Lasso Summit 07

LassoSoft

Point in Space

FLAGSHIP HOSTING.com

HostedStore™

Presented by:
Heunox Corporation

Fort Lauderdale, Florida
Riverside Hotel
March 2-4, 2007
http://www.lassosummit.com

FLAGSHIP HOSTING.COM

800.592.6781

offering:

- Enterprise Email
- Spam/Virus Protection on all Email Accounts
- Webmail
- Custom Configurations
- Professional Stats
- Lasso/PHP
- Server Colocation
- Dedicated Intel Xserve Leasing
- Dedicated G5 Xserve Leasing

Do you have ideas about creating a custom on-line presence? Contact Alex Pilson, alex@flagshiphosting.com for information and solutions for e-commerce, content management and custom web applications.

www.flagshiphosting.com

Contents

WWW.POINTINSPACE.COM

Lasso MySQL FileMaker Solutions Alliance

(800) 664-8610 * (603) 352-3701 * INFO@POINTINSPACE.COM

Lasso Summit 2007 Schedule

Thursday, March 1, 2007

7:00 PM - ? ..... Informal gathering starting at the Cheesecake Factory

Friday, March 2, 2007

Pre-conference Sessions

8:30 AM - 12:00 PM ..... Pre-conference: Lasso First Steps Workshop
12:00PM - 1:00 PM ..... Working Lunch
1:00 AM - 4:30 PM ..... Pre-conference: Lasso Fundamentals Workshop
4:00 PM - 6:00 PM ..... LPA Members Only Intracoastal Cruise of Fort Lauderdale
6:00 PM - 7:00 PM ..... Conference Registration

Lasso Summit Begins In the Champ Carr Room

7:00 PM - 8:00 PM ..... Keynote Address by LassoSoft
8:00 PM - 10:00 PM ..... Cocktail Reception just outside of the Champ Carr Room

Saturday, March 3, 2007

Lasso Summit Continues In the Champ Carr Room

8:00 AM - 8:30 AM ..... Registration & Working Continental Breakfast
8:30 AM - 9:30 AM ..... Time to Stop Hating and Start Loving the Lasso PDF Tag Suite
by Jolle Carlestam
9:30 AM - 10:00 AM ..... Morning Break
10:00 AM - 11:00 AM ..... Lasso: The Future of the Language
by Kyle Jessup
11:00 AM - 11:15 AM ..... Short Break
11:15 AM - 12:15 PM ..... Cutting Through The Hype: Simple Web 2.0 with Lasso and jQuery
by Jason Huck
12:15 PM - 1:15 PM ..... Working Lunch
1:15 PM - 2:15 PM ..... Implementing a Changes Tracking System
by Eric Landmann
2:15 PM - 2:45 PM ..... Afternoon Break
2:45 PM - 3:45 PM ..... ChartFX Designer
by Fletcher Sandbeck
3:45 PM - 4:00 PM ..... Short Break
4:00 PM - 5:00 PM ..... Managing Common Code
by Jonathan Guthrie
5:30 PM - 7:30 PM ..... LassoSoft Reception and Cocktails

Sunday, March 4, 2007

Lasso Summit Continues In the Champ Carr Room

Roundtables

All Roundtables are being held concurrently and each covers a general topic to last approximately 2.5-3.0 hours. Several Hosts will share their expertise providing an excellent chance for one-on-one discussion. Feel free to travel between tables and meet with the people that best fit your needs. Even the roundtable hosts are encouraged to move between tables. We encourage everyone to share their own knowledge at these tables.

8:00 AM - 9:00 AM ..... Working Continental Breakfast
9:00 AM - 11:00 AM ..... Refreshment Table
12:00 PM - 1:00 PM ..... Closing Lunch
1:00 PM ..... Lasso Summit Ends

[i_need]

to parse a complex XML document without learning XPath | strip all the HTML out of a bunch of text | display events in a calendar grid | create a podcast | create an rss feed and manage it at Feedburner | process transactions through Authorize.Net, Monetra,VeriSign, or Google Checkout | get the Word of the Day from Yahoo! | authenticate against an LDAP server | generate interactive charts in Flash | run an Applescript | retrieve stock quotes | translate a mailing address into coordinates | create a Google sitemap | get current weatherconditions and forecast data | find the number of words in a string | detect bots and spiders | filter comment spamthrough Akismet | retrieve album covers from Amazon | use Google to power search results on my site | control my version control system | generate a "next closest tour" to solve TSPproblems | generate a log file in Apache CLF format | zap gremlins | validate a social security number | check to see if an image is an animated GIF | generate an m3u playlist | convert RGB values to hexadecimal and back | convert a string to Soundex code | convert degrees to radians and back | convert Fahrenheit to Celsius and back...

[help]

tagswap.net

// a public exchange for Lasso custom tags

Welcome Letter

Summit Attendees,

It is with great pleasure that I welcome you to Lasso Summit 2007. It is exciting to be able to come together in beautiful Fort Lauderdale, Florida and spend a few days learning about Lasso and sharing ideas with some of the brightest minds in the Lasso community.

I want to thank Jim Van Heule and Tom Wiebe for organizing this event. Their dedication to the Lasso platform is unsurpassed. The Lasso Summits are among some of the best trade shows which I have attended.

Lasso Summit 2007 features a veritable who's who of Lasso developers as speakers. We look forward to hearing from Jonathan Guthrie, Jason Huck, Eric Landmann, Jolle Carlestam, Nikolaj de Fine Licht, Miles, and the other developers who will host round tables. Don't miss out on the opportunity to talk with the speakers before or after their presentations.

Many of the people who work on Lasso itself will be available throughout the Summit. We welcome the chance to hear about what you have accomplished with Lasso and to hear your ideas about how we can make Lasso even better. Please don't be shy about approaching us. We always look forward to meeting our users face to face.

Lasso Summit gives us an opportunity to reflect on the history of Lasso and to look forward to the future of the platform. We hope that the information we can share with you about where Lasso is headed will make you as enthusiastic about the future of Lasso as we are.

Fletcher Sandbeck

on behalf of the entire Lasso team

Time to Stop Hating and Start Loving the Lasso PDF Tag Suite

Jolle Carlestam

The paperless society was just a dream. We produce our texts on computers but depend on paper output for reading, sharing, archiving and for legal reasons.

Web applications based on HTML give very little control over paper output. Different browsers on different operative systems printing on different printers produce - very - different results.

How nice then that Lasso can help us transform screen information into paper with control over layout and content via the Lasso PDF tag suite.

But the PDF tags can look very intimidating, they're far from intuitive to work with. The documentation available is almost as frightening as the actual pdf tags and some of the tags behave in surprising ways. (Some call those behaviors bugs...)

I've struggled with pdf demands since Lasso was just an akward teenager. My first successful solution used Filemaker for the actual pdf creation and profited from Filemaker being single-threaded. Once Filemaker started to create the pdf no other action could take place. Therefore the user could be sure his pdf document was delivered back and not someone else's, who maybe started printing at the same time. My first client still runs the original solution and refuses to upgrade. It's been running for some ten years. He is terrified that it will brake down and hangs on to a couple of old Macs as spare if the original server fails.

This paper introduces four different solutions using the Lasso 8.5 PDF suite. It's a walk through on:

- Mailing labels, and building on the same code; nametags
- Contact list
- Invoice
- Info leaflet

Preparations

All examples described here are built and tested on Lasso 8.5.X. They might work on older versions of Lasso but that is not likely.

On the CD distributed to Summit attendants is a folder containing each demo example. You will need access to that folder in order to follow the explanations. I recommend that you open each of the demo files and read the code parallel to working through this paper. The demos depend on some external files. Some files act as substitutes for DB calls and contain demo data. You could replace those parts of the code with your own DB calls and data preparations. Some are necessary images and pdf templates. These files are located in the folder ex_cont. They need to be in the

same folder as the demo files. Lasso must be configured to handle files with the suffix ".incl" and be able to include pdf files.

Each example described builds on the previous. So even if your interest is only in the Invoice example don't skip the earlier explanations.

Common ground

All the examples build a pdf using the same basic technique. A pdf_doc object is created with this code.

```
var:'pageSize' = 'A4';

var:'pdfTemplate' = (pdf_doc:
    -nocompress,
    -size = $pageSize,
    -title = 'Example Title',
    -author = 'jinaOnline',
    -margins = (array: 0,0,0,0)
    );
```

The variable pdfTemplate is then populated with different pdf objects and finally served to the user.

```
$pdfTemplate->close;
pdf_serve: -content = $pdfTemplate,
-file = 'Jina Demo Example.pdf';
```

The name of the document sent to the user will, with the code above, be "Jina Demo Example.pdf".

If you want a different page size you just set the variable pageSize to your preferred value. Could for example be set to LETTER.

Mailing labels

Demo file:
label.lasso

Included files:
jina_prepPDF_demo.incl
Contacts.incl

The goal with this exercise is to produce a pdf holding as many printable mailing labels as is requested at each time. That calls for a solution that can handle multiple pages. Another feature request is that they should look nice. A detail that Filemaker has been handling with grace for a long time is the ability to have each label content vertically aligned within the label. Let's mimic that.

Some might assume that producing labels, with it's natural disposition in columns and rows, calls for the use of the PDF table tags. But no, I've found no use for them, not in this solution or any other. My personal advice is to leave them in the box. Unpacking them will only aggregate your frustration and perhaps make you leave Lasso all together.

It's easier to build our own grid. And make use of the pdf tag Add.

But first we need some initial settings. Since I'm from Europe (Sweden) I set the paper size to A4, the standard size of office papers all over the civilized world. You who read this are probably from the US and are used to other paper sizes. Well, change the settings or move to Europe.

The example settings fit Avery 7160 that holds 3 columns and 7 rows. If you prefer other labels then you can easily accommodate that by some experimenting with these settings.

```
// The number of rows and columns of labels.
var:'labelRows' = 7;
var:'labelCols' = 3;

// These variables set the font face and leading.
var:'textFont' = (pdf_font: -face = 'Helvetica', -size = 10);
var:'textHeight' = 13;

var:'pageSize' = 'A4';
var:'pageHeight' = 842;

// The top and left margin for the page
// (where the upper left label is located), in points.
var:'marginTop' = 45;
var:'marginLeft' = 25;

// The width and height of each label
var:'labelWidth' = 174;
var:'labelHeight' = 107;

// The width between label columns and height between label rows (if any).
var:'marginWidth' = 14;
var:'marginHeight' = 1;
```

These are all settings needed to adjust for different paper sizes and label layouts. With them we can build a grid that starts in the upper left corner using variables marginTop and marginLeft. The space for each cell in the grid (the label) is defined by labelWidth and labelHeight together with marginWidth and marginHeight.

Here's a little secret: It says in the manual that using Add, -width and -height only applies to the image and barcode tags. Well that's not true. It works with text too by setting the boundaries for the box.

Let's say we want to place one label. First we would need some content, a name and an address.

```
var:'prepLabel' = 'CEO';
$prepLabel += '\rDonald Anka';
$prepLabel += '\rWorld Travels';
$prepLabel += '\r1253 Highland Ave';
$prepLabel += '\rNeedham, MA 02492';
```

Then we place the content in the pdf object using Add.

```
$pdfTemplate->(Add:
  (PDF_Text: $prepLabel,
    -type = 'paragraph', -font = $textFont),
  -left = $marginLeft + 6,
  -width = $labelWidth - 12,
  -top = $marginTop + 6,
  -height = $labelHeight - 12,
  -align = 'left',
  -leading = $textHeight);
```

This places the label content 51 points (points are almost like pixels) from the left edge of the paper and 31 points from the top edge. (Since marginLeft = 45 and we add 6 to that, and marginTop = 25 and we add 6 once more.) It constrains the content to stay within the boundaries set by the -width and -height params creating a box that's 162 points wide (174 - 12) and 95 points high (107 - 12).

We could rest contentedly with this if all we wanted was one label. We don't. Time for some more vars. Let's not change the content of marginLeft and MarginTop. It's best that they keep their original content when it's time for the next page. Instead we create some other variables to hold the temporary position info.

```
var:'colPos' = integer;
var:'rowCount' = integer;
var:'labelLeft' = integer;
var:'labelTop' = integer;
```

ColPos will be used to keep track of which column we're placing labels in and rowCount keep track of the vertical position. They start empty. LabelLeft and labelTop will hold the position in points for each label.

Since we need to change label content and position for each label we need some kind of loop. In the example I use an Iteration. In the real world it could also be a Records container. The example uses dummy data prepared by an include. It changes the content of the variable prepLabel for each iteration so that it contains a new address formatted like the example above. A possible title, followed by a name, followed by a possible organization and ending with an address. Each part is separated by a line feed.

The placement of the label is tracked by altering colPos and rowCount.

```
  if:($rowCount) == $labelRows;
    $colPos += 1;
    $rowCount = 0;
  /if;

  $rowCount += 1;
```

Since labelRows in the example is set to 7 this would increase the column position with one when rowCount = 7. And also reset rowCount. Notice that the first column will be 0, the second will be 1 etc.. The same applies for rowCount.

To set the actual position for the label we do some math using the variables we prepared.

```
$labelLeft = $marginLeft + (($labelWidth + $marginWidth) * $colPos);
$labelTop = $marginTop + (($labelHeight + $marginHeight) * $rowCount);
```

Let's translate that into real numbers. MarginLeft = 45, labelWidth = 174 and marginWidth = 14. If we wanted the left position for the sixth label in the second column then colPos would be 1. The formula would look like this.

```
$labelLeft = 45 + ((174 + 14) * 1)
-> 233
```

If we wanted the position for the first column the formula would look like this.

```
$labelLeft = 45 + ((174 + 14) * 0)
-> 45
```

That is, for the first column labelLeft would equal marginLeft since any value multiplied with zero is still zero.

The same math is applied to the horizontal position setting in labelTop.

Now we could place the content of the label using Add and the values in labelLeft and labelTop as -left and -top params. But we still have some preparations to be made. We want the label content to be vertically aligned within the label. So that no matter if the content spans three rows or seven there's still equal space above and below the content.

To do that we need to know how many rows the content will span. A simple way to do that would be to count the line feeds. But what if some of the content, like a name or an organization is so long that it would span several rows? We need to track that too.

Let's start with the easy part, counting actual line feeds. Split the label content on line feeds and count them.

```
$labelArray = $prepLabel->(split:'\r');
$labelVerSize = $labelArray->Size;
```

Then calculate if each row content will span more than one row.

```
iterate: $labelArray, $labelTemp;
  $labelVerSize += math_floor(($textFont->(textwidth:
    $labelTemp)) / ($labelWidth - 12));
/iterate;
```

Textwidth is a tag that uses a pdf_font variable, like the one we prepared in textFont, to calculate how many points the text in labelTemp will occupy, should it all be written to one row. This is a feature that's used extensively trough-out all examples in this paper. Since we know the width each row is allowed to use, (labelWidth - 12) we can calculate if it will span more than one row by dividing the textwidth result with the row width. If the result is less than one the content will

fit into one row. If not we now know how many rows it will span. Math_floor will give us the integer part of the result. If there's no row span the result will be a decimal value 0.XXX. Using math_floor on 0.XXX adds zero to the variable labelVerSize that keep track of rows. If the text flows over into the second row we would have a value of 1.XXX and math_floor would add the integer part, 1, to labelVerSize.

Now we know how many rows the label content will need. And since we also know how many point each row uses, the value stored in the variable rowHeight, we can easily find out how many points the content uses.

```
$labelVerSize *= $textHeight;
```

Time to add the label to the pdf.

```
$pdfTemplate->(Add:
  (PDF_Text: $prepLabel,
    -type = 'paragraph', -font = $textFont),
  -width = $labelWidth - 12,
  -left = $labelLeft + 6,
  -top = $labelTop + (($labelHeight -
  $labelVerSize) / 2) - ($textHeight / 2),
  -height = Math_Min(($labelVerSize),$labelHeight),
  -align = 'left',
  -leading = $textHeight);
```

The labelTop is the starting point for the label content had we not been concerned with vertical alignment. By adding (($labelHeight - $labelVerSize) / 2) we move the starting point down by half of the free vertical space. And to make it look even nicer we subtract a half row - ($textHeight / 2).

That's about it. The final, vital, part is to keep track of pages. To do this we initially set a variable to the number of labels each page can hold.

```
var:'newPageCheck' = $labelRows * $labelCols;
```

In the iteration we use this variable to tell us when it's time for a new page.

```
if:loop_count % $newPageCheck == 0;
  $colPos = integer;
  $rowCount = integer;
  $pdfTemplate->AddPage;
/if;
```

And of course we reset position variables to their initial values.

That's it! Please try it using the example page label.lasso.

Nametags

Demo file:

nametag.lasso

Included files:

jina_prepPDF_demo.incl
Contacts.incl
Bkgr1_name_tags_Summit.jpg or Bkgr2_name_tags_Summit.jpg

This is an expansion on the label code where we instead of mailing labels produce nice looking nametags. It's more or less a question of adding a picture to each "label". We also want to enhance the look by adding some different font settings.

Let's start with the picture.

```
var:'bkgrImage' =
(PDF_Image:-file = 'ex_cont/Bkgr_name_tags_Summit.jpg');
```

I've found that in order for a picture to have a sufficient quality when printed I need to reduce the size by factor 0.5 when included into the pdf. As far as I know Lassos pdf tags won't honor any document size settings from the image. A 200 pixel wide image will occupy 200 pixels (or actually points) in the pdf if left by itself even if the image file originally have a 300 px per inch setting. But I can force a size when adding the image to the pdf object by setting -width and -height. A suitable size of the nametags is 273 points wide and 173 points high. So to have sufficient quality on the background image the image should be 546 * 346 pixels.

As with the labels we set some initial variables.

```
// Heading displayed on every nametag.
var:'headingText' = 'Lasso Summit 2007';

// The number of rows and columns of nametags
var:'nameTagRows' = 4;
var:'nameTagCols' = 2;

// These variables set the font face and leading
var:'headingFont' = (pdf_font:
  -face = 'Helvetica-Bold', -size = 14, -color = '#FFFFFF');
var:'titleFont' = (pdf_font:
  -face = 'Helvetica-Bold', -size = 14, -color = '#e9a45f');
var:'nameFont' = (pdf_font:
  -face = 'Helvetica-Bold', -size = 20, -color = '#e9a45f');
var:'orgFont' = (pdf_font:
  -face = 'Helvetica-Bold', -size = 18, -color = '#e9a45f');
var:'cityFont' = (pdf_font:
  -face = 'Helvetica-Bold', -size = 14, -color = '#FFFFFF');var:'titleLeading' = 16;
var:'nameLeading' = 22;
var:'orgLeading' = 20
var:'cityLeading' = 20;

var:'pageSize' = 'A4';
var:'pageHeight' = 842;


// The top and left margin for the page
// (where the upper left nametag is located), in points
var:'marginTop' = 45;
var:'marginLeft' = 25;

// The width and height of each nametag
```

```
var:'nameTagWidth' = 273;
var:'nameTagHeight' = 173;

// The width between nametag columns and
// height between nametag rows (if any).
var:'marginWidth' = 0;
var:'marginHeight' = 0;

// If text needs vertical adjustment to fit with background
var:'verticalAdjust' = -15;
```

Some explanations. Since we want different appearance on the different parts of the nametag texts, we need several font variables. And since each pdf_font variable has different sizes we need variables holding the text height (leading) for each font. The variable verticalAdjust allows us to adjust vertical alignment to avoid conflict with the background. If there's for example a logo we don't want text to interfere with.

We iterate all contacts that we want a nametag for. Like with labels this should probably be handled by a Records container in the real world. The example code produces variables for each of the types of info we want on the nametag; name, title, organization etc. But before we handle the dynamic content we add some static stuff, the background image and the heading.

```
$pdfTemplate->(Add:$bkgrImage,
  -left = $nameTagLeft,
  -top = $nameTagTop + $nameTagHeight,
  -height = $nameTagHeight,
  -width = $nameTagWidth);$pdfTemplate->(Add: (PDF_Text: $headingText,
    -type = 'paragraph', -font = $headingFont),
  -width = $nameTagWidth - 12,
  -left = $nameTagLeft + 16,
  -top = $nameTagTop + 3,
  -align = 'left');
```

An odd thing with the coordinate system that Lasso uses in the PDf tag suite is that some objects keep track of the lower left corner instead of the upper left. That's why the image is placed with -top = $nameTagTop + $nameTagHeight and not just $nameTagTop.

Next it's time to find out how much vertical space the content needs. And since we have multiple contents each with their own demands on space we need to repeat the calculation for every content type. Let's start with the title.

```
if:$titleTag->Size > 0;
  $nameTagArray = $titleTag->(split:'\r');
  $titleVerSize = $nameTagArray->Size;
  iterate: $nameTagArray, $nameTagTemp;
    $titleVerSize +=
    math_floor(($titleFont->(textwidth:
    $nameTagTemp)) / ($nameTagWidth - 12));
  /iterate;
  $titleVerSize *= $titleLeading;
else;
  $titleVerSize = integer;
/if;
```

It could be that the variable titleTag is empty so we better check for content before we do anything with it. If it is empty the the vertical size variable titleVerSize will also be empty. If it's not empty then titleVerSize will now hold the amount of vertical points that the title uses.

The same procedure is then repeated for each of the variables nameTag, orgTag and cityTag. After that the variables titleVerSize, nameVerSize, orgVerSize and cityVerSize should each hold a value for how much vertical space each part of the nametag uses. Time to add that up and from that calculate a suitable vertical position for the content. As with the mailing labels we want the content vertically adjusted within the nametag.

```
$nameTagVerSize = $titleVerSize + $nameVerSize +
$orgVerSize + $cityVerSize;
$tempNameTagTop = $nameTagTop +
(($nameTagHeight - $nameTagVerSize) / 2) + $verticalAdjust;
```

Remember, verticalAdjust will move content up or down to allow room for a logo in the background image. With all positioning info in place we can safely add the actual content for this nametag into the pdf object. Since we have different font styles and row height demands (leading) for each part of the content we need to do several Adds.

```
$pdfTemplate->(Add: (PDF_Text: $titleTag,
    -type = 'paragraph', -font = $titleFont),
  -width = $nameTagWidth - 12,
  -left = $nameTagLeft + 6,
  -top = $tempNameTagTop,
  -align = 'center',
  -leading = $titleLeading);
$tempNameTagTop += $titleVerSize;
$pdfTemplate->(Add: (PDF_Text: $nameTag,
    -type = 'paragraph', -font = $nameFont),
  -width = $nameTagWidth - 12,
  -left = $nameTagLeft + 6,
  -top = $tempNameTagTop,
  -align = 'center',
  -leading = $nameLeading);
$tempNameTagTop += $nameVerSize;
$pdfTemplate->(Add: (PDF_Text: $orgTag,
    -type = 'paragraph', -font = $orgFont),
  -width = $nameTagWidth - 12,
  -left = $nameTagLeft + 6,
  -top = $tempNameTagTop,
  -align = 'center',
  -leading = $orgLeading);
$tempNameTagTop += $orgVerSize;
$pdfTemplate->(Add: (PDF_Text: $cityTag,
    -type = 'paragraph', -font = $cityFont),
  -width = $nameTagWidth - 12,
  -left = $nameTagLeft + 6,
  -top = $tempNameTagTop,
  -align = 'center',
  -leading = $cityLeading);
```

The final part of the iteration is to check if it's time to add a new page.

```
if:loop_count % $newPageCheck == 0;
  $colPos = integer;
  $rowCount = integer;
  $pdfTemplate->AddPage;
/if;
```

Unfortunately Lasso will not help us beyond the point of creating the pdf. We will have to print it ourselves and, worst of all, cut out each nametag manually. A job that I prefer to delegate to some teenager close by.

Contact Lists

Demo file:

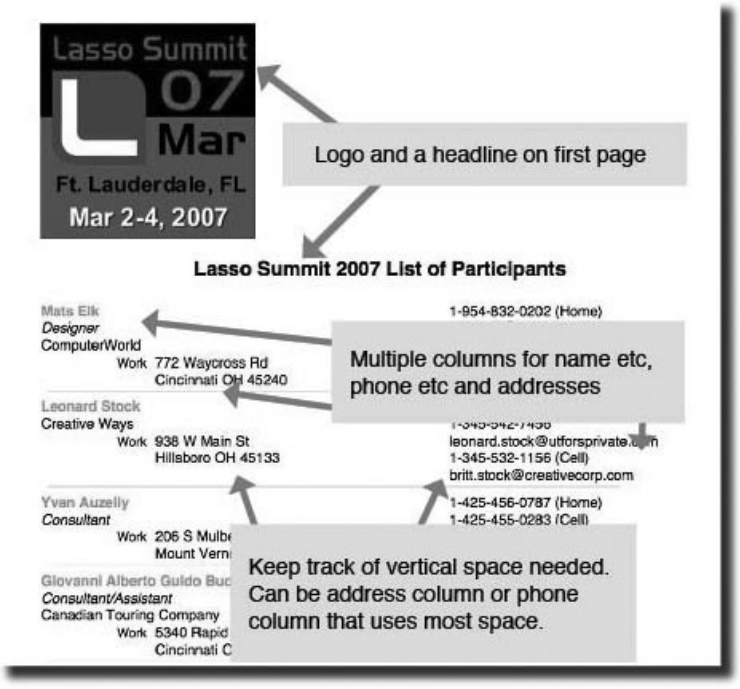
list.lasso

Included files:

jina_prepPDF_demo.incl
Contacts.incl
summit07banner.jpg
line.gif

The previous examples are material that really have to be on paper and formatted according to specific criteria. There needed even if we truly move into the paperless era. Lists like the contact list demonstrated here will maybe disappear. But I doubt it. And at numerous occasions I've seen lists printed directly from a web page. Ugly, poorly formatted, and at times, missing info since it didn't fit into the paper. Better to provide the user with the option to create a pdf. Formed to suit the needs of a printed output.

This is what we want.



The first page will have content that is not repeated on following pages. A logo and a headline. For each contact we need to handle multiple addresses and multiple other contact info like several phone numbers, e-mail addresses, URLs etc. It's also nice to separate each contact with a horizontal ruler.

Each type of content can have it's own font style setting. They don't have to differ but incase we do want to use different styles we might as well prepare different pdf_font variables to be prepared. Even if we set them to different font styles and perhaps color it's easier for future handling if they're all set to the same size. In the example thats -size = 10.

As before we need some variables to hold positioning data. And since we will make use of three different columns we need to set coordinates separately for all three.

```
var:'marginTop' = 45;
var:'marginLeft' = 60;
var:'marginBottom' = 25;
var:'col1Left' = 0;
var:'col1Width' = 260;
var:'col2Left' = 80;
var:'col2Width' = 160;
var:'col3Left' = 286;
var:'col3Width' = 220;
```

The horizontal column setting, col1Left, col2Left and col3Left are all set relative to marginLeft. So should we later decide on a wider margin then the columns will move to the right accordingly, It's also time to set a variable to hold the vertical starting point of the first item.

```
var:'itemTop' = $marginTop + $logoHeight + 10;
```

On all following pages the first items starting point will equal marginTop but on the first page we have to make room for the logo and the headline.

In this example we will be using two images. The first is a logo that will only be used on the first page. As with all other images used we create the original to be at least double the size of the actual space it will use in the pdf. This will ensure sufficient print quality of the image. The image used as logo in the examples is actually a lot larger than needed. It's 1042 * 1042 pixels and should be reduced to 150 * 150 px. This will not have any impact other than that the file size of the finished pdf document will be larger than needed.

Note on images

I've found that jpeg or gif files usually work as inserted images. Other image formats are supposed to work but can give trouble. And it's also wise to make sure that images are saved as RGB. CMYK doesn't always work.

The other image is a straight line used as divider between each contact item.

Let's add the logo.

```
$pdfTemplate->(Add:$logoImage,
  -left = $marginLeft,
  -top = $marginTop + $logoHeight,
  -height = $logoHeight,
  -width = $logoWidth);
```

Image coordinates start from the lower left corner, that's why we have to add logoHeight to the -top param. If we didn't we would only see the lowest 45 pixels of the image. The rest would be above the edge of the paper.

After we place the logo it's a good idea to add the Headline.

```
$pdfTemplate->(Add:
  (PDF_Text: $headingText,
    -type = 'paragraph',
    -font = $headingFont),
  -width = $pageWidth - ($marginLeft * 2),
  -left = $marginLeft,
  -top = $itemTop,
  -align = 'center');
```

The -width param is set so that the text will have an equal margin on both sides. Once the headline is in place we can start focus on the contact items. But first we have to find out how much vertical space the headline occupies and set the variable itemTop to a new position.

```
$listItemArray = $headingText->(split:'\r');
$leftVerSize = $listItemArray->Size;
iterate: $listItemArray, $listItemTemp;
  $leftVerSize += math_floor(($titleFont->(textwidth:
  $listItemTemp)) / ($pageWidth - ($marginLeft * 2)));
/iterate;
$leftVerSize *= $titleLeading;
$itemTop += $leftVerSize + $titleLeading;
```

Now we can start iterating our list of contacts. As before, the example uses a dummy setup of contacts. In the real world the content would be called by an inline and we would loop trough a Records container instead of an iterate. In the example we use these variables to hold data for each contact.

```
var:'titleContact' = string;
var:'nameContact' = string;
var:'orgContact' = string;
var:'addressContact' = array;
var:'infoContact' = array;
```

The variables addressContact and infoContact are arrays since they can hold several items each. A contact can have multiple addresses, and more likely, multiple phone numbers, e-mails, URLs etc. We will start by adding the name in the first column, calculate how much vertical space it occupies. Add a possible title, do the vertical calculation on that and follow up with a possible organization. The vertical math is performed just as in prior examples. But with the creation and handling of the pdf_text objects we introduce a novelty. Let's start by creating the first part of the object.

```
$leftColText = (PDF_Text: ($nameContact + '\r'),
  -Type = 'paragraph',
  -Font = $nameFont);
```

Notice that we don't add the pdf_text object to the pdf_doc yet. Instead we continue with the contact title.

```
$leftColText->(Add:
  (PDF_Text: $titleContact + '\r',
    -Font = $titleFont));
```

Here's the novelty. We add the pdf_text object to the same variable that holds the contact name, leftColText. And we do it using a different font variable! This way we can build a pdf_text object that's hold different fonts and styles within one object. And this is also one reason to not tamper with different sizes in the fonts used. When added to the pdf_doc object the entire content of variable leftColText will be inserted using the same row height (leading). Finally the organization name is inserted the same way.

```
$leftColText->(Add:
  (PDF_Text: $orgContact + '\r',
    -Font = $orgFont));
```

We now have a pdf_text variable leftColText that holds the contacts name, title and organization. And we have another variable leftVerSize that holds the vertical space leftColText will occupy. But we don't add it to the pdf_doc object for a while. We first want to prepare addresses and contact infos.

Let's start with addresses. Beginning with a conditional check. No point of running trough a lot of code if there's no addresses to be dealt with.

```
  if:$addressContact->Size > 0;
    $addressText = (PDF_Text: '',
      -Type = 'paragraph',
      -Font = $addressFont);
    $typeText = (PDF_Text: '',
      -Type = 'paragraph',
      -Font = $typeFont);
    $addressItemTop = $leftVerSize * $textLeading;
```

Notice that we create two empty pdf_text variables. One that will hold the actual addresses and one that will hold the type of the address. The last row gives us a vertical starting point for the addresses.

Time to iterate the address array. As usual we start by calculating the vertical space needed. This value is stored in the variable col2RowCount. It's reset for each iteration.

Addresses in the example can be of different types. Work, Home, Delivery etc. We want to display the type next to the first row of each address and therefore need to store the type in a separate pdf_text object.

```
$typeText->(Add:
  (PDF_Text: $itemTemp->Name,
    -Font = $typeFont));
  loop: $col2RowCount;
    $typeText->(Add:
      (PDF_Text: '\r'));
  /loop;
$addressText->(Add:
  (PDF_Text: $itemTemp->Value + '\r',
    -Font = $addressFont));
```

The loop adds empty rows to ensure that the next type will be horizontally aligned with next address.

More or less the same procedure with phone numbers, e-mails etc. With some small variations. There's only some of the items that need explanations. Anyone recognizes an e-mail address or a URL so no need to point that out. But there's no way to know if a phone number is to the office or to the contacts home. Here's code to deal with that.

```
$rightColText->(Add:
  (PDF_Text: $itemTemp->Value +
  (($itemTemp->Name != 'E-mail' &&
  $itemTemp->Name != 'Web' &&
  $itemTemp->Name->Size > 0) ?
  (' (' + $itemTemp->Name + ') ')) + '\r',
    -Font = $infoFont));
```

The conditional evaluates to true only if the type of info is not "Email" or "Web" and if there actually is a type registered. On true it adds the type between parenthesis. Like this.

1-345-542-0230 (Home)
1-345-542-7456 <- No Type registered
1-345-532-1156 (Cell)
leonard.stock@creativecorp.com
www.creativecorp.com

We now have all content for one contact in a couple of pdf_text variables. Time to do some vertical space checking. There's two variables holding vertical space needs. LeftVerSize holds info regarding, name, title, organization and all addresses. RightVerSize holds info for all phone numbers, e-mails etc. Either one could hold the largest number. It could be that there's several addresses and only one or two contactinfo items. That would make leftVerSize the largest. But it's also possible that there only one or even no address and a lot of phone numbers. Then rightVerSize is the largest. We're only interested in the largest one so we use a math_max formula to find out.

```
$leftVerSize = (Math_Max:$leftVerSize, $rightVerSize);
```

Math_max gives us the largest value of the params sent to it. With knowledge of how much vertical space the contact will use we can check to see if the info will fit into the remainder of the page. If it wont fit, time to add a new page.

```
if:$leftVerSize + $itemTop > $pageHeight - $marginBottom;
  // Add a footer to the present page first.
  $pdfTemplate->(Add:
  (PDF_Text: ($headingText + ' -- Created: ' + date +
  ' -- Page: ' + $pdfTemplate->GetPageNumber),
      -type = 'paragraph',
      -font = $footerFont),
    -width = $pageWidth - ($marginLeft * 2),
    -left = $marginLeft,
    -top = $pageHeight - ($marginBottom * 1.5),
    -align = 'right');
  $pdfTemplate->AddPage;
  $itemTop = $marginTop;
/if;
```

Well, before we add the new page it's a good idea to add a footer to the bottom of the page. And of course we reset itemTop so that items will start at the top of the new page.

Now, finally we can add the content of the contact to the pdf_doc object.

```
  $pdfTemplate->(Add: $leftColText,
    -width = $col1Width,
    -left = $marginLeft + $col1Left,
    -top = $itemTop,
    -align = 'left',
    -leading = $textLeading);
  $pdfTemplate->(Add: $addressText,
    -width = $col1Width,
    -left = $marginLeft + $col2Left,
    -top = $itemTop + $addressItemTop,
    -align = 'left',
    -leading = $textLeading);

  $pdfTemplate->(Add: $typeText,
    -width = $col2Left - 6,
    -left = $marginLeft,
    -top = $itemTop + $addressItemTop,
    -align = 'right',
    -leading = $textLeading);
  $pdfTemplate->(Add: $rightColText,
    -width = $col3Width,
    -left = $marginLeft + $col3Left,
    -top = $itemTop,
    -align = 'left',
    -leading = $textLeading);
```

And as a final touch before starting over on the iteration we add a line using our line image.

```
$itemTop += $leftVerSize + 7;
$pdfTemplate->(Add:$lineImage,
  -left = $marginLeft + 5,
  -top = $itemTop - 2 ,
  -height = 0.5,
  -width = 400);
```

And, yes, we can have a decimal value when setting the -height param. Here's the result from two iterations.

That's it, when all iterations is done we just serve the file and take a brake after a deed well done.

Invoices

Demo file:

invoice.lasso

Included files:

jina_prepPDF_demo.incl
Invoice1.incl (or Invoice2.incl)
invoice_base.pdf

This can be tricky to do right. There's legal aspects on how an invoice can behave. A lot of information need their exact placements. Items can vary dramatically in size. From very short "50 workhours á $120" on the average plumber invoice after changing a dripping packing in the bathroom to very elaborate descriptions spanning over several rows. Graphic departments of larger corporations have very specific rules on how any printed material should look. And you have to keep track if the items are so many that you need to add additional pages to the document.

A piece of cake.

First let's introduce a feature that Lassos PDF tag suite offers. The possibility to include readymade pdf documents. Using that wisely we can take care of a lot of the requirements that the Graphic department hold so dearly. We simply let them create most of the invoice in whatever layout application they use. Including all graphics, static texts, frames, background etc. And when they're satisfied they just export the result as a pdf and send it to you. The pdf should preferably be at least two pages. The first page of the pdf should have room for some itemrows, client info and all summaries. The second page of the pdf will be used as background on all following pages and will have room for more item rows.

Here's part of the first template page.

A pdf document is inserted into a pdf_doc object like this.

```
var:'BkgPDF' = 'ex_cont/invoice_base.pdf';
$pdfTemplate->(insertpage: (pdf_read:
```


```
-file = $BkgPDF), 1);
```

The number "1" at the end of the code tells Lasso to insert page one of the called pdf document. Later we will call the same pdf to insert page two from it.

In order to position content into the right place of the template there's a number of coordinates to be set. To make it more convenient we make an array for each position.

```
var:'datePos' = (array: 107,81,100,30);
var:'refPos' = (array: 236,81,200,30);
var:'invNoPos' = (array: 480,81,50,30);
var:'buyerPos' = (array: 60,110,240,120);
var:'supplierPos' = (array: 313,110,240,120);
var:'sumPos' = (array: 402,630,150,30);
var:'vatPos' = (array: 402,648,150,30);
var:'totalPos' = (array: 402,665,150,30);
var:'notePos' = (array: 60,630,290,120);
var:'pageNoPos' = (array: 361,81,50,30);
```

The order in the array is left, top, width and height. This gives us position and boundaries for a number of "boxes" that we can use later to put content at exact places in the pdf_doc object. Notice that some of the boxes, like datePos has a height set to 30 although it's only supposed to have one row of text. This allows us to give a visual signal to the user that there's to much data in that box. Had the box been set to a height of, for example, 14 then it would only fit one row of text and the surplus would simply be cut at the edge of the box. That would look better but would also be harder to spot for the user and could lead to undesirable results.

Moving on.

```
var:'verSpace' = 376;
var:'verSpaceCont' = 600;
var:'spaceBetween' = 8;
```

We are planning to insert a number of items into the invoice, iInformation about the actual goods or services we are charging for. These variables tell us the vertical space available for these items. VerSpace is used on the first page and verSpaceCont on all following pages. There's more room on the following since we don't need to put any other info on those. SpaceBetween sets how much room we want between each item.

We want a dividing line between each item. But this time we wont use an image. Instead we will let Lasso draw a line. But since coordinates for graphic actions start in the lower left corner instead of upper right we need a different starting point and handling of the line.

```
  // Starting point for horizontal divider line
  var:'lineStart' = 587;
  var:'lineStartCont' = 730;
This sets the color and thickness of the line.
  $pdfTemplate->(SetColor:'stroke','CMYK',0.93,0.59,0.33,0.12);
  $pdfTemplate->(SetLineWidth:0.25);
```

Odd detail

SetColor and SetLineWidth needs to be reset on every new page. They don't stick to the pdf_doc.

And we need some column positions for the items. There should be a Specification column, a Count column (#), a Cost and a Summary column. These are just called col1, col2 etc. All horizontal settings are relative to the page margin.

```
  // The left and width of each column.
  var:'col1Left' = 0;
  var:'col1Width' = 270;
  var:'col2Left' = 295;
  var:'col2Width' = 33;
  var:'col3Left' = 333;
  var:'col3Width' = 75;
  var:'col4Left' = 420;
  var:'col4Width' = 75;
```

Time to start inserting some info into the pdf_doc. Here's for example the date inserted.

```
  $pdfTemplate->(Add:(
    PDF_Text:$invoiceData->'invoiceDate', -Type='Paragraph',
    -Font = $plainFont),
    -left = $datePos->(Get:1),
    -top = $datePos->(Get:2),
    -width = $datePos->(Get:3),
    -height = $datePos->(Get:4),
    -align = 'left',
    -leading = $textLeading);
```

And here's the invoice number.

```
$pdfTemplate->(Add:(
  PDF_Text:$invoiceData->'invoiceNo', -Type='Paragraph',
  -Font = $plainFont),
  -left = $invNoPos->(Get:1),
  -top = $invNoPos->(Get:2),
  -width = $invNoPos->(Get:3),
  -height = $invNoPos->(Get:4),
  -align = 'left',
  -leading = $textLeading);
```

All data that have a fixed position on the first page are inserted in the same way, using the positioning variables we set earlier.

Now it's time for the actual invoice items. We iterate trough them. As usual you will probably want to replace that with looping trough a Records container.

First of all we need to calculate vertical space needs for the item. The columns count (#), cost and sum are safe. They can never span more than one row. But we want to be generous with the item specification and allow that to span for as many rows as needed. The specification comes in two parts. The first is the item name. We perform our usual vertical space calculation, splitting on line feeds and then applying our textWidth/math_floor trick. Since it looks good if the item name stand out somewhat we use the bold font for this part. Once done we have a value stored in variable itemVerSize. Now let's do the same with the second part of the specification, an item description. Calculating vertical space adds to the variable itemVerSize. We now know how much space the item needs. But will it fit? Let's compare it with space available.

```
$itemVerSize = $itemVerSize * $textLeading;
if: $verSpace < ($itemVerSize + 18);
```

The 18 points added are just to make sure that there will be room for a hint that there's more specifications following on the next page. If not we add the hint text, insert the background template (this time using the second page) and reset some positioning variables.

```
$pdfTemplate->(Add:(PDF_Text:
  'Additional item specifications on following page!',
  -Type='Paragraph',
  -Font = $boldFont),
  -left = $marginLeft + $col1Left,
  -top = $itemTop,
  -width = $col1Width,
  -height = 30,
  -align = 'center',
  -leading = $textLeading);
$verSpace = $verSpaceCont;
$itemTop = $itemTopcont;
$lineStart = $lineStartCont;
$pdfTemplate->(insertpage: (pdf_read:
  -file = $BkgPDF), 2, -NewPage='true');
```

And place some additional fixed position content like repeating the invoice number and placing a page number.

Now we can safely place the item content.

```
$itemText = (PDF_Text:
  $itemRaw->(Get:1) + '\r', -Type='Paragraph',
    -Font = $boldFont);
$itemText->(Add: (PDF_Text: $itemRaw->(Get:2),
    -Font=$plainFont));
$pdfTemplate->(Add:$itemText,
  -left = $marginLeft + $col1Left,
  -top = $itemTop,
  -width = $col1Width,
  -align = 'left',
  -leading = $textLeading);
$pdfTemplate->(Add:(PDF_Text:$itemRaw->(Get:3),
  -Type='Paragraph',
    -Font = $plainFont),
  -left = $marginLeft + $col2Left,
  -top = $itemTop,
  -width = $col2Width,
  -align = 'center',
  -leading = $textLeading);
```

After the content is in place we adjust the positioning variables and add a horizontal line.

```
  $itemTop += $itemVerSize + $spaceBetween;
  $verSpace -= $itemVerSize + $spaceBetween;
  $lineStart -= $itemVerSize + $spaceBetween;
  $pdfTemplate->(Line: $marginLeft + 5,
    $lineStart + 6,
    ($marginLeft + $col4Left + $col4Width - 5),
    $lineStart + 6);
```

The pdf_doc->Line params specify the starting point and ending point of the line (X1, Y1, X2, Y2).

Well, that's it. Now all you need to do is serve the pdf doc, relax and wait for all that money to flow into your bank account.

Info Leaflets

Demo file:

leaflet.lasso

Included files:

jina_prepPDF_demo.incl
Leaflet.incl
MultiColumnTemplate.pdf
summit07banner.jpg
leafletImg1.jpg (or leafletImg2.jpg)

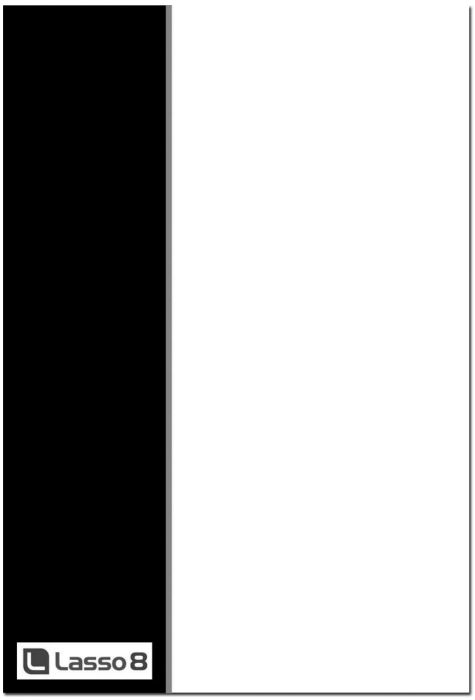
The purpose of this exercise is to produce an information page, for example a sales paper, that's as close as possible to look as if it's been manually and professionally layouted. We wont be able to go all the way. The fine details that a skilled layout person can add to the result is not available for us. We can't do good looking marginal justified columns. There is no way to perform kerning or spacing on individual sections. Since it will all be automated there is no eye that can see that

a headline would benefit by being tweaked in one way or another etc. But we can make a good effort and produce material that will be good enough under a lot of circumstances.

Disclaimer

I make no implied or express statement that the result produced in this example resembles something that's even remotely good looking! Remember, I'm not a designer or AD. I'm a code hacker without any proper education (except for theater history...). But if you collaborate with a skilled layout person you can hopefully use some of this code to produce good looking results.

Starting: as with the invoice example we will hand over some of the more complicated layout elements to a pdf template. I produce mine in Indesign but any layout application that can export pdfs will do just fine. Left for us to handle will be dynamic text and some pictures that will be user supplied for every page. We plan to do a three column layout. Left column is for technical information and the rest is for the article text and image. Here's how the template looks like.



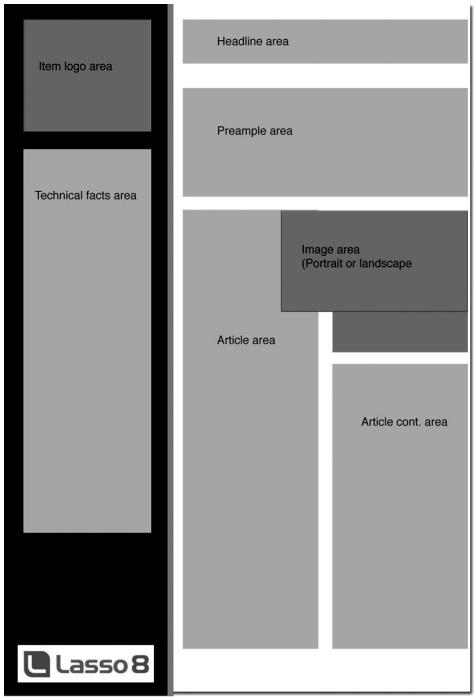
Fairly simple. The left column with technical info should fit in the black area. The white area is for article text and image.

Here's the regions we want to fill with dynamic content. Notice that the image area in the article space is actually two. We wont know if the user will provide us with a portrait oriented image or a landscape. If it's a landscape image we need to make room for it in the left article column.

Here's the finished result.

As usual we start by declaring a number of variables that we will use later. The different parts of the page require different pdf_font variables. For the technical info column we will use two, smallPlainFont and smallBoldFont. The headline could perhaps use the headingFont, the

preamble will look nice in the preambleFont and the article text will just use plainFont. Oh and we will have some text accompanying the image. Good idea to have a special font variable for that too, imgTextFont. Since each font variable sets a different size of the font we need different



leading variables. Leading sets the height of each row in the text.

Then proceed with setting width for all columns.

```
var:'col1Left' = 0;
var:'col1Width' = 170;
var:'col2Left' = 200;
var:'col3Left' = 392;
var:'artColWidth' = 163;
var:'imgColExtra' = 54;
var:'col2Col3Combined' = 345;
```

All left settings are relative to the page margin set in marginLeft. Since column 2 and 3 will have the same width it's enough to set one variable, artColWidth. The variable imgColExtra sets a value that will be added to the column width when we add a landscape oriented image. Headline and preamble will span across both article columns, they will make use of col2Col3Combined.

After the pdf_doc object is created we immediately insert the background pdf.

```
var:'BkgPDF' = 'ex_cont/MultiColumnTemplate.pdf';
$pdfTemplate->(insertpage: (pdf_read:
  -file = $BkgPDF), 1);
```

And continue with placing the logo. To avoid errors we first check that the image file really exist.

```
if:(File_Exists: $logoImgPath);
  $logoImgInfo= (Image: $logoImgPath, -Info);
  $logoImgWidth = $logoImgInfo->Width;
  $logoImgHeight = $logoImgInfo->Height;
  // Reduce the image so it fits into the column.
  $sizeFactor = decimal($col1Width) / decimal($logoImgWidth);
  $logoImgHeight = (Integer:($logoImgHeight * $sizeFactor));
  $pdfTemplate->(Add:(PDF_Image:-File= $logoImgPath),
    -left = $marginLeft + $col1Left,
    -width = (Integer:($logoImgWidth * $sizeFactor)),
    -top = $col1Top + $logoImgHeight,
    -height = $logoImgHeight);
  // Adjust vertical position.
  $col1Top += $logoImgHeight + 10;
/if;
```

The variables logoImgWidth and logoImgHeight will hold the original size of the image. This is so that we can shrink it proportionally when placed. There's supposedly support for a -proportional param in the image tag but I've found it troublesome to work with. Instead it's better to calculate a shrink factor placed in variable sizeFactor. In this case it's the width of the column that govern the factor. By dividing column width with original image width we will know by how much we need to shrink the image. By the applying the shrink factor to both image height and width we have an image that's reduced proportionally to fit within the column. And we also have a value in logoImgHeight that we need in order for the following text to start at a proper distance from the image.

Moving on to the left column text. In the example we use an array of pairs with a name and a value. We want these to be listed with each name in a bold font and each value in plain font followed by a line feed. First we create an empty pdf_text object. And then we iterate through the array placing each name and value into the pdf_text variable.

```
var:'leftColContent' = (PDF_Text:'', -Type='Paragraph',
  -Font = $smallBoldFont);
iterate: $leafletData->'demoItems', $itemRaw;
  $leftColContent->(Add:(PDF_Text:$itemRaw->(Get:1) + ':  ',
    -Font = $smallBoldFont));
  $leftColContent->(Add:(PDF_Text:$itemRaw->(Get:2) + '\r',
    -Font = $smallPlainFont));
/iterate;
```

Once all items are placed in the variable we can insert it into the pdf_doc object.

```
$pdfTemplate->(Add:$leftColContent,
  -left = $marginLeft + $col1Left,
  -width = $col1Width,
  -top = $col1Top,
  -height = $pageHeight - $marginTop - $marginBottom,
  -align = 'left',
  -leading = $smallTextLeading);
```

There! Left column's all done. Let's move over to the article section starting with the headline. Headlines will always start at the same vertical point but they can differ in length and occupy more than one row. So we need to find out where it ends in order for the preamble to start at a proper distance. We will do that with the standard textWidth/math_floor operation.

```
$pdfTemplate->(Add:(PDF_Text:($leafletData->'headingContent'),
  -type='Paragraph',
  -font = $headingFont),
  -left = $marginLeft + $col2Left,
  -width = $col2Col3Combined,
  -top = $col2Top,
  -align = 'left',
  -leading = $headingLeading);
$tempTextCont = $leafletData->'headingContent'->(Split:'\r');
$textVerSize = $tempTextCont->Size;
iterate: $tempTextCont, $tempContainer;
  $textVerSize += math_floor(($headingFont->(textwidth:
    $tempContainer)) / $col2Col3Combined);
/iterate;
$col2Top += ($textVerSize * $headingLeading) +
  ($headingLeading / 2);
```

And then we do exactly the same with the preamble. Both the headline and the preamble span both article columns so we've had no need to keep track of the third columns vertical position until this point. But the rest of the content will need to know so we simply set the third columns variable to the same as the second columns.

```
$col3Top = $col2Top;
```

Now we want to place the article image. Since this is user supplied we can't know how it's oriented. It can be a thin portrait oriented image, almost square, landscape etc. Well, if it higher than it's wider it's simple, let's make it as wide as the column and the use as much of the third column it needs. If it's wider than it's higher then we'll have to steal some of the second column in order for it not to look too small. As with the earlier image we wrap it all in a conditional to

ensure that there really is an image to work with. And then we examine the image to find out it's size and proportions.

```
if:(File_Exists: $leafletData->'imgPath');
  $articleImgInfo = (Image: $leafletData->'imgPath', -Info);
  $articleImgWidth = $articleImgInfo->Width;
  $articleImgHeight = $articleImgInfo->Height;
  $imgColWidth = $artColWidth +
    (($articleImgWidth > $articleImgHeight) ?
      $imgColExtra | 0);
  $sizeFactor = decimal($imgColWidth) /
    decimal($articleImgWidth);
  $articleImgHeight = (Integer:($articleImgHeight *
  $sizeFactor));
```

The variable imgColWidth will be set to the same value as the column width. But if the image is landscape oriented then we add the value stored in imgColExtra. This is then used in finding out the factor by witch we will reduce the image, sizeFactor. With all size and positioning data at hand we can safely add the image.

```
$pdfTemplate->(Add:(PDF_Image:-File= $leafletData->'imgPath'),
  -left = $marginLeft + $col3Left + $artColWidth -
    $imgColWidth,
  -width = (Integer:($articleImgWidth * $sizeFactor)),
  -top = $col3Top + $articleImgHeight,
  -height = $articleImgHeight);
```

Next step is to add some image text beneath the image. We will use the same column settings as the image so that the text will align nicely with it.

```
  $pdfTemplate->(Add:(PDF_Text:
    ($leafletData->'imgTextContent'),
    -type='Paragraph',
    -font = $imgTextFont),
    -left = $marginLeft + $col3Left +
      $artColWidth - $imgColWidth,
    -width = $imgColWidth,
    -top = $col3Top - 14,
    -align = 'left',
    -leading = $imgTextLeading);
```

And of course calculate a new vertical position for the third column. Let's look at that piece of code once more.

```
$tempTextCont = $leafletData->'imgTextContent'->(Split:'\r');
$textVerSize = $tempTextCont->Size;
iterate: $tempTextCont, $tempContainer;
  $textVerSize += math_floor(($imgTextFont->(textwidth:
    $tempContainer)) / $imgColWidth);
/iterate;
```

Finding out this way to use textWidth to calculate how a text will span rows was a revelation for me and have really helped me in a lot of situation. But it's not without flaws. The textWidth/ math_floor trick only delivers an assumption of how many rows a piece of text will occupy.

In rare occasions there will in reality be one line more than we've been told. this is due to that textWidth locks only at each char and how much horizontal space it needs. But when the text is actually placed in the pdf_doc object long words can move down one row to fit, ending in more rows being used.

When it comes to placing the actual article text that kind of uncertainty is not acceptable. Furthermore we plan to split the article into two columns so we need to know when to move over to the next column. I've found no other way to do this than to use my own row building, adding word for word, placing row for row and thus being 100% sure of where I am in the column. If there's someone out there with a better solution, please tell me!

First step is to clean the input data, set the positioning variables and then split on line feeds. We will iterate on paragraphs and then iterate again within each paragraph on words. First of all we need a variable to store each row temporary.

```
var:'tempTextCont' = string;
```

Then in each iteration we check if the content of tempTextCont plus the new word will fit within a row. If it fits all is dandy and we can add the word to tempTextCont. If not, we add tempTextCont to the pdf_doc object, empty the temporary variable and insert the word that didn't fit into it.

```
iterate: $tempTextCont, $tempContainer;
  iterate: $tempContainer->(Split:' '), $wordTemp;
    if:($plainFont->(TextWidth:($thisTextRow + $wordTemp)))
      < $thisColWidth;
      $thisTextRow += $wordTemp + ' ';
    else;
      $pdfTemplate->(Add:(PDF_Text:$thisTextRow,
        -type='Paragraph',
          -Font = $plainFont),
        -left = $thisColLeft,
        -width = $thisColWidth,
        -top = $thisColTop,
        -align = 'left',
        -leading = $textLeading);
      $thisTextRow = $wordTemp + ' ';
    /if;
```

We also have to constantly check to see where the text is vertically positioned. First to adjust column width once the text clears the image and then to move to the next column.

```
$col2Space -= $textLeading;
 if:($thisColTop >= $col3Top - 15) && $imgNotPassed;
  $thisColWidth = $artColWidth;
  $thisColTop += $textLeading;
  $imgNotPassed = false;
  $col3Top = $thisColTop;
 else:($col2Space < $marginBottom) && $colCount == 2;
  $thisColTop = $col3Top;
  $col2Space = $col3Space;
  $thisColLeft = $marginLeft + $col3Left;
  $colcount += 1;
else;
  $thisColTop += $textLeading;
/if;
```

Should there be so much text that it wont fit it will just continue in the third column beyond the paper edge. Depending on circumstances we might want to catch that and throw some kind of warning. I hand that over to you to solve. :-)

Well that's all for today. Questions anyone?

Lasso: The Future of the Language

Kyle Jessup

Kyle Jessup, Director of Development for Lasso, will present a technology preview of the future of Lasso. Details about his presentation will be handed out the day of the event.

Cutting Through The Hype: Simple Web 2.0 with Lasso and jQuery

Jason Huck

Introduction

Anyone in the web development industry has by now gotten used to seeing the term "Web 2.0" making headlines in tech news on a daily basis. In the three years since it was coined by O'Reilly and Medialive International to help promote a conference(1), it's been associated with everything from the next dot-com bubble to AJAX and DHTML, to wikis and web feeds, to the use of large fonts, rounded corners, and reflections on web pages. And, despite being constantly ridiculed by nearly everyone in the business, "Web 2.0" was the most-mentioned article on Wikipedia in 2006(2).

From a practical implementation standpoint, however, a number of interesting and useful technologies have emerged in relation to the Web 2.0 phenomenon, most involving the use of obscure and complicated javascript techniques which turn simple web pages into single-page applications capable of updating themselves and interacting with remote web services, blurring the line between the web and the desktop. Thanks to the ever-increasing rate of adoption on the net, these "bleeding edge" techniques have become the status quo in record time.

Unfortunately, this rapid pace presents a fractured landscape to the developer looking for simple ways to incorporate Web 2.0 functionality into his or her own projects: myriad libraries with vastly different approaches vying for attention, all still under heavy development and usually sparsely documented, if at all.

Thankfully, things are starting to change. Newer libraries such as Yahoo!'s YUI come with copious documentation, thorough code samples, and vibrant community support. One of the most unique and interesting of these new players is jQuery, originally created by John Resig in January of 2004(3). jQuery uses CSS and XPath selectors to traverse and manipulate the DOM, and provides a clear and simple API for binding events to elements. This combination results in an extremely short learning curve and allows developers to become productive almost immediately.

This paper demonstrates how to incorporate common Web 2.0 techniques into an existing web project using Lasso and jQuery, picking apart the acronyms and delving into the file formats and protocols used to communicate with web services, including XML, REST, AJAX, and AHAH.

Using [include_url] To Communicate With Web Services

Lasso provides several tools for communicating with other services on the web. At the most basic level, there is the [include_url] tag, which is actually a wrapper for a command line tool called cURL. In addition, there are special tags for communicating with XMLRPC and SOAP services,

and finally, for anything not already covered, you can control the communication at a very low level using the NET tags.

Of all these methods, [include_url] is both the easiest to use, and often the most robust, providing manual flexibility when more automated methods like the SOAP tags fail, without involving the complexity of using the NET tags. So, for the purposes of this discussion, we'll use [include_url] to facilitate communication with remote servers.

As with many things Web 2.0, what was old is new again. Most Lasso developers have likely used [include_url] in the past to retrieve data from a CGI script by passing parameters on the query string within the URL. In today's Web 2.0 world, the acronym du jour is REST. Although there are academic distinctions between CGI (short for "Common Gateway Interface") and REST ("Representational State Transfer") in terms of exactly what they describe architecturally, in terms of actual implementation, a "RESTful" Web API will require arguments that are passed to the script as simple GET or POST params, just like most CGI's. So, retrieving data from a REST interface is fairly straightforward.

This example uses Google's geocoding service to return a set of coordinates for a mailing address. You pass the address, the preferred result format, and an API key as arguments. In this case, we're requesting an XML response and using the address for Core Five's office in Cincinnati, OH. You can view the result directly in a web browser by passing everything on the query string like so:

http://maps.google.com/maps/geo?output=xml&key=xxxxxxxxxx&q=2245+Gilbert+Avenue,+Cincinnati,+OH+45206

...and, you could just stuff that entire URL into [include_url] and it would work, but it's a little easier to manage the parameters programatically if we take advantage of the -getparams keyword and set them up separately in an array:

```
[//lasso
  // set up arguments
  var('getparams') = array(
    'q' = '2245 Gilbert Avenue, Cincinnati, OH 45206',
    'output' = 'xml',
    'key' = 'xxxxxxxxxxxxx'
  );
  // retrieve the result
  var('result') = include_url(
    'http://maps.google.com/maps/geo',
    -getparams=$getparams
  );

  // display the result
  content_type('text/xml');
  $result;
]
```

Running the above code (with a valid API key) will return an XML document containing the information we're after. But before we move on to parsing data, let's consider some additional best practices when working with remote servers.

For one, cURL, the underlying library that [include_url] uses, has no default timeout. So, if the remote server goes down or takes too long to respond, our script will wait indefinitely, until some other timeout (the Lasso Connector, the browser, etc.) eventually overrides it. Fortunately, you can specify your own timeout. But, there could be other problems, like errors on the remote server, or changes in the API of which you aren't notified. Or perhaps your access is via an annual subscription which expires while you're away. Because you don't want the availability of your site to be tied inextricably to the availability of another, you should always use [protect] around your [include_url] calls. If the data you are retrieving changes on a set frequency, such as once a day, you might also consider caching the results to avoid unnecessary load on the remote server and improve the speed at which your own page renders.

Let's put these ideas into action. The following example retrieves data from Yahoo!'s "Word of the Day" service. Since the data only changes once a day, we cache it locally using a global variable. We include a timeout and use [protect] to keep errors at the remote server from causing us problems, and parse the response using a custom tag called [xml_tree], which we'll discuss later. Finally, we wrap the entire operation up as a custom tag to make it easier to re-use.

```
[//lasso
  define_tag(
    'wordoftheday',
    -namespace='yahoo_',
    -priority='replace',
    -description='Returns the Word of the Day from Yahoo!'
  );
    // check to see if a global map called 'yahoo_wotd' exists,
    // and if so, whether the date stored within it is less than
    // 24 hours old...
    if(
      global_defined('yahoo_wotd')
      && $yahoo_wotd->find('date')->difference(date, -hour) < 24
    );
      // if so, return it to the user
      return($yahoo_wotd);
    else;
      // if not, protect the retrieval operation
      protect;
        // use a 10 second timeout, and store the results as an
        // xml tree type in a variable called 'data'
        local('data') = xml_tree(
          string(
            include_url(
              'http://xml.education.yahoo.com/rss/wotd/',
              -timeout=10
            )
          )
        );

        // parse the xml into a map called 'out'
        // this is explained below
        local('out') = map(
          'word' = #data->channel->item->title
            ->contents->split(' - ')->first,
          'definition' = #data->channel->item->description->contents,
          'link' = 'http://education.yahoo.com/'
            + #data->channel->item->guid->contents,
          'date' = date_gmttolocal(
            date(
```

```
            #data->channel->item->pubDate->contents,
            -format='%a, %d %b %Y %H:%M:%S GMT'
          )
        )
      );

      // set the global 'yahoo_wotd' to the contents of 'out'
      // and return it to the user
      global('yahoo_wotd') = #out;
      return(#out);
    /protect;
  /if;
/define_tag;
]
```

Using [xml_tree] To Parse XML Documents

Like the earlier Google example, this Yahoo! service returns its data in an XML format. This is one of several very common formats used to exchange data between different systems, and you've no doubt come across it in many different places throughout the web. HTML, SOAP, and RSS are all sub-sets of XML. Even Lasso uses XML as its internal serialization format.

Lasso provides a native XML type to help you read and write XML documents. It supports XML's own query language, XPath, as well as a handful of useful shortcuts for common operations. Unfortunately, dealing with XML produced in "the real world" presents problems that aren't always easy to solve based on the examples provided in the Lasso Language Guide. Third party customizations to the loosely-defined standard, multiple imported namespaces, and other variations are often problematic for someone who isn't an XML expert. Consider the following example from a Tip of the Week entitled Extracting Data From XML Using XPath(4):

```
[//lasso
  var('raw' = '\
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:complexType name="anyLassoType">
    <xsd:choice>
      <xsd:element ref="nullType"/>
      <xsd:element ref="stringType"/>
      <xsd:element ref="integerType"/>
      <xsd:element ref="decimalType"/>
      <xsd:element ref="bytesType"/>
      <xsd:element ref="booleanType"/>
      <xsd:element ref="pairType"/>
      <xsd:element ref="arrayType"/>
      <xsd:element ref="mapType"/>
      <xsd:element ref="tagReferenceType"/>
      <xsd:element ref="customTagType"/>
      <xsd:element ref="customType"/>
    </xsd:choice>
  </xsd:complexType>
</xsd:schema>\
  ');
]
```

Let's say we need to retrieve all of the "element" nodes in this document. Using the built-in member tags of the XML type, it is possible, though tedious, to drill down to the element level,

as shown in the first example below. The second example uses XPath to get directly to the target, but since this document uses namespaces, the syntax is far from intuitive. Imagine having to use such conventions for more complex filtering operations.

```
[//lasso
  // retrieve all 'element' nodes using built-in member tags:
  var('xml') = xml($raw);
  var('elements') = @$xml->firstchild->nextsibling->firstchild->nextsibling->children;
  $elements->foreach({
    string(params->first)->iswhitespace ? $elements->removeall(params->first);
    return(true);
  });
  $elements;
]
[//lasso
  // retrieve all 'element' nodes using XPath
  var('xml') = xml($raw);
  var('elements') = $xml->extract('//*[name()="xsd:element"]');
  $elements;
]
```

The custom type [xml_tree] extends the native [xml] data type, and makes it easier to retrieve data from an XML document by handling some of the dirty work behind the scenes. If the structure of the XML document is known, you can drill down to the nodes and/or attributes you need by name, using familiar ->(member tag) notation. Knowledge of XPath and namespaces is not required. Because of these capabilities and the ubiquitous nature of XML on the web, [xml_tree] is an extremely useful tool for dealing with web services.

```
[//lasso
  // retrieve all 'element' nodes using [xml_tree]
  var('xml') = xml_tree($raw);
  var('elements') = $xml->complexType->choice->element;
  $elements;
]
```

Taking another look at the XML response from our original Google geocoding example, it's easy for the human eye to spot the information we're after, but how do we tell Lasso to access this information? With [xml_tree], it's easy. We can see that the data we're after is inside a set of tags (also called a node in XML speak) called "coordinates", which is inside something called "Point", etc. etc. all the way up to the "root" element, "kml." If we create a new [xml_tree] instance called "data" using this document and get its name (a function inherited from the built-in [xml] type), we'll see that it's the kml node. So, if we're starting at kml, what's the path down to the coordinates? Let's plug it in and see what happens:

```
[//lasso
  var('data') = xml_tree('\
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.0">
<Response>
  <name>2245 Gilbert Avenue, Cincinnati, OH 45206</name>
  <Status>
    <code>200</code>
    <request>geocode</request>
  </Status>
```

```
  <Placemark>
    <address>2245 Gilbert Ave, Cincinnati, OH 45206, USA</address>
    <AddressDetails Accuracy="8" xmlns="urn:oasis:names:tc:ciq:xsdschema:xAL:2.0">
      <Country>
        <CountryNameCode>US</CountryNameCode>
        <AdministrativeArea>
          <AdministrativeAreaName>OH</AdministrativeAreaName>
          <SubAdministrativeArea>
            <SubAdministrativeAreaName>
              Hamilton
            </SubAdministrativeAreaName>
            <Locality>
              <LocalityName>Cincinnati</LocalityName>
              <Thoroughfare>
                <ThoroughfareName>
                  2245 Gilbert Ave
                </ThoroughfareName>
              </Thoroughfare>
              <PostalCode>
                <PostalCodeNumber>
                  45206
                </PostalCodeNumber>
              </PostalCode>
            </Locality>
          </SubAdministrativeArea>
        </AdministrativeArea>
      </Country>
    </AddressDetails>
    <Point>
      <coordinates>-84.493147,39.121771,0</coordinates>
    </Point>
  </Placemark>
</Response>
</kml>\
  ');

  $data->Response->Placemark->Point->coordinates;
  // returns: <coordinates>-84.493147,39.121771,0</coordinates>
]
```

Voila! We've retrieved the "coordinates" node. However, since we want just the data inside the node, we can use the ->contents member tag (all of the member tags of the native [xml] type are also available to [xml_tree]) like so:

```
[//lasso
  $data->Response->Placemark->Point->coordinates->contents;
  // returns: -84.493147,39.121771,0
]
```

From there, it's a simple matter to copy the data into a variable and manipulate it to get separate decimal values for longitude and latitude:

```
[//lasso
  var('coords') = $data->Response->Placemark->Point->coordinates->contents;
  var('x') = decimal($coords->split(',')->first);
  var('y') = decimal($coords->split(',')->second);
]
```

Introducing jQuery

So far, we've looked at how to use [include_url] to retrieve an XML response from a REST API and extract data using [xml_tree]. We've seen that with Lasso's trademark ease of use, consuming data from web services on the server side can be simple and straightforward. But so much of the Web 2.0 experience happens on the client side via JavaScript; what about that? What tool can we use on the client side that is as quick, easy, intuitive, and powerful as Lasso is on the server side? Let's talk about jQuery.

I like to say that jQuery turns JavaScript inside out. Instead of extending traditional JavaScript coding methods to solve common web development problems, it takes a whole new approach, with code constructs that are practically self-documenting in the way they describe exactly what you are doing. And it leverages a technology that every web developer already knows how to use -- Cascading Style Sheets -- rather than relying on knowledge of core JavaScript itself. It's true, you don't really even have to know JavaScript that well to accomplish a lot with jQuery. Let's start with the following simple HTML page:

```
<html>
  <head>
    <title>The Book of Lummitations</title>
  </head>
  <body>
    <h1>The Book of Lummitations</h1>
    <h2>Chapter 1</h2>
    <ol>
      <li>Thou shalt have no
      <a href="http://www.php.net/">other languages</a>
      before Me.</li>
    </ol>
  </body>
</html>
```

Before we can write any jQuery code, we have to link to the jQuery library via a <script> tag. This is normally placed in the <head> of the document, but if necessary can be placed elsewhere. This example links directly to the jQuery site for the latest version, but in production you would want to link to a local copy instead:

```
<script
  type="text/javascript" src="http://jquery.com/src/jquery-latest.pack.js">
</script>
```

Manipulating the DOM (short for Document Object Model, the browser's representation of a page) involves binding events to elements. We can't do that until we know the browser has a complete representation ready for us. So, the first thing we'll do is bind a "ready" event to the entire document, like so:

```
<script type="text/javascript">
$(document).ready(function(){
  // where the fun happens.
});
</script>
```

This is similar to using an "onload" event in traditional javascript, except better, since the DOM is actually ready before the page finishes loading. jQuery uses the $ symbol as an alias for the jQuery class, so $(argument) creates a new jQuery object based on whatever criteria is passed in as an argument. In this case, we're creating a jQuery object for the entire document, and binding a "ready" event to it. The argument for the ready event is a generic function. Since a function can contain any other code you wish, we're basically saying, "do whatever is in here as soon as you have a complete representation of the page ready." The syntax may seem foreign at first, but all of jQuery's methods follow this convention:

```
$(selector).command(arguments, callback);
```

- The selector is any valid javascript object, CSS selector, or XPath expression. Because of the CSS support, anything you can point to in your site's style sheet, you can point to with jQuery as well.
- The command is the operation to perform on — or the event to bind to — the selected object. Operations might be visual effects like "hide", "fadeIn", or "slideUp", or setting properties like "text", "html", or "css", or performing AJAX functions via "get", "post", or "load." Events are the usual JavaScript event handlers: "click", "change", "submit", etc.
- The arguments are the parameters which specify how the command should work. For example, "hide" accepts "slow" as an argument to control the speed of the effect. Arguments can be single elements, like a string or integer, or a hash of elements in JSON format.
- The callback is the function to perform after the command is completed. Although you can simply insert the function's name here, you will often see a generic function(){ ... } call in this position, allowing the developer to execute arbitrary code. In Lasso terms, this is similar to how array->foreach can accept either a tag reference or a compound expression.

Now we're ready to have some fun. Let's say we want to change what happens when you click on the link within this page:

```
$('a').click(function(){
  // what do you want to do?
});
```

This creates a jQuery object for all the anchors on the page. Any actions we take on this object will effect all the anchors. In this case, there is only one, but if there were more, we could select the correct one by its ID, its class, its position within the DOM, or the fact that it's the first one on the page, with just slight variations to the selector above.

We've attached an onclick event to the link using the appropriately named .click function, and passed in as an argument another generic function. You'll get used to seeing that, it's just a quick way of allowing the developer to execute arbitrary code right away. If we actually had a function defined elsewhere in our script, we could instead just call it by name, i.e.:

```
$('a').click(myFunction);
```

So, let's say that, on click, we want to add another item to our ordered list. Easy enough:

```
$('ol').append('<li>Thou shalt obey rule number one.</li>');
```

And, let's not allow the original click to go through (you can still use "regular" javascript with jQuery):

```
return false;
```

Here's another look at the page after our additions:

```
<html>
  <head>
    <title>The Book of Lummitations</title>
    <script type="text/javascript" src="http://jquery.com/src/jquery-latest.pack.js">
    </script>
    <script type="text/javascript">
      $(function(){
        $('a').click(function(){
          $('ol').append('<li>Thou shalt obey rule number one.</li>');
          return false;
        });
      });
    </script>
  </head>
  <body>
    <h1>The Book of Lummitations</h1>
    <h2>Chapter 1</h2>
    <ol>
      <li>Thou shalt have no
      <a href="http://www.php.net/">other languages</a>
      before Me.</li>
    </ol>
  </body>
</html>
```

As you can see, you can get a lot of mileage out of a few lines of jQuery. Here are a few more examples:

```
// change the background color of a paragraph:
$('#one').css('background-color','red');
// hide a paragraph, change its text, and then show it again
$('#two').hide('slow', function(){
  $(this).html('New text.')
}).show('slow');
```

The first example is rather straightforward. A method 'css' allows you to modify any CSS property of the selection. However, you'll notice a couple of handy concepts in the second example. First, jQuery's methods are chainable. This means that if you have a series of functions to perform on a single selection, you can string them together in a single statement. In this case, the .hide and .show methods are chained together as a single action.We put the .html command inside the callback function of the .hide command, instead of chaining all three commands together, to make sure it doesn't run before the .hide effect has finished. You'll also notice in the callback function of the .hide method, that jQuery recognizes 'this' as the current selection. So,

it can re-use the current selection in the inner function automatically. Using $(this) is the same as re-specifying $('ol li'), only faster, because you're re-using the existing object instead of re-searching the DOM and creating a new one.

Simple AJAX With Lasso and jQuery

Visual effects and DOM manipulation aside, no modern JavaScript library would be complete without robust AJAX support, and jQuery is no slouch in that department. Depending on your specific needs, jQuery can submit GET or POST requests using either AJAX or AHAH transport methods, and return data as XML, JSON, plain text, or HTML fragments. Let's look at a common need, which is to update a portion of a page with new server-side data without refreshing the entire page.

We'll use another Lasso custom tag in this example, [request_isajax]. jQuery, along with most other JavaScript libraries, including Prototype, LJAX, Laszlo, and others, includes an extra line in the request header when it makes HTTP calls. [request_isajax] checks to see if that line is present, and returns true or false accordingly. Thus, we can execute code conditionally based on whether or not the current request is an AJAX call.

In this example, we'll use jQuery's .load() method to replace the contents of the selected element with new data from the server. In order to keep things simple and self-contained, we'll have the page call itself. On the server side, we'll detect whether the request is an AJAX call using [request_isajax] so that we only return the refreshed data. Finally, as a simple means of making sure we have new data for each request, we'll use [lasso_uniqueid]. Notice that there are no restrictions on the format of the response.

```
[//lasso
  // handle ajax requests
  if(request_isajax);
    // control what is returned by replacing the entire page buffer with our new data
    content_body = lasso_uniqueid;
    // immediately abort to keep anything else from being added
    abort;
  /if;]
<html>
  <head>
    <title>AHAH! That was easy.</title>
    <script type="text/javascript" src="http://jquery.com/src/jquery-latest.pack.js" />
    <script type="text/javascript">
    /* The "$(function(){" is shorthand for the "$(document).ready(function(){"
       construct we discussed earlier. It does exactly the same thing, but
       saves a few keystrokes. */
    $(function(){
      // bind a click event to #rand
      $('#rand').click(function(){
```

```
        // replace the contents of #rand with new content from the server
        $(this).load('[response_filepath]');
      });
    });
    </script>
  </head>
  <body>
    <p id="rand">[lasso_uniqueid]</p>
  </body>
</html>
```

Putting It All Together

We've looked at several different aspects of implementing Web 2.0 features in Lasso and jQuery in a piecemeal fashion. First, we looked at some best practices for retrieving data from remote web services in Lasso using [include_url], and how to parse that data into something useful using [xml_tree]. Next, we explored how to take control of the DOM using jQuery to select and manipulate the style, structure, and content of the document. Finally, we presented a very basic example of combining Lasso and jQuery to create a super simple AJAX enabled page.

This is only a very basic introduction to what can be accomplished by combining the power of Lasso and jQuery. Like Lasso, jQuery benefits from a thriving, active user community, and due to jQuery's extensible architecture, new plugins are released almost daily to make the creation of things like modal dialogs, type-ahead search boxes, and slideshows as easy to implement as the core "hide" and "show" effects.

On the CD accompanying this paper, you'll find a more complete example that combines all of these techniques into a single module which displays the content from a web service and retrieves updates based on user interaction, using AJAX so that the rest of the page does not have to be refreshed with each update.

Resources

Visit the following sites to learn more about the topics covered in this paper:

http://jquery.com/

The central hub for all things jQuery. Maintained by the author of jQuery, John Resig, and key members of the development team, it is the home of the official blog, documentation, plugin directory, and source code repository.

http://www.nabble.com/JQuery-f15494.html

Just like Lasso, jQuery has a highly active user community. Check this archive of the main mailing list for general news, support, and new plugin announcements.

http://docs.jquery.com/

The official documentation, recently redesigned for jQuery's one-year anniversary.

http://visualjquery.com/

Extremely handy third party interface for browsing the jQuery API. Since the library is self-documenting, this interface is generated straight from the source code and should match the official documentation to the letter on content. It even includes documentation for third party plugins that have been checked into the repository.

http://learningjquery.com/

Great collection of articles and tutorials for jQuery coders of all skill levels.

http://www.programmableweb.com/

If you're looking for a particular web service and are not sure whether something like it exists, check here.

http://tagSwap.net/

The latest versions of the custom tags used in these examples are available on tagSwap. In addition, there are many other tags posted there which interact with web services from Google, Yahoo!, Feedburner, WeatherBug, and others.

Tools

You may find the following development tools useful when working with jQuery:

http://www.aptana.com/

Available as an Eclipse plugin or a standalone application, the Aptana IDE contains numerous helpers for HTML, CSS, and JavaScript, including jQuery support.

http://www.macromates.com/

TextMate, billed as "The Missing Editor For Mac OS X", is a full featured text editor with extensible support for a variety of languages. A jQuery "bundle" is available.

http://getfirebug.com/

From the site: "Firebug integrates with Firefox to put a wealth of development tools at your fingertips while you browse. You can edit, debug, and monitor CSS, HTML, and JavaScript live in any web page."

(1) http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html

(2) http://www.nielsenbuzzmetrics.com/release.asp?id=170

(3) http://jquery.com/blog/2006/01/24/jquery-blog/

(4) http://www.omnipilot.com/Tip of the Week.1768.9008.lasso

Implementing a Changes Tracking System

Eric Landmann

Abstract

As websites are becoming more sophisticated and accessed by more people with different roles, keeping a record of who made changes to which data can be important. For nearly any sort of online application, this information can be vital, and is a required component of many sites. This paper addresses how to install a system that tracks data changes for any type of data, how it is constructed, and code examples. Starting from scratch or retrofitting an existing site will be discussed.

The application - lbt (issue tracking)

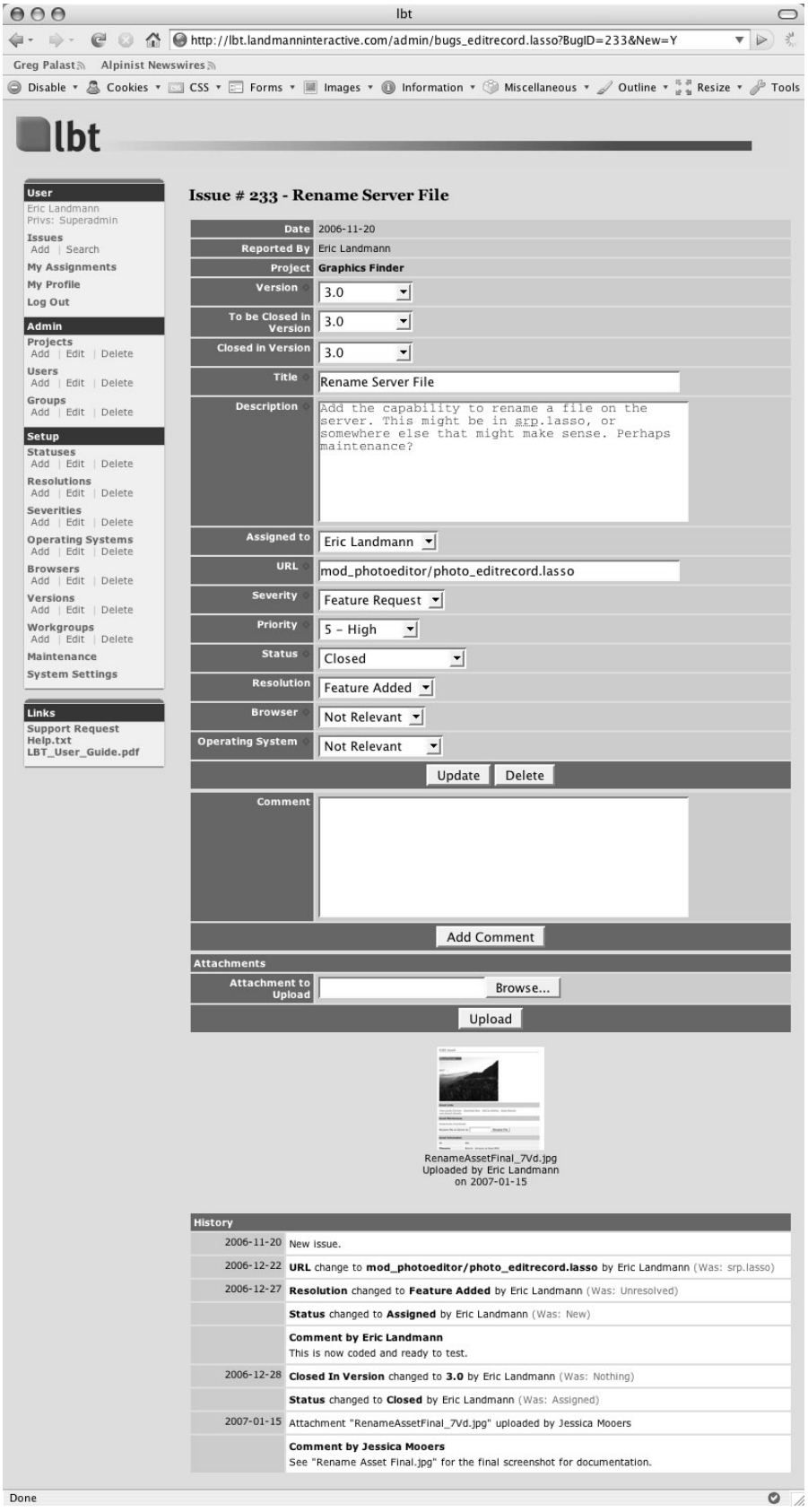
Application Purpose: To collect feature requests and track issues and correspondence relating to a particular issue. Its intended use is for workgroups of programmers, graphic artists, project managers, and testing personnel to all contribute to the project.

The LBT application consists of 24 tables, four of which are described here. The code example we will be discussing tracks changes to 17 fields on the main data table and several other tables. It can easily be extended to handle many more fields or tables.

The data collected by the changes tracker is used as a history of all changes to a particular issue. It records who made a change, the previous value, the new value, and the date it was done. This data could be kept as a permanent record of changes, if the business requirements of the application dictate that.

Key concepts

- Data in the main data table (lbt_bugs in our example) is very dynamic in this application, with changes made by potentially many people in a compressed timespan. This table keeps track of the current state of an issue.
- Data in the changelog table (lbt_changelog) is never changed, and contains the history of the changes to the issue.
- One record is added in the changelog table for every field in the main data table that is changed.
- Other tables can be included in the changes tracker. In this example, comments are added to a comments table (lbt_comment), and attachments are uploaded and recorded in an attachments table (lbt_attachments). This makes it very extensible.
- The changelog table contains a permanent record of all changes made. The only time the data in this table is changed is when the record in the main data table is deleted, then all the changelog entries must be deleted as well.
- Deletion of changelog records will vary depending upon your application requirements.


lbt
http://lbt.landmanninteractive.com/admin/bugs_editrecord.lasso?BugID=233&New=Y
Greg Palast
Alpinist Newswires
Disable
Cookies
CSS
Forms
Images
Information
Miscellaneous
Outline
Resize
Tools
lbt
User
Eric Landmann
Privs: Superadmin
Issues
Add | Search
My Assignments
My Profile
Log Out
Admin
Projects
Add | Edit | Delete
Users
Add | Edit | Delete
Groups
Add | Edit | Delete
Setup
Statuses
Add | Edit | Delete
Resolutions
Add | Edit | Delete
Severities
Add | Edit | Delete
Operating Systems
Add | Edit | Delete
Browsers
Add | Edit | Delete
Versions
Add | Edit | Delete
Workgroups
Add | Edit | Delete
Maintenance
System Settings
Links
Support Request
Help.txt
LBT_User_Guide.pdf
Issue # 233 - Rename Server File
Date
2006-11-20
Reported By
Eric Landmann
Project
Graphics Finder
Version
3.0
To be Closed in Version
3.0
Closed in Version
3.0
Title
Rename Server File
Description
Add the capability to rename a file on the server. This might be in srp.lasso, or somewhere else that might make sense. Perhaps maintenance?
Assigned to
Eric Landmann
URL
mod_photoeditor/photo_editrecord.lasso
Severity
Feature Request
Priority
5 - High
Status
Closed
Resolution
Feature Added
Browser
Not Relevant
Operating System
Not Relevant
Update
Delete
Comment
Add Comment
Attachments
Attachment to Upload
Browse...
Upload
RenameAssetFinal_7Vd.jpg
Uploaded by Eric Landmann
on 2007-01-15
History
2006-11-20
New issue.
2006-12-22
URL change to mod_photoeditor/photo_editrecord.lasso by Eric Landmann (Was: srp.lasso)
2006-12-27
Resolution changed to Feature Added by Eric Landmann (Was: Unresolved)
Status changed to Assigned by Eric Landmann (Was: New)
Comment by Eric Landmann
This is now coded and ready to test.
2006-12-28
Closed In Version changed to 3.0 by Eric Landmann (Was: Nothing)
Status changed to Closed by Eric Landmann (Was: Assigned)
2007-01-15
Attachment "RenameAssetFinal_7Vd.jpg" uploaded by Jessica Mooers
Comment by Jessica Mooers
See "Rename Asset Final.jpg" for the final screenshot for documentation.
Done

Application flow

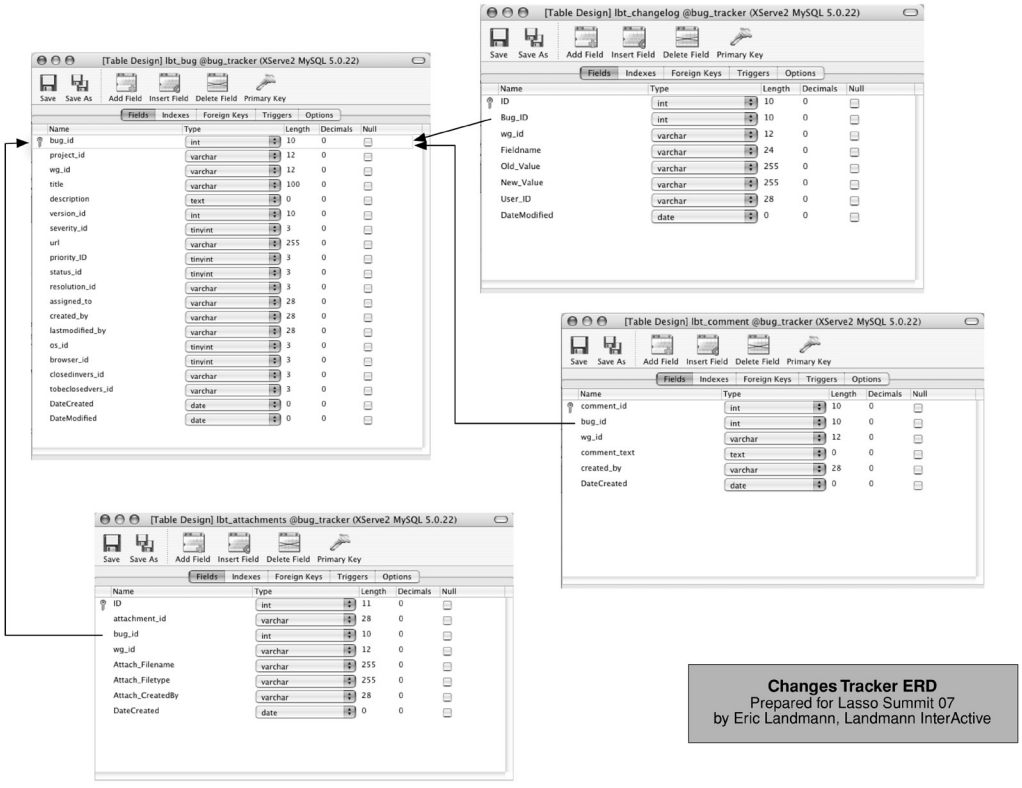
Here is a quick outline of the relevant portions of the application flow:

New Issue (Issues->Add)
- User enters new issue data
- User submits the form
  - New record is added to main data (issues) table
  - New record is added to changelog table
- User is returned to the Issue page

Existing Issue (Issues->Search)
- User changes data
  - Data is updated in issues table
  - New records are added of changes in the changelog table
  - Page is redisplayed with current data and new history

- Add comments (bugs_editrecord.lasso)
  - New record is added in comments table
  - New record is added to changelog table
  - Page is redisplayed with current data and new history

- Add attachment (bugs_editrecord.lasso)
  - New record is added in attachments table
  - New record is added to changelog table
  - Page is redisplayed with current data and new history



Data structure

```
  lbt_bug: The main data table, keeps track of the issue and feature request data
  lbt_comment: Contains comments relating to a particular issue
  lbt_attachments: Contains uploaded attachments (prototypical of related tables)
  lbt_changelog: Contains a record of all the changes
```

Code files

```
bugs_editrecord.lasso - a combination edit/display page for the issue. It includes:
  show_pagehead.inc - the page header
  navbar_main.inc - the left navbar
  frm_bug.inc - the actual form that is displayed. Code to fetch individual issue data
      is at the top of the form.
    bugs_params.inc
      -- converts action_params into vars
      -- checks for missing data, sets error codes
      -- initializes form array variables
  bug_history.inc - the history display
  page_footer.inc - the page footer
bugs_addresponse.lasso - the response page for the bugs_editrecord.lasso
  bugs_addresponse.inc - process logic to process the results of form submission
    bugs_params.inc
      -- converts action_params into vars
      -- checks for missing data, sets error codes
      -- initializes form array variables
    add_changelog.inc - adds the record to the changelog table
lbt_CTags.lasso - Some custom tags that are included are used mainly to get labels
from a provided ID. Also is the definition of the show_error tag and site variables.
```

Logic flow

User edits form and submits. Process logic in bugs_addresponse.inc detects whether the form has been submit from an add or edit page by checking which form parameters have been submit. The controlling form parameter is "Action".

1. If Action is "Add", the request came from the add page
    1a. Convert action_params (line 34)
    1b. Add the record in the main table (lines 43-105)
    1c. Run Changelog routine (line 94)
        *** See explanation below
    1d. Display an error (message) with link to newly-created issue
        (bugs_addresponse.inc, line 333)
2. If Action is "Update" (line 119), the request came from the editrecord page
    2a. Run Changelog routine (line 149)
        *** See explanation below
    2b. Update the record in the main table (lines 153-188)
    2c. Redirect to bugs_editrecord.lasso with error (message)
        (bugs_addresponse.inc, lines 352-365)

3. If Action is “Add Comment” (line 200)
    3a. Convert action_params (line 203)
    3b. Add the record in the comments table (lines 209-261)
4. If Action is “Delete” (line 264)
    4a. Delete the main record (line 275)
    4b. Delete the attachments (line 283)
    4c. Delete the comments (line 287)
    4d. Delete the changelog entries (line 291)
    4e. Redirect to search form
5. If Action is “Upload” (line 307)
    5a. Process the uploads (line 309)
    5b. Add changelog entry (line 312)

```
 1 <?Lassoscript
 2 // Last modified: 1/15/07 by ECL, Landmann InterActive
 3 // Prepared for Lasso Summit 07
 4
 5 // FUNCTIONALITY
 6 // Changelog - Keeps track of changes to multiple database tables
 7 // Create new array called FormArray from the Action_Params
 8 // Then compare the form array with the database array to find differences
 9 // Comparison ignores items beginning with a dash or equaling “Update” or “Bug_
   // ID”
10 // and also anything beginning with a dash ‘-’ which are Lasso operators
11 // Create a new “cleaned” array called FormArrayClean from the items we want to
   // compare
12
13 // Debugging
14 // Var:’svDebug’ = ‘Y’;
15
16 // ============== ADD ========
17 If: $vAction == ‘Add’;
18
19   Var:’SQLAddChangeTableAdd’ =
20     ‘INSERT INTO ‘ $svChangelogTable ‘ SET
21     Bug_ID=”’ $vNewBug_ID ‘”,’ ‘
22     wg_id=”’ $svUser_WGID ‘”,’ ‘
23     Fieldname=”NewIssue”,
24     Old_Value=””,
25     New_Value=””,
26     User_ID=”’ $svUser_ID ‘”,
27     DateModified=”’ (Date_Format:(Date_GetCurrentDate),-DateFormat=’%Y-%m-%d’)
       ‘””;
28
29   Inline: $IV_AddChangelog, -SQL=$SQLAddChangeTableAdd;
30     If: (Error_CurrentError) != (Error_NoError);
31       Var:’vError’ = ‘9001’;
32       Var:’vOption”’ = ‘Writing changelog failed [Error 9001]’;
33     /If;
34   /Inline;
35
36 // ============== UPDATE ========
37 Else: $vAction == ‘Update’;
38
39   If: $svDebug == ‘Y’;
40     ‘<br>\r’;
41     ‘41: <b>Action_Params</b> = ‘ (Action_Params) ‘<br>\n’;
42   /If;
```

```
43
44  // Loop through the FormArray and clean it of all the Lasso operators
45  // Remove “Update” too, which was the value of the form button
46  Loop:($FormArray)->Size;
47    If: !(($FormArray->get:(Loop_Count)->first)->(beginswith:’-’) ||
48      (($FormArray->get:(Loop_Count)->first) == ‘Update’) ||
49      (($FormArray->get:(Loop_Count)->first) == ‘Action’) ||
50      (($FormArray->get:(Loop_Count)->first) == ‘BugID’) ||
51      (($FormArray->get:(loop_count)->first) == ‘Process’));
52        $FormArrayClean-> insert:($FormArray->get:(Loop_Count));
53    /If;
54  /Loop;
55
56  // Sorting the array
57  $FormArrayClean->(Sort:True);
58
59  If: $svDebug == ‘Y’;
60    ‘<br>\r’;
61    ‘61: <b>FormArrayClean</b> is <b> ‘ ($FormArrayClean->Size) ‘</b>
      elements<br>\n’;
62    ‘61: <b>FormArrayClean</b> = ‘ $FormArrayClean ‘ <br>\n’;
63  /If;
64
65  // Getting the bug record
66  Inline: $IV_SearchBugs, ‘Bug_ID’ = $vBug_ID;
67
68    // Creating Array of all fields and values
69    Var:’DBArray’ = (Array:
70      ‘Project_ID’=  (Field:’Project_ID’),
71      ‘Version_ID’=  (Field:’Version_ID’),
72      ‘ClosedInVers_ID’= (Field:’ClosedInVers_ID’),
73      ‘ToBeClosedVers_ID’=(Field:’ToBeClosedVers_ID’),
74      ‘Title’=      (Field:’Title’),
75      ‘Description’=   (Field:’Description’),
76      ‘Assigned_To’=   (Field:’Assigned_To’),
77      ‘URL’=       (Field:’URL’),
78      ‘Severity_ID’=   (Field:’Severity_ID’),
79      ‘Priority_ID’=   (Field:’Priority_ID’),
80      ‘Status_ID’=   (Field:’Status_ID’),
81      ‘Resolution_ID’= (Field:’Resolution_ID’),
82      ‘Browser_ID’=  (Field:’Browser_ID’),
83      ‘OS_ID’=     (Field:’OS_ID’));
84
85  /Inline;
86
87  // Sort the Array
88  $DBArray->(Sort:True);
89
90  // TESTING OVERRIDE
91  /*$DBArray = (array:
92    ‘Assigned_To’ = ‘XXXXXX’,
93    ‘Browser_ID’ = ‘5’,
94    ‘ClosedInVers_ID’ = ‘0’,
95    ‘Description’ = ‘Weasel can\’t log into system. Gets “no user found” message.’;
96    ‘OS_ID’ = ‘33’,
97    ‘Priority_ID’ = ‘2’,
98    ‘Project_ID’ = ‘fssG8EodNLqN’,
99    ‘Resolution_ID’ = ‘4’,
100   ‘Severity_ID’ = ‘4’,
101   ‘Status_ID’ = ‘6’,
102   ‘Title’ = ‘Weasel can\’t log in’,
103   ‘ToBeClosedVers_ID’ = ‘10’,
104   ‘URL’ = ‘/login.lasso’,
105   ‘Version_ID’ = ‘1’);
```

```
106  */
107  If: $svDebug == 'Y';
108    '<br>\n';
109    '109: <b>DBArray</b> is <b> ' ($DBArray->Size) '</b> elements<br>\n';
110    '109: <b>DBArray (sorted)</b> is ' $DBArray '<br>\n';
111  /If;
112
113  // Loop through the two arrays, comparing for differences
114  Loop:($FormArrayClean)->size;
115
116    If: $svDebug == 'Y';
117      '<br>\n';
118      '118: Loop_Count: ' (Loop_Count) '<br>\n';
119      '118: FormArrayClean Fieldname: ' ($FormArrayClean-> Get:(Loop_Count)-
         >first) '<br>\n';
120      '118: DBArray Fieldname: ' ($DBArray-> Get:(Loop_Count)->first) '<br>\n';
121    /If;
122
123    If: (($FormArrayClean->(Get:(Loop_Count))->First) != ($DBArray->(Get:(Loop_
       Count))->First));
124
125      // If fields don`t match, Set error 9001, Update Failed
126      Var:'vError' = '9001';
127      Var:'vOption' = 'Update Failed';
128
129      If: $svDebug == 'Y';
130        '<br>\r';
131        '<b>131: Array Names Out of Sync!</b><br>\n';
132        'Fieldname: ' ($FormArrayClean->(Get:(Loop_Count))->first) '<br>\n';
133        '     131: Old Name: ' ($FormArrayClean->(get:(Loop_
           Count))->first) '<br>\n';
134        '     131: New Name: ' ($DBArray->(get:(Loop_Count))-
           >first) '<br>\n';
135      /If;
136
137    // Here is where the business gets done
138    // If an action_param is different than the database, write out the change
       // record to the Changes table
139    Else;
140
141      If: (($FormArrayClean->(Get:(Loop_Count))->Second) != ($DBArray->(Get:
           (Loop_Count))->Second));
142
143        If: $svDebug == 'Y';
144          '<b>144: Changed Field Hit!</b><br>\n';
145          'Fieldname: <b>' ($FormArrayClean->Get:(Loop_Count)->first) '</b><br>';
146          'Value: <b>' ($FormArrayClean->Get:(Loop_Count)->second) '</b><br>';
147            '144: New Value: ' ($FormArrayClean->Get:(Loop_Count)->second) '<br>';
148            '144: Old Value: ' ($DBArray->Get:(Loop_Count)->second) '<br>';
149        /If;
150
151        Var:'SQLAddChangeTable' =
152          'INSERT INTO ' $svChangelogTable ' SET
153          Bug_ID="' $vBug_ID '",' '
154          wg_id="' $svUser_WGID '",' '
155          Fieldname="' ($FormArrayClean->Get:(Loop_Count)->first) '",
156          Old_Value="' ($DBArray->Get:(Loop_Count)->second) '",
157          New_Value="' ($FormArrayClean->Get:(Loop_Count)->second) '",
158          User_ID="' ($svUser_ID) '",
159          DateModified="' (Date_Format:(Date_GetCurrentDate),-DateFormat='%Y-%m-
             %d') '"';
```

```
160
161         If: $svDebug == 'Y';
162           '162: SQLAddChangeTable: ' $SQLAddChangeTable '<br>\n';
163         /If;
164
165         Inline: $IV_AddChangelog, -SQL=$SQLAddChangeTable;
166           If: (Error_CurrentError) != (Error_NoError);
167             Var:'vError' = '9001';
168             Var:'vOption"' = 'Writing changelog failed [Error 9001]';
169             '169: Error_CurrentError: ' (Error_CurrentError) '<br>\n';
170           Else;
171             '169: Error_CurrentError: ' (Error_CurrentError) '<br>\n';
172           /If;
173         /Inline;
174
175       Else;
176         If: $svDebug == 'Y';
177           '    177: Processed Field, No Change<br>\n';
178         /If;
179       /If;
180     /If;
181
182   /Loop;
183
184 // ============== COMMENT ========
185 Else: $vAction == 'Add Comment';
186
187   Var:'SQLAddChangeTableComment' =
188     'INSERT INTO ' $svChangelogTable ' SET
189     Bug_ID="' $vBug_ID '",' '
190     wg_id="' $svUser_WGID '",' '
191     Fieldname="Comment",
192     Old_Value="' $vNewComment_ID '",
193     New_Value="",
194     User_ID="' $svUser_ID '",
195     DateModified="' (Date_Format:(Date_GetCurrentDate),-DateFormat='%Y-%m-%d')
      '"';
196
197   Inline: $IV_AddChangelog, -SQL=$SQLAddChangeTableComment;
198     If: (Error_CurrentError) != (Error_NoError);
199       Var:'vError' = '9001';
200       Var:'vOption"' = 'Writing changelog failed [Error 9001]';
201     /If;
202   /Inline;
203
204 // ============== ATTACHMENT ========
205 Else: $vAction == 'Upload';
206
207   Var:'SQLAddChangeTableAttach' =
208     'INSERT INTO ' $svChangelogTable ' SET
209     Bug_ID="' $vBug_ID '",' '
210     wg_id="' $svUser_WGID '",' '
211     Fieldname="Attachment",
212     Old_Value="' $vNewAttach_ID '",
213     New_Value="",
214     User_ID="' $svUser_ID '",
215     DateModified="' (Date_Format:(Date_GetCurrentDate),-DateFormat='%Y-%m-%d')
      '"';
```

```
216
217   Inline: $IV_AddChangelog, -SQL=$SQLAddChangeTableAttach;
218     If: (Error_CurrentError) != (Error_NoError);
219       Var:'vError' = '9001';
220       Var:'vOption"' = 'Writing changelog failed [Error 9001]';
221     /If;
222   /Inline;
223 /If;
224
225 ?>
```

Changelog code - how it works

Add Action

1. Add a new record that says "New Issue" to the changelog (lines 16-33).

Update Action

1. Create a new array called FormArrayClean from the action_params contained in FormArray. Ignore any items beginning with "Update", "Action", "BugID", or "Process" (these come from hidden form fields, and are used as actions and not data). Also ignore anything beginning with a dash '-' is a Lasso operators. (line 46-54)
2. Sort the array. (line 57)
3. Get the existing data, and create an array called DBArray consisting of all fields you want to examine. Do not include any fields that are not submit by the form. (line 66-85)
4. Sort the array so the comparison is relevant. (line 88)
5. Compare the form array (FormArrayClean) with the database array (DBArray) to find differences. Check to make sure fields names match so you aren't getting incorrect comparisons. (line 114-182). Code at line 131 is used in debugging to see what is happening.
6. If there is a difference, write out a database records to the changelog table. (lines 141-173)

Add Comment Action

1. Add a record to the changelog indicating a new comment was made. Save the record ID of the comment table to changelog:old_value (lines 187-202). This is captured in bugs_addresponse.inc on line 235.
2. Redirect to bugs_editrecord.lasso with error (message)

Delete Action

1. Delete the main record. (bugs_addresponse.inc line 275)
2. Delete the attachments. (bugs_addresponse.inc line 283)
3. Delete the comments. (bugs_addresponse.inc line 287)
4. Delete the changelog entries. (bugs_addresponse.inc line 291)

Upload Action

Add a record to the changelog indicating a new attachment was made (lines 205-223). Save the record ID of the attachment table to changelog:old_value. This is captured from the process uploads routine (not provided here).

Coding tips

1. The siteconfig has a variable called "svDebug". If this is set to "Y", debugging messages will be output so you can examine the process flow.
2. The numbers in the debug message are simply line numbers, they are for reference only and can be changed.
3. Make sure the number of fields being examined in the DBArray are the same as the number of fields in FormArrayClean, otherwise you will possibly have a "Get" error when the routine tries to get an array element that does not exist.
4. Strip out all of the extra stuff and start with a simple example.
5. The variables vError and vOption are used with an error-reporting custom tag and database of errors/messages. Ask if you want a copy of that.

Supporting files

Structure

Attachments Table Structure.png - A screenshot of the attachments table structure
Changelog Table Structure.png - A screenshot of the changelog table structure
Comments Table Structure.png - A screenshot of the comments table structure
Issues Table Structure.png - A screenshot of the issues table structure

Data Displays

Changelog Table.png - Screenshot of the changelog table showing changelog entries
Comments Table.png - Screenshot of the comments table showing data
Issues Table.png - Screenshot of the issues table showing data
Issue 233 Changelog Entries.png - Screenshot of the changelog entries for the example

Schema

ERD.pdf - A PDF file showing the entity relationships between the tables
Changelog Table Schema.SQL - An SQL file that you can run to create the changelog table

Page Displays

LBT Home.png - Screenshot of the home page of the LBT application
Issue 233.png - Screenshot of the issue page for Issue #233

Lbt information

LBT is a commercial issue-tracking application developed by Landmann InterActive. It is written in the Lasso from LassoSoft. You can find out more about LBT and its issue-tracking and project management capabilities at this address:

<http://lbt.landmanninteractive.com/>

Chart FX Designer

Fletcher Sandbeck

LassoSoft

LassoSoft and Software FX have teamed up to bring enterprise class charting to Lasso. This unique partnership allows Lasso developers to take advantage of advanced charting capabilities on any platform which Lasso supports. The partnership was announced at last year's Lasso Summit and Chart FX for Lasso shipped as a for pay add-on to Lasso 8.5.

This paper presents some of the capabilities of Chart FX and demonstrates how those capabilities can enhance your Lasso-driven Web site. This paper also includes a new data type [ChartFX_Designer] written specifically for Lasso Summit. This data type makes it easier than ever to design graphically rich charts within Lasso.

Note - Complete documentation of Chart FX can be found in the Read Me.pdf file included in the Chart FX folder in each installation of Lasso.

Introduction

This paper introduces a new data type [ChartFX_Designer] which makes it easier to design and serve Chart FX charts within Lasso. The designer makes it possible to utilize many of the features of Chart FX using familiar LassoScript rather than the Java designer or hand-coded XML.

The files which are included with this paper should be installed in your Web server root. The file "Examples.lasso" generates a series of charts which demonstrate some of the features of Chart FX. Each individual chart is generated by a file contained within the Examples folder. The file "ChartFX_Designer.lasso" contains the [ChartFX_Designer] data type which can be used in your own files.

Terminology

Understanding the terminology used in Chart FX can help you see how the different elements of a chart work together. The following terms are used throughout this paper and the Chart FX documentation.

Gallery - The gallery is the type of chart. Examples of gallery types include bar, lines, pie, scatter, etc. Each chart may have a single gallery type for all data series or may have a different gallery type for each data series.

Series - Each sequence of values which are plotted on a chart is a data series. Each data series will be displayed using a single gallery type.

Axis - Many gallery types have an X-Axis and a Y-axis. The attributes of each axis include how values are labeled, whether tick marks or grid lines are shown, etc. A graph can also be modified

to show only a portion of data by scaling or setting the minimum or maximum values for an axis.

Label - Labels are text which identify the values of data points. Usually there are a series of labels on the X-Axis and a series of labels on the Y-Axis. It is also possible to add labels to individual data points or to constant lines on the chart.

Marker - A marker is a symbol which is placed on a line or scatter chart to represent a single data point. Markers can be dots, squares, triangles, circles, x marks, etc.

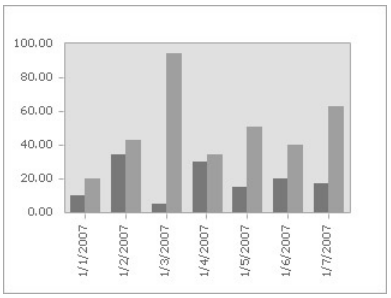
Legend - The legend is a small table which identifies the data series on a chart or the individual values which are plotted.

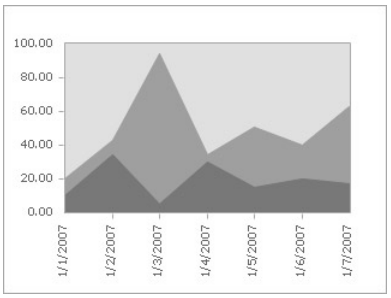
Gallery Types

Chart FX includes the following gallery types. Each type of chart is suited to presenting a different type of data. In addition, many of the charts can be customized or combined for particular situations.

Bar - This is one of the most common types of charts and is what Chart FX produces by default. One data series can be plotted, or several data series can be plotted side by side, or stacked. 3D options include rectangular or cylindrical solids.

Area - This is a line chart where the area under the line is filled in. CurveArea is a variation which is a filled in curve chart.

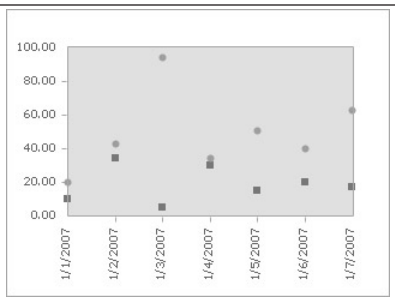


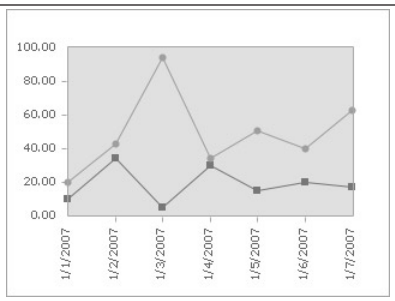


Scatter - Data points are drawn using markers. The marker for each data series can be controlled. A Cube chart is similar, but data points are plotted as a large square or cube. A Bubble chart plots data points as a variable sized circle or sphere where the size of the bubble is determined by a second data series.

Line - This is a chart where the values are connected by a line so that trends can be spotted in the data. Data points can optionally be drawn using markers. Multiple series can be plotted on the same 2D chart or side by side on a 3D chart. Variations include Step which has horizontal line segments connected by vertical line segments and Curve which has a continuous curve connecting individual points.

Pie - This is a circle which is divided into sectors representing fractions of a whole. Each data series represents an individual pie chart with each data point corresponding to a single pie piece.

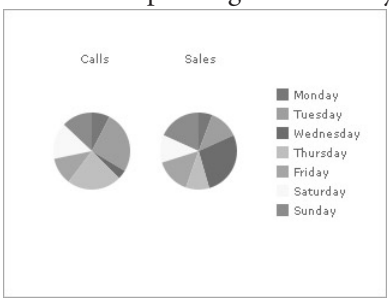


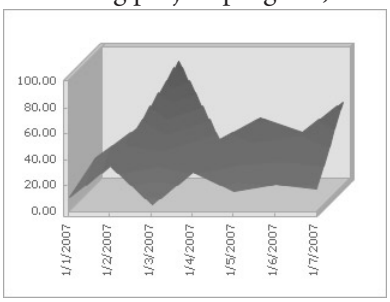


If multiple data series are plotted in a single chart then multiple pies are drawn. Variations include Donut and Pyramid.

Surface - A collection of data series are plotted, all joined by a 3D surface.

There are several additional chart types which can be used for particular purposes including Radar and Polar for plotting data radially, Gantt charts for tracking project progress, and





CandleStick, HiLowClose, OpenHiLowClose, and ParеTo for financial plots.

Chart FX Basics

Chart FX charts are typically designed using an iterative process. First, you select your gallery type and feed in the data that you want to plot. Then, you use the design tools to customize the chart until it appears exactly as you want it to be. Chart FX has intelligent defaults which are automatically configured based on the data which is fed into it. The built-in color schemes, marker selections, 3D settings, and labels usually produce a very nice looking chart.

Every chart is the combination of the data which is to be plotted and a template which defines how the data should be plotted. The [ChartFX_Designer] type represents a complete chart. The member tags of the type allow the data for the chart to be specified, the template for the chart to be designed, and for the completed chart to be output.

A new instance of [ChartFX_Designer] can be created using the following code. It is always recommended to specify a gallery type, width, and height when creating any chart. Data can be added to the chart using the [ChartFX_Designer->SetLassoData] tag. Various design tags such as [ChartFX_Designer->SetFont] can be used to customize the appearance of the chart. And, finally, [ChartFX_Designer->Serve] is used to serve the chart by setting the HTTP content type to image/png. The following tags represent the bare minimum required to serve a chart.

```
[Var: 'myChart' = ChartFX_Designer(-Gallery='Bar', -Width=320, -Height=240)]
[$myChart->SetLassoData($myData)]
[$myChart->SetFont(-Name='Verdana', -Size=7)]
[$myChart->Serve]
```

If the code above were put into a file called "Chart.lasso" then the chart could be displayed by placing the following HTML <img> tag in another file. The [ChartFX_Designer->Serve] tag serves the data of the generated PNG file in place of the "Chart.lasso" page.

```
<img src="Chart.lasso" width="320" height="240" />
```

The remainder of this paper elaborates on the capabilities of the [ChartFX_Designer] type so read on.

Note - The [ChartFX] data type can also be used directly to create and render charts. The [ChartFX] tag is discussed fully in the Read Me.pdf file included in the Chart FX folder in each installation of Lasso.

Data

Since we are using Lasso we are going to assume that the data for our chart is going to be produced using an inline. Chart FX expects to receives its data as a series of database records. Each record includes a label for the data point as the first field followed by one or more fields representing individual data series. The first record in the data contains labels for the individual data series.

The table below shows a collection of data which could be plotted on a chart. The first column includes dates which will form the labels for the X-Axis. The next two columns contain two data series.

```
Labels     Series One   Series Two
1/1/2007   10           20
1/2/2007   34           43
1/3/2007   5            94
1/4/2007   30           34
1/5/2007   15           51
1/6/2007   20           40
1/7/2007   17           63
```

The [ChartFX_Records] tag will format data from an inline in the proper way to feed into a ChartFX tag. For example, the data above might be found in an inline using this code.

```
[Inline(-Search,
  -Database='Marketing',
  -Table='Sales',
  -Op='gte', 'Date'='2007-1-1',
  -Op='lte', 'Date'='2007-1-7',
  -SortField='Date',
  -ReturnField='Date', -ReturnField='Calls', -ReturnField='Sales')]
[Var('myData' = ChartFX_Records)]
[/Inline]
```

For most of the examples in the remainder of this paper, we will use synthetic data. The following code generates exactly the same output as the code above after the hypothetical search has been performed. This code also shows how it is possible to hard-code data series in Lasso, or to programmatically create data series from other sources.

```
[Var: 'myData' = Array(Array( 'Labels', 'Calls', 'Sales'), Array('1/1/2007', 10, 20),
Array('1/2/2007', 34, 43), Array('1/3/2007', 5, 94), Array('1/4/2007', 30, 34), Array('1/
5/2007', 15, 51), Array('1/6/2007', 20, 40), Array('1/7/2007', 17, 63))]
```

The data is fed into the chart using [ChartFX_Designer->SetLassoData] as shown below.

```
[Var: 'myChart' = ChartFX_Designer(-Gallery='Bar', -Width=320, -Height=240)]
[$myChart->SetLassoData($myData)]
[$myChart->SetFont(-Name='Verdana', -Size=7)]
[$myChart->Serve]
```

Note - It is also possible to read in data from an XML file or from a tab or comma delimited text file. Some data also requires the use of the [ChartFX->SetDataType] tag in order to override Chart FX's built-in determination of what type of data has been passed in. These techniques are described in the Read Me.pdf file included in the Chart FX folder in each installation of Lasso.

Template

Most of the tags of [ChartFX_Designer] are used to set the parameters of the template for the chart. The template is an XML file whose format is described in the documentation provided by Software FX. The template generated by the type included with this tip can be output with the [ChartFX_Designer->Template] tag.

A minimal template is shown below. It contains settings for the X-Axis, the default font, the gallery type, and the height and width of the generated chart.

```
<?xml version="1.0" encoding="UTF-8"?>
<CFX6>
  <AXIS>
    <ITEM index="2">
      <LABELANGLE>90</LABELANGLE>
    </ITEM>
  </AXIS>
  <FONT>
    <NAME>Verdana</NAME>
    <SIZE>7</SIZE>
  </FONT>
  <GALLERY>bar</GALLERY>
  <HEIGHT>240</HEIGHT>
  <WIDTH>320</WIDTH>
</CFX6>
```

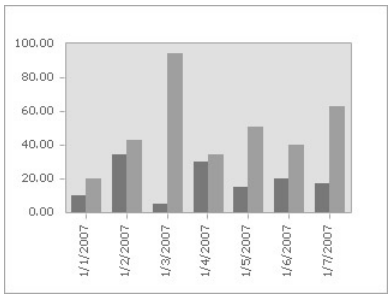
Note - It is also possible to read in templates from an XML file or to generate them using normal Lasso tags. The Chart FX for Java Designer is a graphic environment provided by Software FX which allows chart templates to be generated interactively. These techniques are described in the Read Me.pdf file included in the Chart FX folder in each installation of Lasso.

Serving and Rendering

Charts can be rendered and served using three different techniques. Each technique has its advantages.

Serving - The [ChartFX_Designer->Serve] tag will serve the chart represented by the data type in place of the current Lasso page. The tag will automatically set the HTTP content type, render the chart, and serve the data of the generated PNG file. Using this tag each chart should be designed and served by a single self-contained Lasso page.

```
[Var: 'myChart' = ChartFX_Designer(-Gallery='Bar', -Width=320, -Height=240)]
[$myChart->SetLassoData($myData)]
[$myChart->SetFont(-Name='Verdana', -Size=7)]
[$myChart->Serve]
```



Rendering - The [ChartFX_Designer->Render] tag can be used to render the chart as a PNG file in a temporary folder called chartfx62/temp located in the Web server root. The result of this tag is an HTML <img> tag containing the path to the temporary file.

```
[Var: 'myChart' = ChartFX_Designer(-Gallery='Bar', -Width=320, -Height=240)]
[$myChart->SetLassoData($myData)]
[$myChart->SetFont(-Name='Verdana', -Size=7)]
[$myChart->Render]
-> <img src="/chartfx62/temp/1234567890ABCDEF.png" />
```

Data - The raw data of the generated PNG for a chart can be found using [ChartFX_Designer->Data]. This method is best if you need to store charts in a database or otherwise manipulate them before serving them.

```
[Var: 'myChart' = ChartFX_Designer(-Gallery='Bar', -Width=320, -Height=240)]
[$myChart->SetLassoData($myData)]
[$myChart->SetFont(-Name='Verdana', -Size=7)]
[$myChart->Data]
```

```
-> ... Raw Data of the PNG ...
```

The serve method is used throughout the examples included with this paper. Although the render method seems more convenient, it has two drawbacks. First, a temporary file is created in the chartfx62/temp folder. These temporary files need to be periodically deleted and permissions

problems can make it difficult to generate these files on some systems. Second, the output of this tag is an HTML tag which needs to be parsed if any further processing of the chart is to be done.

Note - The serve and data methods are only supported in Lasso 8.5.2 and higher or Lasso 8.1.1 or higher.

Chart FX Designer

The [ChartFX_Designer] tag creates a new object which represents a Chart FX chart. The type supports all of the member tags of the [ChartFX] type as well as many additional tags which allow the template of the chart to be modified.

The parameters of the [ChartFX_Designer] define the basic parameters for the chart including its gallery type, palette, height, and width.

- -Gallery - Specifies the type of chart. -Gallery defaults to Bar and can also be set to Lines, Area, Scatter, Pie, Curve, Pareto, Step, HiLowClose, Surface, Radar, Cube, Doughnut, Pyramid, Bubble, OpenHiLowClose, Candlestick, Contour, CurveArea, or Gantt.
- -Height and -Width define the size of the chart. If the height and width are not specified then the chart will be 640x480.
- -Palette sets the color scheme for the chart. Chart FX includes several built-in palettes which make it easy to generate charts that fit in a range of different environments. It is also possible to control colors more specifically using the tags below. Palette options include Windows, EarthTones, ModernBusiness, Alternate, HighContrast, Pastels, Sky, Adventure, Mesa, or Vivid.

The default font for the chart can be set using -Name, -Size, and -Bold or -Italic. All labels and titles in the chart will use this font unless instructed otherwise. See the common parameters section below for a discussion of these font parameters.

Other options which can be set for the chart include:

- -Border can be specified to turn on a border around data objects within the chart. Options include -BorderColor and -BorderEffect with values of None, Raised, Light, Dark, Opposite, and Shadow. Note that this option differs from the whole chart border effect.
- -Cluster specifies that values from different series should be stacked rather than shown side by side. This primarily affects Bar charts, but may apply to other gallery types.
- -CylSides gives elements of a bar graph a rounded appearance if set to a high (>25) number. This primarily affects Bar charts, but may apply to other gallery types.
- -InsideColor controls the background color of the graph itself. This is generally defined by the palette choice, but can be overrided here if necessary.

Common Parameters

Some parameters are shared among several of the tags and have pre-defined values. These include:

-Name='Font Name' always specifies the name of a font, -Size specifies the size of the font in points, and -Bold or -Italic can be used to add that attribute to the font. Either -Color or -TextColor may be used to set the color of text depending on the tag.

-Color can be specified as an HTML color value like #ff0000. An additional pair specifies the transparency of the color from #ff000000 for transparent to #ff0000ff for solid. Colors can also be specified by name including White, Light Gray, Gray, Dark Gray, Black, Red, Pink, Orange, Yellow, Green, Magenta, Cyan, Blue, and Transparent.

-Alignment can have a value of Near, Center, Far, or Spread. Near usually specifies that the element should be aligned closer to the 0 point on the axis. Spread specifies that the element should be fit best into the layout. Different values will be the default depending on what is being aligned.

Line styles are set with -Width to specify the width of the line in pixels, -StartCap and -EndCap can have a value of Butt (default), Round, or Square, and -Style can have a value of Solid (default), Dash, DashDot, or DashDotDot.

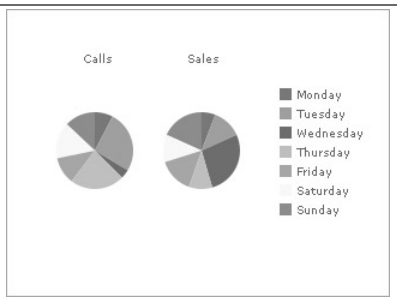
Name/Value parameters such as -Gallery='pie' can be passed to most tags including the creator. The attribute 'GALLERY' will be created with the value 'pie'. This allows attributes which are not anticipated by this tag set to be used. Sub-attributes can be added by including a map as the value of an attribute. Sub-attribute values can be specified as 'font'=(Map: 'name'='Verdana', 'size'=9), however most sub-attributes are handled specially by the individual tags of the type.

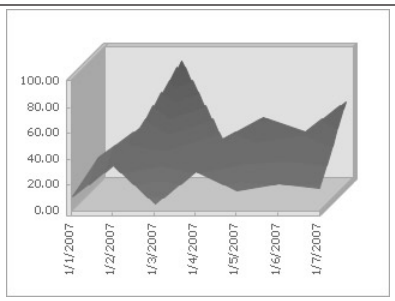
Titles and Legends

[ChartFX_Designer->(AddTitle: 'text')] adds a text title to the chart. Multiple titles can be added. The tag requires one parameter which is the text of the title. The tag accepts font parameters -Name, -Size, etc. The location of the title can be set with -Alignment, -Right, -Center, -Left, or -Top for position. The color of the title can be set with -Color and the background color with -BackColor. If -DrawingArea is specified then the title will be the width of the chart, otherwise it will be the full with of the background.

[ChartFX_Designer->(Legend)] shows the legend for the values in each series. Accepts -Name, -Size, -Bold, -Italic, and -TextColor parameters. -Internal specifies that the legend should be compact and -External specifies the legend should extend to the sides of the chart. The position of the legend is controlled by -Top, -Left, -Right, or -Bottom. -Color specifies the background color for the legend.

[ChartFX_Designer->(SeriesLegend)] shows the legend for the data series. Accepts -Name, -Size, -Bold, -Italic, and -TextColor parameters. -Internal specifies that the legend should be compact and -External specifies the legend should extend to the sides of the chart. The position of the legend is controlled by -Top, -Left, -Right, or -Bottom. -Color specifies the background color for the legend.

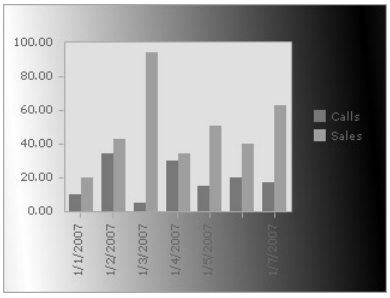




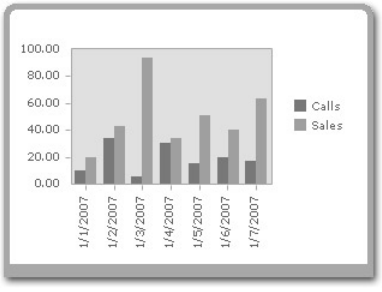
Borders and Backgrounds

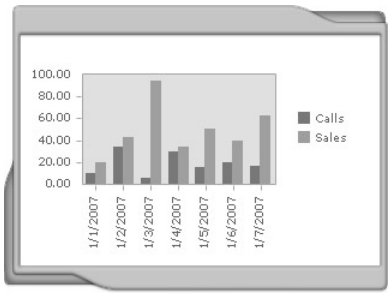
[ChartFX_Designer->(SetBackground: 'color')] sets the background of the chart to a solid color. This is generally defined by the palette choice, but can be overrided here if necessary.

[ChartFX_Designer->(SetBackground: 'color', 'color')] sets the background of the chart to a gradient. Also accepts parameter -Type with a value of Horizontal, Vertical, BackwardDiagonal, ForwardDiagonal, Radial, or Angled. For the last option the angle can be specified with -Angle.



[ChartFX_Designer->(SetBorder: 'type')] sets the border of the chart to the desired type. Built-in types include Inner, Outer, Flat, SunkenInner, RaisedInner, SunkenOuter, RaisedOuter, DarkOuter, and LightOuter. The tag also supports a number of image-based types including Embed, Emboss, Arch, ArchThick, ArchStroke, ArchStrokeThick, Solid, SolidRivet, SolidLines, SolidLinesRivet, Open, OpenRivet, OpenLines, OpenLinesRivet, Colonial, Slide, Rounded, RoundedCaption, RoundedShadow, RoundedCaptionShadow, Aqua, Butterfly, Gel, Museum, and Pulsar. Some of these types accept an optional -Color parameter.

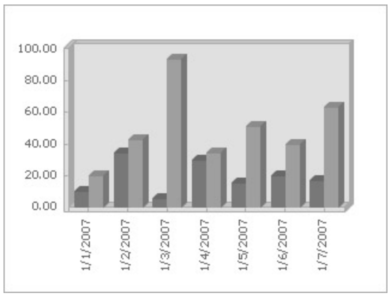


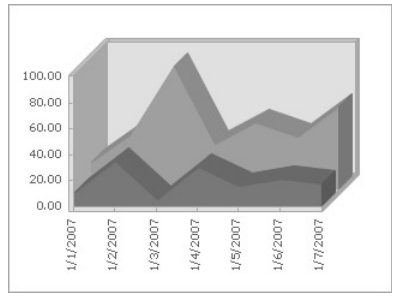


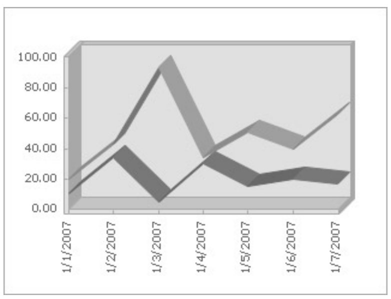
The Third Dimension

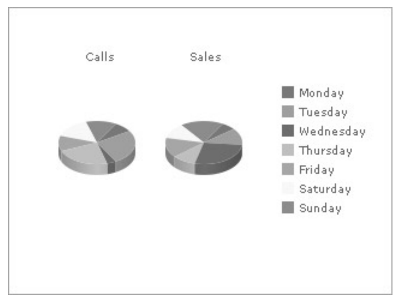
[ChartFX_Designer->(Set3D)] displays the chart in 3D. -View3D can be set to False if you want to render the chart in 3D, but within a flat frame. In addition, several parameter which are

specific to the 3D display of charts can be set here. -View3DLight can be set to Fixed or Realistic. -View3DDepth can be set to a percentage value to control the depth of 3D bar chart solids relative to their thickness. By default, -View3DDepth is set to 100%, a value of 200 will make the bars twice as deep as they are thick. -Perspective can be set to a value from 0 (the default) to 100 to change the ratio of the size of the front to the back of the chart. -AngleX and -AngleY can be set to rotate the chart. -WallWidth controls the width of the wall which surrounds the chart.









Series

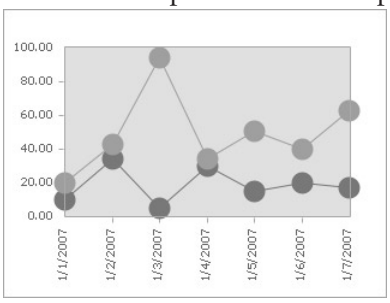
[ChartFX_Designer->(SetMarker: 'shape')] sets the desired marker for all series in line graphs (by default each series will use a different shape). The -Shape parameter can be one of None, Rect, Circle, Triangle, Diamond, Marble, HorizLine, VertLine, Cross, InvertedTriangle, X, or Many. Also accepts -Size and -Step parameters.

ChartFX will automatically use the chart preferences for each data series and will automatically determine the number of data series unless one of the tags here is used. For example, if the data contains three fields the first will map to the X axis and the second and third will be seen as data series. However, if only one series is defined using [ChartFX_Designer->Series] then only the first data series will be rendered.

[ChartFX_Designer->(Series: 1)] sets general preferences for the specified data series. -Color assigns a specific color to the data series. -Gallery allows the data series to be drawn using a different chart type from other data series. -Stacked controls whether the data series is stacked with other series or not. -Visible=false can be used to hide a data series. -LineStyle and -LineWidth can be set.

[ChartFX_Designer->(SeriesLabel: 1)] turns on the point labels for the specified data series. Options include -Angle, -Color, and the font options -Name, -Size, -Bold, and -Italic.

[ChartFX_Designer->(SeriesMarker: 1, 'shape')] sets the desired marker for the specified data series in line graphs. The -Shape parameter can be one of None, Rect, Circle, Triangle, Diamond, Marble, HorizLine, VertLine, Cross, InvertedTriangle, X, or Many. The -Color of the marker can be specified. Also accepts -Size and -Step parameters.



Axis

These tags set the appearance of the axes for the chart. Each tag requires the name of an axis as its first parameter. The axis will usually be "x" or "y", but may also be "x2" or "y2" for certain chart types. The -Step parameter value is shared for each axis so setting it in any tag will affect the value for all tags. The "minor" tags also share a -Step value.

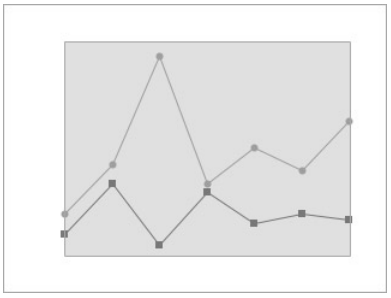
[ChartFX_Designer->(Axis: 'x')] sets basic parameters for an axis. Common parameters include -Max, -Min, -Inverted, -ScaleUnit, -Staggered, -Step, -ForceZero, and -LogBase. If -Visible=False is specified then the axis, its labels, and title are all hidden.

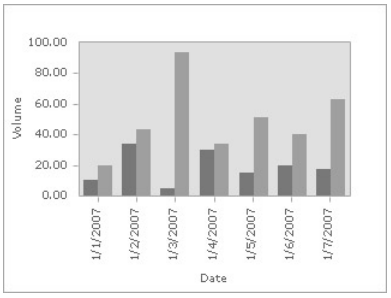
[ChartFX_Designer->(AxisLabel: 'x')] sets options for the automatically generated labels of the axis. -Step controls how often a label is shown. -Angle controls the angle at which the labels are drawn. The font of labels can be controlled with -Name, -Size, -Bold, and -Italic. -Color controls the label color. -Position controls where labels are drawn relative to the axis. -Format can be set to None, Number, Currency, Scientific, Percentage, Date, LongDate, Time, or DateTime and -Decimals controls how many places past the decimal point will be shown. -First allows the first desired label to be specified. -ScaleUnit allows the entire axis to be divided by the specified number and -LabelValue allows the entire axis to be multiplied by the specified number.

[ChartFX_Designer->(AxisTitle: 'x', 'Title')] displays a title by an axis. The font and color can be controlled with -Name, -Size, -Bold, -Italic, and -Color. The -Position of the title can also be specified.

[ChartFX_Designer->(AxisLine: 'x')] sets preferences for the axis line itself. Options include -Color, -EndCap, -StartCap, -Style, -Width.

[ChartFX_Designer->(AxisStripe: 'x', 'color')] allows automatic striping of the data. The background will alternate between the inner color set in the constructor and the -Color specified here. Also accepts the common -Step parameter.

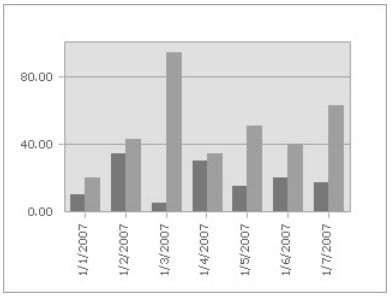


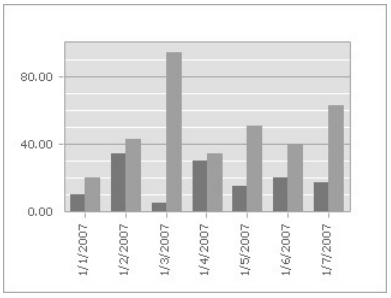


[ChartFX_Designer->(AxisGrid: 'x')] displays a grid from the axis across the chart. Options include -Color, -EndCap, -StartCap, -Style, and -Width. Also accepts the common -Step parameter.

[ChartFX_Designer->(AxisTick: 'x')] displays a tick on each step. The -Tick parameter accepts options None, Outside, Cross, and Inside. Also accepts the common -Step parameter.

[ChartFX_Designer->(AxisMinorGrid: 'x')] displays a grid based on the minor step from the axis across the chart. Options include -Color, -EndCap, -StartCap, -Style, and -Width. Also accepts the common minor -Step parameter.



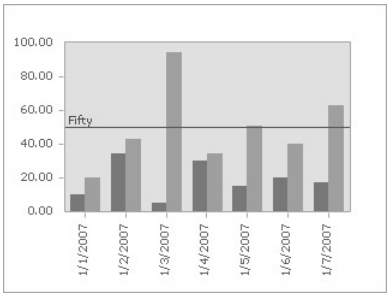


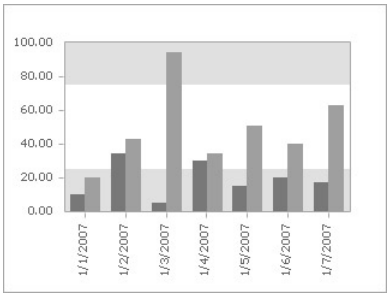
[ChartFX_Designer->(AxisMinorTick: 'x')] displays a tick on each minor step. The -Tick parameter accepts options None, Outside, Cross, and Inside. Also accepts the common minor -Step parameter.

Highlights

[ChartFX_Designer->(AddConstantLine: 'x', '3', 'text')] adds a constant line to the chart. Multiple constant lines can be added. Each line is drawn across the chart from either the X or Y axis. The tag requires two parameters which is the axis for the line and the value at which the line should be drawn. The tag also accepts a -Text parameter to define a title, -TextColor, and font parameters. The location of the title can be set with -Alignment, -LineAlignment, and -OutsideText to draw text in the axis rather than on the chart. The line style can be set with -Color, -StartCap, -EndCap, -Width, and -Style.

[ChartFX_Designer->(AddStripe: 'x', -From=3, -To=5)] adds a stripe to the graph extending from the specified axis. The strip will go from the -From value to the -To value and can have an optional -Color.





Utility Tags

[ChartFX_Designer->(Set: -Name='value')] is a utility tag which accepts the same types of parameters as the creator, but either sets or overrides the attribute within the Chart FX template. Multiple attributes can be set with this tag. If an attribute is given a new value then the new value completely replaces the old value (sub-attributes will not be merged).

Managing Common Code

Jonathan Guthrie

Introduction

There are many different options for implementing common code, and this paper looks at three of these - the use of cTags and cTypes for sharing of common routines and tasks between sites or web applications; the use of version control functionality for new iterations of a site or application that require more than simple customization; and a technique using define_atBegin and file_exists to streamline the decision about files to process.

cTags and cTypes!

The most useful advice for any developer is that if you aren't already using Custom Tags and Types routinely - you need to.

Jason Huck of Core Five Creative stated: *"A custom tag is Lasso's version of a function. It's a way of encapsulating code with standard input and output to make it more portable and easier to re-use."*

The general rule is that if the same code is needed in more than one place then it could probably be abstracted and stored so that it is called in a tag and has the specific parameters passed to it.

An example of this functionality would be where there is an error reporting code block that captures any database or file error and appends it to a variable to output it in a controlled manner at the foot of the page. This is demonstrated below:

```
<?LassoScript
inline($gv_sql,-SQL='SELECT * FROM mytable WHERE a = ');
  // ------------------------------------------------
  if((Error_CurrentError(-ErrorCode)));
      $gv_error += (Error_CurrentError) + '<br />';
    else((File_CurrentError(-ErrorCode)));
      $gv_error += (File_CurrentError) + '<br />';
    /if;
    // ------------------------------------------------
    records;
      // …
    /records;
/inline;
?>
```

Spot the mistake… it will throw an error in MySQL because the comparison value is missing in the statement. That will append the error returned from MySQL. It's unwieldy to put that code block in every place needed to trap for errors. If there are 50 SQL statements per page executed that's a whole lot of additional code!

It makes sense to slot it into a Custom Tag.

```
<?LassoScript
  define_tag('isError',
    -namespace='xs_',
    -priority='replace');
    // -------------------------------------------------
    if((Error_CurrentError(-ErrorCode)));
      $gv_error += (Error_CurrentError) + '<br />';
    else((File_CurrentError(-ErrorCode)));
      $gv_error += (File_CurrentError) + '<br />';
    /if;
    // -------------------------------------------------
  /define_tag;
?>
```

So now we can reduce the previous code substantially!

```
<?LassoScript
inline($gv_sql,-SQL='SELECT * FROM mytable WHERE a = ');
    xs_iserror;
    records;
      // …
    /records;
/inline;
?>
```

There are several ways to load Custom Tags and Types.

- Include the tag in the page, in which case it is not shared with any other page;
- Save in a separate file and load it when you think you may need access to it, in a standard include or a library tag, remembering that a library returns no output;
- Load a separate file as part of every page load by including a common setup file at the head;
- Save the file in LassoStartup for the site instance;
- Save the file in LassoStartup for the server.

The last two options involve less overhead for each page called, and the tags/types are available to the specific site or the entire server respectively. However, if you change them, you have to restart the site (using SiteAdmin or ServerAdmin LassoApps) or the Lasso service (sudo lasso8ctl restart on OS X and Linux).

It makes sense to share a tag like the above error reporting tag globally, but many others would be site specific – the option chosen for each specific tag will depend on the circumstances foreseen for it's usage.

cTags and cTypes have several advantages:

- They can be defined all in one place
- Use less code overall
- Write once, use over and over again
- Easier maintenance due to less places that require the same code
- Make it easy to assemble a common library for sharing between projects

- Plenty of tags & types are available from numerous authors and collated on sites such as tagswap.net so you get the benefit of others experience and shortcuts

*c*Tags and *c*Types have few disadvantages:

- Remember to restart the site after changes if stored in LassoStartup
- Be very careful and consistent in your naming of tags, types and their parameters
- Code must be written to cope with all possibilities that may occur – not just the scenario that led to the tag being written
- Your code will no longer be able to read as a novel by some future paleontologist

Subversion

Version (or source) control is essential to development on many levels. For the purpose of this paper when SVN is referred to the reader can substitute any one of a number of different source control systems. Each has their own specific features, terminology and quirks.

In the authors opinion, SVN is an essential tool for any developer. SVN can save endless grief from the likes of an accidentally deleted file or a portion of a file, corruption, or a plain stupid mistake.

Another tool enable through version control is in the use of branching and tagging. Branching allows you to take a snapshot of a project at a particular point and continue development on both tracks. It's commonplace to call the main directory the "trunk" and the snapshot the "branch".

Now imagine a scenario common to many development companies. One staff member "Harry" is tasked at fixing bugs, and he's working on the trunk and committing those changes there. Another staff member "Sally" has been asked to spearhead the next point-iteration of their product and she needs to be able to have her own sandpit to make a mess of things without causing problems for the work that Harry is doing on squashing bugs and pushing them out to the main product.

At some point Sally will want to incorporate Harry's bug fixes into her branch. If version control was not being used Sally would need to duplicate the trunk's directory, and then run a file comparison on each directory and file to see the differences. This is painful, time consuming and it is also possible to miss things. If source control tools are used Sally can run a "merge" task which will look at the two and either automatically or manually bring them into sync (usually in one direction only). It means that Sally can take advantage of Harry's fixes without having to get him to patch both trunk and her branch at the same time.

When it comes time to rubber stamp Sally's branch as the next point release she can do a merge INTO the trunk and that can get pushed out to the server(s). It's a good idea to also "tag" each release – this means that a given set of files in a given state are logged as a 5.01 release for example.

It is essential to note that most source control systems will store things on the source control server not as physical files, but as a database. They usually store only what's changed, not a

complete copy of each file in each state, so the storage overhead's much lower than if you were trying to keep a copy of all your files at certain milestones.

SVN and similar systems can also assist managing the server side, for both production and development servers.

A production server should only be updated with stable code, and yet the latest version of your work should be available for the client (and dev team) to see on the development server. However the developer also needs a sandpit to play in, typically this will be on a desktop or laptop. Where SVN is not being used it is essential to know what has been changed and to synchronize that up to the specific server. With SVN involved simply running an update command specifying the repository location it will complete this. SVN will work out what needs to be added, deleted, moved, and changed, and just make it happen. This will be done for any media located in that repository – code, PDF, images, and more.

This gives the power to update each development server easily. Updating each production server can now be done without fear and trepidation (assuming a thoroughly tested release of course!). What is even more fantastic is that all developers in the team have access to all your contributions and vice versa! If one developer changes a file already worked on and committed, SVN will tell them and they can selectively merge their changes in with those committed by another.

SVN advantages:

- easier maintenance as you can use discrete places for development versus maintenance
- handles the labor intensive merge tasks
- ability to roll back to earlier versions of code
- perfect for multi developer as well as single developer environments
- the commit comments are likely to be informative as to what has been changed

SVN disadvantages

- requires a little more organization up-front
- developer discipline is needed for change comments to be useful

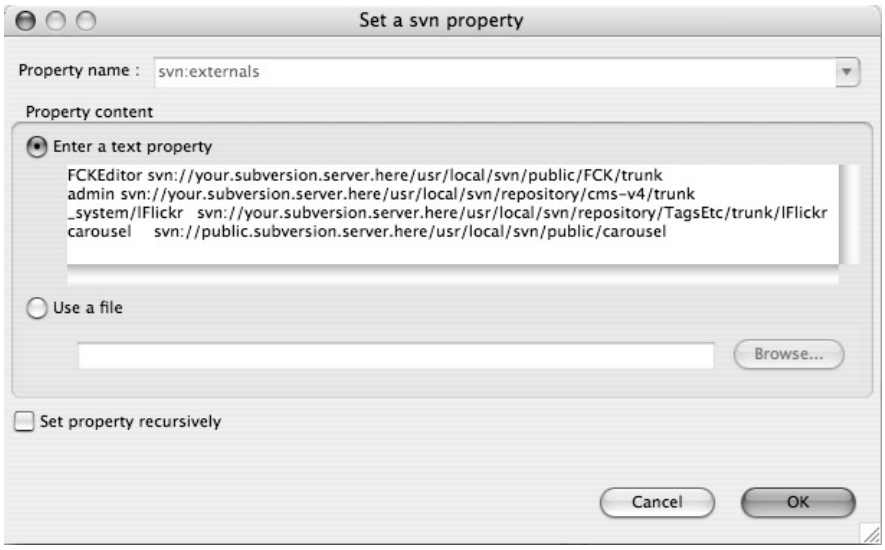
SVN Externals

Prior to discovering SVN Externals, Apache aliases were commonplace for me. I'd have one version of FCKEditor on my server and I'd have an Apache alias to it on every site that needed GUI editing features. The same went for my AJAX libs, Lasso Flickr API's, and a number of other things that I commonly use.

Now if you're not planning on implementing SVN, that might be a valid tactic, and it's one that I wasn't really unhappy with, it was just not particularly convenient.

So roll on the SVN Externals property. As you can see from the following screenshot, Subclipse (an Eclipse SVN plugin) allows you to set the property with ease. In this case the property is set "on" the root of the project, and the directories are specified relative to that location. For

example, FCKEditor is a directory we are placing at the root, and svn://your.subversion.server.here/usr/local/svn/public/FCK/trunk is the URL of the repository it's getting the data from.



Now it is important to note that the SVN Externals command will bring down an entire copy of those directories into your local drive, but when you go to your server and do a new checkout, it will remember the SVN Externals command and download those as well!

No more having to try to remember just what aliases you need to set in your apache virtual hosts file, SVN will simplify it all for you.

Even better, if you have write access to those repository locations in SVN, if you modify one of those files in there on your development machine, and commit that change, it will be reflected in ALL places you have used that repository next time you run the SVN update.

Note also that in the above example I am referencing different repositories – a standard one (read and write protected) and one that is essentially public and cannot be written to anonymously.

A sleight of hand…

atBegin, file_exists and redefining how we include files.

What IS "atBegin"?

According to the Lasso 8.5 Language Guide, define_atBegin "defines a pre-process action. [...] Usually called in a startup item. Requires one parameter which is an invokable object." Define_atEnd follows the same pattern as atBegin but it's a post-process action and we'll not be covering that here.

What does the above statement really mean? It says that we can do some Lasso stuff BEFORE the page gets processed with Lasso.

Generally the user decides they want an object, and then they click on a link in a browser. The browser then sends a request to the webserver (including cookies in that header if there are any

that belong to the domain). The webserver accepts the request and looks to see how it's suppose to be handled – in the case of .lasso pages, having Lasso installed we're going to see the webserver hand off the processing of the object to the Lasso engine. Instead of Lasso calling the page straight up, it looks for any define_atBegin directive specified in the LassoStartup directory for that site, and executes it. This define_atBegin could simply be setting up a page timer, a logging event, or something more complex that can change the resulting code Lasso processes entirely.

The manual warns that: *"Pre-process actions must be thoroughly debugged before they are activated on a live server. A syntax error in a pre-process action can cause every page on a server to fail."* And they're 100% right!

In the LassoStartup directory if we were to place a file with this code in it, and then restart the site:

```
<?LassoScript
  define_atbegin(
    {
      hello world
      var('x' = 1);
    }
  );
?>
```

EVERY page now invoking Lasso will return the error:

"No tag, type or constant was defined under the name "hello world var" with arguments: array: (pair: (x)=(1)) at: _at_begin"

Debugging this will be inconvenient if the site has to be restarted every time.

One way around this is as follows:

```
<?LassoScript
  define_atbegin(
    {
      if(file_exists('/_config/atbegin.lasso'));
        include('/_config/atbegin.lasso');
      /if;
    }
  );
?>
```

The advantage of this is the ability to have different Apache sites going through this same Lasso site instance, and each one will use the appropriate /_config/ directory to look for atbegin.lasso!

It is therefore optional as to whether you in fact use this file – if exists it be incorporated and if not, passed over. A new Lasso site instance does not have to be set up just for that website.

What's more important is that you can make changes to (aka debug, evolve, mutate) the actual atbegin code without having to restart the lasso site each time.

Define_atBegin seems like a nice trick but what does it have to do with managing common code?

Imagine a situation where you have a single "core" engine, and in an ideal world all clients you had signing up for your application were perfectly happy with what you presented to them with no customization whatsoever. All you would have to do is have all those domains pointing to the single core and maybe point the core to access a different database depending on what URL was requested. Simple.

Doh! We don't live in an ideal world. Clients always want something different. They want a different set of features activated. They want a different set of columns in the products table. They want to do things "this" way because they are the client. And of course, they will want to have their own skin on the engine, their own brand, after all they are the ones paying the bill, right?

So you have a few choices:

a) Duplicate the directory and work on 'A. Client's' new site.

b) Create a branch of ALL the data specifically for 'A. Client'.

The difficulty for both is STILL the "pushing out bug fixes to the core to every iteration". Option a) is a nightmare scenario and b) while better is still painful.

Remember the previous mention about how SVN keeps a record only of what's changed? I think it's appropriate to take a leaf out of that book.

What if we have our core and only put into the client site what's changed? If the only thing that is different in the admin between the core and 'A. Client's' site is the file product.lasso then why not store ONLY product.lasso? Then if a change is made to forum.lasso in the core it will automagically be seen in 'A. Client's' site.

Lets look at the code necessary to achieve this:

```
<?LassoScript
if(file_exists('/path/to/file.lasso'));
  $__HTML_REPLY__ = include('/path/to/file.lasso');
  abort;
else(file_exists('///CORE/path/to/file.lasso'));
  $__HTML_REPLY__ = process(file_read('///CORE/path/to/file.lasso'));
  abort;
else;
   $__HTML_REPLY__ = 'File does not exist please try again';
  abort;
/if;
?>
```

First of all determine if the local file exists, if it does the local file is included.

Second, if the local file does not exist we determine if that file exists in the core. If it does, use that.

If the file does not exist in the core then it's likely to be a wrong URL and so the error report is displayed to the user in our flavor of choice.

FYI The above code would theoretically sit in /_config/atbegin.lasso as per the code we've put in LassoStartup.

(Thanks to Adam Randall for the process/file_read solution!)

As part of my ongoing quest to separate the core from the client specific data, I have a standard site_config file that specifies site-specific settings. For us this always lives at /_config/site_config.lasso.

In this case the relevant lines in this file are:

```
<?LassoScript
  var('xsgv_core' = '///Library/WebServer/SiteFiles/CMS_v4.1_CORE');
  var('gv_sql' = array(
    -Database='iamadatabase',
    -Username='iamauser',
    -Password='iamapassword',
    -maxRecords='all'
    )
  );
?>
```

The define_atBegin code can now be extended a little further:

```
<?LassoScript
  local('exclarray' = (:'admin','xml','FCKEditor'));
  local('rfp' = Response_FilePath->lowercase&);
  local('path' = Response_FilePath->split('/'));
  (#path->first == '') ? #path->removefirst;
  local('path_called' = string(Response_FilePath));
  include('/_config/site_config.lasso');
  inline(var('gv_sql'),Action_Params);
    if(#rfp >> '.html');
      redirect_url('/');
    else(file_exists(#path_called));
      $__HTML_REPLY__ = include(#path_called);
    else(file_exists($xsgv_core+#path_called));
      $__HTML_REPLY__ = process(file_read($xsgv_core+#path_called));
    else(
      !(#exclarray >> #path->first) &&
      !(#rfp >> '.lassoapp') &&
      !(#rfp >> '.html')
    );
      $__HTML_REPLY__ = xs_inc('/index.lasso');
      abort;
    else;
      $__HTML_REPLY__ = #rfp;
    /if;
  /inline;
  abort;
?>
```

To explain:

- #exclarray is an array of directories we may not want sent to the "hub". The hub in this case is the place we reinterpret the URL and determine content based on this.
- #rfp, #path and #path_called are slightly different treatments of "response_filepath". #rfp and #path_called are different because #rfp is standardized to lower case and #path_called stays case sensitive due to potential for case sensitive file systems.
- Then include /_config/site_config.lasso - as explained previously this gives us specific settings we rely on.
- Next comes the inline container: This is required as files outside of root are being inspected, and it's not a good idea to give "ANY USER" permissions to access files outside of root, in any circumstance!
- Note the presence of the "Action_params" in the inline container. This makes sure our GET and POST can get through. If this is left out no form will ever get processed, and no param can ever be passed in the URL!
- In the above example there is a wholesale redirection of ALL .html files to the site root. Mix your own treatment of file extensions as you prefer. One site I converted from PHP had all .php extension pages passing to lasso, and in this spot I redirected to the equivalent .lasso page simply by replacing .php with .lasso and redirecting.
- Next is the local and core investigation as explained previously.
- Failing physical files existing in local or core, we check the #exclarray and make sure it's not a LassoApp or .html – if not then direct traffic to the hub file, /index.lasso.
- Last of all, we're assuming there may be a reason to just include the file as requested.

One point I have just glossed over is the cTag as used in this line:

```
$__HTML_REPLY__ = xs_inc('/index.lasso');
```

The xs_inc is very similar to what has just been done. It is a special case replacement for the Lasso standard include tag.

xs_inc defines a replacement from standard include tag. It determines if the requested file exists: if it does then the functionality is identical to "include". If not, it will test for the existence of the core version and include that if exists.

The reason for xs_inc is simple: we need a mechanism for carrying on our local->core crusade even for included files!

So lets have a look at xs_inc:

```
define_tag('inc',
  -namespace = 'xs_',
  -description = '
    Defines a replacement from standard include tag.
    Determines if the file requested exists:
      if it does then the functionality is identical to "include"
      if not, it will test for the existence of the core version
        and include that if exists.
    Requires that either $xsgv_core is set, or pass -core as the SECOND param
    xsgv == xServe Global Variable!
    ');
  inline(var('gv_sql'),Action_Params);
    local(
      'includeme' = null,
      'corevar' = null,
      'req' = null
      );
    if(params->size == 0);
      /* GRACEFULLY degrade */
      return('');
    else(params->size == 1);
      /*
        assume file requested is the only param passed
        just like std include
      */
      local('req' = string(params->get(1)));
      if(var_defined('xsgv_core'));
        /*
          set corevar = top level core
        */
          #corevar = $xsgv_core;
      else;
        /*
          standard file included because no core has been defined.
          no vars need to be set here due to graceful degrade below
        */
      /if;
    else(params->size == 2 && local_defined('core'));
      /*
        User defined core has been requested
      */
      local('req' = string(params->get(1)));
      #corevar = #core;
    /if;
    /* test for local exists, if not, use core */
    if(file_exists(#req));
      /* LOCAL exists so use it */
      #includeme = #req;
    else;
      /* use CORE */
      #corevar->removetrailing('/');
      !(#req->beginswith('/')) ? #req = Response_Path + #req;
      #req->removeLeading('/');
      #includeme = #corevar'/'#req;
    /if;
    /*
      include file contents. protect makes sure it silently degrades
      if you wish to report errors from your include you MUST use vars
    */
    protect;
      return(process(file_read(#includeme)));
    /protect;
  /inline;
/define_tag;
```

The in-code comments hopefully should explain appropriately rather than providing an additional blow-by-blow account.

In all code in following pages, xs_inc is used as a replacement for include.

This method's not perfect, there are one or two disadvantages.

"Scope" is one area that initially caught me out.

Local variables (#myvar) are limited in scope – if you use a local var inside a Custom Tag or Type then it's divorced from any higher level and similarly from any tag or type invoked from within. "Page" vars ($myvar) will be accessible to nested tags and types, as with Global variables.

When you do a simple "include" you can access local variables from the parent, but it turns out that when you use process(file_read('myfile.lasso')) its like calling a tag or type – while page and global vars are accessible, locals are not!

Another drawback is error reporting. It becomes more and more important to handle errors well, as simply putting protect containers around code will stop the app exploding but won't help you work out why you're getting blank data in your default template!

So lets look at a comparison of an actual example.

In theory, in an extreme case there could be NO lasso files in the local. The only things different between local and core might end up being CSS and image files, and if you're not sending .css and .js files to Lasso for processing you need to include them in the local directory anyway.

Other good practices

It's important to realize that your memory is NOT infallible – neither is life so certain that we can be sure that we will be still on a project in a years' time let alone the "hit by a bus" scenario!

A conversation with a friend over a BBQ brought some of this into focus. He's a specialist in process management and disaster recovery planning, currently working for one of our country's energy transmission companies. He told me about their Disaster Recovery Plans (DRP) and how they trial them on a regular basis. To do this and retain institutional knowledge they have a great system of documentation and review, without crossing the line into bureaucratic nightmare.

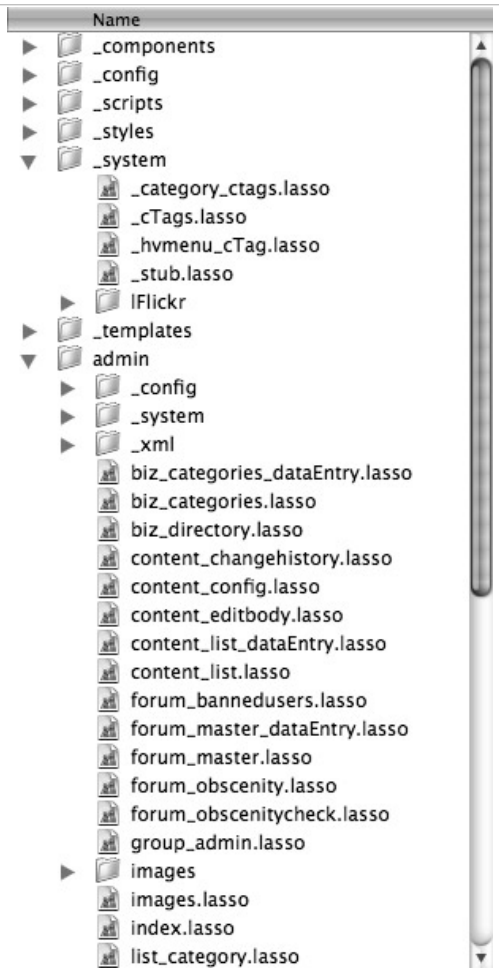
While you could argue that most of what we do is not life-or-death, the same principles should apply to us. We need to document our methods, policies, processes and state of mind!

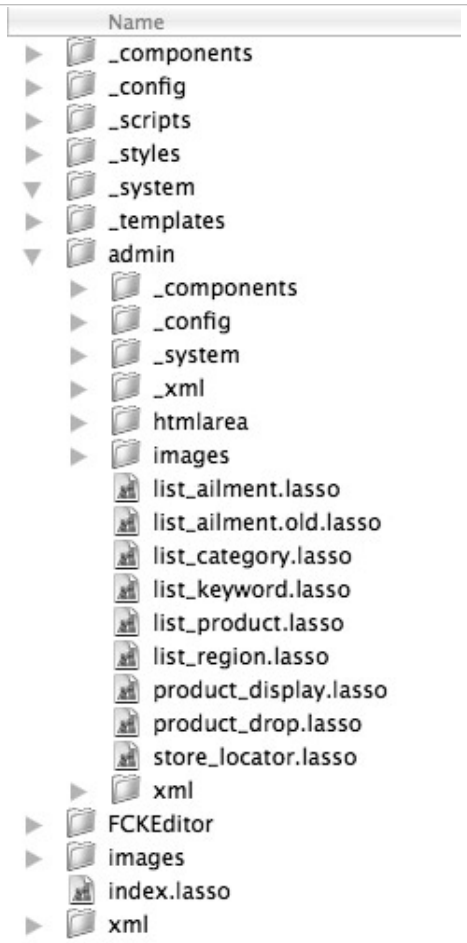
In our quotes and estimates to our clients we must not neglect this component. While it will increase the cost of the project, it also will enhance your professional image to the prospective client as well as potentially prove vital to you both down the road.

Change notes

With each 'release' of your project, it's important to note the changes in the code since last release. Note bug fixes, resolved issues and unresolved issues.

What a "core" might look like:	What a "local" might look like:
This is not a complete list, and only the admin and _system directories have been expanded.	Note the lack of any files in the local _system directory – this means that no lasso files are different between local and core, and core versions will be used. The admin directory is depleted almost to the same degree. Only those files specifically different are included.





Documentation

Explain the structural philosophy for your code layout, the thinking behind the way you are putting the pieces together in the way you are. (It may make perfect sense to you while you are writing it, but time has a way of degrading memory.)

Document how you're controlling your source and versioning systems, how you're expecting to get changes to the staging servers, production servers. Most importantly document the usernames and passwords you use to access all of these. If you're worried about the passwords being "known" then at least provide a method in your documentation as to how to obtain them in a crisis.

Commenting your code should become second nature – even in parts where it seems that it should be commonsense. The way another developer thinks will be different from you, and the difference may be enough that they're going to need your help!

Conclusion

The mix of techniques that will work for you and your organization is going to be different depending on your specific needs and circumstances. Hopefully the discussion and comments in this paper provide a useful direction for making your decisions, and enabling efficient and effective code for multi-site management.

Examples Of the Use of AJAX In Lasso-Based Solutions

Nikolaj de Fine Licht

1.0 Scope

The scope of this roundtable is to provide some practical and - hopefully - ready-to-implement code examples of some common uses of AJAX in a Lasso-context. The term "practical" should be emphasized, the scope is not an in-depth or a theoretical/philosophical discussion of AJAX. Many readers of this manual will be people with more advanced skills than mine, both with regards to AJAX/Javascript and to Lasso, but I would like to share what I have in the hope it can be of some use for people who are where I was about a year ago: wanting to implement some AJAX-tricks but quite unable to find some plain code examples of typical usages, in particular when combining with Lasso.

1.1 Aknowledgements

I wouldn't have gotten anywhere with this if it wasn't for Fletcher Sandbeck from LassoSoft and his "Tips of the Week" and help, and for Paul Melia and Deco Rior from the Lasso community and their code examples and help. Also thanks to Geoff Attrill for a useful tip and to all the other people of LassoTalk for always being there ready to share their knowledge. Thanks to Huberto G. Mayer mayer.art.br/beto for help with the javascripts.

1.2 AJAX And LJAX

AJAX (Asynchronous javascript and XML) describes both an approach to - or a philisophy about - webdesign in general and a technical solution to a number of more specific interface-design problems. Or in other words: a solution can be laid out with AJAX-principles from the very beginning, or it can be a "traditional" solution with a few interface details done with AJAX-technology - and it can be anything in between.

LJAX (Lasso, javascript, and XHTML) is Lasso's implementation of AJAX. LJAX builds on AJAX-philosophy and -code, and you can do largely the same things with both approaches, and also use them in conjunction.

1.3 The DOM

Anything "AJAX" means manipulation the DOM (Document Object Model) - changing the markup and content of a webpage - AFTER is has loaded. Meaning you can't see the changed code by viewing the page's source code! This is bad. Therefore you need tools to see what is actually being done to the DOM, and the best seems to be working with Firefox with these extensions installed:

- FireBug www.joehewitt.com/software/firebug
- Web Developer www.chrispederick.com/work/webdeveloper
- DOM Inspector www.mozilla.org/projects/inspector

1.4 OK, but what does this AJAX-thing mean 100% practically speaking?

For an introduction to the basic problem in webdesign which AJAX offers a solution to please look at Fletcher Sandbeck's introduction to the XMLHttpRequest in Tip of the Week of June 17 2005 http://www.omnipilot.com/Tip%20of%20the%20Week.1768.8801.lasso. He writes, among others: "XMLHttpRequest allows a new workflow. An HTML page can still have anchor tags and forms. But, when a link is visited or a form submitted it triggers a javascript function that replaces only the contents of the page with new content, leaving the surrounding navigation elements intact".

To a certain degree, this is all there is to it: using the socalled XMLHttpRequest and javascript its possible to refresh only a portion of a webpage upon a user interaction without having to reload the entire page. But the rapid development of the use of this approach and technique has lead to the creation of some publically accessible libraries or collections of javascript code, with which its possible to do quite much more than just refreshing portions of content.

As can be seen: javascript-disabled browser - no AJAX. There are other usability problems connected with AJAX, but its outside the scope of this paper to discuss this.

To wrap up, when speaking of AJAX it means solutions which make use of these libraries of javascript code (or of self-rolled javascript), while LJAX means making use of Lasso's implementation of AJAX. This paper is about AJAX, but not because I think its better; I'm afraid its pure coincidence: I started looking into AJAX before LJAX was part of Lasso and ended up sticking with it...

1.5 To Get Started

So, to get started, the first thing to do is to download these libraries. There are two "sets" of libraries:

1. Prototype, the javascript framework which all the rest builds on:
   http://prototype.conio.net
2. Scriptaculous, which builds on Prototype and provides the actual functionality:
   http://script.aculo.us

Prototype is one javascript-file, Scriptaculous is a set of files, 7 in total, plus other files for testing etc. If you follow the instructions for the Scriptaculous download and place the files in a folder in your webserver root you can run the test pages and see all it can do for yourself!

To do anything AJAX-like on a webpage you must load these javascript libraries, calling them in the head of the document. Assuming you keep all javascripts in a folder named "js" in the root, it could look like this if you want to load all of them:

```
<head>
  ...
  <script type="text/javascript" src="/js/prototype.js"></script>
  <script type="text/javascript" src="/js/scriptaculous.js"></script>
  <script type="text/javascript" src="/js/builder.js"></script>
  <script type="text/javascript" src="/js/controls.js"></script>
  <script type="text/javascript" src="/js/effects.js"></script>
  <script type="text/javascript" src="/js/dragdrop.js"></script>
  <script type="text/javascript" src="/js/slider.js"></script>
</head>
```

Its also possible to tell the scriptaculous library which other library to load (given it lives in the same folder as scriptaculous.js) this way:

```
  <script type="text/javascript" src="/js/scriptaculous.js?load=effects"></script>
```

To keep things simple to start with I suggest loading all of these libraries. The downside is that they add up considerably to the total load time of the page, but the browser will, however, keep them in memory after the first load.

1.6 The Files Involved - And The Custom Javascript

In many AJAX-"stories" including a call to a database the files involved are the following:

1. the main document, the webpage
2. the AJAX libraries mentioned above
3. a Lasso file which receives the AJAX-call, does a database lookup and returns the results

Each time an AJAX call has to be issued some custom javascript is needed. It can be very little, it can be a bit more. This custom javascript can be inserted in the webpage just after the HTML-element with which the user interacts, or it can be put away into yet another external javascript file, a custom javascript library. One way (Deco Rior) is to write the custom javascript directly into the webpage while building and debugging and then store it in an external library once its working. But storing it in an external file means defining a javascript function - usually with parameters - and then call that function in your document, so it does add some complexity. The huge advantage is that you can re-use such abstracted functions throughout the entire project. See the examples below.

1.7 Event Handlers

When a user interacts with a webpage - it being a click on a link or on a button, entering values in a form field etc. - that action can be picked up by an event handler or an event observer which fires a javascript. Here is a very simple javascript (no AJAX) which is called by the 'onclick' event handler:

```
<a href="#" onclick="window.close('self');" title="Close this window">Close</a>
```

Simply speaking, 'onclick' tells the browser to close the window instead of following a hyperlink (which would otherwise be the normal behaviour of the <a>-tag).

1.8 Identifying portions of a webpage and CSS

The major point of AJAX is to refresh only portions of a webpage, as mentioned above. In order for an AJAX-function to know which part of a webpage to refresh, the portion has to be identified by a unique CSS 'id'. Examples:

```
<div id="orderlist"> ...here stuff to be refreshed... </div>
<span class="dimmed" id="backorder"> ...here stuff to be refreshed... </span>
```

CSS is also used to style everything coming out of an AJAX-function. This doesn't mean you can't apply AJAX to webpages build without CSS, pages laid out with tables, for example, will do well. But of course there is an advantage of structuring entirely with CSS because you then - probably - will have a structure in which its very simple to add the markup needed for AJAX. Its a good advice (Paul Melia) to run your page through the W3C validator http://validator.w3.org for valid XHTML before adding AJAX-functionality.

1.9 Busy indicator

With anything AJAX its very important to give a feedback to the user telling that a process is going on. Since the entire page is not refreshing there is nothing else to indicate that the user has to wait a little for the result to show up than what the developer provides. Small animated .gifs to use for this purpose can be generated and downloaded here: http://www.ajaxload.info.

A busy indicator needs to show up next to the HTML-element with which the user interacts. There are two ways of calling the indicator:

- enabling and disabling it is included in the AJAX-call itself
- it is enabled and disabled "automatically" by adding this javascript to an external custom javascript library (Geoff Attrill) which must be called on the pages where its needed:

```
Ajax.Responders.register({
  onCreate: function() {
    if ($('busy') && Ajax.activeRequestCount > 0)
      Effect.Appear('busy',{duration:0.1,queue:'end'});
      Element.update('busy','<img src="/path-to-web-grafics/busy.gif">');
  },
  onComplete: function() {
    if ($('busy') && Ajax.activeRequestCount == 0)
      Effect.Fade('busy',{duration:0.1,queue:'end'});
      Element.update('busy','');
  }
});
```

For this script to work you need a busy indicator graphics file named 'busy.gif' stored on the 'path-to-web-grafics', and you place the indicator in the webpage by adding a where you want the indicator to appear. This is a very handy solution, except where you need more than one busy indicator on the same webpage. In this last case you must let the AJAX-calls handle the indicator, identifying the divs where they should appear with unique values, avoiding 'busy' used by the function above.

I always include a class with the -tag holding the busy indicator:

```
<span id="busy" class="busy"></div>
```

which is declared like this:

```
.busy {
  width: 16px;
  height: 16px;
  float: left;
  margin-left: 8px;
}
```

The width and height make sure the tag occupies the space needed for the indicator already at pageload.

1.10 Failure

If an AJAX-call takes too long to return results, or it returns results of the wrong type, the user should receive feedback indicating that something didn't go as expected. I must admit that this is a part that I still need work on, and the below standing examples will not contain much handling of failure.

1.11 Security

Javascript can be injected into form fields interfering with the AJAX-calls, or the secondary Lasso file receiving the AJAX-calls can be called directly - with or without javascript-injection, just to mention two obvious ways of attacking AJAXified solutions. The same rules are valid for all files involved in an AJAX-"story" as for any other web solution you want protected, and validation of params before letting them proceed to database calls is essential. You can pass session ids along with AJAX-calls, and if you use a framework like PageBlocks www.pageblocks.org you can continue applying its strong protection as long as you pass the session id along with the AJAX-call to the secondary Lasso file and program those as single files (no template).

In the examples below I will show how a session id is passed along with the AJAX-call.

Example 1 - Populate Second Menu with Values as Result of Selection in First Menu

2.0 Objective

This example does the same as the example in Fletcher Sandbeck's Tip Of The Week June 24 2005 http://www.omnipilot.com/Tip%20of%20the%20Week.1768.8810.lasso, except it uses the above mentioned AJAX libraries to accomplish the same functionality: User selects a value in a menu, and a second menu is populated with values as a result of this first selection.

2.1 Files Involved

1. The display page, the webpage
2. An external custom javascript library file
3. A Lasso file receiving the AJAX-call and returning the results, 'repertoire_retrieve.lasso'

I have chosen to put the custom javascript into an external custom javascript library here because its easier to add the AJAX-call to the event handler then, and because this code is likely to be useful in more than one place. I have also chosen to handle the busy indicator in the AJAX-call instead of using the general script from 1.9 above.

2.2 The Display Page

The display page with the two menues could look like this:

```
<label>1. Find Composer:</label>
<select id="popup_1" name="popup_1" onchange="update_select('popup_
2','busy2','repertoire_retrieve.lasso?fw_s=[$fw_s]&amp;param=',this.value)">
  <option selected="selected" value="">
    Select Composer...
  </option>
  <option value="myvalue1">
    Johann Sebastian Bach
  </option>
  ...etc.
</select><span id="busy2" class="busy" style="display:none;"><img src="/path-to-web-
graphics/busy.gif"></span>
<br />
<label>2. Find Piece:</label>
<select disabled="true" name="r_id" id="popup_2">
  <option value="" selected="selected">
    (First use menu above)
  </option>
</select>
```

The event handler 'onchange' calls the javascript function 'update_select' with 4 parameters:

- 'popup_2': the id of the second menu to update.
- 'busy2': the id of the element holding the busy indicator.
- the URL of the file receiving the call and returning the results, ''repertoire_ retrieve.lasso?fw_s=[$fw_s]&param='. Notice that a parameter 'fw_s' is passed along with the URL, this is the session id. Another parameter name - 'param' - is passed with the URL.
- the value of the option selected from the menu, 'this.value' - the value of 'param'.

Notice that the second menu is disabled by default. Also notice that the first menu is not enclosed in <form>-tags, this isn't necessary because it only communicates via AJAX. The second menu would typically be part of a plain form but could also issue another AJAX-call, this isn't shown here.

2.3 The Custom Javascript

The event handler calls a function, which in turn issues the AJAX-call, among others. This function must be defined in the external custom javascript library which is included in the head of the page. Following the scheme from 1.5 above, it could look like this:

```
<script type="text/javascript" src="/js/scriptlib.js"></script>
```

The function 'update_select' defined in 'scriptlib.js' looks like this:

```
function update_select(select_id, busy_id, url, opt_val)
{
  if (opt_val == “”)
    $(select_id).disabled = true;
  else
  {
    $(select_id).disabled = false;
    $(select_id).focus();
    $(busy_id).style.display = ‘’;
    var ajax = new Ajax.Updater(
      select_id,     // div id
      url + opt_val, // url with params
      {            // options
        method:’get’,
        onComplete: function(element)
        {
          $(busy_id).style.display = ‘none’;
        }
      }
    );
  }
}
```

I’m not going to fully analyze all custom javascripts in this paper, but essentially what happens here is that the busy indicator is being turned on or off acording to various conditions, and the core of the code is the ‘new Ajax.Updater();’ which performs the call to the Lasso file, receives the results and inserts it into the specified location in the webpage.

2.4 The Lasso File

The small Lasso file will be called with an URL looking like this:

```
repertoire_retrieve.lasso?fw_s=MySessionId&amp;param=SomeValue
```

Input validation should be performed on these two parameter values before anything else. Then a plain database lookup is performed searching for records with a given field containing or matching the value of the parameter ‘param’:

```
inline(...here db-connection, search statement, sorting etc...);
  records;
    ‘<option value=”’ + field(‘MyRecordId’) + ‘”>’ + field(‘MyRecordValue’) + ‘</
option>’;
  /records;
/inline;
```

When testing and debugging its handy to call this Lasso file directly in the browser with an URL like the one shown above and with a valid value for ‘param’. The file should return expected results in the browser, it doesn’t matter that the file isn’t a complete HTML-page.

Example 2 - Simple Autocompleter

3.0 Objective

This example is mainly to show the basics of how to add an Autocompleter or "Live Search" to a form text field. It returns live results but can't add such a result to the form text field and thus can't perform further searching by submitting a form with such a result.

It can be used practically, however. This example shows a small part of a form for entering composers with first name, last name and various other data into a database in an intranet. When the user enters the first name we want to check if a composer with that first name already exists in the database. We return a list showing composers with that first name AND their last names respectively in order to check for composers with the same first name but different last name. If a composer already exists the user simply doesn't enter the data but proceeds to the next composer needing to be entered.

3.1 Files Involved

1. The display page, the webpage
2. A Lasso file receiving the AJAX-call and returning the results, 'composers_retrieve.lasso'

Here I have chosen to put the custom javascript directly into the webpage so there is no external custom javascript library file. I have also chosen to let the general script from 1.9 above handle the busy indicator.

3.2 The Display Page

The form used to enter composres into the database has various fields of course. Here only the field for First Name is shown, which is the field having the Autocompleter attached to it:

```
<label for="NmeF">First Name:</label>
<input autocomplete="off" type="text" name="m_NmeF" id="NmeF" value="[var('m_
NmeF')]" maxlength="20" /><div id="busy" class="busy"></div>
<div id="composers" class="autoshow"></div>
<script type="text/javascript" language="javascript">
  new Ajax.Autocompleter(
    "NmeF",
    "composers",
    "composers_retrieve.lasso?fw_s=[$fw_s]",
    {
      paramName: "nme",
      minChars: 2,
      frequency: 0.5
    }
  )
</script>
```

Lets look at the elements one by one:

1. <input autocomplete="off"...>: hides the <div id="composers"...>-tag on pageload (Paul Melia). I haven't been able to find any visual difference wether or not this attribute is present.

2. <input... name=”m_NmeF” id=”NmeF”...>: its useful in some cases to keep name and id of the input tag different. The name is used to identify the value of the input element upon form submition, while the id is used by the javascript(s) involved to identify the tag itself.
3. : this is where the automatic script from 1.9 above will display the busy indicator. See 1.9 for the declaration of class=”busy”
4. : this is the where AJAX inserts the search results by altering the DOM, it simply changes the markup and content between <div...> and . The id=”composers” is used by AJAX to identify the tag, and the class=”autoshow” is declared as follows:

```
div.autoshow {
  background-color: #ffffff;
  border: 1px solid #000000;
  margin: 0px;
  padding: 0px;
}

div.autoshow ul {
  list-style-type: none;
  margin: 0px;
  padding: 0px;
}

div.autoshow ul li {
  list-style-type: none;
  border-bottom: 1px solid #dcdcd6;
  display: block;
  margin: 0;
  padding: 0.4em;
}

div.autoshow ul li.selected {
  background-color: #ffffff;
}
```

AJAX uses an unordered list <ul> to mark up the content returned by the call, hence the many declarations concerning <ul> and <li>.
AJAX uses the class .selected to style (highlight) the single entries on mouseover (or when selecting with up- and downarrow) in the list of returned results. We don’t want that highlighting because it would suggest that the user can select an entry and have it put into the input field, which isn’t the case in this example. Therefore the declaration div.autoshow ul li.selected {} has the same background-color as the general background-color in div.autoshow {} - that is #ffffff.

5. The custom javascript with ‘new Ajax.Autocompleter’. Between (and) it holds parameter values and function variables:
   - “NmeF”: the id of the input field which the Autocompleter is watching
   - “composers”: the id of the where the returned results should be inserted
   - “composers_retrieve.lasso?fw_s=[$fw_s]”: the URL of the file which performs the lookup. Notice the parameter ‘fw_s=[$fw_s]’ - this is the session id being passed with the URL

- paramName: “nme”: the name of the parameter passing the values entered into the watched input field to the file which performs the lookup
- minChars: 2: the minimum number of characters before AJAX fire
- frequency: 0.5: the delay in seconds between each call issued by AJAX. If this value is too small a fast typist will cause AJAX to fire too often and results will show up flickering or not at all.

3.3 The Lasso File

The small Lasso file will be called with an URL looking like this:

```
composers_retrieve.lasso?fw_s=MySessionId&amp;nme=SomeValue
```

Input validation should be performed on these two parameter values before anything else. Then a plain database lookup is performed searching for records with a given field containing the value of the parameter ‘nme’. If you make a custom sql-statement it should perform the search with ‘LIKE’ in order to get all records containing the value.

```
inline(...here db-connection, search statement, sorting etc...);
  ‘<ul>’;
    records;
      ‘<li>’;
       field(‘firstname’) + ‘<span class=”informal”> ‘ + field(‘lastname’) + ‘</
span>’;
      ‘</li>’;
    /records;
  ‘</ul>’;
/inline;
```

As can be seen, the results are returned as an unordered list. Each list element consists of the first name value (the Autocompleter input field is the first name field) and the last name value surrounded by .

Why this class? It tells AJAX that everything surrounded by this tag doesn’t belong to the field to which Autocompleter is attached. Or in other words, what is surrounded by is filtered out if an entry is selected by the user (which we don’t make use of here, but in the next example we will). The class=”informal” does not have to be declared by the developer anywhere.

When testing and debugging its handy to call this Lasso file directly in the browser with an URL like the one shown above and with a valid value for ‘nme’. The file should return expected results in the browser, it doesn’t matter that the file isn’t a complete HTML-page.

Example 3 - Complex Autocompleter With DOM Update

4.0 Objective

This examples extends the previous example and performs a lookup based on a user-selected entry from the list of returned results. This example is designed for the following scenario:

In a Contacts database the user wants to search for a single record and display all details for this record, all without neither refreshing the page nor using a response page.

One problem we encounter here is that when we want all details returned for a single record, AJAX isn't able to return strings longer than (I think its) 40 characters.

I should stress that this is a solution entirely developed by myself and only tested in Firefox and Safari on Mac, because it was developed for an intranet. I haven't had the time to check it on Explorer.

4.1 Files Involved

1. The display page, the webpage
2. A Lasso file receiving the AJAX Autocompleter-call and returning the results, 'contacts_retrieve1.lasso'
3. An external custom javascript library file, 'scriptlib.js'
4. A Lasso file receiving the AJAX-request for the single record and returning the result, 'contacts_retrieve2.lasso'

Here the first custom javascript which invokes the AJAX Autocompleter is again put into the webpage, while the second custom javascript which invokes the AJAX-call for the single record details is abstracted into a custom javascript library file, similar to Example 1, 2.3. The busy indicator is handled by the general script from 1.9 above.

4.2 The Display Page

The display page is in this case a search page and contains only one input field where the user can enter something of the first name, the middle name, the last name or the company name of the single record she is looking for. For the engine behind it doesn't matter not knowing which kind the input is, so why bother the user?
No <form>-tags surround the input field because we aren't going to submit any form here.

```
<input id="c_id" name="c_id" type="hidden" value="" />
<div id="container" style="display:none;"></div>
<label for="search">Enter Search Text:</label>
<input type="text" name="m_Nmel" id="search" value="" maxlength="30" />
<div id="busy" class="busy"></div>
<div id="results" class="autocomplete"></div>
<script type="text/javascript" language="javascript">
  new Ajax.Autocompleter(
    "search",
    "results",
    "contacts_retrieve1.lasso?fw_s=[$fw_s]",
    {
      paramName: "nme",
      minChars: 2,
      frequency: 0.5,
      afterUpdateElement: function()
      {
```

```
        getContact('search','c_id','[$fw_s]');
      }
    }
  )
</script>
<br />
<fieldset>
  <legend>Result</legend>
  <div id="contact"></div>
</fieldset>
```

Lets look at the elements one by one:

1. <input id="c_id" name="c_id" type="hidden" value="" />: this is a hidden input element used by the scripts involved to temporarily store the value of the variable 'c_id' (see later).
2. : this is also an element used by the scripts involved to store values temporarily (see later).
3. <input... name="m_Nmel" id="search" value=""...>: actually we don't need a name here because there is no form which will be submitted, I think its there for nostalgic reasons. The id is used by the javascript(s) involved to identify the tag itself.
4. : this is where the automatic script from 1.9 above will display the busy indicator. See 1.9 for the declaration of class="busy"
5. : this is the where AJAX inserts the search results by altering the DOM, it simply changes the markup and content between <div...> and . The id="results" is used by AJAX to identify the tag, and the class="autocomplete" is declared as follows:

```
div.autocomplete {
  width: 350px !important;
  background-color: #ffffff;
  border: 1px solid #000000;
  margin: 0px;
  padding: 0px;
}

div.autocomplete ul {
  list-style-type: none;
  margin: 0px;
  padding: 0px;
}

div.autocomplete ul li {
  list-style-type: none;
  border-bottom: 1px solid #dcdcd6;
  display: block;
  margin: 0;
  padding: 0.4em;
  cursor: pointer;
}

div.autocomplete ul li.selected {
  background-color: #ffb;
}
```

As can be seen most declarations are equal to those for the class=”autoshow” shown in 3.2 above, please look there for some explanations. Opposite to class=”autoshow” we do want a highlight colour when hovering or arrowing into entries in the list of returned results, hence the background-color declared in div.autocomplete ul li.selected { background-color: #ffb; }. What is this: div.autocomplete { width: 350px !important; ... }? It seems in my experience that when formatting the list of results returned by the Autocompleter, AJAX sets the width of this list equal to the width of the field to which its attached. This certainly isn't always desireable.

The only way I have found to override this behaviour is to add the attribute '!important' to the width-declaration, otherwise the value isn't taken into consideration when the list is rendered.

6. The custom javascript with 'new Ajax.Autocompleter'. Between (and) it holds parameter values and function variables:
   - “search”: the id of the input field which the Autocompleter is watching
   - “results”: the id of the where the returned results should be inserted
   - “contacts_retrieve1.lasso?fw_s=[$fw_s]”: the URL of the file which performs the lookup. Notice the parameter 'fw_s=[$fw_s]' - this is the session id being passed with the URL
   - paramName: “nme”: the name of the parameter passing the values entered into the watched input field to the file which performs the lookup
   - minChars: 2: the minimum number of characters before AJAX fire
   - frequency: 0.5: the delay in seconds between each call issued by AJAX. If this value is toosmall a fast typist will cause AJAX to fire too often and results will show up flickering or not at all

7. The main difference with the simple Autocompleter in example 3 lays in the function called by AJAX after it has returned the list of results when the user clicks an entry. afterUpdateElement: function() is a function call you can include with the list of variables of the Autocompleter, when used you must add the name of the function to be called when user clicks an entry:

```
{ getContact('search','c_id','[$fw_s]'); }
```

The function getContact takes 3 parameters:
- 'search': the id of the input field which the Autocompleter is watching.
- 'c_id': the id of the hidden input element used to store the value of the variable 'c_id' (see later).
- '[$fw_s]': the session id value.

4.3 The First Lasso File

The first Lasso file, the one which returns the Autocompleter results, is quite similar to the one in example 2 above. It will be called with an URL looking like this:

```
contacts_retrieve1.lasso?fw_s=MySessionId&amp;nme=SomeValue
```

Input validation should be performed on these two parameter values before anything else. Then a database lookup is performed searching for records where one or more of the fields in question contain the value of the parameter 'nme'. If you make a custom sql-statement it should perform the search with 'LIKE' in order to get all records containing the value.

Here is an example of the construction of the 'WHERE'-part of an sql statement:

```
local('where' = 'firstname LIKE "%' + encode_sql(action_param('nme')) + '%" OR ' +
              'middlename LIKE "%' + encode_sql(action_param('nme')) + '%" OR ' +
              'lastname LIKE "%' + encode_sql(action_param('nme')) + '%" OR ' +
              'company LIKE "%' + encode_sql(action_param('nme')) + '%"');
```

The code returning the records found looks like this:

```
inline(...here db-connection, search statement, sorting etc...);
  '<ul>';
    records;
      '<li>';
        field('firstname') + '<span class="informal"> ';
        field('middlename')->size ? field('middlename') + ' ';
        field('lastname');
        '<br /><span style="color:#666666;">' + field('company');
        '</span></span>';
        '<span style="display:none">';
        '#' + field('recordid');
        '</span>';
      '</li>';
    /records;
  '</ul>';
/inline;
```

As can be seen, the results are again returned as an unordered list, but there is more complexity. First, a is again used to surround the parts that don't derive from the input field (excpet the invisible part). Next, each record returned has more information so we mark it up a little: a line break
 is inserted to force the company name to start on a second line, and the company name is styled with a grey colour.

This last part:

```
        '<span style="display:none">' +
        '#' + field('recordid') +
        '</span>';
```

adds an invisible part to the returned result containing the sign '#' and the record id, rendered it looks like this: #SomeRecordId. Why is this? This is because we need to return the record id somehow, but of course we don't want to show it in the list. And in order to later split it from the rest (see below) we use the '#'-sign as a delimiter - meaning that the ids used should never contain that sign.

4.4 The Custom Javascript

The function 'getContact();' called by Autocompleter when the user clicks an entry in the returned results list must be defined, and I prefer to have that definition in an external custom javascript library. The function looks like this:

```
function getContact(inp1,inp2,fws) {
  var str = document.getElementById(inp1).value;
  var strSplit = str.split("#");
  for (var i = 0; i < strSplit.length; i++) {
    var val = strSplit[i];
    if (val) {
      var val0 = strSplit[0]; // search
      var val1 = strSplit[1]; // id
    }
  }
  document.getElementById(inp1).value = val0;
  document.getElementById(inp2).value = val1;

  var ajaxUpdater = new Ajax.Updater({
    success: 'container'
    },
    'contacts_retrieve2.lasso', {
      method: 'get',
      evalScripts: true,
      parameters: 'id=' + val1 + '&fw_s=' + fws,
      onFailure: reportError
    }
  );
}

function reportError(request) {
  alert('Sorry, there was an error communicating with the server!');
}
```

What this function does is that it takes the string returned by Autocompleter and splits it on the '#'-sign, updates the input field value with the first part (except what is inside , and updates the hidden input field <input id="c_id" name="c_id" type="hidden" value="" /> with the last part - the record id.
Then it issues an AJAX call, var ajaxUpdater = new Ajax.Updater(), but it does so by declaring it as a var which should ensure that the javascript returned by AJAX (see later) will run as expected. In this AJAX-call there is a number of parameters and function variables to remember:

- success: 'container': tells AJAX the id of the element where it stores the returned data of the single record. Necessary eventhough the element stays invisible and never is involved in displaying anything.
- contacts_retrieve2.lasso: the URL of the Lasso file which receives the AJAX-call and returns the result of the lookup of the single record
- method: get: the http method used
- evalScripts: true: tells AJAX that the returned result contains javascript
- parameters: 'id=' + val1 + '&fw_s=' + fws: the parameters passed along with the URL. There are two: the record id and the session id

- onFailure: reportError: the name of the function to call in case of a failure (see the function just below the main function).

4.5 The Second Lasso File

The second Lasso file will be called with an URL looking like this:

```
contacts_retrieve2.lasso?id=SomeRecordId&fw_s=MySessionId
```

Input validation should be performed on these two parameter values before anything else. Then a simple database lookup is performed searching for a record with the record id given.

We want the result - all the data for the single record - to be displayed in the main webpage by updating the contents of . This means we need to return the result as javascript, not as HTML markup and content. The code returning the data found looks like this:

```
inline(...here db-connection, search statement, sorting etc...);

  protect;
    '<script type="text/javascript" language="javascript"> ';
    '$(\'contact\').innerHTML=\'\'; '; // zero content of div contact
    (field('firstname')->size) || (field('middlename')->size) || (field('lastname')-
>size) ?
      '$(\'contact\').innerHTML+=\'<label>Name:</label>\'; ';
    field('firstname')->size ?
      '$(\'contact\').innerHTML+=\'<div id="Nmef"> ' + encode_
sql(field('firstname')) + '</div>\'; ';
    field('middlename')->size ?
      '$(\'contact\').innerHTML+=\'<div id="Nmem"> ' + encode_
sql(field('middlename')) + '</div>\'; ';
    field('lastname')->size ?
    '$(\'contact\').innerHTML+=\'<div id="Nmel"> ' + encode_sql(field('lastname'))
+ '</div>\'; ';
    (field('firstname')->size) || (field('middlename')->size) || (field('lastname')-
>size) ?
      '$(\'contact\').innerHTML+=\'<br />\'; ';
    field('jobtitle')->size ?
      '$(\'contact\').innerHTML+=\'<label>Job Title:</label><div id="Prof">' + encode_
sql(field('jobtitle')) + '</div><br />\'; ';
    field('company')->size ?
      '$(\'contact\').innerHTML+=\'<label>Company:</label><div id="Cpny">' + encode_
sql(field('company')) + '</div><br />\'; ';
    ...etc. etc. for all fields you want returned...
    '</script>';
  /protect;

/inline;
```

This looks like a complete mess - certainly because its written in lassoscript - but what it does is quite simple: it returns a declaration wrapped in <script type="text/javascript" language="javascript"></script> which updates the innerHTML of . It uses the shortcut for 'getElementById' provided by Prototype: $. Example: The line

```
'$(\'contact\').innerHTML+=\'<div id="Nmel"> ' + encode_sql(field('lastname')) +
'</div><br />\'; ';
```

when rendered returns this:

```
$('contact').innerHTML+='<div id="Nmef"> Doe</div><br />';
```

Notice the 'encode_sql' before each field content. This is in order to escape signs in the field contents that otherwise would halt the javascript code - like ' and ".

Voilà, because 'evalScripts' is set to 'true' in the Ajax.Updater this javascript updates the DOM of the main webpage and the returned single record data appears in the page body without any page reload.

Example 4 - Sortable List

5.0 Objective

This example is an attempt to show how to work with and collect data from a Sortable List. Sortable List is a feature of the Scriptaculous libraries which makes it possible to create a list out of HTML-elements - -elements in an -list - where list elements can be added and removed, and the order of the list elements can be changed. A main challenge is the exchangie data between javascript/AJAX and Lasso.

Many things have to be accomplished:

- when the page first loads the user must be presented with an empty area where the list will be created and a stock of elements to drag-and-drop
- when the user submits a list the single element ids AND the order in which they appear in the list must be picked up and saved to the database
- when the user loads a page with an existing list the list must be created based on a database lookup and transformed into a Sortable List

5.1 Files Involved

1. The display page, the webpage
2. An external custom javascript library file
3. The logic handling data and database update

In this example all custom javascript is abstracted into a custom javascript library. There is no need for a busy indicator.

5.2 The Display Page

To simplify we divide the main document in two columns, a left column and a right column. We keep the Sortable List in the left column and the stock of elements to drag-and-drop on the Sortable List in the right column. The display page contains as well a form which - in my example - upon submition reloads the entire page and passes the element ids and their order as actionparams to Lasso. Here is an example of what the essential part of the main page could look like:

```
<div class="leftcolumn">
  <div id="listarea">
    <ul id="mylist"><?lassoscript
      if(var('mylist_list')->size);
        iterate($mylist_list, local('i'));
          '<li id="listitem_' + #i->(first) + '">';
            '<a href="" onclick="remove_el(\'listitem_' + #i->(first) + '\');">Remove</
a>';
            '<br />' + #i->(second);
          '</li>';
        /iterate;
      else;
        '<li id="item_insert">Please drop elements here!</li>';
      /if;
    ?></ul>
    <form name="updte" id="updte" method="post" action="[response_filepath]?fw_
s=[$fw_s]">
      <input type="hidden" name="recordid" value="[$recordid]" />
      <input type="hidden" name="ids_serialized" id="ids_serialized" />
      <input type="button" onclick="list_submit();" name="btnSave" value="Save" />
    </form>
  </div>
</div>
<div class="rightcolumn">
  <ul id="myelements">
    <li id="listitem_1">Element 1</span>
    <li id="listitem_2">Element 2</span>
    ...etc
  </ul>
</div>
```

The lassoscript in the middle creates a list of elements if the Lasso-array '$mylist_list' has size, which means that a list already exists in the database and has been inserted into the array '$mylist_list'. If '$mylist_list' has no size the script inserts the element <li id="item_insert">Please drop elements here! instead.
Notice that the ids identifying each -element are constructed like this: a constant value - 'listitem' in this case - and the record id with an '_' (underscore) between them, looking like this when outputted: id="listitem_MyRecordId". This is a requirement for the AJAX Sortable List to work: it picks up the ids by splitting on the constant 'listitem_' in this case. What the constant is named doesn't matter as long as it doesn't contain underscore and is - err - constant.

This line: '(first) + '\');">Remove' constructs a link which will call the function 'remove_el('listitem_MyRecordId');' (see below) with the 'onclick' eventhandler.

The hidden input element in the form, <input type="hidden" name="ids_serialized" id="ids_serialized" /> is used by the scripts involved to pass a javascript-array of element ids, which is submitted with the form (see below). Notice that the button doesn't submit the form but calls the function 'list_submit();' (see below) with the 'onclick' eventhandler.

The most important CSS declarations are these:

```
div.leftcolumn {
  float: left;
}

div.rightcolumn {
  float: right;
}

#listarea {
  background-color: transparent;
  min-height: 200px;
}

ul#mylist, ul#myelements {
  list-style-type: none;
  padding: 0;
}

ul#mylist li {
  margin: 0 0 6px 0;
  padding: 0.2em 0.4em 0.6em 0.4em;
  width: 200px;
  height: 30px;
  background-color: #ffffff;
  cursor: move;
}

ul#myelements li {
  margin: 0 0 4px 0;
  padding: 0.1em 0 0.1em 0.3em;
  width: 200px;
  height: 30px;
  background-color: #e5e5d9;
  cursor: move;
}

.dropover {
  background-color: #fbfbf7;
  border: 1px solid #c8c9c0;
}
```

The class .dropover is a class you can tell the Sortable List to use for highlighting the list area when an element is hovered over the list (see later).

But how are these lists turned into "Sortable Lists"? This is done by a custom javascript - 'create_sortable();' - using AJAX, which loads on pageload - see below.

5.3 The Custom Javascripts

All custom functions are kept in a custom javascript library, 'scriptlib.js'. This avoids - among others - problems with [and] in the scripts.
The function 'create_sortable();' looks like this:

```
function create_sortable()
{
  Sortable.create("mylist",
    {
      dropOnEmpty: true,
      hoverclass: "dropover",
      containment: ["mylist","myelements"],
      constraint: false,
      onUpdate: function(element)
      {
        if ($('item_insert') != null)
          $('mylist').removeChild($('item_insert'));
        if (Sortable.serialize('mylist') == '')
          add_item();
      }
    }
  );

  Sortable.create("myelements",
    {
      dropOnEmpty: true,
      containment: ["mylist","myelements"],
      constraint: false
    }
  );
}

function add_item()
{
  var ul = $("mylist");
  var quest = 'Please drop elements here!';

  var li = document.createElement("li");
  li.setAttribute("id","item_insert");
  li.appendChild(document.createTextNode(quest));
  ul.appendChild(li);
  create_sortable();
}
```

This is getting a little more complex: what essentially happens in 'create_sortable();' is that it uses the AJAX-function 'Sortable.create();' to create Sortable Lists from the two -lists identified by their ids - 'mylist' and 'myelements'. In this example these ids are hardcoded into the script, they could as well be passed as parameters of course. Here are most but not all explanations for the function variables:

- hoverclass: 'dropover': the class for AJAX to use to style the list when hovered with an element
- containment: ["mylist","myelements"]: the two lists between which you can drag-and-drop
- constraint: false: if true you can only drag elements along vertical, horisontal or diagonal lines
- onUpdate: function(element): this defines a function to call when the list is updated, that is, when an element is added or removed from the list. This function checks to see if the element with id 'item_insert' (the element to show on empty list) is present and if yes, it removes it. It also checks to see if the list has content by calling 'Sortable.serialize('mylist')'

- an AJAX-function which serializes the id-values of the list elements - and if not, it calls the custom function 'add_item();' which creates the element 'item_insert' shown on empty list.

The function 'create_sortable();' must be called on pageload in the <body>-tag:
<body onload="create_sortable();">

The function 'list_submit();' looks like this:

```
function list_submit()
{
  var result = Sortable.serialize('mylist');
  $('ids_serialized').value = result;
  document.updte.submit();
}
```

This function again uses the AJAX-function Sortable.serialize(); to create a javascript array from the single element ids in the list identified by the id 'mylist'. It then sets the value of the hidden input element 'ids_serialized' to this array. Finally it submits the form identified by its name 'updte'. This way we get the form submitted with the param 'ids_serialized=SomeJavascriptArray'. This array is what we want to work on with Lasso in order to get the single element ids and the order in which they appear in the list (see below).

Now, what is missing is a way to remove elements from the list. The function used for this named 'remove_el();' looks like this:

```
function remove_el(obj)
{
  var elmnt = $(obj);
  elmnt.parentNode.removeChild(elmnt);
  if (Sortable.serialize('mylist') == '')
    add_item();
}
```

After removing the element which called the function the script checks if the list has become empty, and if yes it calls the function 'add_item();' explained above.

5.4 The Logic Handling Data And Database Update

In this example the database must have two tables: one to hold each list and one to hold data for each list element. The tables are named 'list_tbl' and 'elmnt_tbl'. In the following lassoscript examples I assume we have defined a variable '$dbcon' with all database connection information, table name to ensure correct encoding etc, and that variable names and database field names are identical for simplicity (shouldn't be in a live solution). Most error catching etc. is left out. The code snippets are indeed snippets, they must be inserted in the developer's preferred response construct.

Accessing an existing list. When a call for an existing list is issued we want to prepare a lasso-array - '$mylist_list' - with data for display. Assuming that the table 'list_tbl' has a record id field

named ‘recordid’ and a field (type ‘text’) holding the list element ids named ‘elementids’ we could lookup the data like this:

```
local(‘ids_tmp’ = array); // declare an array to hold element ids temporarily
inline($dbcon, sql=”SELECT * FROM list_tbl WHERE recordid = ‘” +
actionparam(‘recordid’) + “’”);
  if(field(‘elementids’)->size);
    #ids_tmp->(unserialize(field(‘elementids’)));
  /if;
/inline;
```

As seen the list element ids are stored as a serialized array and must be unserialized before any acting upon them.

Assuming that the table ‘elmnt_tbl’ has a record id field named ‘recordid’ and a field holding record details named ‘recorddetail’ we could create an array for displaying the list elements like this:

```
var(‘mylist_list’ = array);    // declare array to hold elements for display
iterate(#ids_tmp, local(‘i’)); // iterate the array from above
  inline($dbcon, sql=”SELECT * FROM elmnt_tbl WHERE recordid = ‘” + #i + “’”);
    $mylist_list->(insert(pair(field(‘recordid’) = field(‘recorddetail’))));
  /inline;
/iterate;
```

The array ‘mylist_list’ is now ready for display, see use for display in 5.2 above.

Saving a list. When the form is submitted by the function ‘list_submit();’ (see above) we get a serialized javascript array as value for the param ‘ids_serialized’ (see of 5.3, last paragraph), but this array has a form which has to be cleaned up before we can do anything Lasso with it. It uses ‘&’ as delimiter and looks similar to this: mylist[]=MyRecordId1&mylist[]=MyRecordId2...etc, where ‘mylist’ is the id of the -list. The code for transforming this array into a Lasso-array, serialize this array and store it could look like this:

```
local(‘ids_tmp’ = array);  // declare an array to hold list element ids temporarily
local(‘ids_ser’ = string); // declare var to hold serialized Lasso-array
if(actionparam(‘ids_serialized’)->size);
  local(‘ids_serialized’ = actionparam(‘ids_serialized’));
  #ids_serialized = string_replace(#ids_serialized, -find=’mylist[]=’, -replace=’’);
  #ids_tmp = #ids_serialized->(split(‘&’));
  #ids_ser = #ids_tmp->serialize;
  inline($dbcon, sql=”UPDATE list_tbl SET elementids = ‘” + #ids_ser + “’”); /inline;
/if;
```

5.5 Extending

This example is somewhat cut down to keep it simple. For example will the -elements typically have more content that the single database field displayed here.
Here is a neat trick: its possible to style the -elements differently depending on which parent -tag they currently reside in. For example:

```
ul#mylist li {
  background-color: #ffff33;
}

ul#myelements li {
  background-color: #cc6600;
}
```

will cause the list elements change color when moved from the stock list to the right to the sortable list to the left.

Some resources

Short, not comprehensive list of resources

General:

Lassotalk thread started by Paul Melia:

http://www.listsearch.com/LassoTalk.lasso?id=214260

Tips of the Week:

http://www.omnipilot.com/Tip%20of%20the%20Week.1768.8801.lasso

http://www.omnipilot.com/Tip%20of%20the%20Week.1768.8810.lasso

Wikipedia:

http://en.wikipedia.org/wiki/AJAX
Prototype:

http://prototypejs.org

http://prototypejs.org/2007/1/18/prototype-documentation-is-here

Scriptaculos:

http://script.aculo.us
Scriptaculous Tags and Pages: http://wiki.script.aculo.us/scriptaculous/tags
Ajaxian:

http://www.ajaxian.com

Busy icons:

http://www.ajaxload.info

Overview of AJAX tutorials:

http://www.maxkiesler.com/index.php/weblog/comments/round_up_of_50_ajax_toolkits_and_frameworks

http://feeds.feedburner.com/ajaxian

Sergio Pereira's Developer Notes:

http://www.sergiopereira.com/articles/prototype.js.html

A couple of public libraries

Lightbox:

http://www.huddletogether.com/projects/lightbox2

Popup windows:

http://prototype-window.xilinus.com

Security

http://searchwebservices.techtarget.com/qna/0,289202,sid26_gci1164745,00.html

http://www.securityfocus.com/infocus/1868/2

Books

Foundations of Ajax:

http://www.amazon.com/gp/product/1590595823/sr=1-1/qid=1138930375/ref=pd_bbs_1/002-9289393-0220041?%5Fencoding=UTF8

Ajax in Action:

http://www.amazon.com/gp/product/1932394613/sr=1-2/qid=1138930375/ref=pd_bbs_2/002-9289393-0220041?%5Fencoding=UTF8

Pragmatic Ajax:

http://www.amazon.com/Pragmatic-Ajax-Web-2-0-Primer/dp/0976694085/sr=1-1/qid=1169242240/ref=pd_bbs_sr_1/105-2013680-1413243?ie=UTF8&s=books

DOM Scripting:

http://www.amazon.com/DOM-Scripting-Design-JavaScript-Document/dp/1590595335/sr=8-1/qid=1169242713/ref=pd_bbs_sr_1/105-2013680-1413243?ie=UTF8&s=books

The Javascript Anthology:

http://www.sitepoint.com/books/jsant1

Another Autocompleter

I have made a walk-through for an Autocompleter very similar to the one in Example 3. The difference is that the data returned upon the user's selection of an entry from the returned list is not displayed in a -tag but is inserted into the appropriate fields of a form. Or in other words, it uses an Autocompleter to accomplish an auto-fill of form fields with existing data from a database lookup: http://www.musicmedia.dk/lasso/ajax_populate.html.

Knop–the framework

Johan Sölve

Knop is a web application framework for Lasso Professional, consisting of a number of core components implemented as custom types, and a defined application flow and folder structure.

Knop is very much work in progress. All planned components do not exist yet, and the components that exist are in alpha or beta stage. The framework as described here has so far been used in one project, however the form generator component has been used in several projects.

Why framework?

An application framework saves time so the developer can focus on the core value of the application instead of dealing with the common and tedious repetitive tasks.

It allows for reusable code, components and modules.

Framework components become well tested over time which leads to higher quality of the application and reduces the risk of errors.

It leads to easier maintenance - changes and improvements can be implemented once and are immediately available wherever the component is used.

Using an application framework should lead to higher productivity and shorter lead times.

Why "Knop"?

"Knop" is Swedish for knot, and a knot is what keeps a lasso together. A good knot makes a good lasso experience.

And as a coincidence, "knop" in Swedish also has the nautical meaning just as knot has.

Knop is pronounced with a sounding "k" and a long "o" just as in groove.

By the way, we reserve the use of the word "Ponk" as well just in case we find the need of the reverse of a framework, whatever that might be.

Knop is created by Johan Sölve, Montania System AB

Credits

Greg Willits' PageBlocks manual has been a valuable inspiration when specifying some of the components of Knop.

Goals with Knop

A lightweight and easy to use framework, mainly targeted for web based applications (as opposed to web sites).

Flexibility to allow for special needs case by case.

Encourage the use of modern standards-based and semantically correct html and css for presentation.

Use client side scripting as progressive enhancement to improve user experience and application responsiveness, but do not rely on client side scripting for critical functionality.

Use Ajax techniques where really motivated

What is Knop?

- A defined application flow, to handle page requests, respond to form submissions and show the resulting page

- A defined folder structure to handle the application logic
- A number of custom types that handle the cure functions of the framework:

    mt_form - a form generator

    mt_nav - handles navigation in the application

    mt_grid - handles record listings

    mt_database - handles database interaction

    mt_user - handles user authentication and authorization

    mt_string - handles user interface strings

- Basic templates for html and css

The framework is entirely based on a onefile structure.

This presentation

The emphasis of this presentation will be the form generator datatype, since that is what I find the most valuable component of Knop. It's a good starting point to grasp the concepts of the framework.

A basic understanding of Lasso custom types is assumed.

Application flow

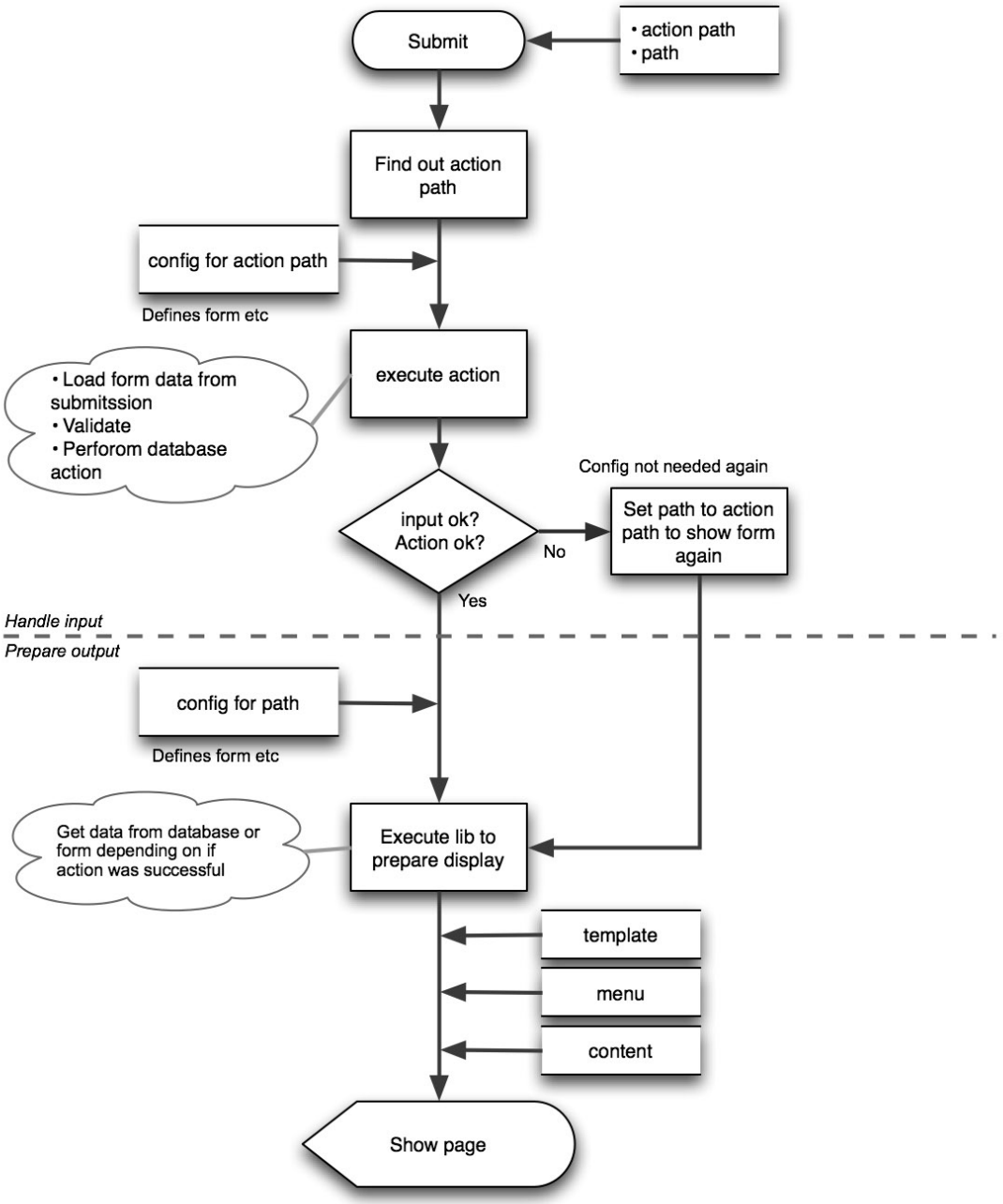
Let's assume the user submits a form. Every page request has two vital parameters:

- action path (optional), this tells the application how to act upon a submission (if there is one). In other words, where we came from. (This is not to mix up with the 'action' html attribute of the form tag itself, that is a completely different thing.)
- path (required), this tells the application where to go next.

So to handle a form submission:

A) Take care of the input

1. Find out 'action path', which is where the submission comes from (this is determined by



mt_nav)

2. Load config for the action path to define forms etc.

3. Run action to load form data, validate input and run all the logic needed in response to the form submission if validation is ok

B) Prepare the output

4. Action ok? ->

4a. Load config for 'path' to define form, grid etc for display

Action not ok? ->

4b. 'path' is set to 'action path' (config does not need to load again)

5. Run lib for 'path' to prepare the page output

6. Include template to build the page html framework with navigation menu, content area, sidebars etc.

7. Template includes content for 'path' to assemble the actual page contents

8. Page is served to browser.

<picture>

Folder structure

```
index.lasso
_config:
  cfg__global.inc
  cfg__nav.inc
  cfg_cust_edit.inc
  cfg_cust_list.inc
_action:
  act_cust_edit.inc
_library:
  lib_cust_edit.inc
  lib_cust_list.inc
_content:
  cnt_cust_edit.inc
  cnt_cust_list.inc
```

Custom Type: mt_form Form Generator

This is what Knop started with over a year ago and is the most mature of the components. The basic idea is that forms are one of the most tedious things to handle manually in a web application. First the form fields should be shown on the edit page. They need labels, proper styling and different properties. They also need initial values to show in the form fields. The values can either come from a database lookup, or from a previous submission of the same form if there was an input error that needs to be corrected, and in that case the erroneous fields or labels needs some highlighting to guide the user. And finally the form submission must be handled by validating the user's inputs and then storing the form data in a database.

All these tasks are a perfect target to make things easier for the developer.

First we define the form. We give it a form action, and we add the fields and other elements such as submit buttons that the form should contain. The fields can have the same properties as regular html form fields do, and they have additional properties to define the options of a select menu, the checkbox options of a checkbox field set, for interaction with databases and other purposes.

Then we fill the form fields with data. It can either come from a form submission or from a database lookup. In the case of a database lookup, the corresponding database field has been declared as property for each form field.

Then we set a template for the form to define how the form should be presented in html. Or we can just use the default template.

Then we render the form on the page. We can render the entire form at once, or specific fields at a time (we can even set different templates for every field), to have the flexibility needed to able to accommodate the form in just about any html context.

The form object even generates some javascript for us that will warn the user if he navigates away from a "dirty" page (a page that has unsaved changes) and other nice features.

The next step is that the user submits the form. Now the form object makes its second entry by taking care of the form submission. Since all form fields are defined in the form object, it knows where to put each field when we tell it to load data from the form. And since it knows what kind of data is allowed in each field the form object can validate itself with a single call.

If the validation encounters an input error, the form objects prepares to show itself again but this time with the erroneous inputs highlighted.

If the validation passed, the form object comes back to our help once again and provides us with a complete pair array with field name and value pairs (the form fields knew what database fields they correspond to, remember?) that we can feed right into an inline to add or update a database record, or we can get an SQL string that we can put in a SQL statement of our liking.

Example member tags of mt_form:

```
-> addfield
-> setformat
-> renderform
-> loadfields
-> validate
-> updatefields
```

When defining the form object we can even give it a reference to a database object, and if we do that then the real magic begins. Now we can just tell the form object to "process" and it will do the right thing with the database, be it add, delete, or update. Just one line of code.

That leads us to...

Custom type: mt_database for Database Interaction

This is a core data type to handle add, update, delete, select with optional record locking (pessimistic locking with timeout). It can handle duplicate prevention to avoid adding a record again if the user reloads the submitted page. It generates key values in the form of random strings in the database, to use as "safe" key values.

mt_database primarily uses pair arrays as field specifications, which makes it easy to integrate with standard Lasso inlines, but can also use sql statements for some of the operations. When interacting with mt_form and mt_grid, pair arrays are normally used to exchange field data and other search parameters. The use of pair arrays for standard inlines is one way to provide greater flexibility.

mt_database works transparently both with SQL (primarily MySQL) and FileMaker databases.

Example member tags of mt_database:

```
-> select
-> getrecord
-> addrecord
-> saverecord
-> deleterecord
```

The result of a select operation (to get multiple records, as opposed to getrecord which gets a single record) is available to the page as a standard inlinename record set, or a records_array. Select supports true found_count even when using LIMIT with raw SQL statements.

mt_database can interact with mt_form and mt_grid.

Custom type: mt_grid for Database Record Listings

This custom type is for showing record listings with pagination, sorting, edit link etc. It requires a reference to a database object because they are so tightly related. It can highlight the affected record when returning to the list after adding or editing a record. We can also give it a reference to a nav object, to get the right pagination links and other things. It can also provide a basic "Quicksearch" function integrated with the record listing.

Quicksearch and the sort headings generate pair arrays (or sql strings) to interact with mt_database. Sort parameters and the quicksearch query is automatically propagated through a form, so the same set of records set is selected after editing a records.

Example member tags of mt_grid:

```
-> insert (to add list columns)
-> sortparams
-> renderhtml
```

Custom type: mt_nav Site Navigation

This data type keeps track of where the user is and where he can go, and validates that a user is allowed to access the current location. It is responsible for displaying the navigation menu and gives us some tools to include the right config, action, lib and content files. The navigation menu is as default rendered as a hierarchical ul/li list, which can be styled as horizontal top navigation, left or right navigation, as tabs, indented lists or whatever. Only css sets the limit.

mt_nav supports both parameter based navigation, where the current location is specified as url parameter "?-path=", as well as "URL design"-style navigation based on the URL path, where the current location is the path (for example using an atbegin url handler).

mt_nav supports navigation hierarchies in multiple sub levels with arbitrary depths.

mt_nav doesn't have to define the navigation for an entire site, it can restrict itself to a section of a site (a sub folder for example)

Example member tags of mt_nav:

```
  -> insert
  -> getlocation
  -> renderhtml
  -> renderbreadcrumb
  -> include
```

mt_nav can interact with mt_grid.

Custom type: mt_string for Language Strings

Not yet implemented

An mt_string object holds all language strings for all supported languages. Strings are stored under a unique text key.

Preliminary specification:

```
  ->onUnknownTag returns the language string for the specified text key  = shortcut
for:
  ->getstring
    required: -key textkey to return the string for
    optional: -language to return a string for a specified language
  ->setlanguage sets the current language for the string object
  ->validlanguage
    required: -language checks if a specified language exists in the string object
  ->browserlanguage returns the most preferred language as specified by the browser's
accept-language q-value
  ->languages
    optional: -language returns an array of all available languages in the string
object (of the -language array if specified)
  ->keys returns array of all text keys in the string object
```

Examples

```
$strings_messages -> loginfailed;
$strings_errors -> recordlocked;
$strings_info -> (setlanguage: 'sv');
$strings_info -> welcome;
$strings_info -> (setlanguage: ($strings_info -> browserlanguage));
```

Custom type: mt_user for User Management

Not yet implemented

Welcome to Lasso Summit 2007

LassoSoft

Special Offer

For Lasso Summit Attendees Only

Receive 10% off one order at the Lasso Store with this coupon.

Please reference **Lasso Summit 2007** in the comments field when you check out.

https://store.lassosoft.com/
lassosales@lassosoft.com • 888-286-7753

Discount will be applied when the order is processed. Billing name must be a Lasso Summit attendee. Please call or use the Comments field if the names do not match. Offer expires 6/31/2007.

Lasso Professional 8.5 – The fastest, easiest way to tie any database to any website on any platform. New version includes native data source connectors, AJAX support, and more.

Lasso Developer 8.5 – This free download allows you to develop and demonstrate Web sites locally. Free download is fully functional and allows up to five user connections

Chart FX for Lasso – Enterprise class charting for Lasso. Create graphically rich charts using a plethora of chart types, backgrounds, borders, and data analysis tools.

Lasso