Lasso Summit 06

Presented by:
Heunox Corporation

Fort Lauderdale, Florida
Riverside Hotel
February 17-19, 2006
http://www.lassosummit.com

Contents

Schedule

Friday, February 17, 2006

4:00 PM - 6:00 PM .....LPA Members Only Intracoastal Cruise of Fort Lauderdale

6:00 PM - 7:00 PM.....Registration

7:00 PM - 8:00 PM.....Keynote Address by OmniPilot

8:00 PM - 10:00 PM .....Cocktail Reception

Saturday, February 18, 2006

8:00 AM - 8:30 AM.....Registration & Working Continental Breakfast

8:30 AM - 10:00 AM .....URL Design - Methods to Use Friendly URLs by Johan Sölve

10:00 AM - 10:15 AM.....Morning Break

10:15 AM - 11:45 PM .....Strategies for Presenting Dynamic Content with Lasso by Peter Bethke

11:45 PM - 12:30 PM .....Working Lunch

12:30 PM - 1:45 PM .....Getting the Most From Lasso Studio for Eclipse by Tom Wiebe and Kyle Jessup

1:45 PM - 2:00 PM .....Short Break

2:00 PM - 3:15 PM .....Database Handling Through Custom Types by Göran Törnquist

3:15 PM - 3:45 PM.....Afternoon Break

3:45 PM - 5:00 PM.....Dynamic Sites with Lasso and AJAX by Fletcher Sandbeck

5:30 PM - 7:30 PM .....OmniPilot StarBar Reception and Cocktails

Sunday, February 19, 2006

8:00 AM - 8:30 AM.....Working Continental Breakfast

8:00 AM - 11:00 AM.....Roundtable Open Discussions

11:00 AM .....Lasso Summit 06 Ends

FLAGSHIP HOSTING.COM

800.592.6781
770.368.4992

offering:

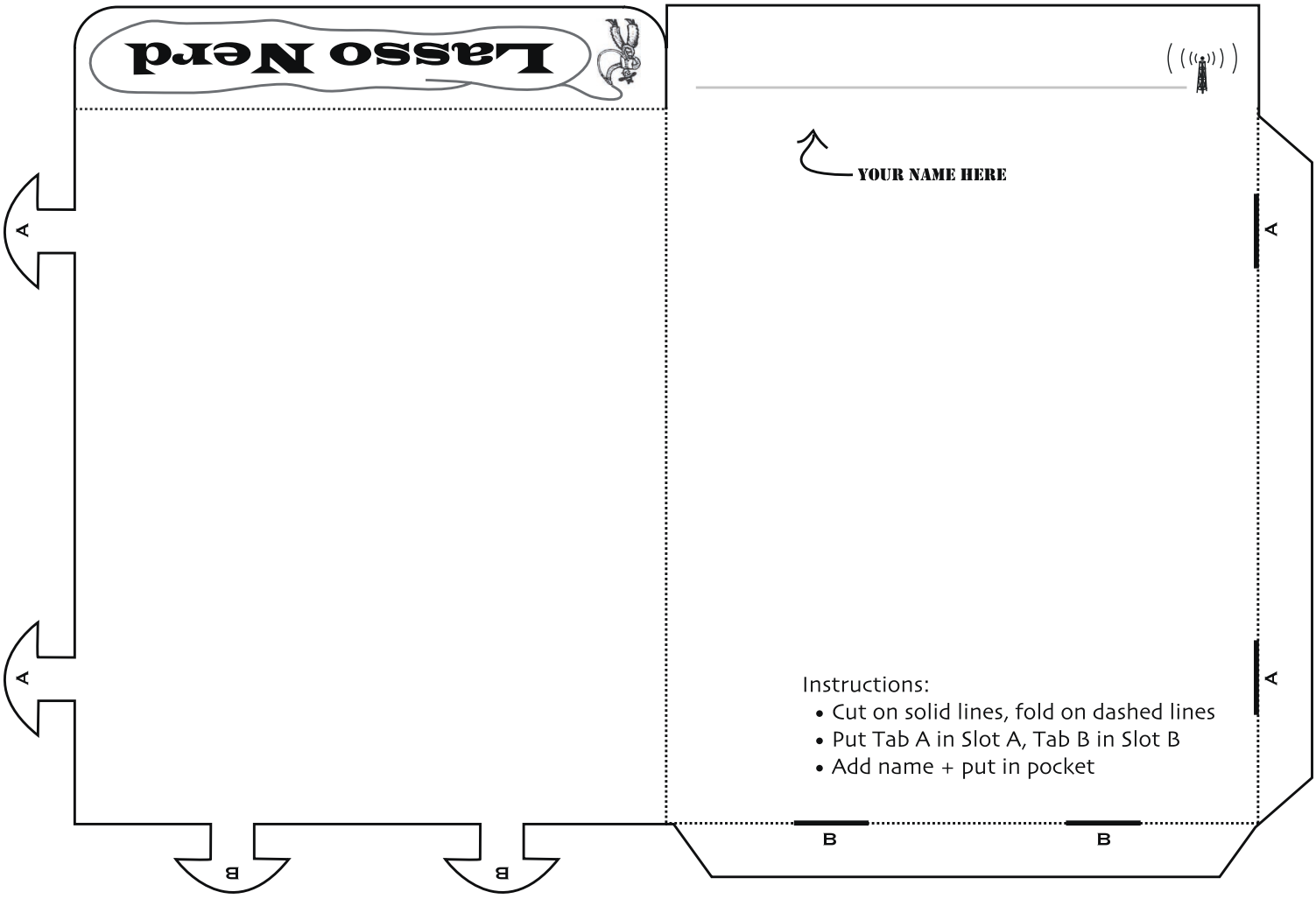
- Enterprise Email
- Spam/Virus Protection on all Email Accounts
- Webmail
- Custom Configurations
- Server Co-location
- Professional Stats
- Free Support
- Lasso/PHP

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

The Official
Lasso Pocket Protector

...spend less time cleaning up after those leaky pens and more time coding!



Wear your Lasso Pocket Protector to the Point In Space booth for a special surprise!

Point In Space

WWW.POINTINSPACE.COM

Lasso MySQL FileMaker Solutions Alliance HostedStore

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

ExecuChoice

Software Development

Founded in 1995 ExecuChoice is a company that combines skills from experts in the internet and graphics industry. We specialize in Lasso driven web applications and have developed many custom tags and plug-ins for use in the Lasso programming environment. We make Lasso do what people thought could never be done.

If you have a custom need please feel free to contact us and we will be glad to be of assistance in satisfying your every need. No matter how big or small of a project we are the ones to make it happen for you.

Steffan A. Cline
Owner / President
Phone : (602) 579-4230 Fax : (602) 971-1694
steffan@execuchoice.net http://www.execuchoice.net

Lasso JavaScript/DOM Java HTML XML CSS C/C++/C# Pascal MySQL FileMaker Pro

When writing your own ecommerce solution from scratch seems like a silly thing to do.

www.hostedstore.com

What is URL Design?

Johan Sölve

A URL is not a Filename

A URL1 is usually tied to a specific filename on the server, but this is not how things are intended. Usually it's the server technology that leads us into thinking that a URL maps to a specific filename, but URLs (or URIs in general) have conceptually nothing to do with a file system.

http://www.w3.org/TR/chips/

By disconnecting the URL from the server's file system, we get much more freedom when choosing URLs. Each URL doesn't have to correspond to a physical directory or file, and the URLs we use don't even need to have a file extension.

Choosing URLs

So what's a good URL? There is a lot of reading to do on this topic around the web. I've collected a few recommendations from different sources.

http://www.w3.org/Provider/Style/URI

http://www.useit.com/alertbox/990321.html

http://www.w3.org/QA/Tips/uri-choose

A URL should be:

- Short

  Keep URLs short to make them easier to communicate or share.
- Memorable

  Make it easier for users to return or remember a URL they've read somewhere.
- Bookmark-able

  Avoid session specific info in the host or path part of a URL.
- An aid in site navigation

  Let the URL reflect the site structure to help the visitor visualize where he is on a site.
- Guess-able

  A URL that can be guessed is a great help for visitors.
- Hackable

  Make it possible and easy for users to "hack" the URL for example to move upwards in the site hierarchy.
- Persistent ≠ Make it easy for people to link to your site

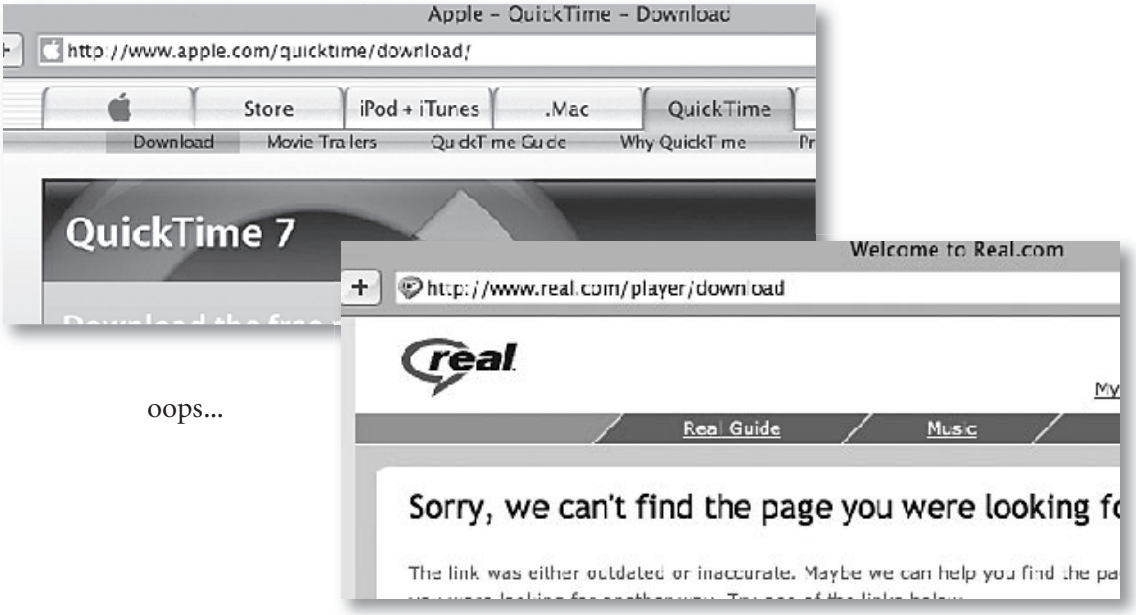
  URLs don't change: people change them.

Let URLs remain forever, do not move pages around.

Avoid causing link-rot.

"Persistent URLs Attract Links, Link-rot equals lost business" (Jacob Nielsen)

If you have to change a URL, either point to the new URL or make it very clear that the page is gone



oops...

- Technology neutral

  Avoid file extensions that is dependent of the server architecture.

  Security - avoids revealing information that is useful for a hacker,

  future proofing - allows changing server architecture without changing any URLs.
- The acid test — can you read a URL to someone over the phone?

Ways to do URL Mapping With Lasso

Apache mod_rewrite

mod_rewrite is a robust and powerful way to parse and manipulate URLs. However it is implemented completely outside of Lasso and is dependent of the web server used. It can also be a bit challenging to master the mod_rewrite rules.

Here isa configuration example that will pass any request for nonexistent files or directories to index.lasso, passing the requested URL as parameter.

```
RewriteEngine On
#RewriteBase /
# Check to see if the request is a real file or directory
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteCond %{REQUEST_FILENAME} !.*/Security
# Everything else gets sent to index.lasso
RewriteRule ^(.*)$ /index.lasso?response_filepath=/$1 [QSA,L,NS]
```

IISRewrite and ISAPI_Rewrite for IIS, WebSTAR Rewrite

Similar to mod_rewrite.

Error.lasso

This method uses a custom error.lasso file to parse the requested URL when requesting a non-existent file. The idea is to have Lasso process requests for non-existent files. However Apache won't pass the requested file path to error.lasso, instead the actual path to error.lasso is returned which makes it impossible to parse the URL that was originally requested. In addition, the request must end with .lasso for error.lasso to kick in.

Lasso Built-In using [Define_AtBegin]

Lasso 8 introduced the tag [Define_AtBegin] which lets us hook into the processing of any Lasso page processed by the server. This lets us use Lasso to parse the requested URL before running the actual page.

How to use Define_Atbegin as Pre-Processor for URL Mapping

Introducing [Define_AtBegin]

[Define_AtBegin] is a tag that defines pre-processing code that will be executed before each page on a site is executed or even loaded - for EVERY request to Lasso. The atbegin code is defined globally for each Lasso Site by calling the tag from a lasso file in LassoStartup.

This pre-processing ability gives us a way to analyze the requested URL and map it to a desired action before loading any page for it.

Since [Define_AtBegin] defines pre-processing code that will be executed for every page on a site, it is important to debug the code carefully before using it on a live server. Otherwise every page requested on the site might fail.

Advantages Over Other Methods

Using a Lasso pre-processor to perform URL mapping makes the process mostly self-contained within Lasso which reduces the complexity of the server setup. Less things to setup, less things that can fail. The URL parsing logic, decisions about not to get involved with the request, all of it can be done within Lasso. This also makes it easier to integrate the URL mapping more tightly with the rest of the site's logic, or even databases.

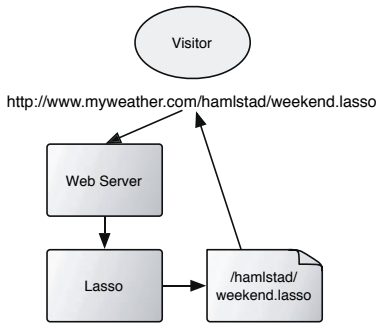
Only a minor configuration needs to be done in the web server. With Apache it can even be configured in a .htaccess file (if allowed on the server) to provide a directory-specific activation.

The pre-processor can also be used for many other useful things.

How does it work?

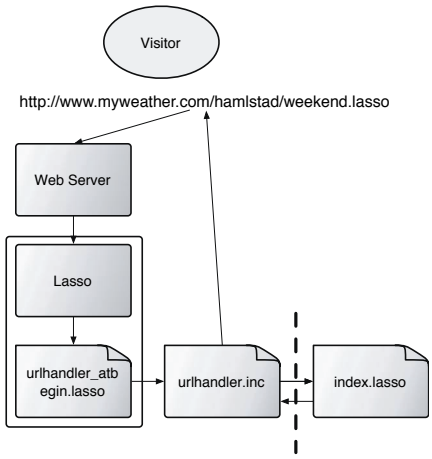
We intend to use a pre-processor to parse the requested URL before executing the page. The pre-processor will be defined by a [Define_AtBegin] call. How will the pre-processor interact with the flow of execution?

Let's begin with looking at what a normal Lasso page request looks like.



- The web server is configured to send URLs with .lasso extension to Lasso.
- The requested URL maps directly to a filename on the web server.
- Lasso executes the requested page.

This is what a page request can look like when we use a pre-processor to do URL mapping.



- The web server is configured to send the request to Lasso even if the requested URL doesn't have a .lasso extension.
- The requested URL has nothing to do with the filesystem on the web server.
- Lasso has a pre-processor defined at startup (urlhandler_atbegin.lasso) which will call a URL handler (urlhandler.inc) to have the requested URL parsed.
- After parsing the URL, urlhandler.inc will execute the site's default page or hub file, passing the parsed URL as argument.

Web Server Configuration

The web server should be configured to have Lasso process all requests for URLs that we want to parse, for example any URL without file extension. We would normally want to exclude files that have file extension to avoid having Lasso process image files, media files, CSS files, javascript files and so on. We could also limit processing of requests to specific directories.

Configuration example for Apache, to put in a virtual host configuration or in a .htaccess file:

```
<LocationMatch "^[^\.]+$">
  # anywhere without file extensions
  SetHandler lasso8-handler
</LocationMatch>
```

See "Additional Apache configuration examples" below for other setups.

For IIS 6.0 on Windows 2003 Server, you can set up a wildcard mapping for which will direct any request to Lasso on a folder by folder basis.

See " Installing Wildcard Application Mappings (IIS 6.0)"

http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/5c5ae5e0-f4f9-44b0-a743-f4c3a5ff68ec.mspx

Also see "Setting Application Mappings in IIS 6.0 (IIS 6.0)"

http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/4c840252-fab7-427e-a197-7facb6649106.mspx

Lasso Configuration

We begin with putting a Define_AtBegin script in a .lasso file in the LassoStartup folder for the current Lasso Site and then restart LassoService for the site.

We want to make the atbegin handler very generic and have as little logic in the actual atbegin-handler as possible to avoid restarting Lasso when changing the logic for URL mapping. We also want it to be completely transparent to not interfere with normal operation for virtual hosts on the same Lasso Site that don't use URL mapping.

urlhandler_atbegin.lasso

```
<?LassoScript
define_atbegin: {
  if: file_exists: '/urlhandler.inc';
    include: '/urlhandler.inc';
  /if;
};
```

This atbegin handler looks for the file urlhandler.inc in the virtual host's root and includes this file to do the actual processing, to either map the URL to an action, pass it on as a normal request or return an error. By keeping the atbegin handler simple as this, it becomes transparent so that virtual hosts that do not have urlhandler.inc in the web root are not affected at all by the atbegin handler.

[Define_AtBegin] use a compound expression to define the pre-processing code. A compound expression is a LassoScript that is placed between { }. It's a way to specify a set of Lasso tags that can be stored and executed later. When [Define_AtBegin] is called at startup, the compound expression is stored internally by Lasso, and will be retrieved and executed before every page request.

Handling the URL

Now that we have all Lasso requests for a host passing through one spot, we get the opportunity to parse the requested URL by looking at [response_filepath]. Now we can do whatever we want to map the URL to a desired action on the web site.

Here is an example URL handler that will parse URLs for the news and products sections of a web site:

urlhandler.inc

```
<?LassoScript
  // this file is called by the atbegin handler, so it is executed
  // before any page is being processed.
  if:response_filepath->(endswith:'.lasso') ||
      response_filepath->(endswith:'.lassoapp');
    // don't do anything for normal .lasso and .lassoapp requests
  else;
    if:response_filepath->(beginswith:'/news/') ||
        response_filepath->(beginswith:'/products/');
      var:'url_path'=response_filepath, 'section'='';
      $url_path->(removeleading:'/');
      if:!$url_path->(endswith:'/');
        $url_path += '/';
      /if;
      if:$url_path->(beginswith:'news/');
```

```
        // check for pattern /news/2004/12/31/keyword/
        //using regular expression
        var:'pathcheck'=(string_findregexp:$url_path, -find=
          '^news/(20\\d{2})/(0[1-9]|1[0-2])/' + '([^/]+)/([^/]*)');
        if:$pathcheck->size >= 5;
          $section = 'news';
          var:'newsdate'=$pathcheck->(get:2) + '/' +
                         $pathcheck->(get:3) + '/' +
                         $pathcheck->(get:4);
          $newsdate = (date:$newsdate, -format='%Y/%m/%d');
          var:'newskeyword'=$pathcheck->(get:5);
          var:'newsextra'='';

          if:$pathcheck->size >= 6;
            $newsextra = $pathcheck->(get:6);
          /if;
        /if;
      else:$url_path->(beginswith:'products/');
        $section = 'products';
        // split up the path in components
        $url_path = $url_path->(split:'/');
      /if;
      // run site
      // use absolute path!
      $__HTML_REPLY__ = include:'/index.lasso';
      abort;
    /if;
  /if;
?>
```

Let's walk through this code bit by bit:

urlhandler.inc

```
<?LassoScript
  // this file is called by the atbegin handler, so it is executed
  // before any page is being processed.
  if:response_filepath->(endswith:'.lasso') ||
      response_filepath->(endswith:'.lassoapp');
    // don't do anything for normal .lasso and .lassoapp requests
```

Since .lasso and .lassoapp URLs are always set to be processed by Lasso, those requests will also get into the pre-processor. But we don't want to interfere with those request so we will just let them pass through. We will only deal with extension-less URLs.

```
else;
    if:response_filepath->(beginswith:'/news/') ||
        response_filepath->(beginswith:'/products/');
```

The URL path begins with either /news/ or /products/, so we know we should look closer at the path

```
var: 'url_path'=response_filepath,
'section'='';
```

We use the variable "section" to keep track of what main section of the site we are visiting. This is used later on.

```
$url_path -> (removeleading: '/');
if: !($url_path -> endswith: '/');
$url_path += '/';
/if;
```

Trim the leading / from the path, and add a trailing / if there is none. We don't deal with paths that have file extensions here, so we don't expect to end up with a path like news/current.lasso/.

```
if: $url_path -> (beginswith: 'news/');
// check for pattern /news/2004/12/31/keyword/
//using regular expression
var: 'pathcheck'=(string_findregexp: $url_path,
      -find='^news/(20\\d{2})/(0[1-9]|1[0-2])/'
            + '([0-2][1-9]|3[0-1])/' + '([^/]+)/([^/]*)');
```

We use a fairly strictly defined pattern for the URL path for news. This breaks the recommendation for "hackable URLs" since the pattern check doesn't allow the visitor to hack off the day or month from the URL, for example if he wants to get to a news archive for a year or month. We should also add rules to allow a URL like `/news/current' to always return the most current news article.

[string_findregexp] returns a nicely split array if the path matches our defined pattern.

```
if: $pathcheck -> size >= 5;
  $section = 'news';
  var: 'newsdate'=($pathcheck -> (get: 2))
      + '/' + ($pathcheck -> (get: 3))
      + '/' + ($pathcheck -> (get: 4));
  $newsdate = (date: $newsdate, -format='%Y/%m/%d');
  var: 'newskeyword'=($pathcheck -> (get: 5)),
  var: 'newsextra'='';
  if: $pathcheck -> size >= 6;
    $newsextra=($pathcheck -> (get: 6));
  /if;
/if;
```

The result of the parsing for a news URL is a few variables:

$section tells the site what main section we are in.

$newsdate contains the publication date that we will use when looking up the news article in our database.

$newskeyword contains an identification string, usually the news headline in lowercase and with spaces removed.

$newsextra contains an optional extra item, for example `comments' that will lead to a page related to the news article.

```
else: $url_path -> beginswith: 'products/';
$section = 'products';
// split up the path in components
$url_path = $url_path -> (split: '/');
```

For the `products' section we have much more liberal rules. We will simply pass on the $url_path variable as array to use later on when looking up the current product category in our product database.

In a real world case we would continue with more rules for different site sections.

```
/if;
// run site
// use absolute path!
$__HTML_REPLY__ = include:  '/index.lasso';
abort;
```

This is where all the action happens. Index.lasso is the central hub page for the actual site, and the variables we have set above will be handled in the site code to show the desired page.

Remember, this file is run by the atbegin handler, and the code run by define_atbegin will never return any output to the web page since Lasso will clear the page buffer once the real page begins processing. Therefore we must explicitly put the page result into the page buffer, which is stored in the variable $__html_reply__.

Once we have stored the page output into the page buffer we must also stop any further processing of the request using the abort tag, otherwise Lasso would try to run the file that was actually requested, which is probably nonexistent since we use virtual paths in the URL.

```
    /if;
  /if;
?>
```

Error Handling

The entire mechanism for URL mapping relies on the fact that a URL is not a filename. That means that we can technically never end up in a "File not found" situation, so there's no natural mechanism to provide file not found errors.

Nevertheless it's quite possible to get a request for a URL that is invalid and that doesn't match anything relevant in our site. For these situations we need to provide a proper error page and most importantly a proper HTTP status code, so search engine crawlers, bookmark checkers and other automated visitors know about the error.

For URLs that are invalid we should serve a "404 Not Found" status code.

For URLs that have been valid but are no longer available we should serve a "410 Gone" status code.

http://www.w3.org/Protocols/HTTP/1.1/spec.html#Status-Codes

Here is a custom tag to easily set the HTTP status code without affecting the rest of the HTTP header:

```
define_tag: 'setHTTPstatus', -required='statuscode';
// replace status code but keep leading HTTP with version
  $__http_header__ =
    (string_replaceregexp: $__http_header__,
          -find='(^HTTP\\S+)\\s+.*?\r\n',
          -replace='\\1 ' + #statuscode + '\r\n');
/define_tag;
```

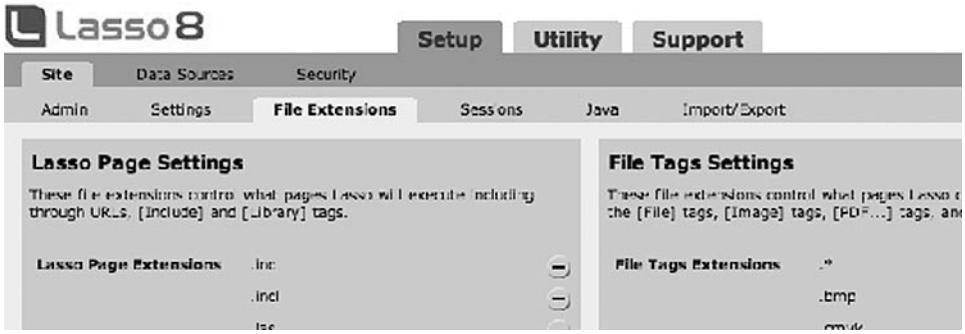
Debugging AtBegin Scripts

One common method to debug tricky code problems is to insert the [abort] tag on different places and look at what has been output on the page before getting to the [abort] tag.

One thing to note about this is that when executing a page by calling it from an AtBegin script as we do in `urlhandler.inc`, we can't use the [abort] for debugging any more. If we put [abort] anywhere in the Lasso code that is executed in index.lasso, it will result in a completely blank page. This is because it will stop any further processing

whatsoever which means that the code in urlhandlerinc that assigns `$_ _html_reply_ _` to the output of index.lasso will not be executed either. The end result is that putting an [abort] tag anywhere in index.lasso will result in a completely blank page.

Lasso SiteAdmin settings



No settings in Lasso SiteAdmin should need to be changed, but apparently it seems to be needed to add wildcard (.*) as allowed extension in the File Tags Settings under File Extensions. It is not entirely clear if and why this appears to be needed sometimes.

Additional Apache configuration examples

This configuration will have Lasso process any non-file request (no periods in the URL) only in a specific folder.

```
<Location ~ "^/site/[^\.]+$">
  # files without file extensions in specific folder
  SetHandler lasso8-handler
</Location>
```

This configuration will have Lasso process ANY request in a specific folder. This can be useful to offer protected file downloads, using the [file_stream] tag.

```
<Location ~ "^/downloads/">
  # any file in specific folder
  SetHandler lasso8-handler
</Location>
```

Other uses for AtBegin

Methods That Can Be Implemented in Urlhandler.inc

- Protection of .inc files

Protecting .inc files from being accessed directly is important to not reveal unprocessed source code, or run include files out of context. This protection requires that .inc files are set to be processed by Lasso.

urlhandler.inc

```
<?LassoScript
  if: response_filepath -> (endswith: '.inc') ;
    // .inc files are not allowed to be called directly
    $__HTML_REPLY__ = '<h1>Not authorized</h1>';
    abort;
  /if;
?>
```

- Locking down siteadmin.lassoapp

  This code makes it impossible to access siteadmin.lassoapp except from the local computer.

urlhandler.inc

```
<?LassoScript
  // make sure we're ok to use siteadmin
  if: response_filepath -> split: '/' -> last -> (beginswith: 'siteadmin.')
   && response_filepath -> (endswith: '.lassoapp')
   && client_ip != '127.0.0.1';
      $__HTML_REPLY__ = '<h1>Not authorized</h1>';
      abort;
  /if;
?>
```

Methods That Are Implemented as Separate atBegin Handlers

- Logging and timing (when used in pair with [Define_AtEnd])

```
<?lassoscript
  define_atbegin: {
    var: '_pagetimerstart'=_date_msec;
    define_atend: {
      log_detail: 'http://' + server_name + response_filepath + ' '
        + (_date_msec - $_pagetimerstart) + ' ms';
    };
  };
?>
```

- Debugging troublesome pages

  [Bil Corry] If it were me, I'd create a define_atbegin script that logs every page request to a database. Then have a define_atend that erases that db entry. The only entries left will be those that never finished executing.

- Branding

  Use an AtBegin handler install an AtEnd-handler on every page that adds an ISP logo, adds site information, copyright info or disclaimer as html comment, adds banner ads or other things by manipulating the `$ _ _ html _ reply _ _` variable before it is served to the visitor.

- Optimizing HTML code and collapsing whitespace

  Use an AtBegin handler install an AtEnd-handler on every page that removes excess whitespace and line breaks from the html source.

- Gzip compression of served HTML pages

  Tip of the week for September 23, 2005 includes a Gzip compression module for Lasso 8.1. The module allows Lasso to automatically compress all pages using Gzip for compatible browsers. The full source code for the modules is included and is a good example of how to create at-begin and at-end processes.

http://www.omnipilot.com/Tip%20of%20the%20Week.1768.8959.lasso

- Protected downloads with real filenames

  Serve a file for download using [file_stream] by reading the original file from outside of the web root, after checking that the user is authorized to download the file. By passing requests for ANY file to Lasso, the user can request the file by it's real name.

- Dynamically generated media files with real filenames

  Generate pdf or image files on the fly and serve them directly, even if the file is requested by its real name.

---

1 URL or URI? In this document we will refer to web addresses with the term "URL", but some of the referred links use the more general term "URI". In short, a URL is one kind of URI. Here we talk about web addresses, so let's stick to URL for now.

Strategies for Presenting Dynamic Content with Lasso

Peter D Bethke

> "Content is where I expect much of the real money will be made on the Internet, just as it was in broadcasting." -- Bill Gates (1/3/96)
> (http://www.microsoft.com/billgates/columns/1996essay/essay960103.asp).

1. Introduction:

In the continuing evolution of the Internet as an information and entertainment medium, the demand for dynamic, audience-driven content has increased dramatically. Moving from a "static" web site to one that can offer fresh, relevant information takes careful planning and application of Lasso's broad and powerful range of web programming tools.

But what is "Dynamic Content" anyway? There seems to be a general agreement that it is the opposite of "static" content, but from that point the definitions differ. There seems to be a general agreement that Dynamic Content is a more advanced form of web programming than "static" html, but in the modern era of complex CSS this too is arguable.

Obviously, deciding to use Dynamic Content takes a careful analysis of the reasons and necessity, and a careful assessment of the possible drawbacks and/or overhead that may result from it, such as increased server load or search engine compatibility. Some developers address these issues by filtering Dynamic Content to select audiences, also called "personalization" or "authenticated site content". Others focus on optimization, eking out every last bit of performance from their server. Many sites combine the two to create the coveted "how did they do that" reaction.

Dynamic Content can take many forms, from textual to graphic. Some is built "on the fly" and some is delivered pre-built but is considered dynamic due to it's delivery. Using Lasso's powerful PDF and Image tags, it can take the form of generated PDF documents and images. Newer applications use cutting-edge display technologies such as AJAX to push performance of web applications to levels found previously only in desktop applications.

Naturally, all Dynamic Content has to "come from somewhere". Some web applications are based on content that is generated internally, frequently in concert with a back-end database such as MySQL, SQLlite, or MS SQL Server. Others aggregate content that is generated elsewhere, using open standards like SOAP, or using simple included urls or even frame-sets.

Each of these types of applications can be built using technologies that are available to Lasso programmers. Omnipilot has provided developers with a very powerful and flexible array of tools to tackle the issue of Dynamic Content. Using Lasso it is possible to serve Dynamic Content in both textual and binary format (eg images, pdf and other formats) from databases, bypassing the need to store this information in the application directory. Further, this information can be optimized and secured using caching and encryption methods, to create truly flexible and powerful results.

2. Definitions of Dynamic Content on the Web:

A simple search in Google (enter "define:Dynamic Content" in search box) returns a number of definitions of "Dynamic Content". Here are a few that are interesting:

Dynamic Content is:

- ...web site content that can be altered or updated very easily. (www.c7.ca/glossary/)
- Information in web pages which changes automatically, based on database or user information. Search engines will index Dynamic Content in the same way as static content unless the URL includes a ? mark. However, if the URL does include a ? mark, many search engines will ignore the URL. (www.dreamweaverresources.com/seo/glossary.htm)
- Content that is generated on the web pages on the fly. Information gathered from databases or other sources depending on user request, when displayed on the web page makes the content dynamic. Pages that are created in Flash or use other animation technique are also sometimes referred to as dynamic. (www.optymise.co.nz/resources/glossary.asp)
- Content that is assembled to meet users' specific needs, providing them with exactly what they are looking for, when they are looking for it, and in the format they are looking for it in. (www.managingenterprisecontent.com/myweb/Glossary.htm)
- Dynamic Content is the ability to have the presentation of information on a web page, or other services, influenced by other factors. The servers that create the web page run computer programs that, according to a sequence of decisions, alter the content of the page in real-time. Dynamic Content could be as simple as putting the current date in a web page. At its most complex it can identify the person using the page, and personalize the information presented to the preferences they gave to the server when they registered to receive that service. (www.fraw.org.uk/library/005/gn-irt/glossary.html).

If there is a common thread with these definitions it might be that Dynamic Content is content that is presented according to some criteria that is set out by the programmer with some thought for the intended audience, and that this content is changeable based on the provided criteria.

In this case I like the spirit of the last definition, as it it speaks to the role of the developer, who through programming can cause a web server to "...according to a sequence of decisions, alter the content of the page in real-time". This criteria can be as simple as pointing to an included url or using the (date) tag to display the current date to something as complex as presenting content in a specific format at a specific time and place to a specific user at a specific ip range.

Obviously, in the case of the included url or tag, the developer has limited control of the actual content, aside from choosing the source or parameters, or the format of the output. However, the content does meet the criteria of being changeable and presented with some thought towards audience. In the case of the date tag, for example, that thought might be "what is the use of knowing the current time and date". In the case of the highly selective, or "personalized" content, it is easy to make the case that the developer put considerable time into defining the criteria that influenced its presentation.

Finally, the term "Dynamic Content" itself can be confusing as it can refer to the page itself, or elements (objects) on the page. When a definition is referring to the difficulties of optimizing Dynamic Content for search engines, it is primarily talking about the whole page, since search

engines (in general) index the page as a whole. When it talks about date tags or Flash animation it is most of the time talking about objects on a page that might be otherwise static.

3. Advantages/Disadvantages of Dynamic Content

Dynamic Content has obvious advantages. The lure of building a web application that can "run itself" with minimal interaction is great, and is the stuff of many an inticing sales pitch. Bill Gates once famously said (in 1996) that in the modern internet era, "content is king". This is becoming more and more true. Dynamic Content is a vital tool for "visitor retention", whereby visitors to a site are encouraged by what they see at first glance to return later or on a regular basis. Static content is by nature poor at true retention -- while a visitor may return a number of times to read all the content that they need, eventually they will exhaust that "well" and look elsewhere. Even worse, a visitor may "snake" (gather via a web tool) a site for offline browsing, grabbing information, taxing the server, but never actually visiting the site.

The key to visitor retention is to provide information that changes and adapts to time, date, audience, etc, and forces (with a "velvet glove" of course) viewers to return repeatedly to get the "latest", while not appearing limited (or limiting) in any way.

Most times Dynamic Content presented as styled text or a mixture of text and html markup (ie imbedded images). However, advanced techniques of Dynamic Content can go as far as to authenticating and delivering binary content "on the fly" for select audiences. This method, discussed later in this paper, requires a database or an application server that can differentiate between visitor types (or groups). Once implemented, though, the results can be quite powerful, as binary content like images, pdfs, word documents etc can be stored as serialized data and served in a controlled, secure manner that is far more efficient than simply showing or hiding links to "physical" (ie on-disk) files.

The actual implementation of a solid, dynamically-generated application can be very complex. A lot of the complexity comes from the creation of the rules that govern the content itself. Once those rules are created, the code naturally needs to mimic them, and at the same time take into account scenarios that are infrequent and out of the normal scope, such as malformed urls or hacking attempts.

As mentioned in one of the previous definitions, Dynamic Content can pose a problem to indexing services such as Google and Yahoo. Indexing services were first introduced in a time when "static" html pages were the norm, and cgi-based pages were used primarily for simple gateway tasks like email forms. In "modern" times entire applications are built with Dynamic Content -- frequently out of smaller objects that are welded together to create a "rendered" page. This type of page assembly usually requires a scripting or compiled language of some kind, like Lasso, Php, .Net, Java etc. These languages by default use extensions like .lasso, .php etc which tip off the search engine to it's content.

All of these engines deal with Dynamic Content differently, and SEO (Search Engine Optimization) is an art entirely to itself. Of course, truly Dynamic Content is antithetical in a way to the function of the search engine. Because search engines work best with data that remains fixed, Dynamic Content (such as the top news stories on a news site for example) poses problems with search engines that crawl the web periodically - an indexed link might be time-dependent and "vanish" by the time a user accesses it from a search list.

There is also the question of performance, i.e. the load that certain dynamic elements can place on a web server. Many application design decisions are based on a delicate balance between power, complexity and performance. If a web page is made up of multiple Lasso inline "blocks", each representing a separate query, the resulting overhead can potentially slow down server and data source performance, even in a multithreaded environment. A lot of times Dynamic Content can be optimized by spreading a wider net with an initial query and filtering the results using arrays, maps and iterations. Further, using the new cache tags, it is possible to cache the results of complex queries as serialized objects and to call the data when needed, or to refresh and replace it when it is no longer current. In this way the developer can foresee the demands that certain tasks may impose and set up routines that minimize those demands.

The bottom line is that no matter how elegant a site may be, if it is too slow it will have difficulty retaining visitors. This defeats one of the primary reasons for using Dynamic Content, visitor retention, and can make all the hard work of a developer (and client) go to waste.

4. Types of Dynamic Content

There are two major classes of data being served on the web - textual and binary. Textual data usually comes in the form of stylized markup, like HTML or a combination of HTML and CSS. Binary data comes in the form of images, video, PDF, flash animations, etc. Dynamic Content can be any combination of these elements. It can refer to the page that serves the image or the image itself - the definition is largely based on context. Again, what makes it dynamic is that there is some rule or set of rules that governs its appearance, placement and/or accessibility.

Dynamic Content can originate internally, i.e. from a database associated with the application, or externally, from another server or servers. Internally accessed data is the most common form of Dynamic Content used by Lasso developers. It is usually generated by a query to a registered data source, like MySQL, and called by using a named inline or passed to an array either in a raw form or encapsulated into a CType. However the storage and delivery method, at some point it ends up on a display page as text or binary content, subject to the rules governing its appearance and duration.

External referencing and/or presentation of Dynamic Content is sometimes called "Content Aggregation" or "Portalled" content. Some content providers such as Google provide hooks into their own system though SOAP (Simple Object Access Protocol), an XML-based "Messaging Framework". Unlike frames or even externally included urls, SOAP-generated content can (usage guidelines permitting) be made to look like it originates internally, complete with formatting, rules etc (Using the SOAP_DefineTag Lasso ctag introduced in LP8, it is possible to use remote SOAP services as easily as calling local tags).

Javascript can be used in conjunction with Lasso to create Dynamic Content. Javascript has the advantage of widespread browser support, and equally important, browser integration. Browser vendors allow javascript varied levels of access to GUI elements and system information that cannot be duplicated by Lasso. In addition, Javascript is a very capable scripting language that can deliver Dynamic Content by itself or in conjunction with Lasso. Similarly Java, which is more than capable of delivering Dynamic Content, can be integrated with Lasso using the LJAPI (Lasso Java API) suite.

Recently there has been increasing buzz the Lasso community about AJAX (Asynchronous JavaScript and XML). AJAX is not Dynamic Content per se but rather a means to displaying Dynamic Content, much like traditional html-based tag systems but infinitely more efficient. AJAX effectively eliminates or greatly reduces one of the major complaints of Dynamic Content, which is load-time. This can bring web application performance more on par with desktop applications, which itself is the goal of those like Microsoft who see the future as a massive distributed system of centrally leased applications tied together by the internet.

AJAX is an amazing technology that is growing in leaps and bounds, and currently it's only drawback is compatibility with older browsers. However, as with any technology (like CSS for example), AJAX will eventually become very widespread if current trends continue, and it will prove to be a valuable tool in an lasso developer's toolbox.

5. Examples/Scenarios for using Dynamic Content in Lasso

The simplest application of Dynamic Content would be a Lasso substitution tag, such as [date] or [server_date]. When placed on a page, this tag will simply indicate the current date as returned by the server. However, suppose that the developer wanted to vary the date based on the approximate location of the client who was browsing the site. After all, the date and time in Charlottesville, Virginia may be useless to someone in Shanghai, China. It is simple enough to grab the client's ip address using the [client_ip] tag. From this step there are databases available that will match ip addresses to physical locations (humorously referred in internet lingo as "cyberspace to meatspace").

It's not an exact science, though. For example, I recently visited one of these services, ip-to-location.com, which showed that I was located in Florida. A quick check out the window at the snow on the ground showed that I was not. The service had keyed to the dynamically-assigned ip address from my provider (Sprint DSL), which probably originates in Florida. However, it did place me on the east coast in the EST time zone, in the USA. This information could be used in conjunction with the [date_format] to adjust the time-zone offset, based on the client's assumed location and the location of the lasso server. Similarly, one might wish to modify the browser header to specify the language or character set used to deliver content based on physical location. This technique can be very inexact due to a variety of factors but it does show one example of building up from a simple substitution tag to something more complex by mixing various lasso tags. Lasso provides a nice set of tags that are derived from the client header (which are all prefixed by "Client_"), such as [client_type], which can tell you the type of browser that the client may be using. Again, this information can be unreliable but it does provide a starting point for thinking about how information can be selectively delivered based on the needs of the client. Google, for example, makes use of location technology to serve ads and to route visitors to its various language-based portals.

More complex examples involve database interaction. For example, a news-portal site may keep a table of news stories, each record containing columns for title, author, body, etc. Simply displaying these records might be considered Dynamic Content. However, adding display date fields like "start_date" and "end_date" allows the developer to make the display of the content time-based. If the manager of a news portal wished to display a story onto from January 23rd to February 18th, s/he could set the start dates and end dates in the record, and then when running the query compare the value of [date] to these values (formatted of course for the correct data

source). If the current date fell between the start_date and end_date, the story would appear. If not, it would not be returned in the row set and thus not displayed to site visitors. In this way, it is possible for a site to "run itself" in terms of display. The data "front-loading" (i.e. gathering and entering the records) is obviously still a manual form-based or programatically automated task. Combined with the previous method of ip-tracking, it might even be feasible to deliver a combination of time and location-based news stories to site visitors.

Altering site content based on external information (i.e. information gathered from the client browser, ip address etc) is inherently tricky and often unreliable because the information originates outside of the application's sphere of influence. If a site user can be differentiated internally, subject to rules completely in the control of the developer, the reliability of Dynamic Content can be greatly increased. Internal differentiation of site visitors is often referred to as "Authentication". It usually involves a site user logging in, by applying username and password vs a table of approved users. The result of this action is that the application can track the actions and location of the user by matching a url-based session id vs a table of session variables. One of these session variables is frequently a user key (most likely an integer), which can be matched to the user record (frequently a primary key field) at any time to get information such as user name, etc.

Once a developer can reliably confirm the identity of a site visitor, s/he can begin to tailor content based on that visitor's needs. One type of information stored and frequently used to tailor content is "site preferences" or "user profile". Traditionally stored in site cookies, site preferences and profiles are much more reliably stored in a site database and related to a site user via authentication. Preferences or profile information may range from age, sex, etc. to geographical location, areas of interest, and language. Because this information is entered by the user, not gathered from the user's browser connection, it can be considered more reliable (assuming the user is telling the truth). The preference or profile values can then be compared to rules set up by the developer and included in a query. Say for example the news manager wanted to tailor information based on age range as well as date, this could be accomplished using preference and/or profile settings and information if that information had been previously entered by the visitor and stored.

Many developers find a need to assign site users to groups, instead of relying on user-specific information. This has two advantages -- first, it means that a developer does not have to rely on user-provided information to tailor content. It's a fact of web development, users sometimes lie or bend the truth when filling in profiles. Second, once a users/groups system is implemented it is possible to mix and match group assignments to give varying, sometimes overlapping levels of user access. Obviously, the users/groups system is fundamental to operating systems like Unix (and MacOSX of course), and is built into Lasso security. However, it is possible for a Lasso developer to "roll his own" users/groups system to govern site content. Once this is done, it becomes possible to assign group access to Dynamic Content on the query level.

How might this be accomplished? First, lets assume that the developer has two tables, one for users and one for groups. The group table includes the group name, description etc, and of course a primary key which may or may not be the group id. The user table contains first name, last name etc.

Group assignment can be done one of two ways. Either a user can be assigned to a fixed number of groups (for example with a group_id column in the user record which corresponds to the

group id key field in the group table), or a variable number of groups using a related assignment table. Either way it should be possible for the developer to query either the user table or the related group assignment table to determine the groups that a user belongs to. Once this is done these group id's can be queried against a table of Dynamic Content to determine what content can be displayed to the visitor.

Determining what content is associated with what group id can be accomplished in a similar manner. If our developer uses the previously built news story table, with author, title, body etc, s/he can link news items with groups via a group_id key field (locking the content essentially to one group) or a related assignment table that associates group_id with news_item_id. The latter method is much more flexible, as it can make an unlimited amount of associations, which can be also removed without touching the data in either the group or news content tables.

The end result of this is that content queries are generated based on user group, and thus content can be tailored to specific groups of site users without having to rely on browser information. If related tables are used, it becomes possible for developers to create management "back-end" GUI applications to handle the assignment of the various combinations of user, group and content id's.

For example, user "A" might belong to group "A", through an assignment table that associates his user id with the group "A" group id. However, the content that he wants to see is currently associated with Group "B". When he logs in, he cannot view it because the query that governs what is displayed only shows that content that he is authenticated to see, and he is not assigned to a group that can see this data. In order for user "A" to see this content, he must be either 1) added to group "B", or 2) the content that he wishes to see must be associated with group "A". This demonstrates the flexibility that the developer can provide for the site administrator in showing or hiding content. If group "B" was a group that required a monthly fee to belong to, this scenario could easily become the basis for a news site that offered "premium content" to paid subscribers. Non-paid site users might all belong to group "A", and "promoting" a user after payment would be as easy as assigning or associating a new group with his user account/profile.

This can be used with textual output naturally but also with binary content as well. As discussed previously, binary content can be images, pdf files, word documents etc. If binary content is stored as a record in a database table, for example a mysql binary field or longtext field, it can be treated with the same criteria as text output. In other words the same kind of query that matches a text-based "news story" might also match a record that contained a serialized image or pdf file. Using the lasso [image] tags, it is then possible to take the serialized information and serve it to site visitors as an image etc, just as a block of text might be.

The actual tags and code used to serve this image or binary data are more complex than displaying simple text, but the concept is the same. Once Dynamic Content, be it textual or binary, is stored in a database, it can be subject to queries that take into account user group assignments, date ranges, user information, or any number of criteria. Once this is accomplished, a developer can build the overlapping layers of display criteria that can lead to a truly powerful application.

One drawback of serving dynamic, binary information is that the overhead can be very taxing on a server, particularly a database server. It might be able to handle a small amount of users triggering a query which returns a large stream of binary or serialized text, but if a large

amount of simultaneous queries hit a server at one time, the result can be a serious slowdown of performance. This can be helped by query caches (for example the MySQL query cache) if they are supported, but since the query is specific to the current user, its only helps if that user views the binary information multiple times (on a page refresh for example), not the group as a whole, or if the query does not specify the user_id (in which case the query might be cached for a whole group of users).

One recent addition to the arsenal of Lasso Developers is the [cache] tag, which allows developers to cache portions of page output. While this can be done a page level, caching can be problematic if a developer relies on the whole site being dynamic. For example, if a whole page is cached, a page refresh might not reflect other dynamic changes in the site, like ad banners, etc which can occur in "real time" with every page refresh or location change.

One solution is to use the [cache_object] tag to cache large binary objects, and to serve these objects from the cache (possibly stored as a serialize array or map) instead of from the database. The Lasso cache tags also allow developers to compress cached data into byte stream format, and to encrypt it as well. The specific methods for this are beyond the scope of this document, which is intended to be a general overview. However, it is worth using the cache tags, in particular the [cache_object] tag in situations where database overhead is anticipated. Effective use of these tags can significantly improve server speed, which of course is one of the primary requirements of customer retention.

Summary:

In the continuing evolution of the internet, "Content is King". The ability for a web developer to deliver fresh, focused content is critical for those types of applications that rely on customer retention and search engine visibility. There are many strategies for approaching the effective delivery of Dynamic Content, but most require an understanding of the needs of site visitors, and the creation of rules that tailor content to these needs. Lasso provides many tools for delivering Dynamic Content, and it is up to the developer to match the method with the needs of his or her application. If the needs of the application environment, such as performance, user tracking, and display criteria, can be effectively balanced against the needs of the client user, the end result can be extremely powerful.

Getting the Most from Lasso Studio for Eclipse

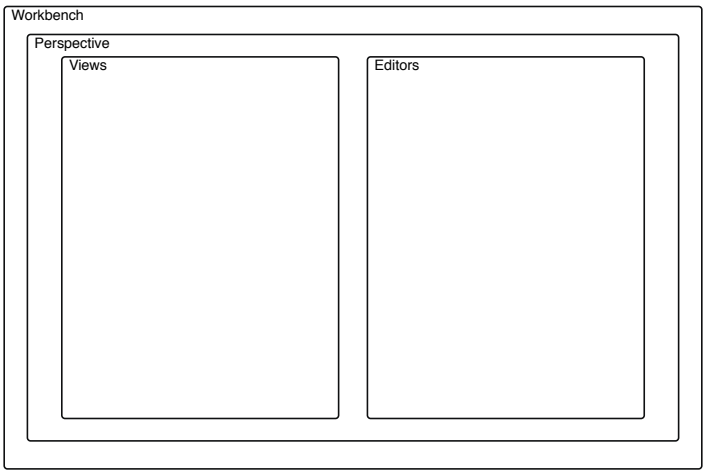
Kyle Jessup & Tom Wiebe

Using the Eclipse IDE

Understanding Eclipse

The Workbench

"The term Workbench refers to the desktop development environment. The Workbench aims to achieve seamless tool integration and controlled openness by providing a common paradigm for the creation, management, and navigation of workspace resources.
Each Workbench window contains one or more perspectives. Perspectives contain views and editors and control what appears in certain menus and tool bars. More than one Workbench window can exist on the desktop at any given time."



Perspectives

"Each Workbench window contains one or more perspectives. A perspective defines the initial set and layout of views in the Workbench window. Within the window, each perspective shares the same set of editors. Each perspective provides a set of functionality aimed at accomplishing a specific type of task or works with specific types of resources. For example, the Java perspective combines views that you would commonly use while editing Java source files, while the Debug perspective contains the views that you would use while debugging Java programs. As you work in the Workbench, you will probably switch perspectives frequently.
Perspectives control what appears in certain menus and toolbars. They define visible action sets, which you can change to customize a perspective. You can save a perspective that you build in this manner, making your own custom perspective that you can open again later.
You can set your General preferences to open perspectives in the same window or in a new window."

- Lasso Studio for Eclipse provides the Lasso Perspective
- Open perspectives from the "Window > Open Perspective..." menu.

View

"Views support editors and provide alternative presentations as well as ways to navigate the information in your Workbench. For example, the Navigator and other navigation views display projects and other resources that you are working with.
Views also have their own menus. To open the menu for a view, click the icon at the left end of the view's title bar. Some views also have their own toolbars. The actions represented by buttons on view toolbars only affect the items within that view.
A view might appear by itself, or stacked with other views in a tabbed notebook. You can change the layout of a perspective by opening and closing views and by docking them in different positions in the Workbench window."

- Also, Fast Views, hidden pop-up views. Handy for seldom used Views.
- Lasso Studio for Eclipse provides the Lasso Script HTML Result View.
- Views may be associated with a particular perspective by default but, can generally be opened in any perspective.

Editor

- Most perspectives consist of an editor and one or more views
- Editors can be associated with a particular File type, if there is no associated editor, Eclipse uses the OS to launch the file in it's default system editor (i.e. bbedit for CSS files)
- The left margin contains error flags, bookmarks, breakpoints for debugging or todo's

Project

Repeat 3 times:
Eclipse is not just a text editor
Eclipse is not just a text editor
Eclipse is not just a text editor

- All your work files must be contained within a project. A lasso project in the case of Lasso Projects.
- The project defined build settings (i.e. integration with Lasso Developer)
- You can open and close projects to save system resources and 'Go into' projects to focus on only the task at hand in the navigator

Installation and Configuration

Thoroughly covered in the Lasso Studio for Eclipse manual.

The Lasso Perspective

Coding

Lasso Project

Associates the file with the Lasso Builder and enables syntax checking, running and debugging of lasso files.

Lasso Script File

Just what it sounds like. Default Lasso Script contents can be defined in the "Lasso Studio > Script File Templates" preference pane. Allows you to add copyright info, SCM keywords or boilerplate code

The Lassoscript Editor

Syntax Colouring

- Works for all 3 styles (Square Bracket, Lassoscript and Parenthesis)
- Even works on a mixture of styles in the same document
- Bracket/Parenthesis highlighting
- Highlights the following elements:

  Tags, Types, Constants
  Keywords (attributes)
  String Literals (quoted strings)
  Numeric Literals (integers and decimals)
  Operators (+,-, etc)
  Page Variables ($myVariable)
  Local Variables (#myLocal)
  Comments (Single line // comments or C-style /*...*/ comments

Code Folding

- Allows you to collapse and expand container tags within your document.
- Makes it easier to navigate complex pages, collapse all but the block of code you're working on.
- Makes it easy to ensure you've closed all container tags properly, your source file becomes an outline.

Code Completion

- Typing the first few characters of a tag or variable name presents a list of tags found in the Lasso Reference, or Ctags within your project
- Typing a $ or # will automatically bring up a list of variable name recommendations.
- At the top of the list will be Code Templates, discussed below
- 'Show Definition' (F3) will open include files or ctype/ctag definitions

Tool Tips

- Draws info from the Lasso Reference
- Also works for Custom Types/Tags using the -Description attribute
- Pressing F2 will present an expandable floating window containing the entire description. You can select and copy text within the description, very handy for copying/adapting example code into your existing page

Code Templates

- Tab between insert elements
- Able to show available datasources and tables in an inline

- Use to help remember the syntax for seldom used/complex tags or ctags
- Can be exported to xml format for distribution and sharing

Example Template File

```
<?xmlversion="1.0"encoding="UTF-8"?>
<templates>
<templateautoinsert="true"context="lasso"deleted="false"
description="Page Var"enabled="true"id="var1"name="var">
var('${name}'=${value})${cursor}</template>
</templates>
```

- Template text will be formatted upon insertion to the current formatting preferences

Caveat*: the current version of LSfE (1.5.1) cannot include $variable style elements in a template definition. i.e. inline($params,${database}... will not work, instead, use the long hand form of the variable inline(var('params'),${ database}...*

Code Formatting

- Keeps your code neat and tidy.
- Helps ensure consistent style across multiple developers.
- Lasso has 3 official syntax styles, known commonly as Classic (Square Brackets), Lassoscript (tag:-property) and Parenthesis tag(-property). The Code formatter can switch between these styles with the click of a button, allowing the developer to code in the style most convenient to them and deliver code in the style preferred by their employer/client/ what have you.
- Can set indentation preferences, Tab or up to 5 space characters.
- Text wrapping to avoid long lines.
- quote character preference, single or double quotes.
- Maximum plain text length - How long a string will Lasso include in a lassoscript element?
- Makes working with Version control systems easier as your code is more consistent.

Outline View

- Drag and drop code sections.
- Cut and paste code blocks directly in the outline.
- Useful for navigating your files, double click on any item to navigate to it/select it.
- insert tag/type templates via contextual menu

Custom Tag/Type Template

- Provides a simple interface to help create Tags and Types
- Easily enter tag name, namespace, descriptio, options, and parameters for your tag or type.
- Custom type template allows automatic creation of CallBack tags as well.
- Create member tags within a custom type via contextual click on the type name in the outline.

Automatic Syntax checking and error reporting

- Checks document syntax upon document save and marks errors with a red X in the editor margin along with a notation in the problems window.
- Errors will show for all open projects, close other projects to show only relevant errors.
- Errors can be filtered by severity, type or resource scope (All Projects, This Project, This File etc).

Bookmarks and Tasks

- Eclipse allows you to set bookmarks in your source files, to facilitate jumping to particular places in your project quickly and easily
- Tasks are useful to note features remaining to implement, code that needs cleaning up or leave notes for other developers
- Create either through right clicking the margin on the line you wish to add the Bookmark/ Task or by simply adding a comment prefixed with “MARK” for bookmarks or “TODO” for tasks
- Better to use the comment method, in case other developers might be editing the project outside of Eclipse. Also handy to keep as much info within your files as possible
- Bookmarks and Tasks for open projects all appear within their respective Views

Running Scripts

- Runs a script and prints output to the Lasso Script HTML Result view
- Speaks to Lasso via the SOAP protocol, can work with either a local or remote Lasso Developer installation.
- Includes files based on the current project definition.

Warning*: This actually runs the script so any database or file tags actions will be executed.*

Run Configuration

- Requires a valid Lasso username/password within the ‘Lasso Studio for Eclipse Users’ group within the current lasso site.
- Edit the Laso Server WSDL parameter to specify a server other than localhost. i.e. a remote server or a locally defined hostname to access a particular lasso site.
- You can define multiple Run configurations for a particular site, each running a specific script or, you can set a ‘Preferred’ configuration along with a default file. A preferred configuration will present you with a selection box to select the script you wish to run on each execution.
- Runtime errors will be reported in the Lasso Script HTML Result View.

Debugging Scripts

- Like running but, interactive. You can view the variables as you step through your file and edit them to test different outcomes
- Use Breakpoints to suspend processing of your file in the debugger
- Tip: break your code up into multiple lines, instead of having a lot of tags on one line of code. The debugger works line by line so, you can’t stop multiple times in the same code

block if it's all on one line, nor can you evaluate values from the middle of a line of code, just the output at the end of the line.

Triggers

- A debugging session can be triggered either directly within Eclipse or via the input of an external browser
- Simulate Request - The request is sent from within Eclipse itself, along with any extra headers configured within the run configuration. Best option if the script doesn't rely on external factors (i.e. form input, cookies, get requests, client type, etc)
- Wait for Web Browser - Awaits a request from an external browser. Allows execution of the script with all environment variables in place.
- Can set a default method in the debugging tab of the current run configuration

Stepping through your program

Resume (F8) - runs the current script to it's end or the next breakpoint.

Suspend - suspends processing of the current script

Terminate - Terminate execution of the current script permanently. ie. 'Abort'

Step Into (F5) - The next expression on the currently selected line is executed and suspends on the next executable expression. i.e. 'run next tag'

Step Over (F6) - Like Step Into but, will run any include or library tags in one step. i.e. 'step over includes'

Step Return (F7) - Allows execution of the current script until it's end. Execution halts when an include or library tag are encountered, after the include is loaded. i.e. 'run to next include'

Debugging Specific Views

- Debug - Control execution of the current script
- Breakpoints - Shows current breakpoints and allows you to turn them on or off, as well as delete them entirely
- Variables - shows the current variables and allows editing of them (via contextual menu. Can also show type names.

Advanced Techniques

Using with Lasso Sites

- If you configure a specific hostname on your machine and within the runtime and debug environments, Lasso will execute the script using the appropriate LassoSite. Likewise for LassoSites defined by path name.
- Remote Debugging
- Edit the Lasso Server WSDL parameter in the run configuration to execute the selected file on a remote Lasso Server
- Note, due to security and performance considerations, it is not wise to debug scripts on a live, production server.
- Integration with other Eclipse Plugins and Command line tools

- Eclipse provides a wide variety of tools for dealing with many different tasks, both open source and commercial. Among these are included:

  Database Editors
  XML Editors
  HTML, Javascript and CSS Editors
  Java development (built in)
  C/C++ development

- Varying levels of support for most scripting languages, i.e. Python, Perl, Ruby, PHP
- This is in and of itself likely one of the strongest features of the Eclipse platform, you can do most of your coding work within one tool, lowering support, upgrade and training costs.

  see http://eclipse-plugins.2y.net/eclipse/index.jsp for more Eclipse Plugins.
- Multiple Platform support — Eclipse Works the same on Mac, Windows and Linux. You can move freely between platforms without notable differences in workflow or features.
- Team Features — Eclipse features excellent built in support for CVS version control and add on tools for other SCM (Source Code Management) tools such as Perforce

Database Handling Through Custom Types

Göran Törnquist

Introduction

Lassoscript allows you to access databases through inlines. The specifics of those databases are setup once through the administrative interface, and then we refer to them using the inlines. This way of deferring settings and then connecting functionality to them is called abstraction of the database handling.

This paper is about how to use Custom Types to make handling of records in database tables. The start is a basic custom type which will show how to create a custom type as such. In the end of the paper, you have a small set of custom types that solves a number of issues that the web developer has deal with on a daily basis.

It is the intention that you will see the benefits from using custom types together with database access to make your database implementation easy to maintain and easier to test for quality assurance. While covering the custom types part, you will also learn some about custom tags.

This covers one way of approaching database access in a object oriented way. It is not meant to be a complete guide to object orientated programming, neither does it employ the perfectly secure implementation of database access - we will actually bypass the Lasso database security scheme as much as possible to make things simpler.

Securing the code will be left as an exercise to the reader since it would be a too complex subject to cover in such a short time. In other words, I've kept the implementation as simple as possible to focus on the custom type and database part.

Error handling has also been left out to leave space for conceptually important code.

If you're more experienced, skip to the last two sections. Otherwise, please read on while I walk you through constructing a simple custom type.

What is a custom type?

Custom types, or commonly ctypes, are the way Lassoscript allows you to define your own data types instead of being limited to the types that are defined in Lassoscript from the start.

Let's look at a custom type using an example:

A rectangle can be defined to cover four points in a coordinate system; a left, top, right and bottom position.

```
<?Lassoscript
Define_Type: 'Rectangle';
  local: 'top' = 0;
  local: 'left' = 0;
  local: 'bottom' = 0;
  local: 'right' = 0;
```

```
/Define_Type;
var: 'myRectangle' = Rectangle;
//creates an instance of the custom type
?>
```

It's great to be able to create a variable that automatically has the four points needed to define the rectangle. But a real world example will prove to us that we need to find out the width, length and area of the rectangle.

This is the procedural way of solving that need:

```
<?Lassoscript
Define_Tag: 'rectangleWidth', -required='left', -required='right';
  return: (math_abs: #right - #left);
/Define_Tag;
$myRectangle->'right' = 100;
$myRectangle->'bottom' = 50;
var: 'myRectangleWidth = (rectangleWidth: $myRectangle->'left', $myRectangle-
>'right');
?>
```

To me the object oriented way feels both easier and cleaner:

```
<?Lassoscript
Define_Type: 'Rectangle';
  local: 'top' = 0;
  local: 'left' = 0;
  local: 'bottom' = 0;
  local: 'right' = 0;
  Define_Tag: 'getWidth';
  return: (math_abs: self->'right' - self->'left');
  /Define_Tag;
/Define_Type;
var: 'myRectangle' = Rectangle;  //creates an instance of the custom type
$myRectangle->'right' = 100;
$myRectangle->'bottom' = 50;
var: 'myRectangleWidth = $myRectangle->getWidth;
?>
```

By defining the custom tag getWidth within the custom type we have been given a way to actively use the data defined in the custom type. This is commonly referred to as encapsulation. The retangle custom type "knows" how to operate on the data.

In due time the programmer will find out that someone might be feeding the custom type with erroneous data and then the rectangle ctype will evolve. Anywhere where we use the width calculation on the rectangle ctype will be benefiting from this change.

```
Define_Tag: 'getWidth';
  return: (math_abs: (integer: self->'right') - (integer: self->'left'));
/Define_Tag;
```

Will I produce faster and more compact code?

No. Abstraction and generic handling all comes with a cost. Since a ctype is concerned about a generic situation, everything that could occur has to be handled in a generic way. The upside is that it will possibly handle quite a few different situations. The downsides are that you will type more and the code will not be as easily optimized as in the procedural way.

So why use custom types at all?

The simple and quite generic answer is: To reach the goals for your web application in a way that makes it easy to produce, maintain and document your solutions.

Predictability: By using custom types, you will be able to create data types that always has a specific set of attributes and behave in a specific way. Therefore, the use of these will lead to results that are easier to predict.

Reusability: Thinking in an object oriented way often leads to generic solutions of problems. Used the right way, this can save lots of development time when similar problems needs to be solved in different parts of your application or applications.

Abstraction: When using an object oriented approach, you speak for example of People and how they behave themselves and relate to other things in your system.

Less complexity: Through abstraction, you can hide complicated tasks and constraints. A good example of this is the PDF_Doc custom type that let you produce a PDF file in relatively easy way. There are quite a few more examples that comes with Lasso from the beginning.

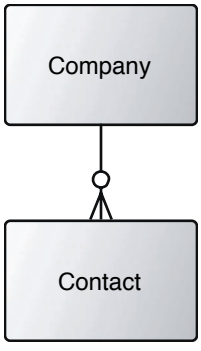
Why, in particular, use custom types for database access?

First of all, code controlling database access is very often created when needed. The need at the moment often defines how far you're thinking when accessing the database. If anything changes with the database definition or the need behind the database access, then you will need to go through every place where you've been accessing the database and walk through the process of creating or changing code, which leads to testing code, which leads to debugging code.

The database access is scattered throughout a web application, and we need a good way to make sure the data is there when we need it, and that it will be valid data. Custom types and database access paired together makes it possible to define the database access in one place, and then allow the scattered code to access the data in a well defined way.

The example

Custom types and the database modeling are forming a good relationship. This will easiest be shown through an ERD (Entity Relation Diagram).



I will use a simple example throughout this paper. It is a database containing contacts belonging to a company. The MySQL schema is given below.

```
CREATE TABLE `contact` (
  `key` bigint(20) unsigned NOT NULL auto_increment,
  `firstname` varchar(64) NOT NULL default '',
  `lastname` varchar(64) NOT NULL default '',
  `company_key` bigint(20) NOT NULL default '0',
  PRIMARY KEY  (`key`)
);
CREATE TABLE `company` (
  `key` bigint(20) unsigned NOT NULL auto_increment,
  `name` varchar(64) NOT NULL default '',
  PRIMARY KEY  (`key`)
);
```

I hope those of you that uses FileMaker or any other type of database will be able to read and recreate the database from the given schema.

A simple custom type

Taken from example01/code.inc

```
Define_Type: 'C_Contact';
  local: 'firstname' = '';
  local: 'lastname' = '';
  Define_Tag: 'setName', -required='firstname', -required='lastname';
  self->'firstname' = #firstname;
  self->'lastname' = #lastname;
  /Define_Tag;
  Define_Tag: 'getName';
  return: self->'firstname' + ' ' + self->'lastname';
  /Define_Tag;
/Define_Type;
```

This example outlines the definition of a simple ctype named Contact. From this moment on we will be able to define a variable to be a Contact.

```
var: 'speaker= Contact;
```

We can also start using it with the same syntax we use with the built in types.

```
$speaker->(setName: -firstname='Göran', -lastname='Törnquist');
```

compared to manipulation of a simple string variable

```
var: 'myName' = 'Göran Törnquist';
$myName->(replace: 'ö', 'o');
```

From the syntax viewpoint there is no difference between the code accessing the Contact and the String variables. Though, obviously, they are not equal in function or in contents. The conclusion is that there's nothing magic or special with ctypes. You can access them in the same way that you normally access any type of variable.

Accessing the attributes

Taken from example02/code.inc

```
Define_Type: 'C_Contact';
  local: 'key' = 0;//unique key to identify record
  local: 'firstname' = '';
  local: 'lastname' = '';

  //*** previous code for member tags taken out for reasons of space ***/
  Define_Tag: 'getKey';  //getting the the key instance variable
  return: self->key;
  /Define_Tag;

  /* You will rarely use the setKey method from outside of this custom type */
  Define_Tag: 'setKey', -required='key';  //setting the key instance variable
  self->'key' = (integer: #key);
  /Define_Tag;
/Define_Type;
```

The member variable 'key' is directly referring to the field that uniquely is used to store values to identify a specific contact. We use member tags to access the key because it makes our code less specific to the type of the key. Even though we won't likely change the type of the key, it is good to use a generic way of handling such a common feature of a database record. If we later create the company ctype, then it will be good to access the key of that ctype in a similar manner.

Retrieving data

To load the contact data from the database, we define a new member tag called 'load'. Since the object representing the record doesn't contain any valid data yet, we have to provide the 'load' member tag with a valid key to retrieve the data from the database.

This is how the 'load' member tag looks like.

Taken from example03/code.inc

```
Define_Tag: 'load', -required='key';
  inline: -search, $gDBspec,
  -table='contact', -keyfield='key', -keyvalue=(integer: #key), -maxrecords=1;
  records;
  self->'key' = (integer: (field: 'key'));
  self->'firstname' = (field: 'firstname');
  self->'lastname' = (field: 'lastname');
  /records;
  /inline;
/Define_Tag;
```

It looks very much like any other inline tag. So, what's so special with this? At the moment it's not about the inline itself, but what the inline contains. All of the database record data is copied to the member variables of the ctype.

Using the ctype data

The way to use the data from the ctype variable can be very much alike any other data used within a html page.

taken from example04/contact.htm

```
<form name="contact_form" id="contact_form" action="" method="post">
  <fieldset>
  <legend>Contact info</legend>
  <label for="inp_firstname">Firstname
  <input type="text"
  name="inp_firstname" id="inp_firstname"
  value="[$speaker->'firstname']" />
  </label>
  <label for="inp_lastname">Lastname
  <input type="text"
  name="inp_lastname" id="inp_lastname"
  value="[$speaker->'lastname']" />
  </label>
  </fieldset>
</form>
```

Using getters and setters

There are different schools telling you their truth about object oriented programming. One way to look at member variables is that you should never access them directly. The main reason for this is that all access to data or functions through the object should be abstracted (or controlled).

While this is generally a very good idea, it adds chores to the programmers todo list: Writing getters and setters as well as testing and debugging the same. Also, the overhead to call a member tag and to access a member variable is completely different. I haven't personally carried out any performance tests to find the difference, but I expect the additional overhead to be significant compared with the simple access of a variable.

In different integrated development environments (IDE's) like the Java editor in Eclipse, you'll find wizards to generate the equivalent to member tags for getters and setters of the member variables. As of now, there is no such feature in Lasso Studio for Eclipse. If there were, then we would have a predictable generator of Lassoscript code for getters and setters. Predictability means consequential behaviour, and such can easily be tested and measured. Knowing what we have and what's going to happen builds stability.

In the code supporting this paper I have chosen to access the member variables directly. There are good sides to this and there are bad sides. As long as you know which they are, you're ready to accept the consequenses.

Storing data

The process of storing data in the database is quite straightforward, and does not yet reveal any special features of object orientation. We will get back to this member tag later to make the handling more generic than at the moment.

Taken from example05/ctypes.inc

```
Define_Tag: 'save';
  if: self->GetKey == 0;  //this is an add operation
  inline: -add, $gDBspec, -table='contact', -keyfield='key', -keyvalue=self->GetKey,
  'firstname'=self->'firstname',
  'lastname'=self->'lastname';
  self->(setKey: -key=keyfield_value);  //it is important to set the key of the object
  /inline;
  else;  //this is an update operation
```

```
  inline: -update, $gDBspec, -table='contact', -keyfield='key', -keyvalue=self->GetKey,
  'firstname'=self->'firstname',
  'lastname'=self->'lastname';
  /inline;
  /if;
/Define_Tag;
```

Finding generic ways to solve common operations

Earlier we used the 'load' member tag for loading the contact data from the database. We are now going to rewrite it to allow for more generic implementation. This way we can reuse the code in other ctypes as well as test it thoroughly through use in other ctypes.

The new version of C_Contact->Load looks like this:

Taken from example06/ctypes.inc

```
Define_Tag: 'load', -required='key';
  inline: -search, $gDBspec,
  -table='contact', -keyfield='key', -keyvalue=(integer: #key), -maxrecords=1;
  records;
  self->LoadX;
  /records;
  /inline;
/Define_Tag;
```

Instead of specifying the fields directly in the C_Contact->Load we split the functionality between one generic member tag and one specific, called C_Contact->loadX. This way we know that most of the code that is specific to the contact table will be found in C_Contact->loadX.

```
Define_Tag: 'loadX';
  self->(setKey: -key=(field: 'key'));
  self->'firstname' = (field: 'firstname');
  self->'lastname' = (field: 'lastname');
/Define_Tag;
```

If you worry about the references to the contact table in C_Contact->load, please don't.We'll come back for another round of making the code generic.

Loading more than one contact

So far we've been concerned with only one contact, but we also need a way to present a list of contacts. It would be good if that code could be both stored with the rest of the code acessing the contact table. Luckily, the object oriented model provides us with a way to do that also.

Returning an array of contacts

The code to return an array of contacts is quite straight forward.

```
Define_Tag: 'getContactList';
  local: 'result' = array;

  local: 'oneRecord' = map;
  inline: -search, $gDBspec, -table='contact', -keyfield='key', -maxrecords='all';
  records;
  #oneRecord = C_Contact;
  #oneRecord->loadX;
  #result->(insert: #oneRecord);
```

```
  /records;
  /inline;
  return: #result;
/Define_Tag;
```

Using an array of contacts

Here follows the contents of the the file example06/list.htm which is included as an html fragment into the main html file.

```
<table id="contact_list">
  <tr>
  <th></th>
  <th class="key">Key</th>
  <th class="firstname">Firstname</th>
  <th class="lastname">Lastname</th>
  </tr>
  [iterate: C_Contact->getContactList, (var: 'oneContact')]
  <tr>
  <td><a href="?key=[$oneContact->GetKey]">Edit</a></td>
  <td class="key">[$oneContact->'key']</td>
  <td class="firstname">[$oneContact->'firstname']</td>
  <td class="lastname">[$oneContact->'lastname']</td>
  </tr>
  [/iterate]
</table>
```

You can see that we're not even creating a variable to access the member tag. This way of using an custom type is very common in pure object oriented languages such as Java, though there the ctypes are called classes.

This way of accessing the code works because Lasso will internally create a temporary variable from the ctype definition and after it's been used - i.e. no code is referring to it - the temporary variable will be forgotten.

Increasing the efficiency of instantiation of custom types

Whenever a custom type is used to create a variable, it will run all the code between the [Define_Tag] and the [/Define_Tag]. The reason for this is that earlier versions of Lassoscript was depending on code being executed at the time of the creation.

Now, much time has passed and the object oriented support in Lassoscript has grown to a more mature version. Today the only code that is expected to be found in [Define_Tag]...[/Define_Tag] is definitions of locals, which are called member variables, and definitions of member tags.

Since we're supporting this more modern way of declaring the ctype, we let Lasso find that out by using the special keyword -prototype.

It looks like this:

```
Define_Type: 'C_Contact', -prototype;
```

This will tell Lasso to immediately create an invisible variable which will be copied whenever we create a new instance of the C_Contact ctype. By doing that the speed will increase more than tenfold when used with ctypes with a large amount of code. You will find out that ctypes tends to be defined by a lot of code - generic code demands that you take a lot of possible cases in account.

Delete a record

To delete a contact we need a C_Contact->delete member tag. It looks almost as the C_Contact->save, except it doesn't make any use of parameters.

Taken from example08/ctypes.inc

```
Define_Tag: 'delete';
  if: self->hasValidKey;  //this is an add operation
  inline: -delete, $gDBspec, -table='contact', -keyfield='key', -keyvalue=self->GetKey;
  self->(setKey: -key=0);  //it is important to reset the key of the object
  //so that it is considered invalid
  /inline;
  /if;
/Define_Tag;
```

The generic question

In C_Contact->delete we use a member tag for questioning the validity of the record key. By doing this we will know that all code, no matter where it's being used will define the invalid key in the same way.

Taken from example08/ctypes.inc

```
Define_Tag: 'hasValidKey';
  return: self->key > 0;
/Define_Tag;
```

This kind of questioning of status or state of member variables is encouraged since the code uses it becomes very easy to read. See also the paragraph about getters and setters.

Compare

```
if: self->key > 0;
```

with

```
if: self->hasValidKey;
```

The latter version is easier to read and has the advantage that no matter the type of key that the record uses, the code will still look the same from the callers' perspective.

What we have done here is to introduce a questioning action in the ctype. With the references to human languages this is often called a predicate.

Hiearchy and inheritance in general

When a ctype is based on another ctype, we use a hierarchical way of defining our conceptual model of the world. Before we start adapting the C_Contact ctype to be using inheritance to pass over the generic code to a base ctype, we'll take a look at a totally different case.

Base custom type

Let's consider a geometric application which needs two different types of shapes: the rectangle and the ellipse.

We could define them like this

```
Define_Constant: 'pi', 3.14;
Define_Type: 'Rectangle';
  local: 'top' = 0.0;
  local: 'left' = 0.0;
  local: 'bottom' = 0.0;
  local: 'right' = 0.0;
  Define_Tag: 'getWidth';
  return: (math_abs: (decimal: self->'right') - (decimal: self->'left'));
  /Define_Tag;
  Define_Tag: 'getHeight';
  return: (math_abs: (decimal: self->'bottom') - (decimal: self->'top'));
  /Define_Tag;
  Define_Tag: 'getArea;
  return: self->getWidth * self->getHeight;
  /Define_Tag;
/Define_Type;
Define_Type: 'Ellipse';
  local: 'top' = 0.0;
  local: 'left' = 0.0;
  local: 'bottom' = 0.0;
  local: 'right' = 0.0;
  Define_Tag: 'getWidth';
  return: (math_abs: (decimal: self->'right') - (decimal: self->'left'));
  /Define_Tag;
  Define_Tag: 'getHeight';
  return: (math_abs: (decimal: self->'bottom') - (decimal: self->'top'));
  /Define_Tag;
  Define_Tag: 'getArea;
  return: self->getWidth * self->getHeight * pi;
  /Define_Tag;
/Define_Type;
```

But I'd rather define them like this

```
Define_Constant: 'pi', 3.14;
Define_Type: 'Shape';
  //the position of the boundary box is defined below
  local: 'top' = 0.0;
  local: 'left' = 0.0;
  local: 'bottom' = 0.0;
  local: 'right' = 0.0;
  Define_Tag: 'getWidth';
  /Define_Tag;
  Define_Tag: 'getHeight';
  /Define_Tag;
  Define_Tag: 'getArea;
  /Define_Tag;
/Define_Type;

Define_Type: 'Rectangle', 'Shape';
  Define_Tag: 'getWidth';
  return: (math_abs: (decimal: self->'right') - (decimal: self->'left'));
  /Define_Tag;
  Define_Tag: 'getHeight';
  return: (math_abs: (decimal: self->'bottom') - (decimal: self->'top'));
  /Define_Tag;
  Define_Tag: 'getArea;
  return: self->getWidth * self->getHeight;
  /Define_Tag;
/Define_Type;

Define_Type: 'Ellipse', 'Shape';
  Define_Tag: 'getWidth';
  return: (math_abs: (decimal: self->'right') - (decimal: self->'left'));
```

```
  /Define_Tag;
  Define_Tag: 'getHeight';
  return: (math_abs: (decimal: self->'bottom') - (decimal: self->'top'));
  /Define_Tag;
  Define_Tag: 'getArea;
  return: self->getWidth * self->getHeight * pi;
  /Define_Tag;
/Define_Type;
```

Please note the empty declarations of getWidth, getHeight and getArea in the Shape ctype. These are there as placeholders so that code that refers to any type of Shape also will be able to perform without error.

When you declare a variable of the type Rectangle, you have to understand that a rectangle is not only a rectangle, but also a Shape.

```
var: 'myRectangle' = Rectangle;
$myRectangle->'right' = 100.0;
$myRectangle->'bottom' = 50.0;
var: 'myArea' = $myRectangle->getArea;
```

The example here will return 5000.0 as the area since it appears frontmost to be a rectangle. But all the calculations are done on the member variables of the Shape. So it seems you have access to everything of the ancestor ctype called Shape.

The space here is too limited to go into the real depths of inheritance, shadowing, overloading and quite a few more object oriented paradigms. If this seems interesting to you, there are a lot of books on the subject.

The base custom type

We have reached a point where we can start to discuss our base custom type. The foundation for the database access through custom types. To get there with the whole picture, we'll skip to example12 in the supporting material.

The goal

First of all, where are we going with the C_Contact custom type? We're limiting it to only consist of attributes and actions that has to be defined within the concept of a contact. The rest is either part of the base ctype or is a special form of Contact itself - a successor to the C_Contact, e.g. C_Salesperson or something along that line.

Taken from example12/ctypes.inc

```
Define_Type: 'C_Contact', 'C_Record', -prototype;
  local: 'table' = 'contact';
  local: 'keyField' = 'key';
  local: 'firstname' = '';
  local: 'lastname' = '';

  Define_Tag: 'loadX';
  self->'firstname' = (field: 'firstname');
  self->'lastname' = (field: 'lastname');
  /Define_Tag;

  Define_Tag: 'save';
  self->parent->(save:
```

```
  'firstname'=self->'firstname',
  'lastname'=self->'lastname'
  );
  /Define_Tag;

  Define_Tag: 'setName', -required='firstname', -required='lastname';
  self->'firstname' = #firstname;
  self->'lastname' = #lastname;
  /Define_Tag;

  Define_Tag: 'getName';
  return: self->'firstname' + ' ' + self->'lastname';
  /Define_Tag;
/Define_Type;
```

The amazing part is that this is the full implementation of the C_Contact in our example. It would be very easy to define a company ctype to complement the C_Contact.

Another ctype created in virtually no time

Let's show the C_Company ctype would look like:

Taken from example12/ctypes.inc

```
Define_Type: 'C_Company', 'C_Record', -prototype;
  local: 'table' = 'company';
  local: 'keyField' = 'key';
  local: 'name' = '';

  Define_Tag: 'loadX';
  self->'name' = (field: 'name');
  /Define_Tag;
  Define_Tag: 'save';
  self->parent->(save:
  'name'=self->'name'
  );
  /Define_Tag;

/Define_Type;
```

What we have done here is to define the ctype, add the member variables to show what is contained in the database table, and specified how we would like these values to be loaded and saved from the database.

The ancestor of all database records

To have our C_Contact and C_Company ctypes working we need the base ctype defined. This is what you would call an abstract class in OOP lingo. It means that there will never be a variable directly defined being this type. There will instead be C_Contact based on C_Record and C_Company based on C_Record.

Dissection of the C_Record custom type

This section is a bit differently structured than the previous ones. It is merely a wrap up of what has been covered in the past pages. Some parts are quite advanced, some are less.

```
Define_Type: 'C_Record', -prototype;
  local: 'key' = null;
  local: 'table' = '';
  local: 'keyField' = '';
  local: 'keyType' = 'integer';
```

The member variables here are telling us what a record consists of. Since there is no concept of a database or a table in this example, I have chosen to include the table name in the record, but left out the database and authentication information. The latter decision has been done to leave place for a Roundtable discussion.

```
  Define_Tag: 'castKey', -required='key';
  return: ((pair: (\(self->'keyType'))=(array: #key))->invoke);
  /Define_Tag;
```

The C_Record->castKey member tag is what makes the record being independent of the type of the keyfield. It will on the fly cast the key that has been passed as a parameter to the specified keyfield type. If we would like to get really in to the depths, this could also be a ctype itself, which would handle more advanced type such as non-predictable record keys.

```
  Define_Tag: 'getKey';  //getting the the key instance variable
  return: self->key;
  /Define_Tag;

  /* You will rarely use the setKey method outside the custom type hierarchy */
  Define_Tag: 'setKey', -required='key';  //setting the key instance variable
  self->'key' = (self->(isValidKey: #key) ? self->(castKey: #key) | null);
  /Define_Tag;
  Define_Tag: 'hasValidKey';
  return:  self->(isValidKey: self->'key');
  /Define_Tag;

  Define_Tag: 'isValidKey', -required='key';
  return: #key != null && ((pair: (\(self->'keyType'))=(array: #key))->invoke) != 0;
  /Define_Tag;
```

The C_Record->getKey, C_Record->setKey, C_Record->hasValidKey, and C_Record->isValidKey forms the interface to access the key and to validate the key in a generic manner. This is a good example of when getters and setters are great to use.

```
  Define_Tag: 'load', -required='key';
  if: self->(isValidKey: #key);
  inline: -search, $gDBspec->getSpec,
  -table=self->'table', -keyfield=self->'keyfield', -maxrecords=1, -keyvalue=#key;
  records;
  self->(setKey: -key=(field: self->'keyfield'));
  self->LoadX;
  /records;
  /inline;
  /if;
  /Define_Tag;
```

The C_Record->load tag does the actual job of accessing the database to locate the record. It is the equivalent of the [inline: -search] container. Since the key handling is a instrumental part of the generic handling of a record, we have put the responsibility to set the key value on the C_Record ctype whenever it can be done.

```
  Define_Tag: 'loadX';
  /Define_Tag;
```

The C_Record->loadX tag is the supporting tag that actually performs the copying of data from the database to the abstracted database record.

```
  Define_Tag: 'save';
  if: self->hasValidKey;  //this is an update operation
  inline: -update, $gDBspec->getSpec,
  -table=self->'table', -keyfield=self->'keyfield', -keyvalue=self->GetKey, params;
  /inline;
  else;  //this is an add operation
  inline: -add, $gDBspec->getSpec,
  -table=self->'table', -keyfield=self->'keyfield', -keyvalue=self->GetKey, params;
  self->(setKey: -key=keyfield_value);  //it is important to set the key of the
object,
  //so the live object is considered valid
  /inline;
  /if;
  /Define_Tag;
```

The C_Record->save tag has a little bit of a different implementation than the C_Record->load tag. The reasons for this is that we might be wanting to save different fields of the database in different situations. However in this implementation, a loaded successor of C_Record must be fully loaded to be considered valid.

```
  Define_Tag: 'delete';
  if: self->hasValidKey;  //this is an add operation
  inline: -delete, $gDBspec->getSpec,
  -table=self->'table', -keyfield=self->'keyfield', -keyvalue=self->GetKey;
  self->(setKey: -key=null); //it is important to set the key of the object
  //so that it is considered invalid
  /inline;
  /if;
  /Define_Tag;
```

The C_Record->delete tag is the opposite of the C_Record->save tag. The most important part is to invalidate the key so that we know that the live object is not associated with a database record anymore. It is therefore perfectly possible to do a [$spokesman->delete] followed by a [$spokesman->save]. Most likely that would have no use in the real world, but the essence is that we can trust a deleted record to be fully represented as deleted.

```
  Define_Tag: 'getList';
  local: 'result' = array;

  local: 'oneRecord' = map;
  inline: -search, $gDBspec->getSpec,
  -table=self->'table', -keyfield=self->'keyfield', -maxrecords='all';
  records;
  #oneRecord = (\(self->type))->Invoke;
  #oneRecord->(setKey: -key=(field: self->'keyfield'));
  #oneRecord->loadX;
  #result->(insert: #oneRecord);
  /records;
  /inline;
  return: #result;
  /Define_Tag;
```

The C_Record->getList tag is the foundation for a general type of loading more than one record from a database. For it to be useful you'd need to define different sorting orders. For performance sake you'd need to define partial loading etc. The technique used to instantiate the ctype is one of the major strengths behind dynamically typed object oriented languages.

```
/Define_Type;
```

Summary

Once you start using ctypes in this way you'll be stuck into the lazy life of expecting things to behave themselves. You are allowed not only to define custom tags that are used to extend the

functionality of Lasso, but also to build new conceptual models of "things" and "objects". Most likely you'll start to speak about the database records in a different manner.

This is the start of a two-tier database solution. Your next step would be a three-tier solution, which can be constructed upon a well built two-tier solution.

AJAX Asynchronous JavaScript and XML

By Fletcher Sandbeck

Introduction

AJAX is an acronym for Asynchronous JavaScript and XML. It refers to a technique of building dynamic Web sites by downloading data as XML fragments through a background process written in HTML. However, AJAX is also shorthand for a new generation of Web sites which allow the contents of the page to be manipulated without reloads.

The idea of building Web sites which are more dynamic and more responsive to users has been around a long time. One prior incarnation was dubbed DHTML or dynamic HTML. Another current buzzword for this type of Web site is Web 2.0.

This paper discusses a collection of techniques which are commonly collected under the AJAX moniker. It shows how Lasso works as the back-end for an AJAX solution and how the LJAX (Lasso JavaScript And XML) framework can be used to make programming a dynamic Web site easier. Many of the techniques are derived from the script.aculo.us JavaScript libraries and the Prototype JavaScript framework.

The examples presented in this paper are collected into an AJAX Examples Pack which is included on the Lasso Summit CD. The folder "ExamplesPack" should be dragged into your Web server folder and then accessed through a URL like:

<http://localhost/ExamplesPack/AJAX/index.lasso>

Web Site Interactions

Traditionally a visitor's interaction with a Web site has been defined in terms of either following a series of links or filling out and submitting forms. These techniques have been sufficient to create Web sites for newspapers, shopping carts, message boards, online banking, and many more applications.

Plug-in technologies including Flash and Java make it possible to embed more dynamic applications within Web sites. Flash-based Web sites are often heavily graphics based with animations. Flash allows for interactivity to the point where even video games can be implemented within a Web browser. Java can similarly be used to create interactivity within a downloaded applet whose UI is displayed within a browser.

JavaScript presents a technique of creating dynamic Web sites without using plug-in technologies. JavaScript can be used to auto-fill forms, to check form values before they are submitted, to submit forms immediately after a value has been entered rather than waiting for a submit button to be pressed, and more.

The goal of many AJAX Web sites is to present a user interface which is as rich and responsive as that of any traditional desktop application, but to present that interface entirely in a Web browser. AJAX sites implement drag and drop of page elements, direct editing of text on the page, and pages that update automatically without reloading.

The conceit of AJAX is that this can be accomplished within a Web browser without any plug-ins like Flash and without downloading Java Applets.

Foundation Technologies

AJAX relies on four key Web browser standards: JavaScript, XHTML, XMLHttpRequest, and the DOM.

JavaScript - This ubiquitous client-side scripting language ties all the other technologies together. JavaScript is used to capture the site visitor's mouse clicks and key strokes, fetch data from the server using XMLHttpRequest, parse the resulting XHTML, and rewrite the page's DOM to reflect the new data.

XHTML - XHTML is an elaboration of the HTML standard which makes it follow the same rigorous parsing rules that define XML. XHTML is important to AJAX since it makes it possible for JavaScript to easily parse and manipulate downloaded data without worrying about all the quirks and inconsistencies of traditional HTML.

XMLHttpRequest - This JavaScript function, originally introduced by Microsoft and since embraced by all other major browsers, allows XHTML content to be downloaded asynchronously from whatever Web site is currently being visited in the browser. This secondary data channel is the key to allowing a site to send and receive data without unnecessary browser page reloads.

DOM - The Document Object Model is the Web browser's internal representation of the current page being shown to the site visitor. The current DOM can be manipulated through JavaScript allowing the page to be modified without reloading. DOM manipulations can be as simple as minor text or form element value changes or as major as CSS overhauls or complete page replacements.

A common AJAX procedure is to use JavaScript to call XMLHttpRequest when the user clicks on a link or submits a form. XMLHttpRequest fetches an XHTML fragment and manipulates the DOM of the current page in order to show the new content without reloading page.

Flow Chart

A traditional Web site using links or forms has the following basic flow chart. A Web page is downloaded and displayed to the visitor, they click on a link or submit a form and that action is uploaded to the Web server, a new Web page is generated in return.

```
-> Download HTML
-> Display Web Page
-> Visitor Clicks on Link or Submit Button
-> Upload Data
-> Generate New HTML Page
-> Download HTML
…
```

An AJAX Web site has a tighter loop. By only updating those portions of the Web site which have actually changed, the Web site seems more interactive. When the visitor clicks a link or submits a form the action is uploaded to the Web server, but only a fragment of the page is downloaded and then merged into the visible page in the browser.

```
-> Download HTML
-> Display Web Page
-> Visitor Clicks on Link or Submit Button
-> Asynchronous JavaScript
  -> Upload Data
  -> Generate New XHTML Fragment
  -> Download XHTML Fragment
  -> Update Page DOM Using Fragment
-> Visitor Clicks on Link or Submit Button
…
```

LJAX

A set of tools have been created in Lasso and JavaScript which work together to make AJAX techniques easy. These tools are collectively called LJAX and consist of the following. Full documentation of these tools is included at the end of this paper.

- Lasso.IncludeTarget(target,options) - This JavaScript function encapsulates the process of sending an action to the Web server, downloading an XHTML fragment, and merging that fragment with the current page's DOM.

- [LJAX_Target] … [/LJAX_Target] - This Lasso tag is used to mark different portions of a site so they are served for LJAX requests, for normal requests, or only for certain LJAX targets.

- LJAX.Lasso - This page is created by the site author and is responsible for serving appropriate XHTML fragments based on the what targets and other parameters are passed to it.

In order to use these tools a Lasso site must validate as XHTML. The site can first be written using traditional forms and links. The elements of the site that need to be dynamic are identified. The [LJAX_Target] … [/LJAX_Target] tags are used to block out portions of the page which are LJAX enabled. The LJAX.Lasso page is created to serve XHTML fragments. Finally, the links and forms which should trigger dynamic updates are modified to use Lasso.IncludeTarget().

AJAX Example

The AJAX example included with this paper shows how these tools can be used to create a dynamic Web site. The example is served as a "one file" solution where the "index.lasso" page handles all requests and uses different include files to generate page contents. The "index.lasso" page includes the HTML template wrapper. The "template.lasso" file from the examples is responsible for generating the page contents by checking the "code" action parameter and serving the appropriate file from the "examples" folder.

```
index.lasso
  examples/template.lasso
    examples/dragdrop.lasso
    examples/dynamic.lasso
    examples/reveal.lasso
    …
```

The "ljax.lasso" page from the example functions similarly to the "index.lasso" page. It also includes the "template.lasso" file to serve page contents. However, this contents is wrapped in <ljax> … </ljax> tags and served as an XHTML fragment rather than as an HTML page for

the Web browser. The "ljax.lasso" file also intercepts several targets and serves specially crafted XHTML fragments.

Within the "template.lasso" file the [LJAX_Target] … [/LJAX_Target] tag is used to signify that the HTML s which define the template should only be served if the current target is "page_frame" or if there is no target. This means that these template elements will not be served for other targets.

```
[ljax_target: 'page_frame', -notarget]
  …
[/ljax_target]
```

Several of the individual examples also use [LJAX_Target] … [/LJAX_Target]. The "dragdrop.lasso" file has a section which is served if the target is "nu_content" or "page_frame" or if there is no target.

```
[ljax_target: (array: 'nu_content', 'page_frame'), -notarget]
  …
[/ljax_target]
```

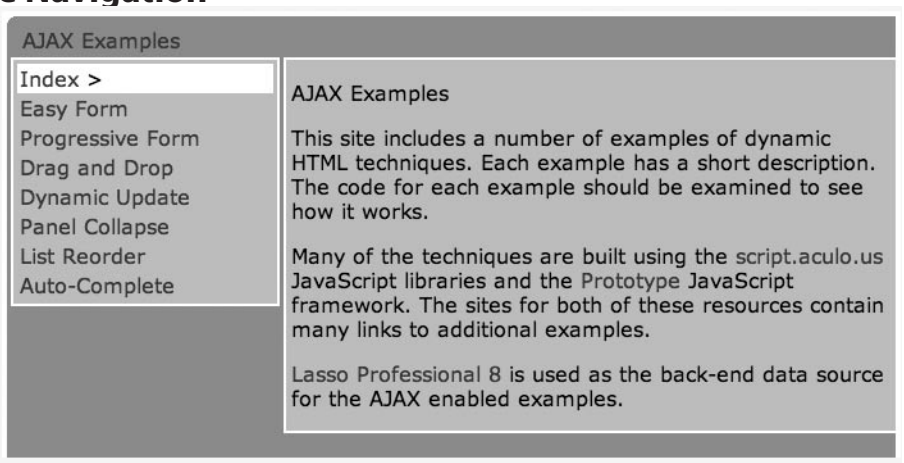
These [LJAX_Target] … [/LJAX_Target] tags create a situation where if the target is "nu_content" then only a small portion of the "dragdrop.lasso" file is served. If the target is "page_frame" then the complete template and page contents is served. If there is no target then the complete HTML page is served. Loading these URLs in your browser lets you see how the contents changes for the different -Target values.

<http://localhost/examplespack/ajax/ljax.lasso?page=dragdrop&-target=page_frame>

<http://localhost/examplespack/ajax/ljax.lasso?page=dragdrop&-target=nu_content>

Each of the examples demonstrates a different LJAX principle or script.aculo.us technique. In order to understand each example you should first try the user interface, then look at the appropriate page from the "examples" folder and any associated JavaScript functions in the "ajax.js" file. Loading the "ljax.lasso" file directly can also help to understand how XHTML fragments are being used to update the page.

Example Navigation



The menu on the left side of the AJAX example is itself a dynamic element. Each time a link in the menu is selected the page content is refreshed dynamically rather than having the entire page reload.

```
<a href="index.lasso?page=dragdrop" onclick="Lasso.includeTarget('page_frame', {args:
this, afterFunc: nu_decorate}); return false;">Drag and Drop</a>
```

The links in the menu each have an href that would reload the page normally. If the browser doesn't support JavaScript (or something goes wrong while executing the JavaScript handler) then the page will simply reload with the new contents like any traditional Lasso Web site.

The onclick handler allows a JavaScript function to be called when the user clicks on the link. The Lasso.includeTarget() function is called with a target of "page_frame". It uses the parameters from the current anchor tag "this" as its args option. And, it asks that nu_decorate() function be called after the page is refreshed dynamically. Finally, false is returned to prevent the browser from handling the click (there is no need since we handled it dynamically).

When the link is clicked by a site visitor the XMLHttpRequest method is used to fetch the following URL asynchronously (in the background).

<http://localhost/examplespack/ajax/ljax.lasso?page=dragdrop&-target=page_frame>

The contents is an XHTML fragment which includes new contents for the HTML div tag with an ID of "page_frame". The Lasso.includeTarget() tag swaps in the new contents and the user sees the new page contents without the entire page refreshing.

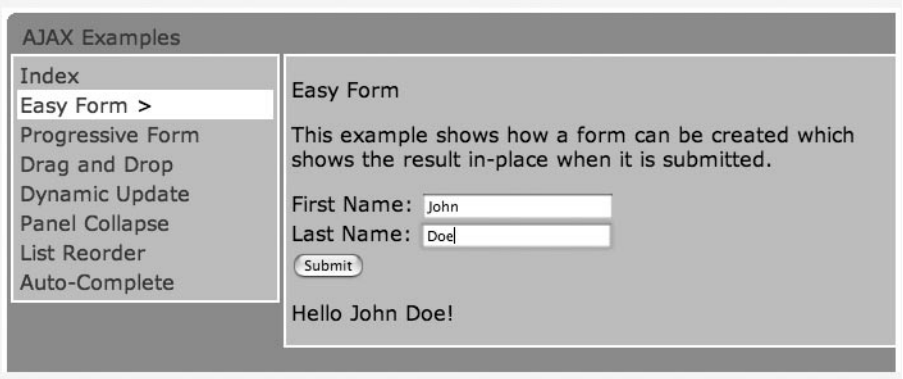
```
<ljax>
  <div class="group_list" id="page_frame">
    …
  </div>
</ljax>
```

Finally, the nu_decorate() function is called to initialize some elements that the drag and drop example needs (see below for details).

It is interesting to turn JavaScript off and try the site. This returns the site to its non-dynamic behavior where each page reloads. The performance of the site is similar in both cases (since the

site is quite simple), but with the AJAX version the URL of the site never changes and the browser user interface doesn't "flash" as much.

Simple Form



The simple form demonstrates how JavaScript can be used to intercept a form submission and change it into an LJAX dynamic page refresh instead. The results of the form are shown below the form dynamically without refreshing the page. Submitting the form multiple times simply updates the results.

The form from the page is shown below. The form is typical except for the onsubmit handler in the <form> tag and the submit <input> tag. The handlers call the JavaScript function Lasso.includeTarget() with a target of form_display. The args for the handler in the <form> tag references "this" to pass the values of all the form elements to the called page. The args for the handler in the <input> tag references "this.form" to accomplish the same thing. Each handler returns false in order to prevent the browser from submitting the form usuing the usual method.

```
<form action="index.lasso" method="post"
    onsubmit="Lasso.includeTarget('form_display',{args:this}); return false;">
  <input type="hidden" name="page" value="easyform" />
  <p>First Name:
    <input type="text" name="form_first" value="[var: 'form_first']" />
  <br />Last Name:
    <input type="text" name="form_last" value="[var: 'form_last']" />
  <br /><input type="submit" name="form_action" value="Submit"
    onsubmit="Lasso.includeTarget('form_display',{args:this.form}); return false;" /></
p>
</form>
```

The Lasso.includeTarget() tag calls the solution's ljax.lasso file with a target of form_display. The [LJAX_Target] … [/LJAX_Target] tags ensure that only the portion of the page which has changed is served. The LassoScript at the top always runs since it is not contained in [LJAX_ Target] … [/LJAX_Target] tags. The introduction text and form are only served if the target is page_from or if there is no target. That is if the page is being loaded by the user clicking on a menu option or by visiting the URL of the page directly.

```
<?LassoScript
  var: 'form_first' = (action_param: 'form_first');
  var: 'form_last' = (action_param: 'form_last');
?>
[ljax_target: (array: 'page_frame'), -notarget]
  <p>Easy Form</p>
  …
  <form action="index.lasso" method="post"
```

```
    onsubmit="Lasso.includeTarget('form_display',{args:this}); return false;">
    …
  </form>
[/ljax_target]
```

The results of the form are wrapped in [LJAX_Target] … [/LJAX_Target] tags which display the results if the target is page_frame or form_display or if there is no target. In particular, when the target is form_display this portion of the page is displayed, but the top of the page is not. The contains results if either the first or last name have a value or is empty if both the first name and last name are empty. The first time the page is loaded this is served empty, but on subsequent loads the contains the "Hello" message. The xmlns attribute of the is required only if the is being served as part of an XHTML fragment.

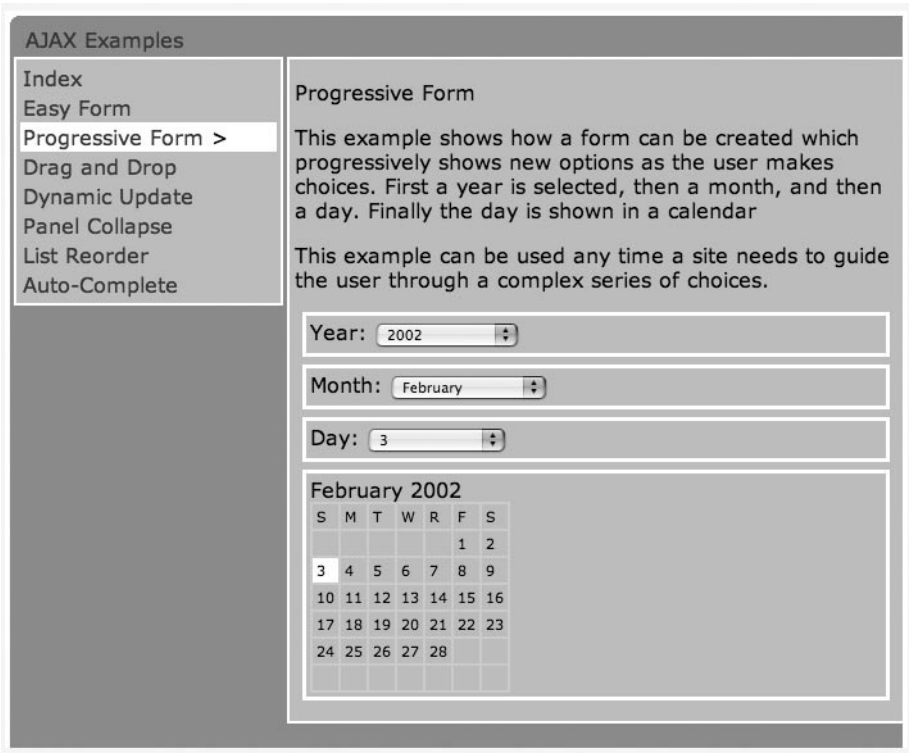
```
[ljax_target: (array: 'page_frame', 'form_display'), -notarget]
  [if: $form_first != '' || $form_last != '']
    <div id="form_display"[if: (var: 'ljax') == true]
        xmlns="http://www.w3.org/1999/xhtml"[/if]>
      <p>Hello [var: 'form_first'] [var: 'form_last']!</p>
    </div>
  [else]
    <div id="form_display"[if: (var: 'ljax') == true]
        xmlns="http://www.w3.org/1999/xhtml"[/if]></div>
  [/if]
[/ljax_target]
```

If the form is submitted with the values "John" and "Doe" then the ljax.lasso page ends up serving code like that shown below. The Lasso.includeTarget() tag is responsible for finding an element with the same ID of form_display and replacing its contents with the contents of the from this fragment.

```
<ljax>
  <div id="form_display" xmlns="http://www.w3.org/1999/xhtml">
    <p>Hello John Doe!</p>
  </div>
</ljax>
```

This example shows the most basic form of LJAX, a form is submitted and a portion of the page is dynamically refreshed. This example can be used as the basis for a wide range of solutions which need to provide user feedback, but don't need the entire page to be refreshed each time the user submits additional data. For example, Lasso could actually be adding records to a database each time the form is submitted and the feedback could be a confirmation that the record was added.

Progressive Form



The progressive form demonstrates how a guided form can be created which shows additional options as the visitor makes choices. The visitor is asked to choose a year, then a month, and finally a day. A calendar of the selected month is shown below with the selected day highlighted. As the visitor progresses only the parts of the page which need to be updated are.

This example uses the same basic idea as the easy form shown above. When the year is chosen an onchange handler calls Lasso.includeTarget() to refresh the dynamic portion of the page. The dynamic portion encompasses the month, day, and calendar displays. The afterFunc option is used to perform a task after the dynamic contents of the page has been updated. In this case the with ID cal_month is made visible by manipulating its CSS display attribute.

```
<select name="cal_year"
    onchange="Lasso.includeTarget('cal_display',{args:this.form,
    afterFunc:function(request){
      document.getElementById('cal_month').style.display = 'block';
    }});">
  <option value=""[if: $cal_year == 0] selected="selected"[/if]>
    Select a Year...
  </option>
  [loop: -from=2000, -to=2010]
    <option value="[loop_count]"
        [if: $cal_year == loop_count] selected="selected"[/if]>
      [loop_count]
    </option>
  [/loop]
</select>
```

The for cal_month has its CSS display attribute initially set to none. This prevents the from being shown on the page. After a year is selected this attribute is set to block (the default for s) in order to have the be visible on the page again.

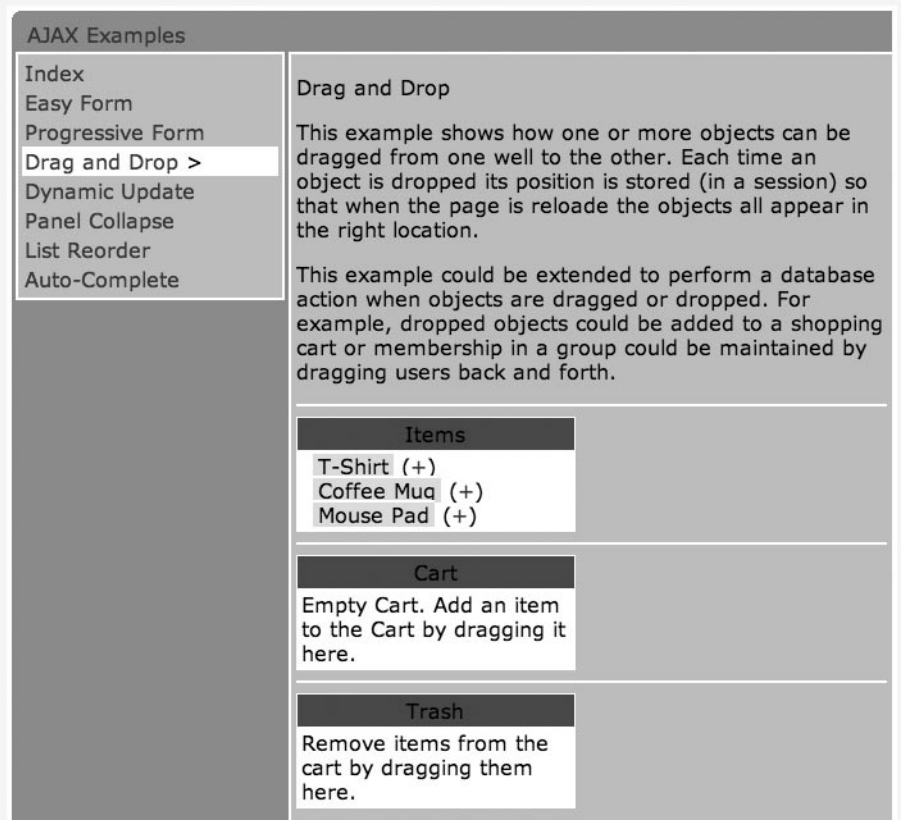
```
<div id="cal_month" style="[if: $cal_year == ‘’]display: none; [/if]border-color:
white;“> … </div>
```

After a month is chosen the day is shown using a similar technique and after a day is chosen the calendar itself is displayed. There is no provision in this example for rolling the inputs back up so the user can de-select a day and de-select a month. Instead, the day and calendar are simply hidden if the month is set to an invalid value.

This technique can be used on sites which need to present the user with a series of questions or hierarchical choices. As the user makes choices the remaining elements of the page are refreshed giving either the next question or the next level of choices.

Note - The browser will not interpret the inline CSS style attributes of dynamically loaded elements in all browser. When this is refreshed it will not have a display attribute of none so it should be shown on the page, but most browsers will remember that the was hidden and won’t make it visible again. Instead, it is necessary to use afterFunc to change the display attribute of the explicitly.

Drag and Drop



One of the most impressive examples from the script.aculo.us library is the drag and drop functionality. It is possible to make any HTML div a draggable object and any other div a drop target. Some Web sites are now using this technique for shopping carts so that users can simply drag items into their cart rather than having to click on links.

The drag and drop example starts as a simple shopping cart interface consisting of two s. The first has a list of items and each item has a + button which calls a URL with an action and item to add an item to the cart. The second div lists items in the cart and has +/- buttons for changing the quantity of items in the cart and an x button for removing items from the cart.

index.lasso?page=dragdrop&action=add&item=nu_one

In order to make elements draggable the function nu_decorate() from the "ajax.js" file is called at the bottom of the page (this function must be called after the s that it decorates have already been defined). This function calls several functions from the script.aculo.us library to create draggable items and drop targets.

Draggables are created by simply referencing their ID in the Draggables() creator function. Each item from the store is marked as draggable and the nu_cart_draggables() function marks each item which is contained in the cart as draggable as well.

```
new Draggable('nu_item_one', {revert:true});
new Draggable('nu_item_two', {revert:true});
new Draggable('nu_item_three', {revert:true});
function nu_cart_draggables()
  {
    if (document.getElementById('nu_cart_one'))
      new Draggable('nu_cart_one', {revert:true});
    if (document.getElementById('nu_cart_two'))
      new Draggable('nu_cart_two', {revert:true});
    if (document.getElementById('nu_cart_three'))
      new Draggable('nu_cart_three', {revert:true});
  }
```

Droppables require that the ID of the drop target be identified as well as the class of draggable item that the drop target should accept and the function that should be called after each item is dropped on the cart. The function in this case is Lasso.includeTarget() with the target of "nu_content", args generated by the + link for the item, and the function nu_cart_draggables() called after the dynamic refresh.

```
  Droppables.add('nu_cart',
    {
      accept:'item',
      onDrop:function(element){
        Lasso.includeTarget('nu_content',
          {args: document.getElementById(element.id + '_add'),
          afterFunc: nu_cart_draggables});
      }
    });
```

A similar call is used to make the trash a drop target for draggable items in the class "cart" which uses args generated bythe x link for the cart item.

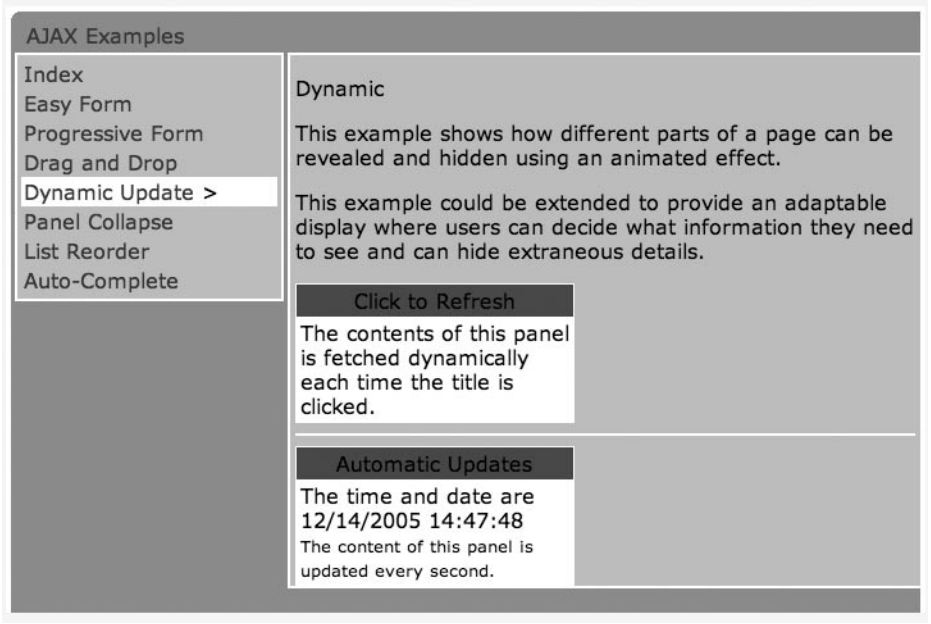
```
  Droppables.add('nu_trash',
    {
      accept:'cart',
      onDrop:function(element) {
        Lasso.includeTarget('nu_content',
          {args: document.getElementById(element.id + '_rem'),
          afterFunc: nu_cart_draggables});
      }
    });
```

Amazingly, that is all the JavaScript required. Now, dragging an item from the "Items" div to the "Cart" div works the same as clicking the + link next to an item. If an item is already in the cart then its quantity is incremented. Items which are dragged to the trash are simply removed from the cart.

The cart can be easily customized by changing the look of the three div's. The items can be enhanced with product pictures. The cart could show the quanitity of items visually. The trash can be made to look like an actual trash can.

For backward compatibility the +/-/x buttons can be used to perform the same basic operations in a browser that does not support JavaScript.

Dynamic Update



This example shows two methods of dynamically refreshing the contents on a page. Each example shows the current date/time within a small panel. The upper panel is refreshed when the title is clicked. The lower panel is refreshed automatically every second.

The upper panel has the following basic structure. An onclick handler is added to the title div so that it functions like a clickable link. Lasso.includeTarget() is used to refresh the lower div. Note that the "args" option is generated simply as target=theta_content.

```
<div style="background: blue;">
  <div id="theta_title" onclick="Lasso.includeTarget('theta_content',
    {args: 'target=theta_target'}); return false;">Click to Refresh</div>
  <div id="theta_content" style="background: white;">
    The contents of this panel is fetched dynamically
    each time the title is clicked.
  </div>
</div>
```

Within the "ljax.lasso" file this action parameter "target" is checked for the value "theta_target" and a special XHTML fragment is created with the following content. This demonstrates how the

"ljax.lasso" file can quickly serve just the content required for a dynamic update without loading the rest of the site.

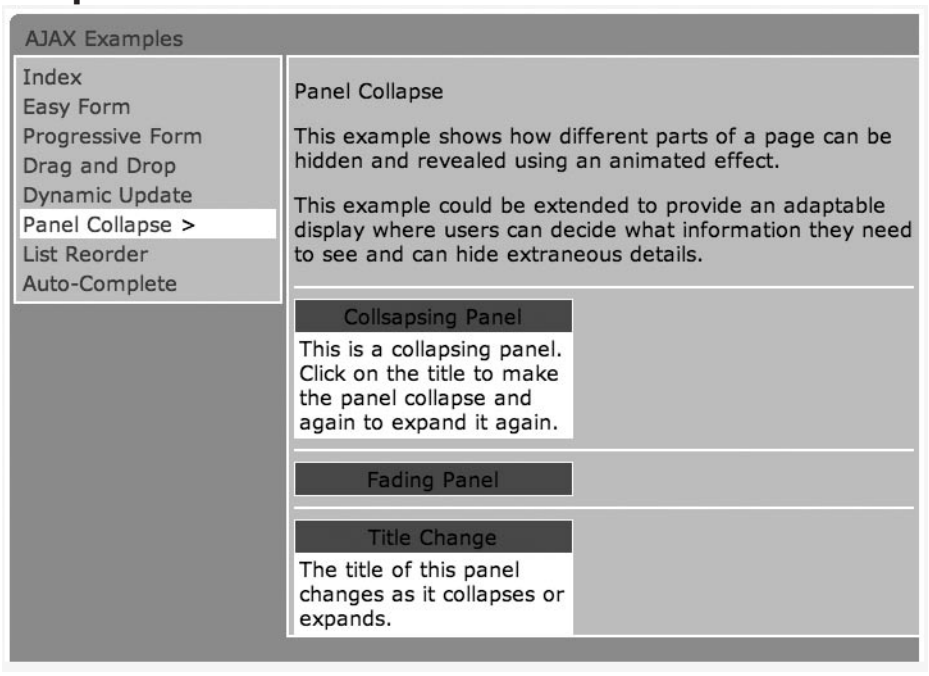
```
'<div id="theta_content" xmlns="http://www.w3.org/1999/xhtml">';
  'The time and date are';
  '<br />' + date->(format: '%D %T');
'</div>';
```

The lower target uses a JavaScript setTimeout() function to schedule an update to run every second. The function eta_refresh() checks if an element with ID "eta_content" is contained on the page. If it is the contents is updated using Lasso.inclueTarget() and the function is rescheduled to run a second later (1000 milliseconds). Otherwise, the function simply exits.

```
function eta_refresh()
{
  if (document.getElementById('eta_content'))
  {
    Lasso.includeTarget('eta_content', {args: 'target=eta_target'});
    window.setTimeout("eta_refresh()",1000);
  }
}
```

These techniques could be used to create a section of a site which is updated each time it is clicked to provide a random "fortune", to provide the answer to a question, or to allow the user to check on the progress of a background activity. The timed version could be used to poll a site for a dynamic status message periodically or to display the ongoing progress of a background activity.

Panel Collapse



This example is a script.aculo.us technique. It shows how to create panels on the page which can be opened and closed by clicking on the title. The panels are animated as they open or close providing good visual feedback for the site visitor.

The first panel appears to roll up into the title of the panel. The panel is defined as follows. The onclick hander in the title calls a function beta_blind() which is defined in the "ajax.js" script file.

```
<div style="background: blue">
  <div onclick="return beta_blind();">Collsapsing Panel</div>
  <div id="beta_content" style="background: white">
    This is a collapsing panel. Click on the title to make the panel collapse
    and again to expand it again.
  </div>
</div>
```

The beta_blind() function is defined as follows. It finds the div with ID "beta_content" and then uses a pre-built script.aculo.us effect to hide the div using an animated effect. Once the div is hidden its height is "0px" and calling the function again will show the div again.

```
function beta_blind()
{
  var beta_content = document.getElementById('beta_content');
  if (beta_content.style.height != "0px")
  {
    new Effect.BlindUp(beta_content);
  }
  else
  {
    new Effect.BlindDown(beta_content);
  }
  return false;
}
```

The second panel both rolls up and fades to blue. This is accomplished in the gamma_blind() function by combining two pre-built script.aculo.us effects.

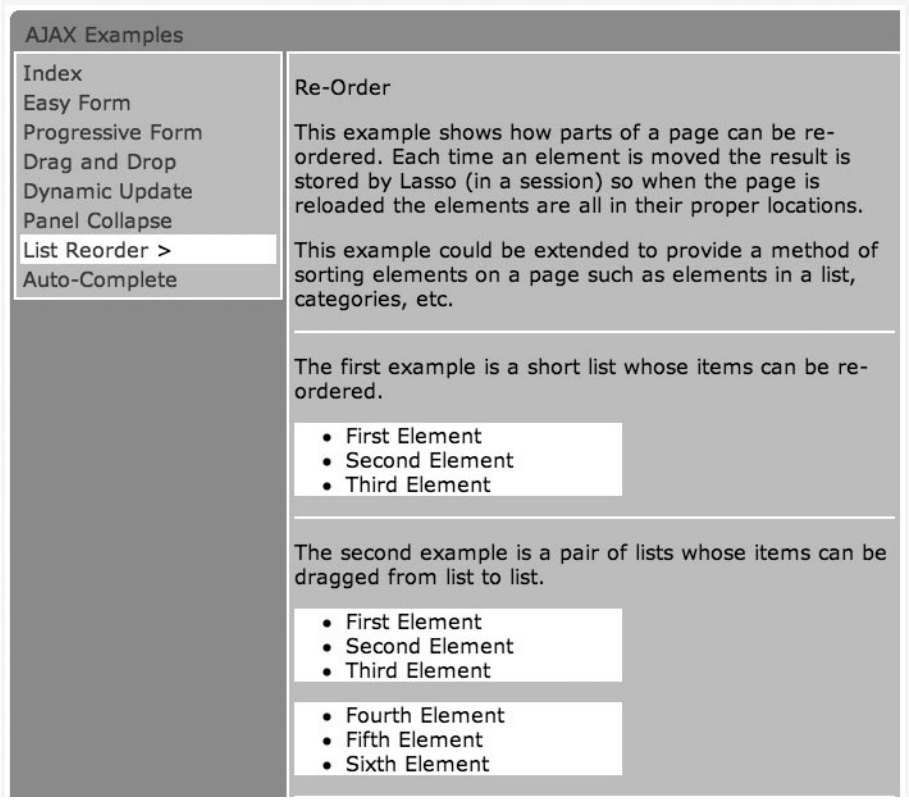
```
new Effect.BlindUp(gamma_content);
new Effect.Fade(gamma_content);
```

The third panel also changes the title of the panel when it is expanded or contracted. This is accomplished in the delta_blind() function by changing the innerHTML property of the "delta_title" div.

```
var delta_title = document.getElementById('delta_title');
if (delta_title)
  delta_title.innerHTML = 'Collapsed';
```

These effects can be used to create a page which has revealable content. The title of each panel could be a summary and the details can be shown only if the panel is expanded.

List Reorder



This example is a script.aculo.us technique. It shows how to create re-orderable lists. The site visitor can re-order the list in the top panel and can drag items from one list to the other in the next two panels. The third part is a sentence which is re-orderable.

The panels are all made re-orderable by the function list_decorate() in the "ajax.js" page. The top panel is made sortable by the following code. The containment parameter specifies that only elements from within "rho_list" can be dragged within the list. The onChange handler logs the new value of the list (and could be used to initiate a database operation, etc.). The onChange function calls Lasso.includeTarget() to have Lasso log the new order of the list.

```
Sortable.create('rho_list',
 {containment:['rho_list'],
   onChange:function(element){
     Lasso.includeTarget('', {args: 'target=none&log=rho:' +
       list_order('rho_list')});}});
```

The next two panels are made sortable by the following code. The containment parameter her specifies both lists. This allows items to be dragged from one list to the other. The onChange function calls Lasso.includeTarget() to have Lasso log the new order of the list.

```
Sortable.create('sigma_list',
  {
    containment:['sigma_list','tau_list'],
    onChange:function(element){
      Lasso.includeTarget('', {args: 'target=none&log=sigma:' +
        list_order('sigma_list')});
    }
  });
Sortable.create('tau_list',
```

```
  {
    containment:['sigma_list','tau_list'],
    onChange:function(element){
      Lasso.includeTarget('', {args: 'target=none&log=tau:' +
        list_order('tau_list')});
    }
  });
```

The third part contains a sentence whose words are re-orderable. The new sentence is logged each time it is changed. The or-orderable element in this case are span tags. The onChange function calls Lasso.includeTarget() to have Lasso log the new order of the list.

```
Sortable.create('upsilon_list',
  {
    containment:['upsilon_list'],
    tag:'span',
    onChange:function(element){
      Lasso.includeTarget('', {args: 'target=none&log=' +
document.getElementById('upsilon_list').innerHTML.replace(/<\/?span.?>/gi,'')});
    }
  });
```

This can be used to have the user prioritize elements on a site or to move users between groups. The Lasso.includeTarget() call could perform a database or session operation to store the new value of the list rather than simply logging the new order.

Auto-Complete

This example is a script.aculo.us technique. It shows how to create an auto-completing text input. The text input is named "zeta_input" and is immediately followed by a "zeta_complete" div which is initially empty.

```
<input type="text" id="zeta_input" name="test" value="" />
<div class="complete" id="zeta_complete"></div>
```

Auto-complete is enabled by calling the zeta_decorate() function in the "ajax.js" file. This function uses the script.aculo.us Autocompleter object with a list of explicit terms to be used in the auto-complete div when it is shown.

```
new Autocompleter.Local('zeta_input', 'zeta_complete',
  ['one','two','three','four','five','six','seven','eight','nine','ten'],
  {});
```

This example could be extended with a list of words generated from site-specific data. The script.aculo.us library also offers several different Autocompleter classes which can be used in conjunction with live data fetched through AJAX methods.

Details

This section contains more detailed documentation of some of the functions used by the AJAX examples.

Lasso.includeTarget(target,options)

The function requires one parameter which is a target (or array of targets) to fetch. Additional options can be specified in a second parameter (described below). The function uses XMLHttpRequest to call a file LJAX.lasso on the current site with a series of -Target=target parameters.

The LJAX.Lasso file is expected to return a valid XHTML fragment surrounded by <ljax> … </ljax> tags. This fragment is parsed and any sub-elements which have an ID are used as replacements for elements with the same ID from the current Web page.

The most common option is "args" which allows any anchor tag or form to be specified. The URL or form parameters of the specified element are passed to the LJAX.Lasso file. Alternately, a string of parameters can be specified. This option allows Lasso.IncludeTarget to essentially simulate a link or form submit.

The option "afterFunc" allows a JavaScript function to be named which will be called immediately after any matching XHTML elements have been replaced. The option "argsoverride" allows specific values to be sent in place of the actual parameters from the link or form specified as "args".

Finally, the option "func" allows a custom function to be used in place of the default behavior of replacing elements with matching IDs. A second optional "param" is passed to the function when it is called.

The Lasso.IncludeTarget() function is typically called in an onclick handler for an anchor tag or in an onsubmit handler for a form tag. In either case the anchor or form can be passed as "this" for the "args" option. The function should be followed by "return: false;" in order to avoid the natural behavior of the tags (reloading the page).

```
<a href= [response_filepath]?name=value"
    onclick="Lasso.IncludeTarget('target',{args: this}); return false;">
  …
</a>
<form action="[response_filepath]
    onsubmit="Lasso.IncludeTarget('target',{args: this}); return false;">
  <input type="hidden" name="name" value="value" />
</form>
```

[LJAX_Target] … [/LJAX_Target]

These tags mark how an area of a Web page should be served. Areas can be marked to serve when no LJAX target has been specified, when any LJAX target has been specified, or when a specific LJAX target has been specified.

[LJAX_Target: 'target'] … [/LJAX_Target] will serve the contents only when the specified target is being asked for through an LJAX request. The contents will not be served for normal HTML

requests. An array of targets can also be specified as in [LJAX_Target: (Array: 'target1', 'target2')] … [/LJAX_Target].

[LJAX_Target: -NoTarget] … [/LJAX_Target] will serve the contents only when an LJAX target has not been requested. This is most useful for HTML-only parts of the Web page including the head elements, title, stylesheet, etc. -NoTarget can be combined with a specific target to serve the contents when either that target or no target has been requested.

[LJAX_Target: -AnyTarget] … [/LJAX_Target] will serve the contents when any LJAX target has been requested. This is most useful for LJAX-only parts of the page.

The most common form of the tag is [LJAX_Target: (Array: 'target1','target2'), -NoTarget] … [/LJAX_Target]. This form serves a portion of the page when any of the targets within the array or no target is specified. This marks an area of the page so it will be served as part of the initial HTML Web page and also as part of an LJAX request for an XHTML fragment.

Note that nested [LJAX_Target] … [/LJAX_Target] tags can be specified, but every outer tag must share at least one target with the inner tags. The following example works fine since "target1" is specified both in the outer tag and the inner tag.

```
[LJAX_Target: (Array: 'target1','target2')]
  [LJAX_Target: 'target1']
    …
  [/LJAX_Target]
[/LJAX_Target]
```

However, in the following example the inner contents will never be served at all since a call to "target3" will cause the contents of the outer tag to be skipped and the code to check whether the inner tag should be served will never be seen by Lasso. If "target1" or "target2" is requested then the contents of the inner tag won't be served anyway.

```
[LJAX_Target: (Array: 'target1','target2')]
  [LJAX_Target: 'target3']
    …
  [/LJAX_Target]
[/LJAX_Target]
```

LJAX.Lasso

The LJAX.lasso page must be created for each site which wants to make use of the Lasso.includeTarget() JavaScript function. The minimum implementation of an LJAX.Lasso must ensure that it be served as an XHTML fragment with appropriate XML header and content-type. The outermost XML tag in the file should be a simple <ljax> … </ljax> tag. And, the file should set the variable "ljax" to be true.

```
<?xml version="1.0" encoding="UTF-8"?>
[content_type('text/xml; charset=UTF-8')]
[var('ljax'=true)]
<ljax>
  …
</ljax>
```

The contents of the <ljax> … </ljax> tags should be one or more XHTML fragments tagged with appropriate IDs. The Lasso.includeTarget() function will scan for all IDs in the file and replace like elements in the current HTML page with their contents.

One of the easiest ways to create an LJAX.lasso file (and the method the AJAX example uses for the most part) is to use the same site includes in the LJAX.lasso file as in the actual site, but to mark up the site with [LJAX_Target] … [/LJAX_Target] tags.

However, it is also possible to craft all of the XHTML within the LJAX.lasso file specially. For some applications this may result in a speed boost, but at the cost of maintaining separate code bases which perform much the same tasks.

Browser Support and Backward Compatibility

The LJAX method and script.aculo.us methods (and included Prototype library) have all been tested in the most recent versions of Safari 2 for Mac OS X, FireFox for Mac OS X and Windows, and Internet Explorer 6 for Windows. The methods may be supported in additional standards compliant browsers as well.

Many of the methods are backward compatible with older browsers and there are notes throghout this paper and the provided code which show how to provide the best backward compatibility possible. In particular, the drag and drop and menu navigation examples are fully backward compatible.

If the XMLHttpRequest method isn't supported by a browser then the Lasso.includeTarget() tag will fail. If the browser doesn't support JavaScript (or JavaScript is turned off) then the callback in the link, div, or form will simply never be called and the mouse click will be recorded as a normal Web browser action.

Examples which are not backward compatible include the re-orderable lists, the expanding and collapsing panels, and the text auto-complete. In general, these features should be incorporated into a site in such a way that their loss in older browsers does not cause the site to be completely unusable.

Additional Resources

These are links to the script.aculo.us Framework and Prototype JavaScript Library. Many of the examples in this paper are built using these tools.

```
script.aculo.us Framework - <http://script.aculo.us/>
Prototype JavaScript Library - <http://prototype.conio.net/>
```

These are links to the Web standards for XHTML, CSS, and a JavaScript resource.

```
XHTML - <http://www.w3.org/MarkUp/>
CSS - <http://www.w3.org/Style/CSS/>
JavaScript - <http://wp.netscape.com/eng/mozilla/3.0/handbook/javascript/>
```

Finally, these are some useful Web links which you probably already know.

Lasso Professional - <http://www.omnipilot.com/lasso>
Lasso Examples - <http://www.omnipilot.com/addons>
Lasso Reference - <http://reference.omnipilot.com>
Lasso Tips of the Week - <http://www.omnipilot.com/totw>
OmniPilot Software - <http://www.omnipilot.com>

Covering the Basics Roundtable

Miles

INTRODUCTION: EMAIL LOG CREATION: A CASE EXAMPLE

Where would the world be without email. If it weren't for email, I would be outta business in short order. As would most of you. Today's tip is not a retrospective of miles musings about email. No in fact its actually about the email capabilities of Lasso 6, 7 and now 8! But rather than feed you what you already know about the tags, assuming you do...let's take a slightly different tack. And remember folks...you gotta pay for the soup first: TANSTAAFL!

Today's Lasso Tip is about sending log files to an end user. It includes some sample (read that as FREE) code for you to play with and to use, and it includes a descriptor email to go with. Trust me this is a must read today. Not as cohesive as my previous LTfN but definitely a readworthy piece of text. Besides, remember ITS FREE!

Moving on.....

Lasso has some amazing capabilities and one of them is its ability to use email as an effective method to notify users, delayed feedback if you will. To send out everything from username/password pairs to PDF coupons!

Until rather recently (within the last 3 years Im talking), Lasso's email capabilities by today's standards (and even of the day itself) was limited to just text, and not even HTML text at that, not unless you tweaked the hell out of it, it was just text. Suffice it to state that Lasso's climb from email text to email superiority has been long and slow for one reason or another. Time was that if you got Lasso to email at all it was a miracle and you thanked your lucky stars (L2 days). Stepping into the wayback machine for a second, I refer to the Lasso 3 days when you'd set up a msg to get sent out under certain conditions and if Lasso couldn't talk to its intended mail server, after 3 attempts it would drop the issue entirely. Lasso 3 would go so far as to make a log entry like this:

```
05/22/03 14:33:03 ERROR: -104.  Failed 3 times to connect to email host.
```

And that's about as far as you got. And if you got that, you were doing good, at least you knew Lasso TRIED, whether or not the msg actually contained anything was another matter entirely. Today Lasso is much much much better at telling you that an email wasn't sent for a variety of reasons....it doesn't just drop the msg, but tells you exactly what error response it got back from the email server, and instead of 3 tries it stops after the 5th, and instead of discarding the msg, it holds it in memory until YOU do something! Trust me my friends, Lasso has come a long way in terms of email.

Today Lasso can send not just text, but HTML mail as well, and of course it can send attachments....DUH! However I'm willing to wager that you've looked at LP8 and gotten about half way down on the page of specs and said, "WAIT! Hold the phone, it does WHAT ? It will EVEN check a POP3 server ? Where's my CREDIT CARD ?!?!?!?!?". For those of us that have been with Lasso since v1.0 (I was there with 1.1), we understand just what a big deal that

is...however if you're coming from another platform. That's a different story altogether. When I saw that LP8 actually will check a POP server for mail, no 3rd party plugins, no tweaking, no fiddling, just straight forward -> GET THE MAIL! I got out my credit card. That's pretty cool stuff when you sit and think about it. That puts Lasso on par with PHP, ASP or ColdFusion.

Of course all three of those had email connectivity about 3 years ago, but better late than never at all I say! (smile)

Ok so Lasso's history with email hasn't been shall we say stellar but its here now and let's take advantage of that for our own nefarious purposes! (giggle)

PART ONE: EMAIL BASICS

Before we get to the fun stuff...the code, there are some rules of the road, Lasso does require that you actually set an SMTP host, however you can specify in the tag itself the host that will be used to send your msg. Because I run a B2B ISP/ASP and my own CMS, I bypass the requirement of having to set the host SMTP. My reason is that because 99 % of the apps Im running I wrote, but for different domains, and each of those domains use their own mail host, not mine, so its pointless to actually set a host, when the individual tag will allow me to set it, that and because of the fact I am running multiple domains off my servers requires that each site have its own host requirements. Another requirement is that you can't send anonymous email. I know that sounds DUMB but there's some validity to it. You actually have to have a valid host to send from, however that host does NOT need to be a valid domain, it can be an IP address. I'll tell you this much though, as anyone who hosts their own stuff will tell you, in today's world sending email out via IP address just isn't gonna fly, its just not gonna happen, mostly because in today's world most spam protection routines prevent that kind of mail from ever getting to an intended recipient by IP address alone. Another rule of the road is that Lasso does actually have an upper limit to the amount of data you can send via email, however you'll probably NEVER reach it, but know that there is a limit. Which is to say, try not to send out dissertations, just the facts please.

On to the fun stuff...

Sending an email with Lasso is pretty simple stuff. Time was it was 5 subtags as an inline. Today, its a single tag with a few requirements. For those of you upgrading your solutions, there is ONE major change (which was made for security reasons so I gather) the BODY of the msg is NOW an include. So lets take a gander at the email tag itself:

```
[email_send:
    -host='domain.com',
    -to='info@domain.com',
    -from='site@domain.com',
    -subject='MY SUBJECT',
    -body=(include:'email.txt')]
```

1.) **-HOST**, this is your sending host. Or the host that will be used to relay this msg. Lasso doesn't actually send out the msg, the host does that. Keep that in mind when sending MASSIVE amounts of email!

2.) **-TO** is who you're sending this to...pretty simple.

3.) -**FROM** is WHO the msg is from. There's a caveat here that you need to be aware of, which we'll get to in a second but be aware that there's something we'll revisit later on.

4.) -**SUBJECT** line....pretty straight forward, but again we're going to revisit this when we kick it up a notch.

5.) -**BODY**, here's the actual msg. And note that the its a FILE INCLUDE.

The actual body of the msg is contained elsewhere...

Let's break this down a bit....these are the 5 basic subtags you need in order to get Lasso to send a msg. However, there's a whole lot there that you're probably not seeing. Not the least of which is this is a single tag, it can be modified to meet your needs, and that the actual body of the msg can contain Lasso directives!

Which leads me to the actual INCLUDE itself....this is the result of log file that we will send out later on, its the file include that is the aggregate result of a (what you will see) long series of SQL calculations.

```
    Good Morning,
    Here is your daily web log for [date_format: ($v_today), -format='%A, %B %-d %Y'].
    Total Hits: [$v_hits].
    Distinct IP addresss: [$v_ip].
    Hits By Platform
    MAC: [$v_mac]
      PC: [$v_pc]
    UNIX: [$v_other]
    Hits By Browser Type
              msie (mac/pc): [$v_mmsie]/[$v_msie]
                 other: [$v_other]
    Hits By Page[records: -inlinename='sql_pagecount']
[field:'page_name'] [field:'page_count'] [field:'ref_count'][/records]
    your report in excel, go here -> [$v_filename_xls]
    your report in txt, go here -> [$v_filename_txt]
    Thank you.
    Your Web Server.
```

Most of this stuff is useless to you, however note that the include is full of Lasso directives and defined variables, and not to mention an INLINE command! So what is the email you're looking at ? Well its part of a larger system that computes log stastics. I didn't like the apache logging routine so I wrote my own. Every time someone hits a page in my CMS it logs to a client database. That data is calculated via a series of SQL statements and then put into variables and then placed here in their final form. What is of further note is that in the course of a day thousands of hits are registered across my client websites, and this email is sent out in a matter of seconds to all my clients by midnight. It takes about 3 seconds to compute the hits for the day and then to send out this msg. And its done automatically via an event trigger!

I digress, the only reason Im telling you all this is to show you what can be done with the simplest of email tags!

PART TWO: TAKIN IT TO THE NEXT LEVEL!

So we've got the basics down, there are a few things we need to be aware of...like for instance that because Lasso isn't actually sending out these msgs itself, but the email host is, there are

situations (because of the world we live in) where you need a method to VALIDATE who is actually sending the msg that you're wanting to send, and for that reason Lasso has built into it an authentication routine to login to an SMTP server and validate the sending msg:

```
[email_send:
-host='domain.com',
-port= 1-OPTIONAL,
-username= 2-OPTIONAL,
-password= 3-OPTIONAL,
-to='info@domain.com',
-from='site@domain.com',
-subject='MY SUBJECT',
-body=(include:'email.txt')]
```

1.) With the advent of LP8 you can NOW specify the PORT your mail server will respond on. In previous versions LP6/7 this wasn't possible, unless you changed the start up lassoapp. Lasso had port 25 hard coded into itself. So you'd have to modify it to allow for another port.

2.) username. This is the username of the SENDING account! In the case of EIMS, that sending account is: user%domain.com.

3.) password. this is the password of the username above.

Having got past the inane parts of the msg structure...let's move on to the actual FUN parts of this...the HTML mail...so how do you send plain text AND HTML mail at the same time ? Again with the advent of LP8, sending HTML mail is pretty damned simple...you see there's a reason why you upgraded (and no its not because of the hottie on the Lasso box, wait, there is NO hottie on the Lasso box.....OP! Hello let's get with the program here...as software developers its a requirement that you throw some scantily clad woman on the box to signify that this is sexy stuff...and that by buying your product we are buying sex sex sex...oh and a really cool app server....AHEM, I digress).

Adding HTML to the mix is this simple:

```
[email_send:
-host='domain.com',
-port= 1-OPTIONAL,
-username= 2-OPTIONAL,
-password= 3-OPTIONAL,
-to='info@domain.com',
-from='site@domain.com',
-subject='MY SUBJECT',
-body=(include:'email.txt'),
-html=(include:'email_html')]
```

How easy is that. One caveat when sending HTML mail, is that any images you wish to embed must be in absolute format, ie: http://www.youdomain.com/images/xxxxx.jpg

What will get sent out is actually a two part msg. Well three if you really sit and think about it. The msg will have to primary parts one in MIME format and the other in TEXT. So what's the 3rd part, the headers which are defining the msg itself.

Lastly, something to keep in the back of your mind, is that in the Admin for Lasso 5/6/7, the email sweep was automatically set to 300 seconds, that means that every 5 minutes, Lasso would check the EMAIL queue for NEW msgs that had to go out. You could of course change this to be whatever you want it to be. With LP8 things are just a lil different. You have more options in the form of an email delay between sweeps as well as space for SMTP AUTH User/Pass pair, and space for a default port.

PART THREE: ITS IN THERE!

Lasso 5 - 8 has some wonderful tools in it, and one of those tools is the event manager. This is how we're going to accomplish sending out our LOG files to an end user. Scheduling an event to happen at a specific date, time, and using a specific URL is pretty damned easy, a lot easier than makin an appointment to go see a dentist and a helluva lot less painful. What does this tip about EMAIL have to do with EVENTS ? The long story short is that there will be times in developing your applications where you're not going to want to send a msgs immediately or during the next email sweep.

In the email above that Im sending out to my clients every morning, the email is the RESULT of an event, and the event as I said earlier is the process of a VERY LONG laundry list of SQL calls and calculations to determine browser type, file sizes, distinct IPs, etc....it then compiles all of that and sends out a msg to the end user to display their morning report...which most clients absolutely LOVE.

There is one tiny hiccup to this process running form another domain OTHER than the client's domain. And it will require you to open up the compatibility.lasso file inside your Lasso Documentation > 3-Language Guide > LassoApps > Startup folder. Find the compatibility file and make 3 modifications to the LOG tag. You're going to comment out the following lines, like so (lines 115 - 118):

```
  //fail_if: #_file->(beginswith: '///'), -9956, '[Log] Cannot write to fully qualified
paths';
  //fail_if: #_file->(endswith: '/'), -9956, '[Log] Cannot write to a directory';
  //fail_if: (#_file >> '//'), -9956, '[Log] Cannot write to fully qualified paths';
  //fail_if: (#_file >> '..'), -9956, '[Log] Cannot use .. in log paths';
```

You're doing this so that the log tag will be able to write to any path. When you're done, recompile the Startup folder, and then replace the resulting app with your new startup app, then restart lasso. Next up is you'll have to enable permissions for the admin user to run the log tag. Also you'll have to add .txt as a viable file tag that Lasso can manipulate (as well as, XLS...excel files).

The Event in question is below....Ive removed some of the more esoteric items and aggregates...but you'll get the general gist:

```
  <?LassoScript
  //  MAGICMILES SOFTWARE LOG REPORTING. © 2006 magicmiles software.
  //
  //  this is a site report for domains on xxxxxxxxxxxxx.com
  //  each domain is run in 15 second intervals as an event from the
  //  lasso admin.
  //
  //  It sends a report to the users which is culled from the sql statements below for
  //  today's date, each record in the counter db ($v_db_connection) below is uniquely
```

```
tagged with
  //  their unique site_id marker.
  include: 'insert-dbvalues.inc';
  var: 'id'           = (action_param:'id'); //passed site id variable
  var: 'v_today'      = (date_format:(server_date), -format='%Y-%m-%d'); //date
variable
    inline: ($v_db_connection) 'auth_siteid'=($id), -search;
     var: 'v_domain' = (field:'auth_domain');   //domain to run from and to
     var: 'db_name'  = (field:'auth_db');     //database name
     var: 'tb_name'  = (field:'auth_tb');     //table name
     var: 'site_id'  = (field:'auth_siteid');
//site_id matches database field name of same name
     var: 'users'    = (field:'auth_users');    //users to send report to
     var: 'v_user'   = (field:'auth_username'); //username
     var: 'v_pass'   = (field:'auth_password'); //password
    /inline;
  //counts page name values
  var:'sql_pagecount'=(string);
  $sql_pagecount +='SELECT DISTINCT page_name, count(page_name) as page_count FROM '
+ ($tb_name) + ' WHERE site_id = "' + ($site_id) + '"';
  $sql_pagecount +=' AND date_time LIKE "%' + ($v_today) +'%" GROUP BY page_name
ASC';
  //counts referrer hits
  var:'sql_pagereferrer'=(string);
  $sql_pagereferrer +='SELECT DISTINCT referrer_url, count(referrer_url) as ref_count
FROM ' + ($tb_name) + ' WHERE site_id = "' + ($site_id) + '"';
  $sql_pagereferrer +=' AND date_time LIKE "%' + ($v_today) +'%" GROUP BY referrer_
url ASC';
  //counts page ip hits
  var:'sql_pageip'=(string)
  $sql_pageip +='SELECT DISTINCT ip_address FROM ' + ($tb_name) + ' WHERE site_id =
"' + ($site_id) + '"';
  $sql_pageip +=' AND date_time LIKE "%' + ($v_today) +'%" GROUP BY ip_address';
  //similar to pagecount
        var:'sql_hits'=(string);
  $sql_hits +='SELECT a_browser FROM ' + ($tb_name) + ' WHERE site_id = "' + ($site_
id) + '"';
         $sql_hits +=' AND date_time LIKE "%' + ($v_today) +'%"';
        //counts macintosh platform
        var:'sql_pagemac'=(string);
  $sql_pagemac +='SELECT a_browser FROM ' + ($tb_name) + ' WHERE site_id = "' +
($site_id) + '"';
          $sql_pagemac +=' AND date_time LIKE "%' + ($v_today) +'%"';
          $sql_pagemac +=' AND a_browser LIKE "%MAC%"';

        //counts windows platform

        var:'sql_pagepc'=(string);
  $sql_pagepc +='SELECT a_browser FROM ' + ($tb_name) + ' WHERE site_id = "' +
($site_id) + '"';
          $sql_pagepc +=' AND date_time LIKE "%' + ($v_today) +'%"';
          $sql_pagepc +=' AND a_browser LIKE "%WIN%"';
  //counts unix/other platform
  var:'sql_pageother'=(string);
  $sql_pageother +='SELECT a_browser FROM ' + ($tb_name) + ' WHERE site_id = "' +
($site_id) + '"';
         $sql_pageother +=' AND date_time LIKE "%' + ($v_today) +'%"';
         $sql_pageother +=' AND a_browser NOT LIKE "%WIN%" AND a_browser NOT LIKE
"%MAC%"';

       //counts ie-windows browser
    var:'sql_pagemsie'=(string);
       $sql_pagemsie +='SELECT a_browser FROM ' + ($tb_name) + ' WHERE site_id = "' +
($site_id) + '"';
          $sql_pagemsie +=' AND date_time LIKE "%' + ($v_today) +'%"';
          $sql_pagemsie +=' AND a_browser LIKE "%MSIE%"';
          $sql_pagemsie +=' AND a_browser LIKE "%WIN%"';
          $sql_pagemsie +=' AND a_browser LIKE "%WINDOWS%"';
```

```
       //counts ie-mac browser

        var:'sql_pagemacmsie'=(string);
   $sql_pagemacmsie +='SELECT a_browser FROM ' + ($tb_name) + ' WHERE site_id = "' +
($site_id) + '"';
          $sql_pagemacmsie +=' AND date_time LIKE "%' + ($v_today) +'%"';
          $sql_pagemacmsie +=' AND a_browser LIKE "%MSIE%"';
          $sql_pagemacmsie +=' AND a_browser LIKE "%MAC%"';

       //counts other browser
  var:'sql_pagebrowser'=(string);
   $sql_pagebrowser +='SELECT a_browser FROM ' + ($tb_name) + ' WHERE site_id = "' +
($site_id) + '"';
          $sql_pagebrowser +=' AND date_time LIKE "%' + ($v_today) +'%"';
          $sql_pagebrowser +=' AND a_browser NOT LIKE "%SAFARI%"' ;
          $sql_pagebrowser +=' AND a_browser NOT LIKE "%NETSCAPE%"';
          $sql_pagebrowser +=' AND a_browser NOT LIKE "%MSIE%"';
          $sql_pagebrowser +=' AND a_browser NOT LIKE "%MAC%"';
          $sql_pagebrowser +=' AND a_browser NOT LIKE "%WIN%"';


        //compiles all sql calls into inlines for variable processing.

  inline: -inlinename='pagereport', ($v_db_connection), -op='cn', 'date_time'=($v_
today), 'site_id'='xxxxxxxxx', -search;/inline;
  inline: -inlinename='sql_pagecount', ($v_db_connection), -SQL=($sql_pagecount);
error_currenterror; /inline;
  inline: -inlinename='sql_pagereferrer', ($v_db_connection), -SQL=($sql_
pagereferrer);/inline;
  inline: ($v_db_connection), -SQL=($sql_pageip);var:'v_ip'=(found_count);/inline;
  inline: ($v_db_connection), -SQL=($sql_hits); var:'v_hits'=(found_count);/inline;
  inline: ($v_db_connection), -SQL=($sql_pagemac);  var:'v_mac'=(found_count);/inline;
  inline: ($v_db_connection), -SQL=($sql_pagepc); var:'v_pc'=(found_count);/inline;
  inline: ($v_db_connection), -SQL=($sql_pageother); var:'v_other'=(found_count);/
inline;
  inline: ($v_db_connection), -SQL=($sql_pagemsie); var:'v_msie'=(found_count);/
inline;
  inline: ($v_db_connection), -SQL=($sql_pagemacmsie);var:'v_mmsie'=(found_count);/
inline;
  inline: ($v_db_connection), -SQL=($sql_pagebrowser); var:'v_browser'=(found_count);/
inline;
  inline: ($v_db_connection), -SQL=($sql_pagemb2);  var:'v_mb2'=(found_count);/inline;
  //this next section defines variables for the report values: date of report, and
logname files.
          var:'v_subdate'=(string_concatenate:'WEB LOG REPORT FOR: ',(date_format:
($v_today), -format='%A, %B %d %Y'));
    var:'v_logname_txt'='///Library/WebServer/Documents/~' + ($v_domain) + '/weblogs/
weblog-' + ($v_today)+'.txt';
    var:'v_logname_xls'='///Library/WebServer/Documents/~' + ($v_domain) + '/weblogs/
weblog-' + ($v_today)+'.xls';
    var:'v_filename_txt'='http://www.' + ($v_domain) + '.com/weblogs/weblog-' + ($v_
today) +'.txt';
    var:'v_filename_xls'='http://www.' + ($v_domain) + '.com/weblogs/weblog-' + ($v_
today) +'.xls';
    //this next section actually creates the reports in the users directory in two
formats: text and excel!
    log: ($v_logname_txt);
      $v_subdate + '\r\r';
      string_uppercase: ($v_domain) + '\r\r';
      '#' + '\t' + 'page' + '\t' + 'date' + '\t' + 'ip' + '\t' + 'domain' + '\t' +
'browser' + '\t' + 'referrer_url' + '\n';
      records: -inlinename='pagereport';
      (loop_count) + '.)' + '\t' + (field:'page_name') + '\t' + (date_format:(field:
'date_time'), -format='%h:%M %p') + '\t' + (field:'ip_address') + '\t' + (field:
'domain_address') + '\t' + (field:'a_browser') + '\t' + (field:'referrer_url') + '\r';
      /records;
    /log;
```

```
    log: ($v_logname_xls);
      $v_subdate + '\r\r';
      string_uppercase: ($v_domain) + '\r\r';
      '#' + '\t' + 'page' + '\t' + 'date' + '\t' + 'ip' + '\t' + 'domain' + '\t' +
'browser' + '\t' + 'referrer_url' + '\n';
      records: -inlinename='pagereport';
      (loop_count) + '.)' + '\t' + (field:'page_name') + '\t' + (date_format:(field:
'date_time'), -format='%h:%M %p') + '\t' + (field:'ip_address') + '\t' + (field:
'domain_address') + '\t' + (field:'a_browser') + '\t' + (field:'referrer_url') + '\r';
      /records;
    /log;

    //this next section sends the file report.inc to ($users) variable.

      email_send:
        -From='xxxxxx@xxxxxxxxxx.com',
        -to=($users),
        -bcc='xxxxxx@xxxxxxxxxx.com',
        -host='xxxxxxx%xxxxxxxx.com||xxxxxxxxxx||xxxxxxxxxx.com',
        -Subject=($v_subdate),
        -Body=(include: 'report.inc');
email_send:
        -host='domain.com',
        -username=($v_smtp_u),
        -password=($v_smtp_p),
        -to=($users),
        -from='site@domain.com',
        -subject=($v_subdate),
        -body=(include:'report.inc'),
        -html=(include:'report-html.inc');
?>
```

If you can decypher all of that, skipping the SQL parts, you'll note a few things not the least of which is that there is nothing magical about what's being done here...absolutely NOTHING. One thing to note is that because this app runs on both a LP6 server and an LP8 server I had to allow for the proper language for the email send statement. Look at the host line for the commented out LP6 code, note that its username||username||hostname, whereas in LP8 those values are divided out. This logging tool also writes out an actual log file in two formats, txt and excel.

So how is this all run, by an event url, like so:

http://www.xxxxxxxxxxxxx.com/customersite.lasso?id=XXXXXX

In short this is a wonderful logging tool even if I do say so myself...and the best part is that you're getting it for free. If you'd like the DB structure, Im happy to send it to you so that you can start using it today.

Simplifying your Coding Life with Custom Types

Göran Tornquist

Introduction

In my other paper created for this Lasso Summit, I have described the benefits gained when using custom types together with access to database records. The focus on this paper is to give the foundation for discussions about general benefits about custom types and how they could be implemented.

I have defined three custom types to introduce you to the benefits of custom types. They will be briefly described by the following subtitles.

- What is the purpose of the custom type?
- Member tags.
- Examples of usage.

Which is the Faster? Custom Types or Custom Tags?

That is a question that has both a simple and a more elaborate answer. The simple, and generally true answer is that custom tags are faster than custom types.

Everything comes with a price

Custom types come with two extra costs: extra development time to handle the general case instead of the specific case, and inefficiency due to lack of optimization.

A custom type is a definition of a "thing" that has the attributes and the knowledge to manipulate those attributes. Normally, optimizations are not present. They would require information outside of the scope for the custom type. That would break the rule that says that custom types (or classes as in other OOP languages) should be self-contained.

There's more inefficiency as well. Lasso has to handle the encapsulation of the custom type member variables and member tags. This requires some overhead, as every object – a created instance of the custom type – does need a general preparation. To minimize this, there is the `-prototype` keyword that can be used together with the definition of a custom type. When doing this you tell Lassoscript that there is no code that needs to be executed at the time of the creation of the object. In other words, there is no outside dependency that has to be accounted for. Using `-prototype`, simply makes Lassoscript copy a ready-made template, which makes it almost as efficient as the native types that are built-in into Lassoscript.

What is the Definition of Fastest Code? Quick and Buggy Code or Slower and Correct Code?

If you feel challenged by this subtitle, please understand that I do not prefer OOP at all times. There are a lot of cases where it does not apply for reasons of efficiency, complexity and interchange with other languages – just to name a few. Now that you've read this disclaimer, please proceed.

The purpose of OOP is to make a simplified and general abstraction of a certain entity that is found in a solution. Almost every thing in this world appears more complex when we look a bit closer at it. Custom types give you the possibility to create a complex operation that will look very simple from the outside.

The question above is a rhetoric question with the assumption that it is easier to create bug free code with custom types. Well-defined behavior can be tested through unit testing and therefore we can predict the most of the results of the behavior. Also, since the code only relies on outside information passed on through the member tags, we know that the consistency of the object is preserved at all times.

Are Custom Types Cost Efficient?

The answer of the question above depends on the resources and the usage of the solution in a whole. Which boundaries do you operate within? The two main factors are time and cost – and both results in money in the end.

- If you have a lot of time before the solution needs to be there, then you can afford to create optimized and specific code.

- If you can afford multiple resources working on the solution, then you will have a high performance solution. Resources is regarded to be many developers working on the problem, or many servers co-operating to deliver the solution.

Time and cost are often restricted by budgets or deadlines, and can be regarded as absolute in the specific case. But there is actually one more factor that gives us the boundaries for the solution. The quality of the solution as whole must not be negotiable, however completely bug-free code is impossible and therefore there's no such thing as an absolute value of the quality factor.

The main cost benefits from custom types can be found in code reuse and less quantity of bugs. Full code reuse takes place when you can create a custom type for one solution, and you use it in another solution without any changes. Whether you will be able to take advantage of full code reuse or not, depends on the balance between the depth of investigation of the abstraction and the cost (in time and resources) you that can afford.

Often there will be room for extension of the custom type, which will affect and benefit both the old and the new solution. Obviously that demands that the extension does not violate the boundaries of the custom type abstraction. If it does, then we have a back compability problem.

C_URL – a Custom Type for handling URL's

You find the custom type C_URL in the file "ctypes.inc" that comes with this paper.

What is the purpose of the custom type?

To deliver correct handling and evaluation of URL's (or URI's) as defined by W3C. Since this is a simple example, the full purpose has not been fulfilled.

Member Tags

You use the custom type trough one of the following member tags. All getters will return a valid value if the URL as a whole is considered valid.

Member Tag	Purpose
getURL	Guaranteed to either deliver a working URL that is consistent with the definition of a URL and the usage depending on the supported protocol, or an empty string if the url is not valid.
setURL: 'url string'	Accepts a string that defines the URL, and stores it within the object.
getProtocol	Returns the protocol part of the URL.
setProtocol:'protocol string'	Sets the protocol part of the URL with the given protocol string.
getHost	Returns the host part of the URL.
setHost: 'host string'	Sets the host part of the URL with the given host address.
getPath	Returns the path part of the URL.
setPath	Sets the path part of the URL.

Examples of Usage

example a

```
var: 'homepage' = (C_URL: 'http://www.cortland.se/summit/');
'a: ';
$homepage->getURL;
'<br />';
//example b
var: 'html' = '<a href="' + $homepage->getURL + '">';
'b: ';
if: $homepage->getProtocol == 'mailto:';
  $html += 'Send a mail';
else;
  $html += 'Click here';
/if;
#html += '</a>';
#html;
'<br />';
//example c
'c: ';
$homepage->setProtocol: 'ftp:';
$homepage->setHost: 'ftp.cortland.se';
$homepage->getURL;
'<br />';
```

C_String – an Extension of the Native String Type

You find the custom type C_String in the file "ctypes.inc" that comes with this paper.

What is the purpose of the custom type?

Two purposes:

1. To prove the possibility of using strings that has been extended by use of inheritance.
2. To simplify code that manipulates the string – and thereby avoid bugs.

Comment: When the [String_CountFields] and [String_GetField] tags where deprecated, I was forced to change my solutions in quite a few places to accommodate for the new and more efficient [String->Split]. But as I was changing those, most of the time, there was not time and resources available to change the structure of the code. Without a structure change, the benefits of performance were missing. Also I found myself checking for just one field to be present, and

then continue with some processing. The code became more complicated and less readable, which is a threat to future development and handling of bugs.

So, here they are again. Built with [String->Split], with the promise of low performance, but simpler looking code.

Member Tags

You use the custom type trough one of the following member tags.

Member Tag	Purpose
left: 'length'	Returns a maximum of 'length' characters from the left side of the string.
right: 'length'	Returns a maximum of 'length' characters from the right side of the string.
countFields: 'delimiter'	Returns the number of string fields that are divided by the given delimiter.
getField: 'delimiter', 'position'	Returns the string field at the given position, delimited by the given delimiter. If not found, it will return null.

Examples of Usage

```
var: 'name' = (C_String: 'Göran Törnquist');
//example a
'a: ';
$name->(left: 5);  //returns 'Göran'
'<br />';
//example b
'b: ';
$name->(left: 0);  //returns 'Göran'
'<br />';
//example c
'c: ';
$name->(right: 5);  //returns 'quist'
'<br />';
//example d
'd: ';
$name->(right: 256); //returns 'Göran Törnquist'
'<br />';
//example e
'e: ';
$name->(right: 0); //returns the empty string
'<br />';
//example f
'f: ';
var: 'address' = (C_String: 'ftp://ftp.cortland.se/summit/');
$address->(countFields: '//'); //returns 2
'<br />';
//example g
'g: ';
$address->(getField: '//', 2); //returns 'ftp.cortland.se/summit/'
'<br />';
```

C_Integer – an Replacement of the Native Integer Type

You find the custom type C_Integer in the file "ctypes.inc" that comes with this paper.

What is the purpose of the custom type?

Two purposes:

1. To prove the possibility of replacing the native integer, in certain cases, yet be able to fully calculate without implicit conversions.
2. To make formats, that has been set by `C_Integer->setFormat`, not be reset everytime we perform a calculation or assign a value to the integer.

Comment: There was no way I could implement this using inheritance from the native integer type. The reason for this is possibly the optimizations done in the implementation of the member tags of the integer type.

Member Tags

There are no special member tags that differs from the normal integer. The main feature of this custom type is that it is not any different from an integer when it comes to calculation or the alike. However, once a format has been set using C_Integer->setFormat, it will stick. No matter the assignments being done to the variable that contains the object.

Examples of Usage: The native integer behavior

```
var: 'salary' = 1000;
$salary->(setFormat: -groupChar=' ');
//example a
'a: ' + $salary + '<br />';
//example b
$salary += 5;
'b: ';
$salary + '<br />';
//example c
$salary = 2000;
'c: ';
$salary + '<br />';
```

Examples of Usage: The C_Integer behavior

```
var: 'salary' = (C_Integer: 1000);
$salary->(setFormat: -groupChar=' ');
//example a
'a: ' + $salary + '<br />';
//example b
$salary += 5;
'b: ';
$salary + '<br />';
//example c
$salary = 2000;
'c: ';
$salary + '<br />';
```

Epilogue

I expanded my own knowledge while researching for this paper. I had a clear and distinct idea what I wanted to prove with the paper and the presentation. The last custom type, C_Integer, took quite some time to realize and to work 100% the way I wanted it to. The path to fully understanding the inheritance and overloading of callback member tags and symbols was a long and troublesome one.

If I would be lazy, I'd blame either the implementation of Lasso, or the documentation for this, but I don't. When it comes to the inner workings of inheritance and the full operations of a cascading member tag, it is not an easy subject.

One of the reasons for my troubles was the implicit conversion of data, which made me suffer from a infinite recursion until I finally understood how it works.

The first two examples are actually quite simple. The advanced part in them lies in understanding how the callback member tags are working.

The third example is more of a subject for the fearless. The reason I developed that custom type was that I was tired of keeping track of whether formatting was done or not. With this custom type in place, I can rely on the formatting being in place when I'm converting to strings for presentation.

Happy ctyping

/Göran

Using ImageMagick, Lasso and Passthru

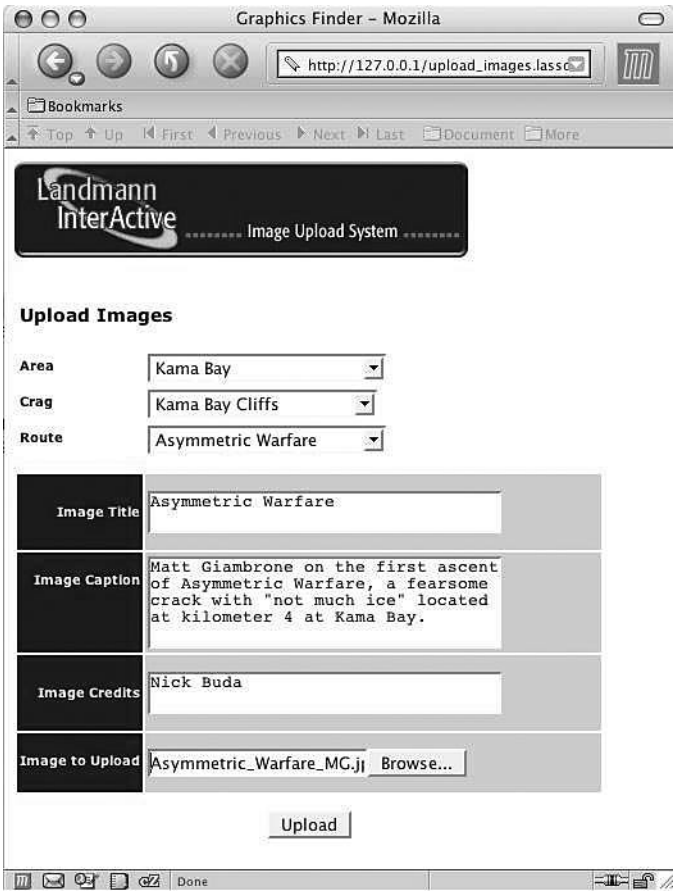
Eric Landmann

INTRODUCTION

Photo upload systems that "do it all" for the user are becoming increasingly common on websites. Systems that allow a user to upload a photo and capture some additional information are used in many types of applications. Some of these include an employee directory or staff listing, an image bank, a blog, a members area of a chat site, a used equipment site, or a real estate listings system.

The examples provided arefor a site providing information about climbing routes. The photo uploaded might show a picture of the climb which would be of interest to other climbers attempting the route.

This functionality would typically be found in an administrative or user area to restrict access from the general public. The example provided does not show any user authentication code or scheme; this would have to be added by the coder. Below is a screenshot of the application just before the user submits the form.



Why do it this way?

Lasso comes bundled with ImageMagick, a very useful suite of utilities that can manipulate images in many ways. Lasso ties into ImageMagick through the use of Lasso's Image tags.

Using Lasso's builtin Image tags, you can resize an image, get info on an image, crop images, sharpen, and do many other things. Our solution does not use all of this functionality. Instead, we use PassThru, a plugin that issues Terminalstyle commands, to pass commands directly to ImageMagick. It is a fair question why we would use this method instead of using the builtin Lasso image tags. Here are some reasons:

1. Speed — Working with ImageMagick directly is faster than using Lasso's tags
2. Error Reporting — We can capture more descriptive errors directly from ImageMagick
3. Flexibility

   — by using a separate install of ImageMagick, you can take advantage of new releases

   — You can tailor ImageMagick commands to specific image situations

   — Upload and image directories are easily tailorable

FEATURES OF THIS APPLICATION

Requires the user to make selections from dropdowns to be able to upload an image. This process identifies a record in the database to which the photo will be related. Cleans up troublesome filenames and creates a unique filename for the uploaded source image file.

Creates three versions of the image, with modifiable sizes.

Created images are placed in three separate folders, allowing one database entry for the filename.

Folder paths called in HTML determine which size photo is accessed; the filename is the same.

This version works with .jpg files, but is easily modifiable to include other file formats by calling the appropriate ImageMagick convert command.

Employs a siteconfig file, which has sitewide settings and makes moving code from one site to another relatively easy.

Original file can be archived if desired (set in the siteconfig file).

The upload page is selfposting (less code maintenance).

SOFTWARE USED

Lasso — To communicate with the webserver, databases, and do various utility tasks
ImageMagick — To analyze images and create derivative images. We are using the Lasso installed version in this example for ease of use, but an external installation certainly could easily be used.

PassThru — Aplugin that communicates between Lasso and the Terminal to pass commands through to and receive message from the operating system. It is a commercial addon. See the "References" section for where to get PassThru.

MySQL — To store the record information for the routes and images.

OPERATING PRINCIPLES

Here is how this application operates. The user is presented with a form with a dropdown dialog. When an "Area" is selected, the form selfposts, a database lookup is performed, and a "Crag"

dropdown list is presented. When a crag is selected, the form selfposts again, and a list of climbs are presented. When a climb is selected, the form selfposts again, and presents a form to fill out an image title, caption, credits, and a browse button to select the photo to upload. After the photo is selected and submit, the form selfposts one final time.

If all the form submissions pass the tests, the process_uploads.inc file does all the heavy lifting. This include file contains all the code to clean up the submitted filename of improper characters, create a unique new filename for the uploaded file, create the three sizes of thumbnails (called large, small, and thumb images) stored in their respective folders, make the database entry, and take care of file maintenance. Here is a code snippet that produces the large image (line 128 of process_uploads.inc). It also checks whether the image is wider or taller, and issues the appropriate ImageMagick command:

```
// <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
//
Make Large version
// <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
If:(#Height < #largeheight) || (#Width < #largewidth);
Local('ResizeTest') = 'Image Height or Width LESS THAN large image, make large
version, no resize<br>\r';
Local:'MakeLarge' = ($svPathToIM 'convert -density 72x72 -colorspace RGB "'
($svWebserverRoot) (#ThisFilePath) '" "' (#ULPathImageLargeOUT) '"');
Else;
Local('ResizeTest') = 'Image Height or Width GREATER THAN large image, make
large version, resize 600x600<br>\r';
Local:'MakeLarge' = ($svPathToIM 'convert -density 72x72 -colorspace RGB
-resize ' (#largewidth) 'x' (#largeHeight) ' "' ($svWebserverRoot) (#ThisFilePath)
'" "' (#ULPathImageLargeOUT) '"');
/If;
```

Immediately after this, and sprinkled throughout the code are various places that display information when debugging is on. This particular bit of code displays the value of the variables ResizeTest and MakeLarge:

```
If: $svDebug == 'Y';
'<p class="debug">\n';
'141: ResizeTest = ' (#ResizeTest) '<br>\r';
'141: MakeLarge = ' (#MakeLarge) '<br>\r';
'</p>\n';
/If;
```

Immediately after this (at line 146), the PassThru command is run, and the output of this command is captured in the variable "converting":

// Run the PassThru command to convert the file
Local('converting') = PassThru(#MakeLarge, -username=$svPassThruUsername, -password=
$svPassThruPassword);

At this point, the command has been passed to ImageMagick, and it should be doing its thing. If you are watching the server when executing this, you should see the newlycreated files appear in the image folders under /site/images.

FILE STRUCTURE

SQL Files

A filecalled beta_content_demo.sql contains the structure and data of the tables used in this application. To use this, create a table in MySQL called "beta_content_demo", and run the beta_content_demo.sql file. It will create the four tables listed below and populate them with test data.

Database and Table Descriptions

beta_content contains information for the climbs

beta_crag contains information about the crags (climbing areas)

beta_errors contains information accessed by the Show_Error tag

beta_images contains information about the uploaded images

Code Files

Files of note are as follows:

upload_images.lasso — The entry point for this application

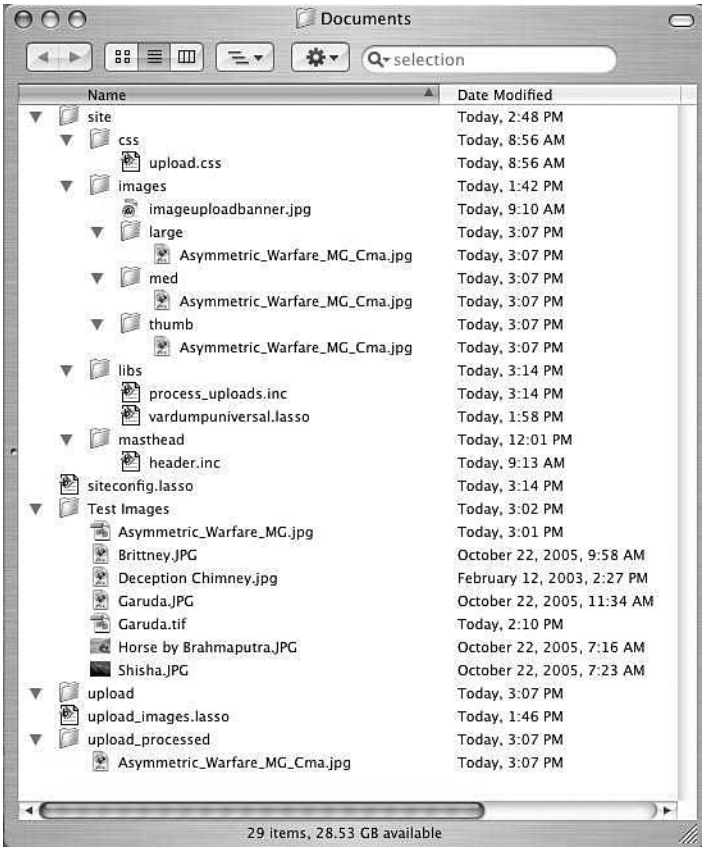
siteconfig.lasso — controls site and serverspecific specific information

process_uploads.inc — does the actual image file creation, communicating with ImageMagick via PassThru; also creates the database entry and handles error and message generation

vardumpuniversal.lasso — A handydandy variable dump utility that you can put anywhere in a page to display some information about the current file and the value of variables. It is written to not display Lasso "reserved" variables (those starting with "_") or siteconfig variables (those starting with "sv"). Used in debugging and site development.

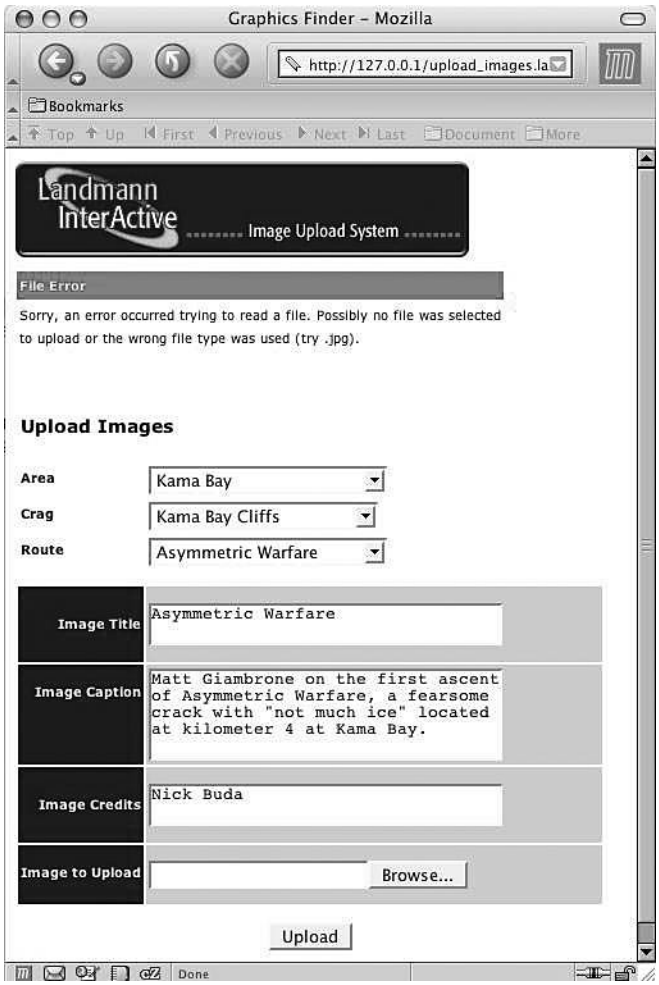
Directory Display

The list below shows a typical image upload system directory after one image (named "Asymmetric_Warfare_MG.jpg") has been uploaded. The three generated image files (a large, medium, and thumb version) are each in their own subfolders of /site/images. The original file has been renamed and moved to upload_processed.



Error Checking and Display

A custom tag called "Show_Error" is defined in the siteconfig.lasso file. It takes several parameters and returns a formatted table with a message. It could be an error or it could be a success message, depending upon the code that is passed to the tag. The code that is passed to the tag is typically a fourdigit number, like "5061" which means "Upload Successful." Below is a typical error message displayed at the top of the page:



This tag can easily be used with a redirect by passing a parameter "error" through the URL.

For example:

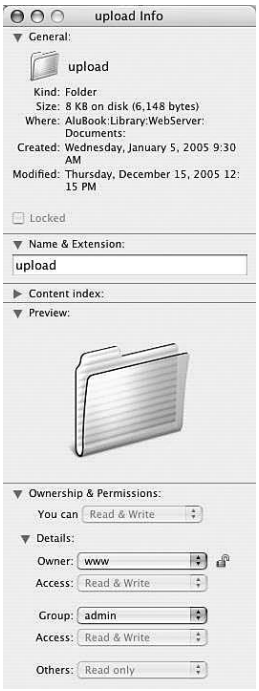
```
'upload_images.lasso?Error='$vEr
ror'&Option='$vOption;
```

The error is then displayed by calling the Show_Error tag as follows:

```
Show_Error: -ErrNum=(Var:'vError'), -Option=(Var:'vOption'),
-PosColor='0099FF', -NegColor='FF0000', -BgColor='CCCCCC;
```

See the tag definition in the siteconfig for info on what the parameters mean.

FILE AND FOLDER PERMISSIONS



Lasso is very persnickety about permissions to perform file manipulations. Rather than repeat the documentation about how to do it, check the Lasso documentation regarding file tags and folder permissions. The example files make use of a user ID and password for this purpose, set in the siteconfig. One commonly overlooked aspect is setting folder permissions for the enclosing folder where the graphics are stored. See screenshot below for example folder permissions.

DEBUGGING

This can be controlled either globally through a variable declaration at the top of the siteconfig (by setting svDebug = Y), or can be turned on for any page or code block. Turning debug on will allow you to see all the commands sent to ImageMagick, all the database calls and variable states.

Debugging also calls the vardumpuniversal.lasso file. This file displays some page information, action_params that have been submit to that page, and the variable values. It is typically used at the bottom of the page. A sample of debugging output is below:

```
PAGE INFO
Response_Filepath = /upload_images.lasso
Referrer_URL = http://127.0.0.1/upload_images.lasso

ACTION_PARAMS DUMP
Area = Orient Bay
Crag = Orient Bay
RouteID = 6

VARIABLE DUMP
vImageTitle =
vCrag = Orient Bay
vOption =
vImageCredits =
SQLSearchAreas = SELECT DISTINCT Crag_Area FROM beta_crag ORDER BY Crag_Area
vRouteID = 6
SQLSearchCrags = SELECT DISTINCT Crag_Crag FROM beta_crag WHERE Crag_Area = "Orient Bay"
newparams =
vAreaSelect = Thunder Bay
SQLSearchRoutes = SELECT DISTINCT Route_Name,ID FROM beta_content WHERE Crag="Orient
Bay" ORDER BY Route_Name
vError =
vArea = Orient Bay
vCaption =
vProcess =

vardumpuniversal.lasso loaded
```

REFERENCES

The Lasso documentation provides examples of how to set up the file tags to allow manipulations of the sort needed here. See Chapter 29 of the "Lasso 8 Language Guide."

PassThru

PassThru has its own reference manual (included with the software).

http://www.execuchoice.net/

ImageMagick Examples

http://www.cit.gu.edu.au/~anthony/graphics/imagick/

Lasso

http://www.omnipilot.com/

LassoTalk Archives

http://www.listsearch.com/

CREDITS

Thanks to the following Lasso coders for putting up with my learning curve, and contributing code bits to the community and to me directly:

— Pier Kuipers for the Create Unique ID tag
— Keith Schuster for the original idea about the error tag
— Greg Willits and Bil Corry for bits of the upload code
— Steffan Cline for help with PassThru
— Chris Corwin for help with ImageMagick
— the Lasso community, for being there

Development Tools Roundtable

Tom Wiebe

Eclipse Plug-ins

Eclipse offers a wealth of plug-ins, so much so that it can be a little daunting to know what to try and what to avoid.

MyEclipseIDE offers a huge number of features for a minimal cost of $29.95 for an annual subscription and can be an excellent way to test the waters. Most of the features within MyEclipseIDE, however, are open source utilities that have been packaged by the developers, Genuitec.

Many of the included features are windows only or highly focused on Java development.

Does include decent HTML, CSS, and Javascript editors. Current Windows version includes a Javascript Debugger. SQL editor is flakey on OS X, seems to work for some, not for others (not for me).

Good starter package for new Eclipse users, you can ignore the java features if you don't need them

There is a huge number of standalone plug-ins, that can accomplish just about any task you could imagine within the Eclipse environment.

Finding Eclipse Plug-ins

The Eclipse community is so large, there are hundreds of plug-ins available and at least a few good sites listing them

http://eclipse-plugins.2y.net/

http://eclipseplugincentral.com/

Some of my Favourites

Logfile View

Tail multiple log files directly within Eclipse. Very handy to watch LassoErrors.txt or apache logs while you run/debug scripts within Eclipse

http://www.mimo.ch/plugin.htm

Database Tools

Eclipse SQL

Decent SQL Editor within Eclipse

http://sourceforge.net/projects/sql-editor-plug

Azzurri Clay Database Modeler

Great Database modeling tool, development seems to have slowed on it, only supports Mysql 4.0 syntax (along with a variety of other databases via JDBC) but, does work great with Eclipse 3.1 and Java 1.5

http://www.azzurri.jp/en/software/clay/index.jsp

Free (Pro version available in Japan?!?)

AnyEdit tools

Adds some handy text manipulation features to Eclipse Editors. One less reason to open another editor. Converts tabs to spaces, html entities, capitalization in a small, simple tool. Extends existing Eclipse editors.

http://andrei.gmxhome.de/anyedit/index.html

Source Code Management

Eclipse includes excellent CVS integration right out of the box.

CVS VersionTree offers a graphical look at your cvs repository

http://versiontree.sourceforge.net/

Subversion support provided by the Subclipse plug-in, developed by the same people who developed subversion itself, also excellent integration

http://subclipse.tigris.org/

Plug-in for many other SCM systems (perforce, Visual Source Safe, Clearcase, etc) available at http://Eclipse-Plugins.y2.net

HTML, XML, CSS and Javascript Editors

The Eclipse project itself provides the Web Standard Tools, which provide HTML, Javascript and CSS editors. They offer code completion, syntax colouring, formatting and basic validation for HTML, Javascript and CSS files. Many other tools are built upon this platform.

http://www.eclipse.org/webtools/wst/main.html

XML editors are many in number, especially for Eclipse, given it's Java heritage and the common use of XML configuration files by nearly every Java Framework.

I like XMLBuddy for editing XML files, unlike most, it's simple and to the point and, pretty much works like you'd expect a text editor to work. There's a free version that offers basic XML editing capabilities or, the Pro version ($ 35.00 US) adds XPath features and an XSLT editor amongst other things. The developer is very responsive to queries as well.

Standalone Apps

This list is primarily Mac oriented but I've tried to include Java and cross platform apps where possible, so there'll still be something there for Windows and Linux users.

Text Editors

While Eclipse is an excellent environment, it is not primarily a Text editor. Sometimes, it's preferable to work in a standalone editor for expediency's sake. As well, if you 'just don't like Eclipse' as some are bound to do, you can still use it for project management and debugging and do your primary editing in another program by choosing the "Edit in System Editor" from the contextual menu when opening a file. This setting is remembered on a per-file basis so, once you open a file this way, it will always launch the system editor when you open it, until you choose to open it in one of Eclipse's editors (Also via Contextual menu)

BBEdit is still the trusty old standby it's always been. My 'go to' editor. I find, however, that it tends to get used less and less these days, for smaller and smaller tasks. I think this has a lot less to do with BBEdit itself and a lot more to do with the veritable cornucopia of other editors available now.

If you can only have 1 text editor for everything, BBEdit is still likely your best choice but many of the other editors excel within their various specialties.

Of note, the very cool SubEthaEdit, which allows multiple users to edit the same file. It also offers syntax colouring for LassoScript files, amongst many other languages, via Adam Randall's excellent LassoScript mode.

http://codingmonkeys.de/subethaedit/index.html

A recent addition to the editor pantheon is TextMate. It is an interesting product (Mac only) that makes rather bold moves with common user interface guidelines. With its file management features, snippets, macros and more, it approaches the features of an IDE. I have only played with it a bit but, while it's very different, it also felt nearly instantly familiar to me. It is a product to watch.

http://macromates.com/

TextWrangler is the replacement for BBEdit lite and, provides a much closer experience to that of BBEdit itself than lite ever did. Probably 80% of the functionality of BBEdit, for free! Definitely worth checking out if you haven't got the full version of BBEdit, Also handy to install on extra workstations and servers, where you might want a decent GUI editor but not want to purchase extra BBEdit licenses.

http://www.barebones.com/products/textwrangler/index.shtml

Database Tools

There is a plethora of tools available here, especially for MySQL.

From MySQL themselves come MySQL Administrator and MySQL Query Browser (both cross platform).

MySQL Administrator provides nearly complete GUI control of a MySQL Server, virtually eliminating the need to ever use the command line in administrating your MySQL server. Also provides backup functions, a must have

http://www.mysql.com/products/tools/administrator/

MySQL Query Browser is used to create, execute and optimize SQL queries. It is nice, free and cross platform. However, I find that some of the third party SQL Browsers are much nicer to use.

http://www.mysql.com/products/tools/query-browser/

Aqua Data Studio is something of a Swiss army knife of database browsers. A very well written Java app, it runs anywhere, talks to nearly every DBMS out there and is generally very full featured.

It is free for personal use or, $149 for commercial use.

http://www.aquafold.com/

DBVisualizer is another Java SQL editor that, like Aqua Data Studio, connects to virtually every DBMS out there. Worth checking out just to see it's interface, which is the best I've ever seen on a Java app.

Has integrated charting features, very nice for quick reports.

CocoaMySQL is very popular open source MySQL interface amongst Mac users, although the original developers seem to have stopped work on it since 2003 at this point.

http://cocoamysql.sourceforge.net/

Development seems to have been picked up by a third party though, an example of open source software working at it's best.

http://www.theonline.org/cocoamysql/

YourSQL is very nice, simple, free tool to build and query your MySQL databases, a native cocoa app under regular development, YourSQL has become my favourite tool to use in creating databases. Its feature set is much smaller than the other tools listed here but it is just fast and simple. Definitely worth having in your toolkit

http://yoursql.ludit.it/

PHPmyAdmin — If you need to manage a remote MySQL installation and don't want to allow direct access to MySQL over the network, PHPmyAdmin is the ticket. A feature rich web app for managing MySQL.

http://www.phpmyadmin.net/home_page/index.php

Command line tools

Too many to list them all, if you are on a Mac, install DarwinPorts and browse through the available programs. If you are on Windows, most of the unix command line tools will build and run under Cygwin.

http://darwinports.opendarwin.org/

http://www.cygwin.com/

Summary

There are almost as many interesting development tools out there as there are developers. Given the opportunity, I suppose most carpenters would become expert hammer makers as well. In software, everyone can be a hammer maker.

There is no 'Best Tool' out there for any one task. Use what you are comfortable with. There are general-purpose tools like Eclipse or BBEdit that work well for a variety of tasks. It is very useful to maintain a good working knowledge of a tool like this. Often, even if there is a 'better' tool for the specific task you are working on, using something like Eclipse for HTML editing eliminates the need to learn yet another tool.

Sometimes though, a tool will provide functionality that isn't available elsewhere, such as SubEthaEdit, or just work so much better than the alternatives that it its worthwhile to keep another program around for a specific type of job.

Variety is the spice of life, don't be afraid to experiment with new tools. You never know what might lurk around the next corner.

How to make sense of your hierarchical content

Seeing the Wood for the Trees — Categories, Menus and More.

Jonathan Guthrie

In pretty much everything we do online, we're dealing with organizing content, and mostly it's hierarchical content. Site content, menu systems, product categories and endless subcategories, forums, examples are everywhere we look.

We can just ignore it, put physical files in folders and let the client author hard files and put them in the right place, letting the user navigate something you hope is not broken because the client, one of your staff, or heaven forbid, you yourself, put something in the wrong directory. That's the first approach many of us started with, and yet once you get above just a few pages it becomes a nightmare to maintain.

XML is a perfect example of a data model that is hierarchical in nature, but not a lot of people store dynamic content as XML in situations where it's read from and written to frequently, and for very valid reasons.

Relational databases are essentially flat data with relationships, and yet while creating a flexible hierarchical data model can be challenging, using a database to store and retrieve data fits well with the way most of us work.

Multi-table approach

A system I used in the early days of content management was to have a separate table for each level of navigation. I had primary, secondary and tertiary nav tables, and while it made many things drop dead easy, it was severely limited in flexibility, and I had to have an additional table for content.

Primary	Secondary	Tertiary	Content
ID	ID	ID	ID
Name	Name	Name	Page Name
Display_order	Primary	Primary	Primary
Status	Display_order	Secondary (optional)	Secondary
	Status	Display_order	Tertiary
		Status	Content
			Status

It's a workable system, if limited. It has the advantage of being simple to understand, but laden with snafus such as...

"Can I have another level in my menus please"

"Sorry, that's all you get" became a frequent story during deployment of new sites, and we often justified it successfully with "Best Practice Guidelines" and all that garbage. Then again Best

Practice is only Best Practice as long as you have need of it, or no-one else has developed a better practice.

Getting the data out

To generate the menus or list of categories, you can either nest your queries (inlines) or try the one-hit SQL join. Either way gets ugly, and as much as it pains me to admit, the nesting of queries/inlines is often the only way to truly control sense of place etc.

Should I be seeing this?

It's wise in any CMS or product catalogue to allow content to be worked on while not actively viewable by Joe Average, so you need a status flag. However unless you join your SQL to ascertain the status of the content, you have to store status in the nav tables. That gets painful to maintain, and the entire situation can escalate out of control.

Data Integrity (Referential)

If you have a page that's attached to a tertiary nav row, what happens if the tertiary row's deleted? You get an orphaned content row. It makes your job as a developer harder because at every step you have to ensure the integrity of multiple table's data, and there's no easy way to lock that down on the database side.

The Adjacency Model

There are quite a few other variations that rely on multiple tables and on single tables, but we're going to focus on just 2 more. The easier of the two to understand and hence the more popular one is the Adjacency Model.

If you look at a typical off-license, the product catalogue might go something like this: (abbreviated!)

```
Alcoholic Beverages
  Beers
  Wines
    White Wines
      Chardonnay
        Oaked
        Unoaked
Sauvignon Blanc
Pinot Gris
Riesling
Gewurztraminer
Viognier
Semillon
    Red Wines
    Champagne
    Sparkling
    Sweet Dessert
    Ports/Sherries
  Spirits
Non-Alcoholic Beverages
Snacks
```

Immediately you can see that the multi-table model will barf on this - we need something much more adaptive.

In the adjacency model you want to store the element's immediate parent element's id. This enables an easy inheritance reference.

Id	Name	Parent	Status
1	Alcoholic Beverages	0	1
2	Beers	1	1
3	Wines	1	1
4	White Wines	3	1
5	Chardonnay	4	1
6	Sauvignon Blanc	4	1
7	Pinot Gris	4	1
8	Riesling	4	1
9	Gewurztraminer	4	0
10	Viognier	4	1
11	Semillon	4	1
12	Red Wines	3	1
13	Champagne	3	1
14	Sparkling	3	1
15	Sweet Dessert	3	1
16	Ports/Sherries	3	1
17	Spirits	1	1
18	Non-Alcoholic Beverages	0	1
19	Snacks	0	1
20	Oaked	5	1
21	Unoaked	5	1

As you can see, one table handles all categories. Now you can simply have products that have a single parent reference, or if this were a CMS then your body content, template reference etc, could live in the row with your category. Even a custom non-alphabetical display order can be handled neatly.

Simplicity is also one of it's strengths: one can see what the immediate parent is at a glance.

So lets look at some of the ways to work with this data. Please bear in mind that there will be other ways to do this, an exhaustive display is outside the scope of this presentation!

Displaying the tree

When Jane Average is browsing your site, if she's not searching for the product she already knows she wants, she often starts at the top and follows a path to the category she's interested in, so we'd like to get the full tree from the database.

Assuming she's selected "Alcoholic Beverages":

```
SELECT c1.name AS cat1, c2.name as cat2, c3.name as cat3, c4.name as cat4
FROM adjacency AS c1
  LEFT JOIN adjacency AS c2 ON c2.parent = c1.id
  LEFT JOIN adjacency AS c3 ON c3.parent = c2.id
  LEFT JOIN adjacency AS c4 ON c4.parent = c3.id
WHERE c1.id = 1;
```

Returns:

Cat1	Cat2	Cat3	Cat4
Alcoholic Beverages	Beers	[Null]	[Null]
Alcoholic Beverages	Wines	White Wines	Chardonnay
Alcoholic Beverages	Wines	White Wines	Sauvignon Blanc
Alcoholic Beverages	Wines	White Wines	Pinot Gris
Alcoholic Beverages	Wines	White Wines	Riesling
Alcoholic Beverages	Wines	White Wines	Gewurztraminer
Alcoholic Beverages	Wines	White Wines	Viognier
Alcoholic Beverages	Wines	White Wines	Semillon
Alcoholic Beverages	Wines	Red Wines	[Null]
Alcoholic Beverages	Wines	Champagne	[Null]
Alcoholic Beverages	Wines	Sparkling	[Null]
Alcoholic Beverages	Wines	Sweet Dessert	[Null]
Alcoholic Beverages	Wines	Ports/Sherries	[Null]
Alcoholic Beverages	Spirits	[Null]	[Null]

Now there's a few obvious problems here:

- it's not returning Wines as a distinct row because it has at least 1 child, hence Beers returning and not Wines.
- If you try to do this query originating at the top of the tree you get some funky unintended results.
- Note that the categories Oaked and Unoaked are missing from Chardonnay, they're judged 5th level and we only requested data down to 4. You have to know in advance, at the code stage, how many levels deep the client's going to want to show Jane Average.

Around about now it's obvious the solution is Lasso. Create a recursive tag that simply calls itself until the whole tree's assembled.

For example:

```
define_type('adjacency',-priority='replace',-namespace='demo_');
    define_tag('drill', -required='id',-optional='level');
      local('out' = string);
      inline(
-database='summitDemo',
-SQL='SELECT * FROM adjacency WHERE parent = '#id';');
        records;
          #out += #level'. '(field('name'))'<br \>';
          #out += demo_adjacency->(drill(
```

```
-id=(field('id')),
-level=#level+1));
       /records;
     /inline;
     return(@#out);
   /define_tag;
 /define_type;
 // action and output the above tag
 demo_adjacency->(drill(-id=0,-level=0));
```

So you could in theory do that from any point in the tree and get all siblings and children. You could also build in a depth limiter which would stop it crawling more than 2 levels deep so that you only returned what was appropriate for a shallow top level menu system.

Difficulties with the Adjacency Model

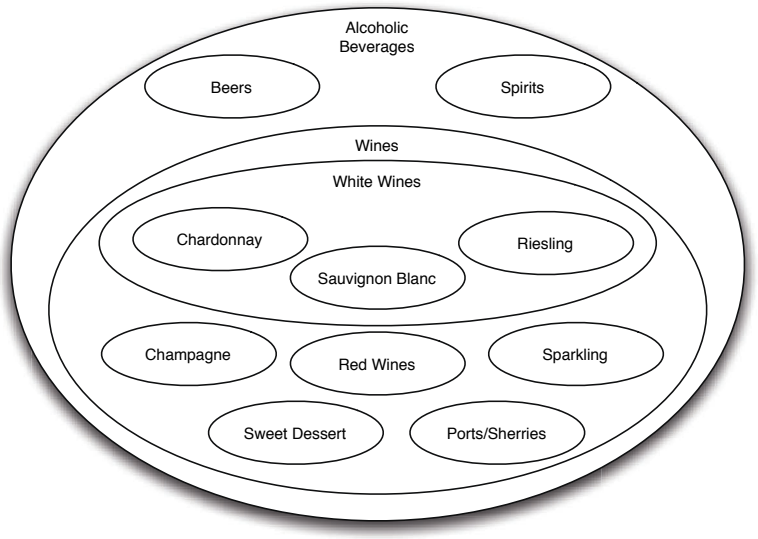
Where the adjacency model falls down is that you have to build in exactly how you want to present it into the recursive presentation tag, unless you want to make it into a nested array but when you think about it if you do that you're no further ahead at all.

Referential integrity is not easily taken care of unless once again you do these checks in lasso. It would be all too easy to accidentally orphan nodes, however if that does happen, it's a piece of cake to fix by a bit of manually editing.

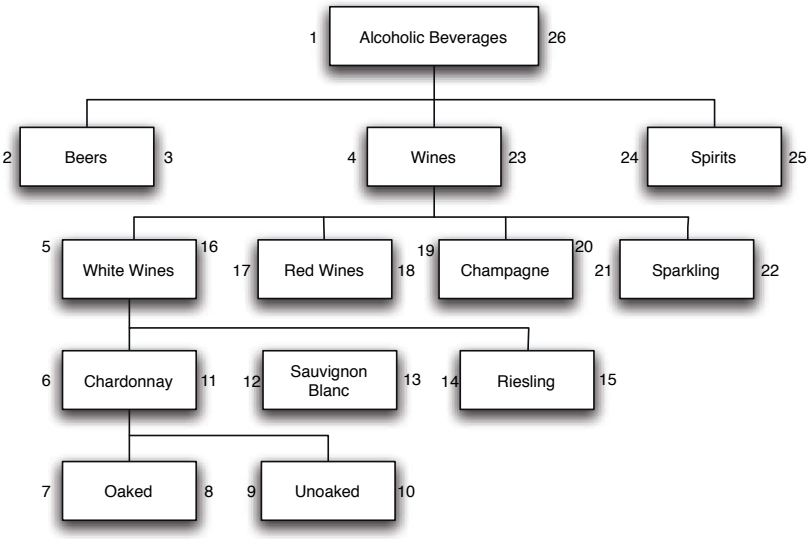
My main objection to this model is that it forces us to use a recursive approach. Not only is it slow, but it will be prone to getting bogged down with extremely large data sets, as I found out with a certain scientific research application.

The Nested Set Model

The previous examples are so simple (easy to understand) because they're either constrained or linear. However rather than conceptualizing data as linear objects, think of hierarchical data as groups, or nested sets. A group that is not fully contained in a parent group might need to be re-classified.



So what we need to do is work out a way to represent these groups, or sets. If we represent this data in a tree diagram, to quote Mike Hillyer in an article on mysql.com, "When working with a tree, we work from left to right, one layer at a time, descending to each node's children before assigning a right-hand number and moving on to the right. This approach is called the modified preorder tree traversal algorithm."



About now I feel I need to offer up a warning: This method requires MySQL 4.1 (or a datasource that utilizes subqueries and variables) and a clear, sober head. Some of what I am presenting here is very close to the first major article I absorbed on the subject and I make no apologies for similarities... however I will be providing LP8-specific examples that are in use IRL. The full article by Mike Hillyer can be found at http://dev.mysql.com/tech-resources/articles/hierarchical-data.html

The database would look something like this:

```
CREATE TABLE `nestedset` (
  `id` int(11) NOT NULL auto_increment,
  `name` varchar(64) collate utf8_bin NOT NULL default '',
  `lft` int(11) NOT NULL default '0',
  `rgt` int(11) NOT NULL default '0',
  `status` int(2) NOT NULL default '1',
  PRIMARY KEY  (`id`))
insert into `nestedset` values('1','Alcoholic Beverages','1','38','1'),
 ('2','Beers','2','3','1'),
 ('3','Wines','4','35','1'),
 ('4','White Wines','5','24','1'),
 ('5','Chardonnay','6','11','1'),
 ('6','Sauvignon Blanc','12','13','1'),
 ('7','Pinot Gris','14','15','1'),
 ('8','Riesling','16','17','1'),
 ('9','Gewurztraminer','18','19','1'),
 ('10','Viognier','20','21','1'),
 ('11','Semillon','22','23','1'),
 ('12','Red Wines','25','26','1'),
 ('13','Champagne','27','28','1'),
 ('14','Sparkling','29','30','1'),
 ('15','Sweet Dessert','31','32','1'),
 ('16','Ports/Sherries','33','34','1'),
 ('17','Spirits','36','37','1'),
```

```
('18','Non-Alcoholic Beverages','39','40','1'),
('19','Snacks','41','42','1'),
('20','Oaked','7','8','1'),
('21','Unoaked','9','10','1');
```

Note ids 20 and 21, they belong inside Chardonnay, and the lft and rgt values of Unoaked and Oaked neatly fit inside Chardonnay's lft and rgt.

Selecting the ordered data, all the data is returned ordered.

```
SELECT node.id, node.name, node.lft, node.rgt
FROM nestedset AS node
WHERE node.status = 1
ORDER BY node.lft;
```

In multi-table and adjacency models the path root through child has to be a known maximum depth. In the Nested Set model it's not required.

```
SELECT parent.id, parent.name
FROM nestedset AS node,nestedset AS parent
WHERE node.lft BETWEEN parent.lft AND parent.rgt AND node.id = 20
ORDER BY node.lft;
1     Alcoholic Beverages
3     Wines
4     White Wines
5     Chardonnay
20      Oaked
```

Remember the adjacency model and our nested cTag that burrowed though the hierarchy, returning a "formatted" string, nested and with depth? We're about to annihilate it with one foul query.

```
SELECT node.id, node.name, (COUNT(parent.name) - 1) AS depth
FROM nestedset AS node, nestedset AS parent
WHERE node.lft BETWEEN parent.lft AND parent.rgt
GROUP BY node.name
ORDER BY node.lft;
```

ID	Name	Depth
1	Alcoholic Beverages	0
2	Beers	1
3	Wines	1
4	White Wines	2
5	Chardonnay	3
20	Oaked	4
21	Unoaked	4
6	Sauvignon Blanc	3
7	Pinot Gris	3
8	Riesling	3
9	Gewurztraminer	3
10	Viognier	3
11	Semillon	3
12	Red Wines	2
13	Champagne	2
14	Sparkling	2
15	Sweet Dessert	2
16	Ports/Sherries	2
17	Spirits	1
18	Non-Alcoholic Beverages	0
19	Snacks	0

Yeah I know this is supposed to be a lasso paper, but imagine what we can do when the hard part's taken care of in SQL and we can concentrate on writing code rather than untying the knots we've tied ourselves in with recursive tags and so on!

Tying this in with Lasso

Adding a new node

It's relatively easy to create new nodes in multi-table and adjacency models, but it's more complex in the nested set model. Fortunately the hard work has been done by those with higher IQ's, and I've adapted some of these to cTags.

So we want to get the right value of the item we're inserting AFTER, and shuffle those rows to the RIGHT up by 2 to make way for the new row, then inserting the new row with appropriate lft and rgt values. Simple once you get used to it!

```
define_tag('addSibling',
    -Required='cattable',
    -Required='txt',
    -Optional='othersmap',
    -Required='id',
    -Description='Adds entry after another child, same level'
      );

  local('xtraFields' = string);
```

```
  local('xtraValues' = string);
  if(local_defined('othersmap') && local('othersmap')->(IsA('Map')));
    iterate(#othersmap,local('temp'));
      #xtraFields += ','+#temp->name;
      #xtraValues += ","'+#temp->value+'"';
    /iterate;
  /if;
  local('sSQL' = '
    LOCK TABLE '+#cattable+' WRITE;

    SELECT @myRight := rgt FROM '+#cattable+' WHERE id = '+#id+';

    UPDATE '+#cattable+' SET rgt = rgt + 2 WHERE rgt > @myRight;
    UPDATE '+#cattable+' SET lft = lft + 2 WHERE lft > @myRight;

    INSERT INTO '+#cattable+'(name, lft, rgt'+#xtraFields+') VALUES("'+(encode_
sql(#txt))+'", @myRight + 1, @myRight + 2'+#xtraValues+');

    UNLOCK TABLES;
  ');
  inline($gv_sql,-SQL=#sSQL);
  /inline;
/define_tag;
USAGE:
xs_cat->(addSibling(
  -cattable='nestedset',-txt=#txt,-othersmap=(map('f1'=#f1)),-id=#id));
```

Points to note:

- The table name is dynamic as we use this in situations where there may be more than one nested set table in a given client solution.
- "Othersmap" is a method of getting the "other" data into the row at creation time. Remember, this is a generic tag that gets used in a number of solutions.
- Note the use of multiple statements in the executed block.
- The LOCK TABLES is used to ensure no other query either gets in the way or gets wrong data.

The tag to add a nested child is almost identical, see the tag "addChild" in the cTags file supplied for this presentation on your Lasso 2006 Summit CD.

However this time we select the LEFT value of the item we're inserting INTO, and again shuffle those rows to the RIGHT up by 2 to make way for the new row, then inserting the new row with appropriate lft and rgt values. The biggest difference is that the shuffling is relative to the LEFT of the identified row rather than the right.

Deleting nodes

Remember "Referential Data Integrity"? We've pretty much got that sorted here too, especially if the row also includes the content data. If you delete the navigation item, you're deleting the content. You can also set it up so that if you're deleting a row that has child nodes, then the child nodes are history too.

```
define_tag('deleteNode',
      -Required='cattable',
      -Required='id'
      );
  local('sSQL' = '
    LOCK TABLE '+#cattable+' WRITE;
```

```
    SELECT @myLeft := lft, @myRight := rgt, @myWidth := rgt - lft + 1
    FROM '+#cattable+'
    WHERE id = '+#id+';

    DELETE FROM '+#cattable+' WHERE lft BETWEEN @myLeft AND @myRight;

    UPDATE '+#cattable+' SET rgt = rgt - @myWidth WHERE rgt > @myRight;
    UPDATE '+#cattable+' SET lft = lft - @myWidth WHERE lft > @myRight;

    UNLOCK TABLES;
  ');
  inline($gv_sql,-SQL=#sSQL);
  /inline;
/define_tag;
USAGE:
xs_cat->(deleteNode(-cattable='category',-id=#id));
```

The actual DELETE is not on an ID but on lft BETWEEN the 2 values. If the node has child nodes, they're zapped too. Dangerous, but effective. I often disallow deletion of nodes if they have child nodes.

Returning Full Hierarchy including depth

This tag returns SQL only, due to the complexities of what we'd want to do with it - it's not appropriate to always do the action in an inline, so this simply returns fully assembled SQL.

```
define_tag('fullCatSQL',
    -Required='cattable',
    -Optional='xtraReturn',
    -Optional='xtraWhere',
    -Optional='depth'
    );
  (!(local_defined('xtraReturn'))) ? local('xtraReturn' = string);
  (!(local_defined('xtraWhere'))) ? local('xtraWhere' = string);
  if(!(local_defined('depth')));
    local('depthComp' = string);
  else(integer(#depth) > 0);
    local('depthComp' = 'HAVING depth <= '+#depth);
  else;
    local('depthComp' = string);
  /if;
  return('
SELECT
  node.id, node.name, (COUNT(parent.name) - 1) AS depth '+#xtraReturn+'
FROM '+#cattable+' AS node,
'+#cattable+' AS parent
WHERE node.lft BETWEEN parent.lft AND parent.rgt '+#xtraWhere+'
GROUP BY node.id
'+#depthComp+'
ORDER BY node.lft');
    /define_tag;
```

This SQL essentially is selecting all those rows in the table that conform to #xtrawhere, which could be as simple as "node.status = 1". The order of node.lft sets us up for the order we've specified when we numbered it all, and the #depthComp helps us restrict the query to a set number of levels below root.

Note the "(COUNT(parent.name) - 1) AS depth". This and the join to the crawling up the tree to count the levels, something I've seen some people doing manually each nested statement (shudder), or caching it which is reasonably sensible if you have no alternative.

The xtraReturn and xtraWhere are used to broaden the usefulness. An example of an xtraReturn is as follows:

```
  (
     SELECT COUNT(*)
    FROM asset, category AS subc
    WHERE asset.category_id = subc.id
    AND subc.lft BETWEEN node.lft AND node.rgt
  )AS chqty,
  (
  SELECT COUNT(asset.name) FROM asset WHERE asset.category_id = node.id
  )AS qty,
  (
  SELECT COUNT(*) - 1
  FROM category AS nnode
  WHERE nnode.lft BETWEEN node.lft AND node.rgt
  )AS nchild
* chqty counts the number of items belonging to that category;
* qty counts number of items belonging to that node only;
* nchild counts the number of child nodes to that category.
```

Once you call this tag you've got all you need to execute an inline and you've got your menu data, category data, or forum list... you get the idea?

Returning just the subtree

OK, selecting a full nested list from the root is useful, but that SQL doesn't give us what we often need: the subtree descending from a given node, with either absolute depth or relative depth.

This tag is slightly more involved. It requires all the normal parameters plus 'relative', which is Boolean true or false.

```
define_tag('subTreeSQL',
      -Required='cattable',
      -Required='id',
      -Optional='depth',
      -Optional='relative',
      -Optional='xtraReturn',
      -Optional='xtraWhere'
      );
  (!(local_defined('xtraReturn'))) ? local('xtraReturn' = string);
  (!(local_defined('xtraWhere'))) ? local('xtraWhere' = string);
  (!(local_defined('relative')) || (local('relative') == false)) ?
    local('relative' = '1') | local('relative' = '(sub_tree.depth + 1)');
  if(!(local_defined('depth')));
    local('depthComp' = string);
  else(integer(#depth) > 0);
    local('depthComp' = 'HAVING depth <= '+#depth);
  else;
    local('depthComp' = string);
  /if;
  //(sub_tree.depth + 1) - makes the depth relative to the one requested
  //HAVING depth <= 1 - limits how many subs it pulls in
  local('out' = 'SELECT node.id, node.name, (COUNT(parent.name) - '+#relative+') AS
depth '+#xtraReturn+'
    FROM '+#cattable+' AS node,
      '+#cattable+' AS parent,
      '+#cattable+' AS sub_parent,
      (
      SELECT node.name, (COUNT(parent.name) - 1) AS depth
      FROM '+#cattable+' AS node,
      '+#cattable+' AS parent
      WHERE node.lft BETWEEN parent.lft AND parent.rgt
      AND node.id = '+#id+'
```

```
      GROUP BY node.name
      ORDER BY node.lft
      )AS sub_tree
    WHERE node.lft BETWEEN parent.lft AND parent.rgt
      AND node.lft BETWEEN sub_parent.lft AND sub_parent.rgt
      AND sub_parent.name = sub_tree.name '+#xtraWhere+'
    GROUP BY node.id
    '+#depthComp+'
    ORDER BY node.lft
    ;');
  return(#out);
/define_tag;
```

The joined 'parent' is the same as the full tree retrieval - it gets the bubble-up, but the 'sub_parent' join and 'sub_tree' subquery join set us up to get the nested levels. It might seem overly complex but consider the alternative: if you try to restrict the "full tree" query to a given id as a parent node, then it won't even return the children, it will only return that node. If we try to restrict it using the lft and rgt values of that node (which is what we actually are doing here anyway), it doesn't accurately retrieve the subtree.

Moving a node

Here's where I regretted ever getting into this, but by now I was hooked. Moving a node was simply not documented properly anywhere online. It was described, but no one could really succinctly show it. The "step by step" instructions proved that some of these guys found it confusing as well. I admit I made copious notes trying to work the theory out on paper, and still got it wrong - it took trial and error in a couple of places before I understood what I was actually doing!

```
define_tag('moveNode',
    -Required='cattable',
    -Required='id'
    );
  local('id2' = 0);
// (A)
  // get immediate prior sibling
      local('sSQL' = (
    'SELECT node.id, node.name,
     (COUNT(parent.name) - (sub_tree.depth + 1)) AS depth,node.lft,node.rgt
    FROM '+#cattable+' AS node,
    '+#cattable+' AS parent,
    '+#cattable+' AS sub_parent,
    (SELECT node.name, (COUNT(parent.name) - 1) AS depth
    FROM '+#cattable+' AS node,
    '+#cattable+' AS parent
    WHERE node.lft BETWEEN parent.lft AND parent.rgt
    AND node.id =
        (SELECT parent.id
          FROM '+#cattable+' AS node,
          '+#cattable+' AS parent
          WHERE node.lft BETWEEN parent.lft AND parent.rgt
          AND node.id = '+#id+' AND parent.id != '+#id+'
          ORDER BY parent.lft DESC LIMIT 1)
    GROUP BY node.name
    ORDER BY node.lft
    )AS sub_tree
    WHERE node.lft BETWEEN parent.lft AND parent.rgt
    AND node.lft BETWEEN sub_parent.lft AND sub_parent.rgt
    AND sub_parent.name = sub_tree.name
    GROUP BY node.id
    HAVING depth = 1
    ORDER BY node.lft ASC;'
```

```
      ));
      inline($gv_sql,-SQL=#sSQL);
        records;
          if(integer(field('id')) != #id);
            #id2 = integer(field('id'));
          else;
            loop_abort;
          /if;
        /records;
      /inline;

      if(#id2 == 0);
// (B)
        // here we are trying to ascertain if it's a root node!
        #sSQL = '
        SELECT node.id, node.name, (COUNT(parent.name) - 1) AS depth
        FROM '+#cattable+' AS node,
        '+#cattable+' AS parent
        WHERE node.lft BETWEEN parent.lft AND parent.rgt
        AND node.id = '+#id+'
        GROUP BY node.id
        ORDER BY node.lft';
        inline($gv_sql,-SQL=#sSQL);
          records;
            if(integer(field('depth')) == 0);
              // yay, it's a root node!!!
              #sSQL = '
                SELECT node.id, node.name,
                 (COUNT(parent.name)-1) AS depth
                FROM '+#cattable+' AS node,
                '+#cattable+' AS parent
                WHERE node.lft BETWEEN
                  parent.lft AND parent.rgt
                GROUP BY node.id
                HAVING depth = 0
                ORDER BY node.lft
              ';
              inline($gv_sql,-SQL=#sSQL);
                records;
                if(integer(field('id')) != #id);
                  #id2 = integer(field('id'));
                else;
                  loop_abort;
                /if;
                /records;
              /inline;
            /if;
          /records;
        /inline;
      /if;
      if(#id2 > 0);
      #sSQL = ('
LOCK TABLE '+#cattable+' WRITE;
-- 1
SELECT @myLeft := lft, @myRight := rgt, @myWidth := rgt - lft + 1 FROM '+#cattable+'
WHERE id = '+#id2+';
-- 2
UPDATE '+#cattable+' SET rgt = (rgt*-1), lft = (lft*-1) WHERE lft BETWEEN @myLeft AND
@myRight;
-- 3
UPDATE '+#cattable+' SET rgt = rgt - @myWidth WHERE rgt > @myRight;
UPDATE '+#cattable+' SET lft = lft - @myWidth WHERE lft > @myRight;
-- 4
SELECT @myLeft2 := lft, @myRight2 := rgt, @myWidth2 := rgt - lft + 1 FROM
'+#cattable+' WHERE id = '+#id+';
-- 5
UPDATE '+#cattable+' SET rgt = rgt + @myWidth WHERE rgt > @myRight2;
UPDATE '+#cattable+' SET lft = lft + @myWidth WHERE lft > @myRight2;
```

```
-- 6
SELECT @x := lft FROM '+#cattable+' WHERE id = '+#id+';
SELECT @y := rgt FROM '+#cattable+' WHERE id = '+#id+';
-- 8
UPDATE '+#cattable+' SET rgt = (rgt - (@y - @x + 1)) WHERE rgt < 0;
UPDATE '+#cattable+' SET lft = (lft - (@y - @x + 1)) WHERE lft < 0;
-- 9
UPDATE '+#cattable+' SET rgt = rgt * -1 WHERE rgt < 0;
UPDATE '+#cattable+' SET lft = lft * -1 WHERE lft < 0;
UNLOCK TABLES;
      ');
      inline($gv_sql,-SQL=#sSQL);
      /inline;
    /if;
  /define_tag;
USAGE:
xs_cat->(moveNode(-cattable='category',-id=#id));
```

The in-SQL comments are kept there for my own reference as to the various logical steps that they represent.

In the placeholder comments, (A) and (B) are in the lasso, --1 and so on are SQL comments.

A few notes about the process:

- This particular tag is meant fro moving a node UPWARDS in the ORDER, not changing level, although in this code can be adapted to move a node and all it's children from one part of the tree to any other part of the tree with a little effort.
- (A) - In order to move a node up the order you need to know what it's immediate previous sibling is, proved to be complex to solve in SQL, so I've captured all child nodes with depth 1 of the parent of the node we're moving (ie, siblings!). In Lasso we then set the #id2 to the column id until we hit the moving node and we abort the loop.
- (B) Special case is if the node we're moving is at depth 0, therefore no parents, so the (A) code blows apart and #id2 will remain at zero.

  1: We retrieve the lft, rgt and width of the node we're exchanging "places" with.

  2: Make the entire node+children we're exchanging with, negative, as a placeholder.

  3: Collapse the rest of the tree to assume the removed node's place. Note the similarity with the delete node code.

  4: Retrieve lft, rgt, width from the node we desire to move "up"

  5: Make space for the exchanged node (that which we negative-valued in step 2) by shifting everything to the right of our moving node outwards.

  6: Reselecting the lft and rgt of the moving node, partly out of consistency with other tags that do similar things.

  7: yes you are right, there's no 7 out of respect to those very same similar tags

  8 & 9: We take every lft and rgt value that is less than zero and make them positive with the newly calculated lft and rgt that fit into the gap we created. I had some issues with doing this as one step in the SQL so accepted a minute performance hit with doing this as 2 sets of statements.

It looks hideous, but apart from ascertaining the immediate prior sibling's id, it's pure SQL.

Summary

I've played with a lot of data models over the years, and have looked at (and worked with) the way some large content management systems - you know the type, that cost hundreds of thousands to license and implement, manage their categorized data. I believe we can learn a lot from they way these guys have missed the mark in certain areas.

Where we need to be prepared to jump through hoops is on the back end - we can afford a speed hit there, not on the front-end. Surprisingly, I found the "big solution providers" were constantly seeming to convince their clients it was easy to maintain, and yet in reality the front-end code was often clumsy and slow.

The easier we can categorize our data and make it cohesive, the better we're setting ourselves and our clients up to win. When the client wins, they will recommend us to others and so we win, as we will have work walking in our doors.

I believe the more we talk to each other about how we're dealing with our data, hierarchical content being just one piece in the spectrum, the more we will be able to move forward as a community.

I hope this exposé of the way I've begun dealing with this area will open doors for many of you and promote active discussion. One thing's for sure, we live in a constantly evolving environment - by the time my son is earning his keep by coding, these techniques will probably be well outdated. Collaborative development and discussion will keep us all ahead of the curve.

References

Managing Hierarchical Data in MySQL by Mike Hillyer http://dev.mysql.com/tech-resources/articles/hierarchical-data.html

MySQL 4.1 User Variables, http://dev.mysql.com/doc/refman/4.1/en/variables.html

More links at http://xserve.co.nz/hd_links.lasso

Author: Jonathan Guthrie

xServe Limited, Wellington NZ. jono@xserve.co.nz

For Lasso Summit, Ft Lauderdale, Miami, February 2006

Structured Dynamic Content Roundtable

Peter D Bethke

Overview

It is an unavoidable truth of web application development that the more complex a site is, the more difficult it becomes to implement site-wide changes. In the "early" days of web application development, when sites were primarily static html documents, individual pages were coded one at a time, and often pages were added by cloning and altering a previous page. While this was effective in creating a site with only a few pages that seldom changed, it became increasingly inefficient in handling sites where information had to shared between pages, and more importantly changed and updated consistently between pages. Many an html coder has (hopefully in the past) experienced the tedium of adding the same information over and over again to multiple web pages, or relying on "search and replace" operations while praying that the search pattern has remained constant throughout all the pages.

With the advent of "dynamic" web applications, developers began to overcome these issues by taking advantage of two fundemental tags that are common to Lasso and virtually all other scripting languages - variables and included files (most often called "includes"). An included file in LDML is called, using the [include] tag, by another file, and the contents of that file is then essentially "folded" into the body of the first file at the point in the script (proceeding in a linear fashion from top to bottom) where the [include] tag appears. Most importantly, if there are variables on the "included" file that are global in scope (in other words cabable of being recongnized as proper variables in the body of the "calling" page), they too are "folded" into the body of the calling page, and can be used on that page for a number of purposes. The simplest of these is of course declaring and calling simple string variables. For example, if the developer had a file, "A", which included the variable "foo" with value "bar", and this file was called via the [include] tag into the body of another file, "B", the developer could call (at any point below the "point of insertion", ie the place where the [include] tag was declared) the "foo" variable and get the value "bar". At any point after that, changing the value of "foo" in file "A" would result in the value instantly changing in file "B".

These very simple concepts lead to the use of included files to provide, for example, "header" and "footer" files for web sites. The "header" file might contain a common logo, and the "footer" might contain contact information. If these files were called as dynamic includes by any number of pages, essentially "shared" by the whole site, it greatly reduced the developer's effort when it came time to change any of the information contained therein. Building further on this concept, it became advantageus to create a "site configuration" file, or "siteconfig" as it is sometimes called, in which common variables were placed. This siteconfig file was then included by all pages on the site, and these pages could call any of the global variables declared in the siteconfig at any time, and similarly those variables could be changed and their values would update instantly across the site.

One important function of the [include] tag is that it allows "nesting" of files. Included files are not limited to one "level" of inclusion -- much like a Russian Doll that has many dolls contained within, multiple files may be "chained" or "nested" together to produce complex structures. Even

more important to the creation of dynamic systems, the global variables contained in included files become available to files that are part of the nested structure (though always in a "top-down" fashion, as discussed earlier).

The combination of the "nesting" function of included files and the fact that the file path specified by the [include] tag can itself be a variable, enabled the creation of Structural Methodologies such as the Corral Method. Corral had the distinction of being the first Lasso-based structural method proposed, but it was by no means perfect. Rather, it served the purpose of "opening a dialogue" on the roll of structural methods at a time when a number of Lasso developers were pursuing similar ideas. It was eclipsed soon after by systems like [Framework:] (now called "Pageblocks") and other much more robust systems ported from other languages (like Fusefox). A quick distinction should also be made - Corral was a Structural Method, ie a set of ideas that produced a result, and open to broad interpretation and reinvention. This stands in contrast to a Structural System, sometimes called an API, which is much more devoted to defining a specific system and set of guidelines to follow. Pageblocks is a terrific example of a very mature Structural System, as is Fusebox. Much more effort has gone into these systems to provide developers with specific tools to enable rapid application development.

In the end, however, neither Corral or Pageblocks or Fusebox or any other of the multitude of Structural Methods and Systems could function without the [include] or [variable] tag (or their equivalents in other languages) and the way that they nest together to enable complex and dynamic systems.

ExecuChoice Roundtable Discussion

Steffan Cline

This discussion is meant to be more of a question and answer session to discuss how ExecuChoice tags can be used to assist developers needs of functionality that is not provided by Lasso out of the box. A brief overview of some of the features in the planned release of the ExecuChoice Lasso Summit 2006 Master Installer is listed below to provide some ideas of topics that may be discussed.

[PassThru]

PassThru is probably the most common and widely used third party LCAPI tag which allows a lasso user shell access on any one of the three Lasso operating platforms available. This tag has been used for so many features that it is hard to list them all. Recent usages have been for using Lasso to access CVS tools from the command line to manage files etc. Additionally are for importing large files via FTP or some other means and then using a utility like stuff/unstuffy/zip to open/compress files for various purposes.

[ShortString]

ShortString is a simple little LCAPI tag which was built to simplify the display of long blocks of code in small areas. A demonstration is planned to demonstrate just how useful this tag will be for everyone to use. Currently this is available for all three Lasso platforms.

[PDF_xxx]

The PDF suite is the next most widely used set as it provides blazing fast shortcuts to manipulating PDF files. Demonstrations are planned to show the difference in speed of drawing a PDF manually using the built in Lasso tags versus using the [PDF_fromHTML] or the [PDF_SetFieldValue] tags. In addition will be to show how a Lasso server is capable of faxing and printing directly from the server itself.

[XLS_XXX]

Demonstrations of this tag set will show the ease of and many methods of importing and manipulating the data within the Lasso environment. The tags have been built so that the data can be returned in several common data types so that the data can be inserted into a table easily or simply displayed. Software is being developed at this time to actually write out the excel files as well.

[DOC_xxx]

Discussion of this tag set will show how easy it is to extract data from this popular word processing format to be archived in a table or saved to files. Examples will be given as to how these tags can be used to output the data in the most popular formats as well.

HostedStore Roundtable Discussion

Brian K. Middendorf

This discussion is meant to be more of a question and answer session to discuss how HostedStore can be used to meet the ecommerce related needs of your current and/or future clients. A brief overview of some of the features in HostedStore is listed below to provide some ideas of topics that may be discussed.

Extensibility

HostedStore was built to provide a base application that could be used as is or customized to meet specific requirements. The store template and all email/html content files can be modified. A system of hooks allows for insertion of custom code before core code is executed, while code core is executed, or after core code is executed. In many cases, core code can be disabled and replaced with custom code. For example, the product search script can be completely replaced with a different method written be the developer.

Storefront

The storefront contains features that have become standard in ecommerce solutions: static shopping cart, tax and shipping estimates in the cart, address books, wish lists, shopping lists (prior orders that can be reordered), saved carts, sale pricing, member pricing, volume pricing, price groups, coupons/promotions, affiliate management, product reviews, etc. Most of these features can be enabled or disabled with a single parameter on the store configuration screen, thereby making it easy to add features as the needs of the store and its customers grows.

Product Management

Honestly, there are just too many options to list. Multiple pricing levels options including member, sale, volume, and price groups. Multiple attributes can be assigned, each of which can adjust the overall product pricing as well as the inventory of other products. Inventory thresholds can be set remove a product from publication, publish a product but not sell it, and even track customers who would like to be notified when a product is back in stock. Purchase limits can be set for a product to prevent hoarding. Inventory enforcement can be applied at the last moment before checkout in case more than one customer has the last available item in their cart. Product dependencies/restrictions can be set so that a product may only be purchased if another products is purchased or another product from a product group is purchased.

Order Management

Orders may be assigned multiple statuses. Even down to the item level. For example, this allows for back ordering or canceling specific items while shipping those items in stock. Items with multiple quantities can even be separated to assign different statuses.

Affiliate Management

The affiliate system allows for payout of a fixed amount or a percentage for first time orders, subsequent orders, first time autoship orders, and/or subsequent autoship orders. Affiliates have

the capability to login into an area of the store and view all orders for which they have been paid or are awaiting payment.

Coupon/Promotion Management

This versatile system allows for applying discount to an order subtotal, shipping, or a specific item. Attributes can be set that require specific dollar amounts, item counts, a specific product, a specific customer, etc to be associated with an order before the coupon may be applied. This allows for a powerful system which could be automated by the end developer to automate marketing promotions by auto-sending coupons to new customers, customer who purchase specific items, customers who have not ordered for awhile, etc.

Lasso Professional Server 8.5

the fastest, easiest way to tie any database to any website on any platform.

Lasso Professional – the next generation of building data-driven Web sites – is simply the fastest, easiest way to tie **your** databases to the Web. It supports multiple databases simultaneously, eases the sequential migration of databases and it's fully supported on Mac, Windows®and Linux.

Advanced features for advanced developers

The Lasso Professional 8.5 upgrade delivers new features including:

- Native data source connectors for PostgreSQL, Oracle, OpenBase, ODBC, and more.
- An LJAX framework which makes creating state of the art dynamic Web sites easier.
- A free developer mode!

Special Offer

Introducing

Free Lasso Developer 8.5

- Five IP connections
- Fully Functional

www.omnipilot.com

lassosales@omnipilot.com

800-678-9958

OMNI PILOT® SOFTWARE

Lasso