need to call the AddFont() method:

```
$pdf->AddFont('Comic','','comic.php');
```

And the font is now available (in regular and underlined styles), usable like the others. If we had worked with ***Comic Sans MS Bold*** (comicbd.ttf), we would have written:

```
$pdf->AddFont('Comic','B','comicbd.php');
```

Example

Let's now see a complete example. We will use the font Calligrapher. The first step is the generation of the font files:

```
<?php
require('makefont/makefont.php');

MakeFont('calligra.ttf','cp1252');
?>
```

The script gives the following report:

```
Warning: character Euro is missing
Warning: character zcaron is missing
Font file compressed: calligra.z
Font definition file generated: calligra.php
```

The euro character is not present in the font (it's too old). Another character is missing too.

Alternatively we could have used the command line:

```
php makefont\makefont.php calligra.ttf cp1252
```

or used the online generator.

We can now copy the two generated files to the font directory and write the script:

```
<?php
require('fpdf.php');

$pdf = new FPDF();
$pdf->AddFont('Calligrapher','','calligra.php');
$pdf->AddPage();
$pdf->SetFont('Calligrapher','',35);
$pdf->Write(10,'Enjoy new fonts with FPDF!');
$pdf->Output();
?>
```

About the euro symbol.

The euro character is not present in all encodings, and is not always placed at the same position:

Encoding	Position
cp1250	128
cp1251	136
cp1252	128
cp1253	128
cp1254	128
cp1255	128
cp1257	128
cp1258	128
cp874	128
ISO-8859-1	N/A
ISO-8859-2	N/A
ISO-8859-4	N/A
ISO-8859-5	N/A
ISO-8859-7	N/A
ISO-8859-9	N/A
ISO-8859-11	N/A

ISO-8859-15	164
ISO-8859-16	164
KOI8-R	N/A
KOI8-U	N/A
	I

ISO-8859-1 is widespread but does not include the euro sign. If you need it, the simplest thing to do is to use cp1252 or ISO-8859-15 instead, which are nearly identical but contain the precious symbol.

Reducing the size of TrueType fonts

Font files are often quite voluminous; this is due to the fact that they contain the characters corresponding to many encodings. Zlib compression reduces them but they remain fairly big. A technique exists to reduce them further. It consists in converting the font to the Type1 format with ttf2pt1 (the Windows binary is available here) while specifying the encoding you are interested in; all other characters will be discarded.

For example, the arial.ttf font that ships with Windows Vista weights 748 KB (it contains 3381 characters). After compression it drops to 411. Let's convert it to Type1 by keeping only cp1250 characters:

```
ttf2pt1 -b -L cp1250.map c:\Windows\Fonts\arial.ttf arial
```

The .map files are located in the makefont directory of the package. The command produces arial.pfb and arial.afm. The arial.pfb file weights only 57 KB, and 53 after compression.

It's possible to go even further. If you are interested only by a subset of the encoding (you probably don't need all 217 characters), you can open the .map file and remove the lines you are not interested in. This will reduce the file size accordingly.