Using Lasso to Create Dynamic PDF documents

This is an introduction to some convenient ways in using Lasso's pdf tags to produce nice looking pdf documents. Examples presented are tested on Lasso 8.5.5. It's possible that most code will work on earlier versions but some of the techniques described are dependent on features not introduced until 8.5.5. Furthermore, all development and testing has been done on an OS X server. Everything should work on other platforms, but that's for others to find out.
The paper is accompanied by a number of demo files that each illustrates a feature described here. The code examples are well commented and hopefully provide additional insights in how to use Lasso's pdf tag suite at it's full potential.

Basic facts

Pixels, Points and halftoning

A **pixel** (**pict**ure **el**ement) is the smallest information carrying element available for computers screens. It can hold only the color value that the element represents. In the context we're interested in, that is using Lasso to produce pdf documents, a pixel is always square. We need to be familiar with pixels if we want images used in our pdfs to look good when printed.
In the early days of computer graphics (Macintosh black and white screens) computer screens had a resolution of 72 pixels/inch. Today they tend to be at least 96 pixels/inch and objects presented on screen appear smaller than in real life.
Points are a typographical unit for measuring font size and leading (row height) and other minute items on a printed page. It unfortunately doesn't represent the same size depending on where the user comes from. It could be a french, an american or a postscript point. Among others. When producing pdfs we're only interested in the postscript version. That, like old day screen resolutions, also happen to be 72 points/inch. Coincidence? I think not. It was one of the factors behind WYSIWYG.
Halftoning is a printing process that enables images and graphics to be represented on paper when printed. The halftoning process uses a resolution, called Lines per inch (LPI), that varies depending on paper quality and type of printing technique used. A newsprint is usually around 85 LPI where a high-quality print could use up to 300 LPI. What's important for us to think about when producing pdfs is that if the document is to look good when printed we need to create images that have at least four pixels for every line in the halftone grid. If we expect the document to be printed on a laser printer using 105 LPI then we need images that have a resolution of 210 pixels/inch.
Did you get lost in the math here? Simple. If you double the resolution going from 105 to 210 pixels/inch you get four times the information. So one pixel becomes four.

Page sizes

The standard page size for most of the civilised world is A4. Why is this a good standard? Because if you divide a paper in the "A" series in half you get a paper that have the exact same aspect ratio. And that's smart. So smart that all the world uses it. Or to quote Wikipedia: "today only the U.S. and Canada have not adopted the system". :-)
The A4 page is 210 x 297 mm. Translated into postscript points it's 595 x 842 points. And if you place images without size params, it's the same in pixels since Lasso's pdf tags presumes 72 pixels/inch.
For those that insist on using US-Letter here's the sizes for that format:
216 x 279 mm or 612 x 792 points.

Placing mysteries. Some start at the bottom. Some at the top.

Lasso's horizontal coordinate grid starts at the left edge of the page and then increments in points. But the vertical grid sometimes start at the upper edge and sometimes at the lower edge depending on what kind of object you're placing. It's possible to place text and image objects on the page using -> Add and the params -left and -top. For

both of these objects the grid starts at the upper left corner. There's no problems using the -left param. It always point on the same location, the left of the object to be placed. But the -top param behaves differently depending on what kind of object you want to place. For text, -top points to the upper left corner of the object. For images, the lower left. If you place graphics like rectangles, lines, arcs or circles the grid starts at the lower left corner of the page.
Confusing? Yes!

Placing text

If you want to place text on the page you can do so using -> Add. It's perfectly all right to use -top, -left, -width and -height params together with -> Add when used on text. The text object will be placed using the objects upper left corner. Where that is exactly depends on the leading used for the object. Say you have a text using a leading of 18.

```
$pdf -> add( $text,
        -width = 300,
        -height = 200,
        -left = 1,
        -top = 1,
        -leading = 18);
```

The text baseline will start 18 points below the -top param and flow row by row down within the box boundaries created by the -width and -height params until there's no more text or there won't fit more text in the box.
Note, if you leave out the -width param, Lasso will ignore -top and -left and instead insert the text in the flow of the PDF_Doc.

In short, text are placed using the objects **upper** left corner on a grid starting on the **upper** left corner of the page.

DrawText?

Why not use the documented technique for exact placement of text, DrawText? Well, in some rare circumstances it is the best choice. But PDF_Doc -> Add allows you to handle more complex text objects. Like when mixing different fonts and styles. Using -> Add also allows you to confine the text to an invisible box set by the placement params. So, get used to using PDF_Doc -> Add whenever possible and resort to PDF_Doc -> DrawText only when necessary.

Placing images

Images can also be placed on a page using -> Add. But in order to make life interesting they are placed using the lower left corner of the object. So placing an image that's 100 pixels high and 200 pixels wide like this:

```
$pdf -> add( $image, -left = 1, -top = 1);
```

will only show a tiny pixel row of the image in the upper left corner. If you wanted to place the image at the very top/left corner of the page you would have to take the image height into consideration.

```
$pdf -> add( $image, -left = 1, -top = 100);
```

In short, images are placed using the objects **lower** left corner on a grid starting on the **upper** left corner of the page.

Adding images with sufficient print quality

As mentioned above a rule of thumb is that an image should have double the resolution in pixels as the printers LPI. If you're used to producing images for printing purposes you know how to set the resolution in for example Photoshop. But, this information is nothing that Lasso cares about. No matter what resolution the image is set to, Lasso will place the image using 72 pixels/inch if not instructed otherwise by the params -width and -height.
Say you have an image that's 100 by 200 pixels. If you want the image to look good on a laser printer or for newsprint it's sufficient to divide the image size by 2 when placing it.

```
$pdf -> add( $image,
        -height = 100 / 2,
```

```
        -width = 100 / 2,
        -left = 100,
        -top = 100);
```
This equals a resolution set in Photoshop to 144 pixels/inch.

If the pdf is to be used for high-quality printing using 150 LPI, divide by 4.

```
$pdf -> add( $image,
        -height = 100 / 4,
        -width = 100 / 4,
        -left = 100,
        -top = 100);
```
That would give you a resolution equalling 288 pixels/inch, good enough for 150 LPI.
It could be that there's someone in the crowd shouting: "But the Language Guide states that I can use -height when creating the image object and thus scale the image proportionally!"
True. That is, it's true that the Language Guide says so. But it's not true that it works.

Adding images with unknown size

The above tip is all good and dandy when you know the size of the image you're placing. But if it's a user supplied image just uploaded to the server the image could be of any size. So in order to place it you first need to find out the size. Well, we can do that using Lasso's image tags. In this example we want to confine the image proportionally to an invisible box that's 200 pixels wide and tall. And to produce a pdf that can be printed with high quality thus preparing it for 150 LPI halftoning.

```
$imginfo = image( 'largeimage.gif', -info);
$height = $imginfo -> height;
$width = $imginfo -> width;

// Find the size factor to use. If the image is to small for chosen
print quality use standard factor (0.24).
// Since we don't know the orientation of the image we use math_max
to find out the largest one of the height and width
$factor = 200.0 / decimal( math_max( $width, $height));
(($factor > 0.24) ? ($factor = 0.24));

var('image' = pdf_image(
        -file = 'largeimage.gif'));

// Place the image within the "box".
$pdf -> add( $image,
        -height = $height * $factor,
        -width = $width * $factor,
        -left = 100,
        -top = 300
        );
```
Why multiply with a factor 0.24? Well since 72 (pixels/inch) divided with 300 (pixels per inch) equals 0.24. And 300 pixels/inch gives 4 pixels per line in a 150 LPI.

Adding graphics

Graphics that are drawn by Lasso on the other hand are placed on the page using a grid that starts from the lower left of the page. This applies to rectangles, circles, lines and arcs etc. The placement params looks a bit different than when placing text and images. Instead of setting -top and -left params you simply write the coordinates in a specific order.
Rectangles are placed like this:

```
$pdf -> rect( 100, 300, 200, 150); // (X, Y, Width, Height)
```

This will draw a rectangle with the lower left corner of the rectangle 100 points from the left edge of the paper and 300 points from the bottom of the page. It will have a width of 200 points and a height of 150 points.
If you want a solid rectangle use the -fill param.

```
$pdf -> rect( 100, 300, 200, 150, -fill);
```

Circles are placed in a similar fashion but the horizontal and vertical coordinates specify the center of the circle instead of the left/bottom. The third param is the radius length in points.

```
$pdf -> circle( 400, 600, 50, -fill); // (X, Y, Radius)
```

In order to draw lines you need to know where it starts and where it ends. Both ends need a horizontal and a vertical coordinate.

```
pdf -> line( 100, 100, 200, 300); // (X start, Y start, X end, Y end)
```

In short, graphics are placed on a grid starting from the **lower** left corner of the page.

Layers

The concept of layers are not used per se by Lasso's pdf tags. But you get the same results when placing objects into a PDF_Doc since object are placed layered on top of each other depending on what kind of objects you're placing and the order you add them to the PDF_Doc.
Placing text using -> Add without placement params puts text at the bottom "layer". Any graphic objects drawn will appear on top of the text regardless if the text is added after the graphic.
Text added with placement coordinates and graphics are layered in the order they are placed. The first added object will appear underneath objects added later.
Images are placed in their own "layer" at the very bottom. They will always appear underneath other objects regardless of the order they are added to the PDF_Doc. Even text placed using -> Add without placement coordinates are layered on top of images.

Same again using other words. Text and graphic objects can cover each other based on the order they are added to the PDF_Doc. Images can never cover text or graphic objects regardless of the order they are added. Images can only cover other images.

Except... There actually exists a layer that's even further down. If you add a pdf file to your PDF_Doc using -> InsertPage the inserted pdf will appear beneath any other objects on the page even if it's the last thing you add.

Changing the look of graphics

Line width

You can control the line width of graphics drawn by Lasso by using PDF_Doc -> SetLineWidth. The setting will apply to all drawing actions taking place after you set the width as long as you stay on the same page.
The width can be a decimal or integer value that Lasso will translate to points. You can have very thin lines. Using for example 0.1 as value is no problem. If you don't specify a line width Lasso will use 1 point as standard.
In order for a setting to not spill over on an object that it wasn't intended for, I tend to set the line width each time I draw a graphics object.

```
$pdf -> setlinewidth( 2);
```

```
$pdf -> rect( 100, 680, 200, 100);

$pdf -> setlinewidth( 4.5);
$pdf -> circle( 400, 630, 50);
```

Note that a thick line will expand in two directions. It will grow into the object and expand outside it equally. Lets say you have a rectangle that's set to be a 100 pixels wide and a 100 pixels high. If you set a line width to 10 pixels, the area that the rectangle, including the border, will occupy grows to 110 by 110 pixels. At the same time the filled part of the rectangle shrinks to 95 by 95 pixels. Take that into account when you create graphics with thick borders. The space they occupy will grow.

Rounded corners

As of Lasso 8.5.5 it's possible to draw rectangles with rounded corners. It's easy to apply. Just set a fifth parameter to a rectangle object specifying the radius in points for the corner.

```
$pdf -> rect( 350, 500, 200, 100, 16, -fill); // This rectangle will
have rounded corners with a radius of 16 points.
```

Rotate it

Text and images can be rotated as of Lasso 8.5.5. The amount of rotation is set by the optional param -rotate in degrees clockwise.

Images

To rotate an image you set the rotation param when you create the PDF_Image object, not when you place it.

```
var('image' = pdf_image(-file = 'ball.gif', -rotate = 40));

$pdf -> add($image, -left = 300, -top = 250);
```

Note that the -left and -top params points at the rotated image bottom left edge created by drawing a vertical line from the pixel furthest to the left and a horizontal line from the pixel furthest down.

Tilted text

Text can be rotated either when inserted using plain -> Add.

```
var('text' = pdf_text('Rotate me!'));

$pdf -> add( $text, -rotate = 90);
$pdf -> add( $text, -rotate = -20);
$pdf -> add( $text, -rotate = 190);
```

Or when placed using DrawText.

```
$pdf -> drawtext('Rotate me and put me at a specific place!',
        -left = 320, -top = 520, -rotate = 20);
```

It would be nice to use rotation when placing text using -> Add and coordinate params but alas it's a no no. Using -> Add with placement for text objects was an undocumented feature at the time so the developer that enhanced the pdf tags with the rotation possibility didn't know that it was possible and thus didn't implement support for rotation for that particular action.
I wouldn't mind seeing support for rotated rectangles too, the only graphic object that could benefit from rotation. (Rotating a circle or a line is rather pointless.) But that's not a supported feature at the moment.

Defining and using color on graphics and text

"You can have any color you want as long as it's black". Henry Fords bold offer to his car buyers holds true for Lasso as well. Or at least the first part of his statement. Because Lasso really can offer any color. And as of Lasso 8.5.5 it can be RGB, CMYK or a spot color. There's several ways of defining the color of an object. Using a -color param, setting the PDF_Docs standard color using -> SetColor or by creating a color object using the new PDF_Color.

RGB, CMYK & Spot?

RGB stands for **Red**, **Green** and **Blue** and it's the way screens (and TVs) create their colors. They do it by adding colored light. By mixing various amounts of red, green and blue light they can produce a wide range of other colors. If you turn on all three lights to 100% you get white. If you turn off all lights, that is, use no colors, you get black. RGB is an Additive Color System since it adds colors (light).

RGB is OK to use if the pdf created is mostly aimed at being read on screen or printed on office printers.

CMYK is the process used when colors are printed, usually on paper. It stands for **Cyan**, **Magenta**, **Yellow** and **Black** (The K is designated to avoid confusion with Blue). The colors are created by applying pigments of color on a paper. The pigments absorbs certain wavelengths of color while reflecting others and thus creates the appearance of certain colors. It's called Subtractive Color System since it subtracts (absorbs) colors. By mixing the colors Cyan, Magenta and Yellow you can create a wide range of other colors. If you use a 100% of each color you get Black. (In theory that is, in practise you get a dark brown, mud like slush.) If you use 0% of each color you get white (or paper color).

Use CMYK if the pdf will be sent to a high quality print shop and you need accurate control over the colors used.

Spot color is another technique for printing colors on paper. In theory using CMYK should give you all the color nuances you want. In reality it won't work. If you want for instance a really bright red, or a vibrant green, gold etc, CMYK won't deliver. Instead you have to print using a specifically blended color, a spot color. This can be the only color used or you can also use black or other spot colors. Or you can use CMYK and add one or several spot colors depending on needs. (And budget...) Spot colors can only be used by traditional printing techniques such as offset printing or screen printing. Digital printing techniques are usually CMYK only. (Yes, yes, I know. There are exceptions. But as a general rule you can't expect spot color handling from a digital print store.)

Use spot colors when there's specific need for them and the resulting pdf is to be printed using offset or screen techniques.

Since screens only handle RGB colors all representations of CMYK or Spot colors on screen are simulated.

Adding color to graphics

Graphics get their color from a PDF_Doc -> SetColor setting. Once set it will affect all graphic objects drawn on the same page. If you add a new page you need to set it again. Since it does affect all drawing operations I tend to set it each time I draw a graphics object.

Pre Lasso 8.5.5 you set the color params directly in SetColor.

```
$pdf -> setcolor( 'both', 'rgb', 0, 0.9, 0.1);

$pdf -> rect( 100, 300, 100, 100, -fill);
```

As of 8.5.5 you can use the new PDF_Color object.

```
var('colorRGB' = pdf_color('rgb', 0, 0.9, 0.1));
```

```
$pdf -> setcolor( 'both', $colorRGB);

$pdf -> rect( 100, 300, 100, 100, -fill);
```

Values can be set separately for lines and fill color.

```
var('greenRGB' = pdf_color('rgb', 0, 0.9, 0.1));
var('redRGB' = pdf_color('rgb', 0.9, 0, 0.1));

$pdf -> setcolor( 'stroke', $greenRGB);
$pdf -> setcolor( 'fill', $redRGB);
```

PDF_Color

Lasso 8.5.5 introduced the PDF_Color object. With it you can define color variables that you reuse on text or graphic objects anywhere on your PDF_Doc.
PDF_Color objects can be set for Gray, RGB, CMYK or Spot colors.

```
var('gray08' = pdf_color('gray', 0.8)); // A color that's 80% gray
var('greenRGB' = pdf_color('rgb', 0, 0.9, 0.1)); // An RGB color
that's rather green
var('E64_1CMYK' = pdf_color('cmyk', 0.3, 0.9, 1, 0)); // A CMYK
color that emulates the Pantone color PANTONE E 64-1
var('E64_1Spot' = pdf_color('spot','PANTONE E 64-1',
1,$E64_1CMYK)); // A spot color with a 100% tint
```

As mentioned above due to how the two color systems work in real life setting all RGB values to a 100% will produce white and doing the same in CMYK will produce black.

```
var('blackRGB' = pdf_color('rgb', 0, 0, 0));
var('whiteRGB' = pdf_color('rgb',1,1,1));

var('whiteCMYK' = pdf_color('cmyk', 0, 0, 0, 0));
var('blackCMYK' = pdf_color('cmyk',1,1,1,1));
```

Tip! In real life a graphic artist would never set all four CMYK colors to a 100%. To get black it should be sufficient to only set the black color to full. Usually though the other colors are used too with values around 30% since that deepens the black effect.

Gray is handled as RGB. Setting it to a 100% will produce white.

```
var('gray10' = pdf_color('gray', 1)); // white
var('gray08' = pdf_color('gray', 0.8)); // 80% black (a light gray)
var('gray03' = pdf_color('gray', 0.3)); // 30% black (a dark gray)
var('gray0' = pdf_color('gray', 0)); // black
```

Remark: This is not so consistent. It would be better to treat gray as the black part of CMYK and thus having a 100% gray equal black.

More fun with fonts

Default font for text if no font settings is provided is Helvetica Plain 12 points using 16 points leading.
Leading, according to my Swedish sources, is pronounced like the metal lead. "Ledding". It defines the space between each row of text.

The default setting can of course be altered. You can use as many different fonts on a pdf as the readers can stand to look at. And then some. Not only can you set different fonts. You can add color to the text to spice it up even more. There's a number of standard fonts that's always at hand when creating pdf documents and you can add whatever other fonts you want by giving Lasso access to the font file on the server. But, in case you want to use non standard fonts, the file does require to be of specific formats that's not always so easy to arrange.

Setting a font variable

To use any other font than Helvetica 12 points you have to define and use a PDF_Font variable.

```
var('HelvBold14Black' = (pdf_font:
        -face = 'Helvetica-Bold',
        -size = 14,
        -color = '#000000'));

$pdf -> add(pdf_text('Text using Helvetica-Bold as font.',
             -font = $HelvBold14Black,
             -leading = 18));
```

Remember, if you set a PDF_Font, also set a leading that corresponds to the font size. The leading is set when creating the PDF_Text object.

Standard fonts included

Lasso's pdf tags comes with a number of pre installed fonts. As does the pdf standard itself. These fonts are:

- Courier
- Courier-Bold
- Courier-Oblique
- Courier-BoldOblique
- Helvetica
- Helvetica-Bold
- Helvetica-Oblique
- Helvetica-BoldOblique
- Symbol
- Times-Roman
- Times-Bold
- Times-Italic
- Times-BoldItalic
- ZapfDingbats

Spelling is important when you include a font. "Helvetica" is OK. But "Times" is not. It will produce a nonsense error message. Instead you have to write "Times-Roman".

I prefer to initiate all font variables that's to be used at the top of the page and give them names so that they convey what kind of font the harbour.

```
var('HelvBold' = (pdf_font:
        -face = 'Helvetica-Bold'));
var('TimesPlain10' = (pdf_font:
        -face = 'Times-Roman',
        -size = 10));
var('ZapfDingbats14' = (pdf_font:
        -face = 'ZapfDingbats',
        -size = 14));
```

Using you own fonts

To use your own non standard fonts in a pdf document is in theory quite simple. You just point Lasso to a font file of an approved format and that's it.

```
var('WarnockPro' = (pdf_font:
        -file = '/fonts/WarnockPro-Regular.otf',
        -embed));
```

The font file can reside in the usual place where your system keeps them. Just make sure Lasso have read access to that folder. Or you can put font files that you intend to use in a folder within the web servers realm. (My preferred choice.)
It's a good idea to always use the param -embed when adding your own fonts. If the font is part of a standard install then it's a good chance that it's also available on your clients computers but there's no guarantee. Using -embed when creating the font variable ensures that the needed font info is always embedded into the document and sent along with the other content.

But, there's a catch to the theory when it comes to using your own fonts. "Font file of an approved format". Most fonts installed on a server are not of a format that Lasso can use straight away.
(Disclaimer: I have done no font testing on any other systems than OS X. It's possible that it's much easier to get hold of correct font files on Windows or Linux.)

My testing has shown that Lasso can handle fonts that are of type TTF (True Type Fonts) or OTF (Open Type Fonts). And good news are that a lot of fonts available are of either type. Even the fonts that's part of the standard install of OS X are generally of a usable format.
Bad news are that they are often enclosed in different types of packages. Like Dfont files or suitcases. I have found no way for Lasso to get inside a font package in order to get to the actual font file. So you have all these nice looking fonts on your server but no way of using them. And it's not so easy to unpack and extract font files using standard tools available. In order to get to the actual files you need special applications that can look inside font packages and extract font files. Personally I use TransType. Available for both Mac and Windows from FontLab. It's a commercial application with a standard edition for $97 and a Pro version for $179.

Once you have an extracted font file that's either True Type or Open Type all you need is to keep it in a place where Lasso can read it.

Adding color to fonts and texts

Color can be added to a font variable by setting a Hex Color String when creating the PDF_Font object or by pointing to a PDF_Color object.
Using a Hex Color String is the older, pre Lasso 8.5.5 way.

```
var('HelvBold10Green' = (pdf_font:
        -face = 'Helvetica-Bold',
        -size = 10,
        -color = '#00FF00'));
```

Hex Color Strings are the same as used by web browsers so if you're familiar with creating colors for the web you'll feel right at home.

But, there's some benefits in using the new PDF_Color type. First, you use the same color vars for text that you've created for other content in the pdf. Second, you have access to all the types of color that PDF_Color offers. Gray, RGB, CMYK and Spot. And third, you don't have to specify colors in the PDF_Font variable. Instead you can add the color when adding the text.

```
var('HelvBold' = (pdf_font: -face = 'Helvetica-Bold'));

var('gray03' = pdf_color('gray', 0.3));
```

```
var('E64_1CMYK' = pdf_color('cmyk', 0.3, 0.9, 1, 0));

$pdf -> add(pdf_text( 'Lorem ipsum dolor sit amet.',
              -font = $HelvBold,
              -color = $gray03));

$pdf -> add(pdf_text( 'Lorem ipsum dolor sit amet.',
              -font = $HelvBold,
              -color = $E64_1CMYK));
```

If you do specify colors in the PDF_Font var it's easy to override the color value when creating a PDF_Text object.

```
var('HelvBoldGreen' = (pdf_font:
        -face = 'Helvetica-Bold',
        -color = $greenRGB));

$pdf -> add(pdf_text( 'Using the font objects color value',
              -font = $HelvBoldGreen));

$pdf -> add(pdf_text( 'Overriding the color',
              -font = $HelvBoldGreen,
              -color = $E64_1Spot_10CMYK));
```

A hidden feature!!

Adding text to a pdf with style and color is easy using the techniques described earlier. But you soon stumble into limitations. What if you want only one word in bold in the middle of a sentence and not the whole paragraph? Or if you want a smaller section colored but not all of it? Can it be done? Yes, but you won't find it in the Language Guide.

The trick is that you can add PDF_Text objects to another PDF_Text object!

Say you have a sentence like "Roll on, deep and dark blue ocean, roll. Ten thousand fleets sweep over thee in vain. Man marks the earth with ruin, but his control stops with the shore."

We want it in our pdf but also spice it up a little by emphasising parts of it.
"Roll on, **deep and dark blue ocean**, roll. Ten thousand fleets sweep over thee in vain. Man marks the earth with ruin, but his control stops with the shore."

To do this we first need some font variables:

```
var('TimesPlain18' = (pdf_font:
        -face = 'Times-Roman',
        -size = 18));
var('TimesBold18' = (pdf_font:
        -face = 'Times-Bold',
        -size = 18));
```

Then we need a color object:

```
var('blueCMYK' = pdf_color('cmyk', 1, 0.5, 0, 0));
```

We create an empty PDF_Text object that eventually will hold the text. It's important that the type is paragraph or it won't take the leading param:

```
var('pdftext' = pdf_text( '',
        -type = 'paragraph',
        -leading = 20));
```

This far into the process we can add the first part of the content:

```
$pdftext -> add(pdf_text( 'Roll on, ',
        -type = 'chunk',
        -font = $TimesPlain18));
```

Why type "chunk"? Well using chunk will enable us to add content without it starting on a new row. The next part is added with different font and color values:

```
$pdftext -> add(pdf_text( 'deep and dark blue ocean',
        -type = 'chunk',
        -font = $TimesBold18,
        -color = $blueCMYK));
```

And the last part of the quote is added with the plain font settings again:

```
$pdftext -> add(pdf_text( ', roll. Ten thousand fleets sweep over
thee in vain. Man marks the earth with ruin, but his control stops
with the shore.',
        -type = 'chunk',
        -font = $TimesPlain18));
```

Now all we have to do is add the PDF_Text object to the PDF_Doc, close it and serve it.

```
$pdf -> add($pdftext);

$pdf -> close;
pdf_serve( -content = $pdf, -file = 'Famous_quote.pdf');
```

Read more

The examples provided in this paper is kind of a potpourri. The presentation I had on the Lasso Summit 2007 was based on some more ready made solutions. They showed step by step how to construct Labels, Lists, Invoices and Leaflets.

The paper presented and the examples provided at that time are still available on Lassotech.
Visit www.lassotech.com/TotW_20080229 and you'll find a download link to an archive with my stuff and material from all the others that spoke on the Summit. My material together with the tips presented here should give you a good start on getting to terms with Lasso's excellent PDF tag suite.

Lund July 2008

Jolle Carlestam