[Var_Set: 'Lasso Summit'='August 10 & 11, 2001']

Lasso Summit 2001

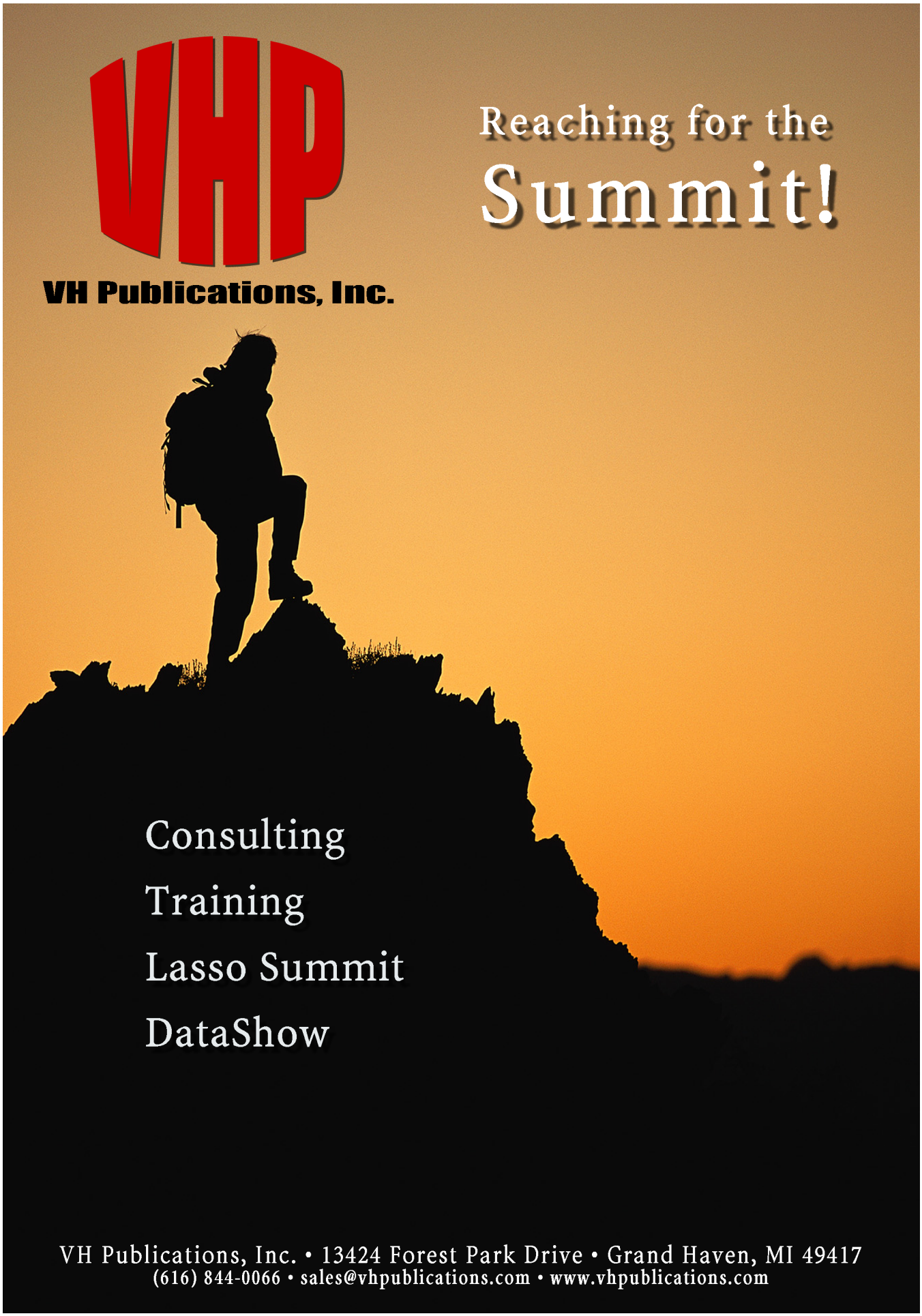
Presented By

VH Publications, Inc.

Database Building & Training

August 10-11, 2001
Disney's Boardwalk Inn
2101 Epcot Resorts Boulevard
Lake Buena Vista, Florida, USA

Conference Manual


VHP
VH Publications, Inc.
Reaching for the
Summit!
Consulting
Training
Lasso Summit
DataShow
VH Publications, Inc. • 13424 Forest Park Drive • Grand Haven, MI 49417
(616) 844-0066 • sales@vhpublications.com • www.vhpublications.com

Lasso Summit

2001 Developers' Conference

Conference Task Force

Jim Van Heule
VH Publications, Inc.

Kris Knox
VH Publications, Inc.

Graham Van Heule
VH Publications, Inc.

Kirk Bowman
MightyData, L.L.C.

Liesl Bowman
MightyData, L.L.C.

Keynote Speaker

Bill Doerrfeld
Blue World Communications, Inc.

Conference Speakers

Peter Bethke
Lassosmart.com

Doug Burchard
Metro Technologies, L.L.C.

Johan Sölve
Montania Solutions

Kirk Bowman
MightyData L.L.C.

Michael Collins
Kuwago Web Services

Jim Van Heule
VH Publications, Inc.

© Copyright 2001 by VH Publications, Inc. and MightyData, L.L.C.
All rights reserved.

Printed in the U.S.A.
$169.00 (US)

Lasso Summit

2001 Developers' Conference

Table of Contents

Lasso Summit

2001 Developers' Conference

Sponsor Center

Lasso Summit 2001 is presented in cooperation with the following sponsors.

Blue World Communications, Inc.

10900 NE 8th Street, Suite 1525
Bellevue, WA 98004
phone: 425.646.0288
fax: 425.646.0236
url: http://www.blueworld.com
email: blueworld@blueworld.com

Blue World's mission is to provide the best tools to build and serve data-driven Web sites. Blue World provides Lasso Web Data Engine, Lasso Studio and Blue World ListSearch service in fulfillment of its long-term mission to bring business to the Internet.

VH Publications, Inc.

13424 Forest Park Drive
Grand Haven, MI 49417 U.S.A.
phone: 616.844.0066
fax: 616.844.6373
url: http://www.vhpublications.com
email: sales@vhpublications.com

VH Publications, Inc. provides first-class project management and web site development services for clients throughout the world. Our complete package of web development includes project management, web design and dynamic database connectivity using the latest Lasso technologies.

Lasso Summit

2001 Developers' Conference

Sponsor Center

Lasso Summit 2001 is presented in cooperation with the following sponsors.

MightyData, L.L.C.

2025 Vista Crest Drive
Carrollton, TX 75007 U.S.A.
toll free: 800.287.0845
phone: 972.492.7523
fax: 972.394.9945
url: http://www.mightydata.com
general email: info@mightydata.com
training email: training@mightydata.com

MightyData offers information technology design, development and training services to a broad client base including Fortune 1000 companies, education institutions, small business, and non-profit groups. MightyData specializes in leading technologies including FileMaker Pro, Lasso, SQL database integration, AppleScript and Palm OS.

Point In Space

2026 East 8th Street
Davis, CA 95616 U.S.A.
phone: 530.759.9105
url: http://www.pointinspace.com
email: info@pointinspace.com

Point In Space offers complete Web, FileMaker Database, Lasso, Ch-Ching, E-Mail, FTP and Listserv internet hosting solutions on powerful Macintosh G3 and G4 servers. Come see what sets us apart from other hosting providers - including our custom Database Manager software to allow you to remotely update your databases, in-depth logging reports for your site, as well as many hosting packages to serve your exact business needs.

Lasso Summit

2001 Developers' Conference

Sponsor Center

Lasso Summit 2001 is presented in cooperation with the following sponsors.

iSolutions, Inc.

1924 East McFadden
Santa Ana, CA 92705 U.S.A.
phone: 888.212.4620
url: http://isolutions-inc.com
email: pj@isolutions-inc.com

iSolutions, Inc. is an information technology consulting company. It is our belief that a company that blends information with knowledge based employees will dramatically improve its ability to both develop and act upon strategic business opportunities. iSolutions helps to provide this management of information through the development and deployment of FileMaker Pro and Lasso solutions.

Starmark International, Inc.

701 South Federal Highway
Fort Lauderdale, FL 33316 U.S.A.
toll free: 888.280.9630
phone: 954.761.1600
fax: 954.761.1615
url: http://www.starmarkintl.com/lasso/
email: lasso@starmarkintl.com

Starmark, developers of Web Site Publisher, Rotating Ads and Bulletin Boards for Lasso, has been developing commercial solutions for Lasso since 1997. All of our solutions have powerful online administration tools and are tightly integrated with each other to comprise an entire suite of solutions for the Lasso community.

Feature Presentations

Structural Design for LDML

Table of Contents

Structural Design for LDML

by Peter Bethke

Peter will discuss best practices for creating well-structured LDML-based web sites. He will explain the Corral Methodology and how it relates to Lasso 5. Alternative approaches will also presented.

The goal of the Corral Method is to create a recognized and fully endorsed open-source structural standard. Peter's presentation will put the Corral Method through a "trial by fire" and move towards acceptance of the Corral Method among the broadest range of developers.

Biography

Peter D. Bethke has been involved in Internet, Multimedia and Video production for over ten years. He is the President of Lassosmart.com, a company specializing in Lasso-based web development and tools for Lasso-based web hosting companies. Before becoming involved in Lasso, Mr. Bethke spent several years as a Producer and Developer of CD-ROM based multimedia. He has held senior-level positions at several prominent Internet and Multimedia companies, and has extensive experience in project management and product development. Mr. Bethke has a Masters Degree in Screenwriting.

Notes

Structural Design for LDML

Introduction

As Lasso Developers move into a new era of power and complexity with the release of LP5, it becomes more evident that a method for organizing dynamic sites built with LDML needs to be adopted and supported. With the addition of many "mature" Internet technologies, such as sessions, enhanced error trapping, encryption, customized tags, and robust user management, the possibility that a new Developer could become quickly overwhelmed is very real.

As a Developer community, we have a responsibility to help fellow Developers along this learning path. Although we may compete at times for the same contract pool, one of the things that makes Lasso Developers stand out is their willingness to guide fellows along paths that they themselves once took, and to provide the benefits of hard-earned techniques and technical wisdom.

Similarly, although the documentation provided with the Lasso Web Data Engine shows Developers how to use the language to accomplish certain tasks, it does not (nor should it) tell Developers how to organize and implement that code most effectively on a site-wide basis. This task should fall to Developers to work out themselves. To use a construction metaphor, Blueworld provides the tools for building the house; it's up to the architects and carpenters to actually build it.

There are many approaches Developers can take to structuring their LDML-powered sites, from the most rudimentary to the highly complex. In general, though, most approaches should do at least two things: Reduce the amount of time a Developer requires to implement new features and layouts, and increase the flexibility of existing code in response to global design changes.

One methodology, referred to as the Corral Methodology, has recently emerged that has generated a lot of excitement among Developers. Its popularity comes not because its ideas are necessarily new or radical; rather, the remarkable thing is how the methods have already been in use by individual Developers for several years. It's only through Developer initiative that all these methods can be brought together for the good of the community.

Whether a Developer adopts a methodology is the Developer's prerogative. While the community that has arisen around the Corral Methodology is seldom in 100% agreement, the discussions that have come from the topic have been beneficial to most if not all participating. This further strengthens what can be considered the central point of this paper, and of Corral; only through discussion and application of a consistent structural method, whatever it may be, will a Developer be able to manage large and complex web applications with a minimum of repetition and a maximum of flexibility.

Assumptions

The Author makes assumptions that those reading the document are familiar with certain basic concepts in Lasso, such as Includes, Variables, Inline actions, form parameters, and Tokens. Familiarity with two of these topics, Includes and Variables, is crucial to understanding the methods presented here.

Acknowledgements

The Author would like to thank all the fellow Developers who have contributed to the development of Corral, and in particular to those active on Corral Talk list. Special thanks to Jim Van Heule and Kirk Bowman for supporting Corral's presence at the Lasso Summit. Also, special thanks are also due to Duncan Cameron and Greg Willits for their individual contributions to the "movement", and to John May for almost 2 years of ongoing and fruitful collaboration. Also, of course, to Bill Doerrfeld and Blueworld for creating and maintaining this amazing product.

And finally, of course, to my wife, who persists in being amazed how I can code in such an organized manner and still never know where I last put my keys.

Purpose of this Paper

This paper is designed to be an overview of different methods of structuring websites using LDML, with an emphasis on the Corral Methodology. It is the hope of the author that this discussion will benefit Lasso Developers of all experience levels who have at least a basic understanding of Lasso coding techniques (see "Assumptions", above).

What is a Structural Methodology?

Different Developers might have different answers for this question. I liken it to the construction (home building) process. Over time, accepted and tested methods for building houses have gained acceptance among carpenters. Out of these common methods there has arisen a shared "lexicon" of terms and techniques. This shared knowledge enables different builders on a project to collaborate and communicate more efficiently.

It should be no different for web programmers. Simply because Developers compete against each other in the commercial realm does not mean that they should keep all their cards close to the vest. It is true, and very common, that individual Developers might have their own specific coding techniques. But it is only in the sharing that these techniques can become a

true Structural Methodology, and approach in function the "common sense" status that certain house building methods enjoy.

Building a web application is similar in many ways to building a house. Key to success is to have a "blueprint" to work from. Success, as they say, is largely due to good preparation, and having a set of "generic" base-level blueprints can be an immense time-saver in both houses and web applications.

So a true Structural Methodology, in the Author's opinion, is a tested and generally accepted method for creating web applications that is a product of the collective experience of the Developer community. A good Structural Methodology should also be well documented and supported and there should exist "blueprints" for creating sites based on this methodology. The true litmus test of a Structural Methodology is the extent to which it is shared. If not, it's just simply another coding technique.

Why use a Structural Methodology?

Coding for the web, like any programming, can be a messy business. This is particularly true with large web applications that may contain many subsections and re-used elements such as graphics, common tables, headers and footers. It is easy to lose track in the development process of all the appearances of a specific block of code. Even in the era of global search and replace, minor changes in a block of code can render it invisible to a pattern-based search routine, and lead to headaches later in the development or Q&A process.

If frequently used code can be consolidated in specific locations and re-used only when needed, it makes the Developer's job much easier when faced with global site changes. Obviously, this is an advantage to both Developer and Client. It saves the Developer valuable programming time and allows the Client to submit cosmetic and programmatic change requests that otherwise might eat up a lot of development resources.

Similarly, global changes can be made to a site "on the fly" through the use of global variables or subroutines. This flexibility directly enhances a site's *portability*, the ability of one site to be subject to base-level changes (such as operating system changes or hosting changes) that might otherwise require large scale recoding. It also helps Developers use previously existing sites as the basis for new ones, or to create freeware or commercial products that can be easily adapted to new layouts or requirements. In addition, if a Structural Methodology is based on fundamental API features, such as includes and variables, it can carry forward through major software upgrades such as the move from Lasso 3.x to LP5.

Another advantage of using a Structural Methodology is that in many cases layout-based, or "display" code, can be segregated from "functional" code such as database queries and subroutines that produce no visible output. If done correctly, this division can allow different

Developers to work on areas of expertise without impacting other Developers. For example, Graphic Designers using a WYSIWYG editor such as Adobe Golive may make graphic changes to a site template without disrupting "pure" Lasso code imbedded in the page or elsewhere in the application hierarchy.

Enabling Developers to collaborate efficiently on projects is another major advantage to adopting a Structural Methodology. Like carpenters and architects who benefit from a shared set of common blueprints and terms, Developers who share knowledge of a specific set of methods can communicate all the more efficiently since they all speak the same "language". Solutions developed in one location can be plugged into another location without disrupting the overall project.

Finally, apart from programmatic advantages, a Structural Methodology with broad Developer support can draw on the same kind of mass "gestalt", or shared conceptual vision, that the Open Source (ie Unix/Linux) movement does. In this, the end product becomes larger than the sum of its parts and serves as an invaluable tool for those just beginning to use the language. Lasso already benefits from an amazingly supportive Developer community, as expressed through the Lasso Talk list. Creating and nurturing community-benefiting, Open-Source tools like Corral is a natural offshoot of this same "vibe".

Common Structural Techniques

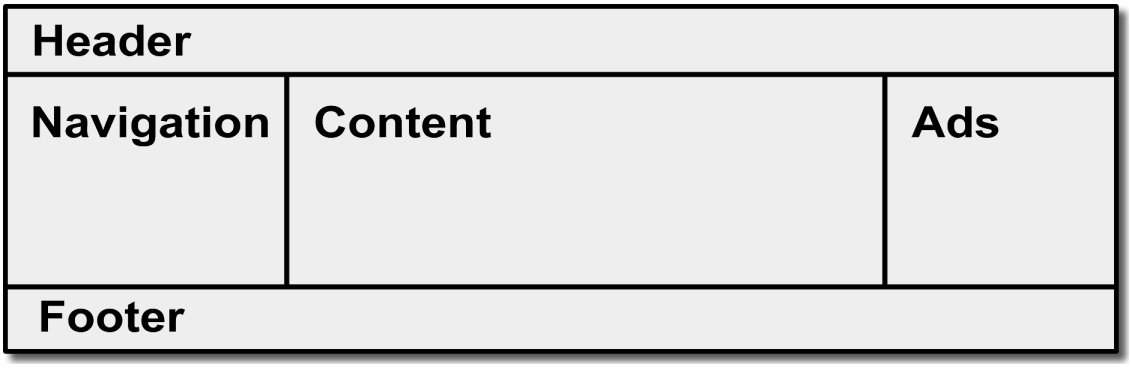
HTML-Based Static Templates

The most basic structural technique is simply the division of static html-based web sites into distinct directories based on some criteria, such as function. For example, in the case of the fictional company, ABC Widgets, Inc., the Developer chooses to break the site up into three directories based on the 3 primary areas of the company - *sales, support, and R&D*.

```
/index.html
/sales/sales.html
/support/support.html
/rd/rd.html
```

Dividing the site in this manner helps the Developer organize the information logically.

The web site at its outset is rather simple – each page is comprised of a single html table divided into specific sections:



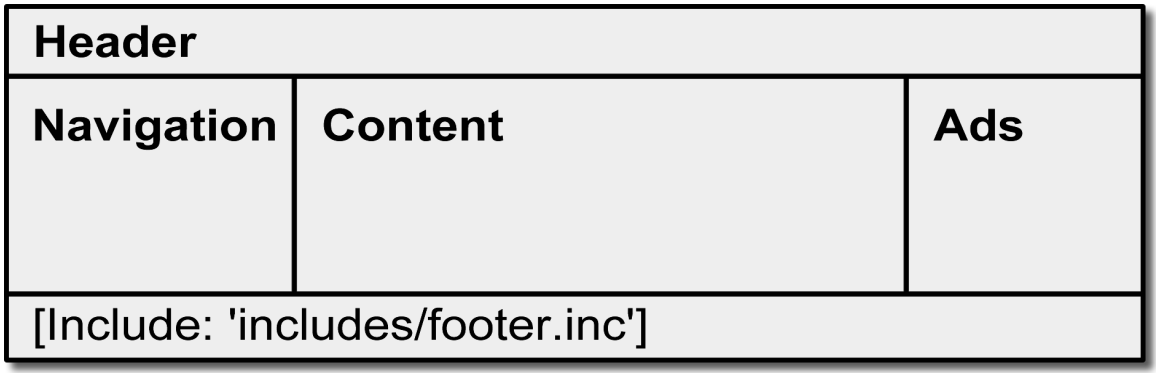
1. A top section, also commonly called a header, which contains the company logo.
2. A middle section, which contains the main page content.
3. A left column, for navigation links.
4. A right column, for ad banners.
5. A "Footer" section, for copyright information.

After the Developer creates the first page, she realizes that it will be easier to create subsequent pages by first creating a blank page with only the table defined, and no content, and re-using it. The concept of creating and re-using an empty table layout, and then filling it with data as needed, is one of the key concepts of a Structural Methodology. Also called a template, this "blank slate" page provides Developer with her first measure of reproducible, consistent code.

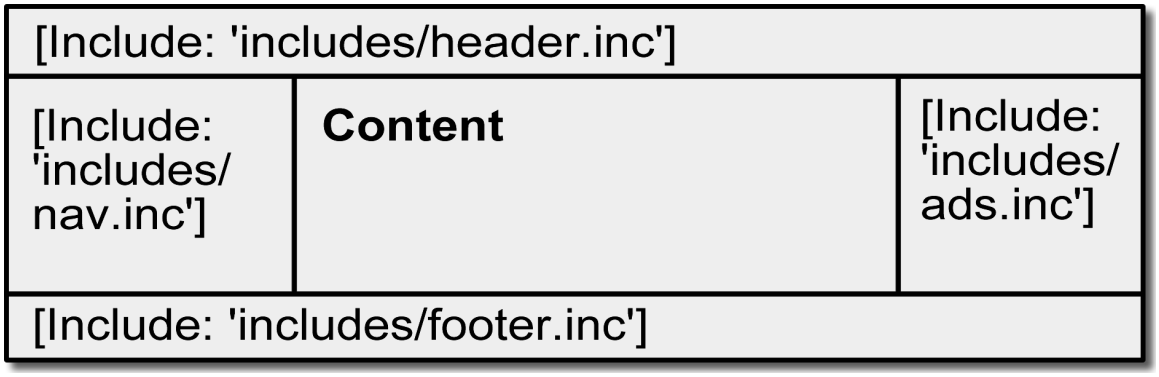
Later that year, after the site has expanded to over twenty pages, the front office tells the Developer that the main phone number has changed. After she cuts and pastes the code into the last page, she vows that there must be an easier way. Fortunately, at this very same time the company wisely invests in a copy of Lasso….

Simple Includes

After reading up on Lasso, our Developer decides to use Lasso's functionality to enhance the site. In order to have the Lasso plug-in parse every page, she changes all her files to end in ".lasso", instead of ".html". She then taps the [include] tag to solve her first problem: applying global changes to the site. In each of the pages of the site, she replaces the footer code with an include statement:



She creates a new directory, /includes, at the root level of her site, and creates a new file there, “footer.inc”. In this file she places the copyright information and the company phone. Then, the next month, when the front office changes the company PO box, she’s ready. One change to the footer.inc file, and all the pages are updated instantly. To reward her amazing productivity boost, the upper management introduces a whole new set of site changes. Undaunted, our Developer creates a whole set of includes to be shared by every page on the site…



The one thing she can not figure out is how to handle the Content section of her pages, since all the other files are global in nature, and the content section of each page changes from page to page. She shelves this thought for a while and move on…

Variables, Scoping, and the Site Configuration File

The more our Developer starts to think about how [include] tags can make her life a whole lot easier, the more she becomes determined to use them in a more flexible manner. About that same time, she begins to dig into variables and how they function in Lasso in respect to [Includes].

The most important concept she learns is how variables are available to both the page that declares it and all external code below it that is “called” into the page using the [Include] tag.

This concept, referred to in traditional programming languages as *scoping of variables*, is at the very heart of advanced Structural Methodologies such as Corral.

It can be said that variables in Lasso are *scoped*, ie "available", to all included and imbedded code from the declaration point on "downwards".

For example, in the following example, our Developer creates two files, "samplefile.lasso", and "sampleinclude.inc". The first file, "samplefile.lasso", has the following code:

```
[var_set: 'mytext' = 'hello world']
[include: '/includes/sampleinclude.inc']
```

the second file, "sampleinclude.inc", has this code:

```
[var: 'mytext']
```

When "samplefile.lasso" is viewed (parsed), the output of the two "combined" files is, of course, "hello world". This is an example of how a variable declared in one page is transferred to an included file called within its body.

This process of scoping does not stop at a single level deep – it will be available to as many "layers" of includes as desired. For example, the "sampleinclude.inc" file could contain the following code:

```
[var: 'mytext']
[include: '/includes/sampleinclude2.inc']
```

And if the file "sampleinclude2.inc" contained this code:

```
<p>
[var: 'mytext']
```

The parsed output of the "master" encapsulating file (ie the first file in the sequence), "samplefile.lasso", would be:

```
hello world
hello world
```

Finally, it is important to understand that at any point in the "include" chain a variable can be re-declared, or in a sense "overridden".

In the above example, if the file "sampleinclude.inc" had this code:

```
[var: 'mytext']
[var_set: 'mytext' = 'hello blue world'] fl- This is where the variable is overridden
```

```
[include: '/includes/sampleinclude2.inc']
```

The final output would be:

```
hello world
hello blue world
```

One of the easiest ways of imagining this process is to envision water flowing downhill. In a sense, variables "flow" downwards through consecutive includes. At any point, they can be re-declared or altered using any number of techniques (such as text manipulation tags or substitution tags).

Our Developer decides to create a single file that will be loaded before any other data, which can be used to declare variables that will be used throughout the site. Although she has yet to discover Corral, by a remarkable coincidence she names the file "siteconfig.inc", and places it in the /includes directory. Then at the top of every page of the site, she puts a single include statement:

```
[include: '/includes/siteconfig.inc']
```

In the siteconfig.inc file she declares a variable:

```
[var_set: 'site_title' = 'ABC Widget Company, Inc.']
```

Finally, on each of the pages that call the siteconfig.inc file, she edits the <head> tag to insert the variable declared in the siteconfig.inc file.

```
<head>
<title>[var: 'site_title', encodenone]</title>
</head>
```

The result of this is that each of the pages now inserts the variable declared in the siteconfig.inc file into the <title> tag dynamically. Any change made to the siteconfig file is reflected in all the pages that share the siteconfig file and call it through a [Include] tag.

Now, our Developer begins to pile all sorts of variables into the siteconfig file, such as variables for background color, text color, etc. But she soon grows tired of inserting all the [variable] tags into each of her individual pages. Also, she's frustrated by the fact that she can't differentiate the variables based on location. Every page has the same title, or she has to manually edit each <head> tag to insert the page name after the site_title variable.

It's about this time that she joins the Lasso Talk list and first hears about the Corral Methodology, and she realizes that it represents a combination of the techniques that she has been developing on her own. She decides to dive right in…

The Corral Methodology – An Overview

The Corral Methodology uses a series of cascading includes, templates, and variables to construct dynamic web applications using LDML. The official sources for what constitute Corral are the Corral Methodology web site, located at http://www.corralmethod.org, the official "white paper", located at same, and the Corral Talk mailing list, where the latest issues involving Corral are discussed by some of the most experienced Lasso Developers in the field. Since the Corral Methodology is by definition a community initiative, any and all are invited to participate in its development.

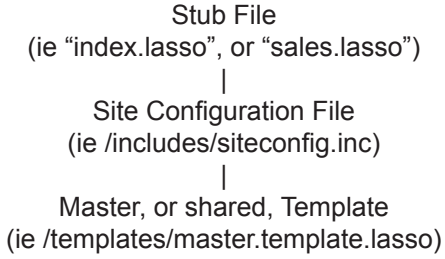
The roots of the Corral Methodology movement started with a white paper published by the Author, Peter D Bethke, in March 2001, and grew out of his extensive experience in creating Lasso-driven web sites. Although he was the first Developer to put the methods down in a formalized structure, and coined the term *Corral Methodology*, Mr. Bethke was pleased to find that many fellow Developers already used many of the techniques for their own creations. And so it grew from there…

The purpose of this paper is not to teach all there is to know about Corral; this is the job of the three mentioned sources, in particular the white paper. Rather, presented here is a general overview of how Corral works, and more importantly how Corral represents an evolution of the methods discussed before.

A web site built using the Corral Methodology usually has the following common elements (specific terms will be described following this list):

1. A site configuration file
2. One or more master, or shared, templates.
3. One or more page, or display, templates.
4. A "includes" directory, with one or more shared elements, such as headers, footers, and navigation elements.
5. One or more "Stub" files, each representing a physical "page".

Described visually, a typical Corral-based page looks like this:



|
Page Template
(ie /pages/index.page.lasso)

The vertical lines here represent the "flow" of information, in the form of variables, as it passes through the hierarchy defined by a series of [include] tags.

The stub file is a small file whose job it is to "start the ball rolling". It declares a series of variables that are used by the files that are included down the "chain". This is similar to the technique described earlier using a single site configuration file, but this method takes it one step further. It allows the Developer to define specific information relevant to individual pages (even though the "stub" file may be physically small, it can be considered the actual "page" since its parsed ["rendered"] output is a whole html-formatted page), instead of having to define only that information that is relevant to the entire site.

The typical "stub file" might contain the following code:

```
[include: '/includes/siteconfig.inc']
[Lasso_comment]
[var_set: 'page_title' = 'Home Page for My Site']
[var_set: 'page_template' = '/pages/index.page.lasso']
[var_set: 'header' = '/includes/header/mainheader.inc']
[var_set: 'footer' = '/includes/footer/mainfooter.inc']
[var_set: 'navbar' = '/includes/nav/mainnav.inc']
[var_set: 'adbar' = '/includes/ads/ads.inc']
[/Lasso_comment]
[include: '/templates/master.template.lasso']
```

In this example you see the stub file literally "building" the elements of the page through a series of [var_set] tags.

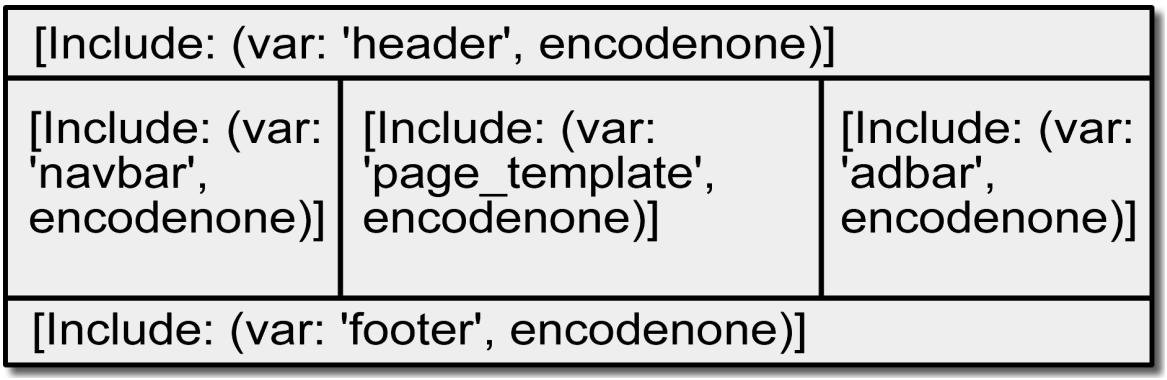
Typically, the first thing a stub file does is call the site config file. Where the role of the stub file is to direct traffic and define page-specific values, defining site-wide variables is the job of the site config file. For example, a common site config file setup includes sub-routines for determining the current page name and current directory. Though the cascading behavior of variables, the values generated by these subroutines are available to all pages that call the siteconfig file.

The final thing that the stub file does is call the master, or shared template. This file usually contains, in html formatting terms, the "skeleton" of the site. For example, if all the pages on a site share the same table layout, this table would be defined in the shared template.

This brings us back to our Developer and her first use of includes in her page structure. The master template functions just like this, except the path references in the [include] statements are not hard-coded. Rather, they are "placeholders", waiting for values that are

declared in the stub file and passed down through the [Include] hierarchy. This is where the flexibility of Corral really comes into play. The stub file "fuels" the master template with values for all its [include] tags. Without the stub file to guide it, the master template is literally "directionless".

So the previous table layout created by our Developer, in Corral, looks something like this:



The "shared" template above can be shared between many pages by simply changing the variables that populate the [Include] tags. These variables are changed in the "stub" file, or can be declared in a global fashion in the siteconfig file.
The "content" section, located in the middle cell, contains the main body of the page. This solves the Developers problem of finding an easy way to vary page content while still re-using common code such as is contained in the master template.

Alll the other elements, headers, footers, etc. function like they did in the simple [include] method. They become true components, or objects, for placement within the master template.

Again, the key to how this all functions lies in the *scoping of variables across [include] tags*. Variables declared in the stub file are available to all elements that follow. These variables populate the "placeholder" [include] tags with values, and through this process "build" the page piece by piece from re-usable components. The final output is a standard html-formatted page that is nevertheless "modular" behind the scenes.

Corral and Community

As previously mentioned, Corral represents a community initiative to create a recognized methodology for creating LDML-based web sites. The strength of Corral can be directly linked to the amount of Developers that are actively participating in its growth.

There are several initiatives under way to help foster the growth of Corral. The first is the Corral web site, located at http://www.corralmethod.org. The most recent copy of the white

paper can always be found at this location. In addition, the actual source code used to create the site, naturally %100 Corral-compliant, is available for download.

The Corral Talk mailing list welcomes all participants. Its members read like a "who's-who" of active Developers in the community. "Newbies" to Lasso or Corral are also very welcomed. In fact, the input of those not as familiar with Lasso or Corral is critical to the development of the Methodology. New members are invited to join by sending a blank email to corraltalksubscribe@corralmethod.org.

One final noteworthy mention is the site http://www.lassodevelopment.com. While its author, Duncan Cameron, is an active contributor to the Corral Talk list, he is also supporting Corral by using "Corral Compliance" as a distinguishing feature of individual Developers listed in his directory. It is important to note that simply because a Developer lists herself as Corral-compliant does not mean that she has to code her site in %100 compliance with the published standards. Simply being aware of the techniques used in Corral is sufficient to call oneself Corral-compliant. And this is an important point to stress. Corral is not about arm-twisting. Its about enabling. Its about sharing knowledge, and passing on accumulated wisdom and techniques. The advantages to using a system like Corral are truly what the Developer chooses to make of them.

Corral – Commercial Advantages

One of the advantages of following a Structural Methodology is that it provides a guideline and "road map" for others to follow when adapting solutions created with it. This applies to the development of commercial applications as well. Just as some products, including LP5, make portions of their code open source to assist 3rd party development, so to should a Structural Methodology assist Developers in creating solutions that can fit easily into existing structures. The modular nature of Corral and its emphasis on creating a common lexicon assists Developers in creating solutions that can plug into larger applications, and provides the "language" for describing how the modules work and what pre-existing conditions (or example, site config variables) they rely on to work.

Also, adherence to an accepted methodology can provide a nice "value-add" to Developers when pitching services to prospective Clients. One of the biggest fears from the Client side is being given a "black box" application, where the code is so dense that only the original Developer can understand it. The traditional, responsible method that Developers use to alleviate these fears is through thorough documentation. However, in some cases, there might not be enough space or time in the budget to make this feasible. When faced with this dilemma, or even as a matter of course, it is beneficial for Developers to assure their Clients that the code they are receiving is created using an accepted and publicly documented methodology, with a large pool of Developers to fall back on should the need arise.

Corral and LP5

The most exciting upgrade in Lasso history, LP5, proves to be all the more remarkable when you realize the pains that Blueworld has taken not to fundamentally change the base-level tags like [var] and [Include]. This means that although methods for accessing data may change from Lasso 3.x to LP5, the methods used to structure sites with Corral or any other methodology should remain unaffected.

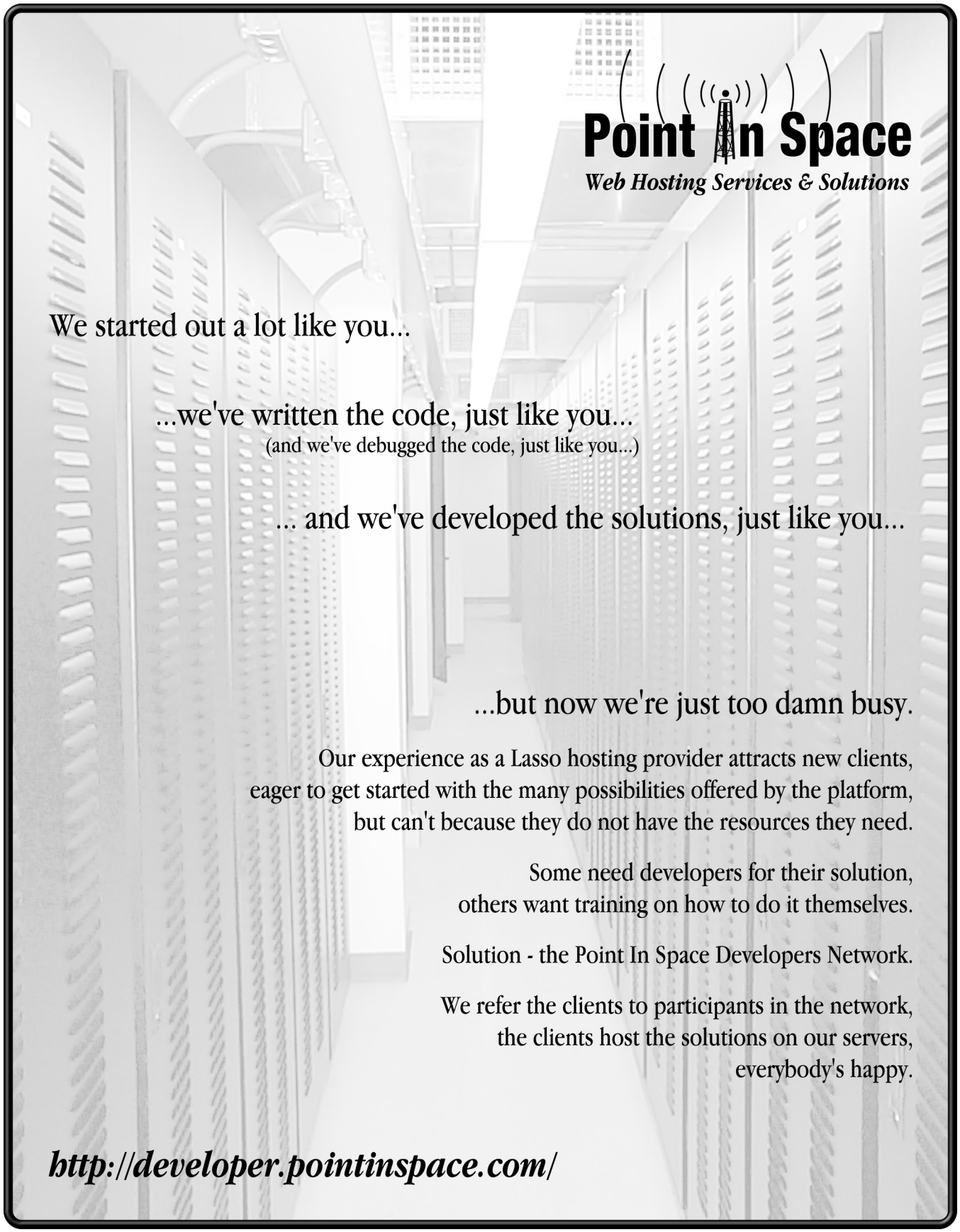
As the beta version of LP5 is circulated throughout the Developer community, new ways of incorporating LP5 functionality into Corral will undoubtedly be discussed and become part of the methodology going forward. The best thing about LP5 and Corral to date is what hasn't happened; the power and flexibility of Corral hasn't been compromised by any of the new features. In other words, it's a great time to be a Lasso Developer, and an even better time to be a Corral-compliant one.

Conclusion

Adopting some kind of structural method for managing large web applications is an essential step in becoming an effective Lasso Developer. Developers have the choice of either adopting their own methodologies, or adopting (or even creating) a documented and supported method such as The Corral Methodology. While the advantages that come from adopting a widely supported Methodology are many, perhaps the greatest advantage comes from the community aspect. Instead of being an "island to one's own", the Developer has the opportunity to draw from and participate in something larger than the sum of its parts. Corral, as the first largely supported Structural Methodology to arise in the Lasso Developer community, may not be the best solution to everyone's problems, but it's free, it's flexible, and it's open to suggestions from anyone who might want to participate.

Copyright Notice

The term "Corral Methodology" is copyright 2001 by Lassosmart.com, a wholly owned subsidiary of Spring Hollow Publishing, Inc. The methods used by the Corral Methodology are Open Source to any and all who wish to use them. All other products mentioned herein are property of their respective owners.


Point In Space
Web Hosting Services & Solutions
We started out a lot like you...
...we've written the code, just like you...
(and we've debugged the code, just like you...)
... and we've developed the solutions, just like you...
...but now we're just too damn busy.
Our experience as a Lasso hosting provider attracts new clients,
eager to get started with the many possibilities offered by the platform,
but can't because they do not have the resources they need.
Some need developers for their solution,
others want training on how to do it themselves.
Solution - the Point In Space Developers Network.
We refer the clients to participants in the network,
the clients host the solutions on our servers,
everybody's happy.
http://developer.pointinspace.com/

Debugging Your Code

Table of Contents

Debugging Your Code

by Douglas Burchard

Do you and your client, or manager, agree on what is a bug? Have bugs in your web application caused you to exceed your schedule or budget? Less than twelve percent of the most costly bugs are caused by faulty code. Doug will introduce a manageable definition for a bug that you, and those you work for, can agree on. He will also present a series of processes that help avoid the inclusion of bugs in your Lasso Professional 5.0 web application, and procedures for eliminating the bugs that do slip in.

Biography

Douglas Burchard has been in web development since 1994, and creating data-driven web applications with Lasso since before it's initial launch in 1996. He has held several senior level positions with prominent web application consulting firms, including nearly three years with Blue World Communications, Inc. Prior to gaining experience as a web developer Doug earned a degree in Graphic Design and spent six years in the graphic design industry. A frequent contributor to the Lasso Talk list, Doug recently opened his own consulting practice, partnering with firms to launch their data-driven applications on the Internet.

Notes

Debugging Your Code

The cause of over half of all bugs is the specification document, or lack there of. Nearly another quarter of all bugs are caused by poor design. Less then an eighth of all bugs, in any given project, are caused by errors in programming. And most of those are so obvious as to be localized and eliminated before the code of the system they're in is finished.

The longer a bug survives in a project, the more costly it is to fix. A bug in the specification document on the upstream end of a project can cost as much as 200 times more to fix downstream as when the bug was first introduced. So the goals of debugging efforts must be to avoid bugs when possible, detect early the bugs that do occur, be able to localize quickly the source of any bug, and be able to easily eliminate a bug when localized.

To be effective against the majority of all bugs, this must start at the time the specifications are being drawn up, and continue through to release of the final product.

Recognizing a Bug

Developers often try to obscure the fact that bugs exist at all. They define a bug in such a way as to make bugs non-existent, while "issues" and "variances" abound around them. Some believe the word bug reflects poorly on their work. But that's the wrong attitude to encourage in developers. A bug needs to be blameless, to engender teamwork towards achieving our goals. Call a bug a bug, and get on with the work of producing quality products.

As a normal side effect of the creative process, a bug is any instance where one of the three following rules is true:

- The product doesn't do something that the specification says it should do, or does something that the specification says it shouldn't do.
- The product does something the specification doesn't mention, or doesn't do something the specification doesn't mention but should.
- The product is hard to use, slow, difficult to understand, or will be viewed by the end user as just plain not right.

Bugs Caused by Bad Process

The development cycle is a group of processes that deliver a project from idea to final product. They include determining what end-users want from the final product, distilling this list of desires into a list of requirements, and writing specifications, which meet these distilled requirements. Also among these processes is a mechanism to ensure the final product satisfies these written specifications.

If you leave one of these processes out of the development cycle, you may launch a final product, but debugging a product with no clear definition of a bug may also overwhelm you. Inattention to the development cycle, or bad process, creates bugs. Attention to good processes avoids and removes bugs.

Debugging is a sub-group of processes that run throughout any project's development life cycle. The goals of these processes is to avoid bugs when possible, detect early the bugs that do occur, be able to localize quickly the source of any bug, and be able to easily eliminate a bug when localized.

The Development Cycle

There are many ways to deliver a project from idea to final product. The development cycle presented here is one evolved from experience with many projects, and personal research. The objective of this paper is not to explain why this development cycle is any better than another, only how processes woven throughout development contribute to debugging code.

Development Plan

The development plan provides a project-specific map from concept, through continuing changes, to finished product. Continually evolving throughout the development cycle, this document identifies what exactly the final product will do, what it won't do, who is responsible for each high-level aspect of development, and what constitutes the end of each development phase.

What the final product will and won't do is defined by the written specifications. This document helps define what is and is not a bug. It makes our first two rules for defining a bug possible. Without a specification document, there's nothing to verify the product against, and no systematic method for avoiding or detecting bugs.

Test Plan and Verification

Using a collection of distinct tests, each deliverable work-product is verified against the written specifications. This is usually tracked by using a trace-ability matrix. This matrix is created over the life of the project, starting as early as possible. Specifications are listed along the top in columns, and tests are listed along the left side in rows. Where a particular test verifies a particular specification, a one (1) is entered at the intersection of that row and column. Tests are created and added to the matrix until the sum of each column is at least one (1). Rows representing tests that do not verify a specification can be safely deleted. Running every test against every delivered build identifies bugs defined by our first rule, where specifications are not met.

Change Orders and Change Control

Creating a final product is a sequence of discovery. As the product develops, changes to the development plan, specifications document, test plan, user-interface, architecture, and finished code become necessary. To keep control of these changes, so that they do not overwhelm the budget and schedule, written change orders are created for any changes necessary to a previously delivered and approved work-product. A change order describes the change required, the reason for the change, and any effect on schedule or budget.
This process of change orders and change control addresses our second rule for defining a bug. If the written specifications fail to mention something they should, or mention something they shouldn't, a controlled process is in place to change those specifications.

User-Interface Prototype

End-users view the interface presented to them as the entire application. Experiencing the product, at the user's "eye level", early in the development cycle, often leads to changes that would be more costly to address in a later phase of development. This addresses the latter half of our third rule for defining a bug. If the web application is difficult to understand, or will be viewed by the end user as just plain not right, changes need to be made.

Application Architecture

Before writing actual code, the general architecture of the application is outlined. This phase allows the developer to design how different parts of the user-interface, code, database, web server, and any additionally required hardware or software will interact.

Staged Delivery of Individual Systems

Individual systems are delivered in stages as finished, releasable, builds. Each stage consists of the detail design, code writing, testing, and delivery of an individual system within the final product. The stages are ordered so that systems with the most desired functionality are delivered first. This helps identify early the first half of the bugs defined by our third rule, If the web application is hard to use, or slow, changes need to be made. This also allows for earlier deliverable work-products to be verified against the written specifications. By taking each stage to a completely finished state before delivery, interested parties have the best possible chance of identifying bugs.

Installation and Launch

Once development is complete, the web application is moved from the development server to the production server. The final product is tested and verified in its final production environment against the written specifications.

Bugs Caused by Bad Syntax

The first section of this paper dealt with the majority of bugs caused by bad planning and process. This next section deals with the more commonly talked about, but less prevalent, type of bug, the humble syntax error.

For all the planning described in the first section of this paper, sometimes the final product simply doesn't do what it's suppose to do, or does something it is not suppose to do. Syntax errors are usually so obvious that they're identified before the code of the system they're in is finished. But identifying that they exist, and localizing them to a specific section of code are two different tasks. And eliminating them can sometimes be difficult as well.

A debugger is an application that helps in locating and correcting programming errors. A good debugger recognizes errors in the coding language, variable names, data-types, and other assorted problems. Better debuggers look through an entire work-product and display all found errors at once, providing file names and line numbers where the errors are located. Whether or not you have a debugger available to you, there are measures you can take to help localize and eliminate bugs caused by bad syntax.

Custom Logging

During the application architecture phase, described above, define a custom logging routine to be used throughout the final product. This routine is a custom tag or function that saves a snapshot of the current state to a file, window, or database. Saving to a file has the benefit of continuing to work even if the database is unavailable.

The snapshot should include a date and time stamp, the current file name, and some sort of name identifying the instance of the routine. This information helps to connect a line in the resulting log with an actual location in your code.

The current name and values of all instantiated variables can be helpful in identifying a line in the log that corresponds to a reported instance of a bug. The current error message, error code, databases, tables, and details of current transactions help detect bugs as well.

Design into your routine a Boolean flag, set universally, to turn the logging on and off. Several different flags can be used to turn logging on for some criteria, and off for others.

Custom Debuggers

While building a debugger from scratch is well beyond the scope of this paper, there's a lot to be said for building a routine into your product to display error codes and state information

on each screen.

Similar to the routine for custom logging discussed above, information can be gathered and added to an array for display at the bottom of each resulting screen. The necessary information is slightly different. But the routine instance, current error message, and current error code provides a wealth of information. Designing your custom debugging routine to accept specific static and dynamic data for each instance is helpful as well.

As with the custom logs, you'll want to be able to turn these debugging messages on and off with a universally set Boolean flag. Restricting their display to a specific end-user allows debugging to take place after release, without unduly disrupting normal activity.

Encapsulation and Modularity

Encapsulation means to provide a programming interface to a section of code, such that even if the enclosed code changes how it performs a task, the interface it provides to other functions doesn't change. The antithesis of this is often called spaghetti code and recalls the way in which a plate of noodles shifts on one side, when a single noodle on the other side is pulled. Modularity refers to the sectioning of code into logical divisions for greater flexibility and easier repair.

The more encapsulated and modular you can design your product, the easier it is to debug. This is because code is protected from changes in one location, affecting functionality in another location.

Shared Code

Complimenting the encapsulated and modular design discussed above, sharing routines across screens means only fixing a bug once. Store your code in libraries, small collections of routines stored in separate files, and call only those libraries you need for each screen. Not only does it mean fewer bugs, if you reuse a library from a previous project, chances are you have already debugged that library before.

Code Editing Environments

Many code-editing environments, whether an Integrated Development Environment (IDE), a WYSIWYG editor, or an HTML editor, provide some means of building a glossary or adding additional functionality. Glossaries provide an alternate means of typing frequently needed syntax. They assist in avoiding spelling errors and can speed up the task of keying in code. Some third parties have created plug-ins and glossaries for the most popular HTML editors.

Conclusion

Debugging code is not a single step in the development cycle, or a menu item in a code-editing environment. Debugging is a sub-group of processes that run throughout any project's development cycle. The goals of these processes is to avoid bugs when possible, detect early the bugs that do occur, localize quickly the source, and easily eliminate a bug when localized.

If you shortcut the development cycle by leaving out a critical process, the greater down-stream cost of fixing a bug may overwhelm the entire project. Inattention to the development cycle, or bad process, creates bugs. Attention to good process avoids and removes bugs.

The cause of only an eighth of all bugs, syntax errors are usually so obvious that they're identified before the code of the system they're in is finished. Debuggers are very useful in localizing and eliminating syntax errors. But, whether or not you have a debugger available to you, there are additional measures you can take.

iSolutions
forward thinking, simple solutions

iSolutions, Inc. Lasso Development Services

One of the Nation's Largest FileMaker® and Lasso® Consulting Firms

~ We provide Lasso design, development, and project management services for companies of all sizes.
~ We are an innovator and industry leader in FileMaker web technologies.

Proud Sponsor of:

Lasso Summit 2001

Lasso Summit

FileMaker

Lasso blueworld POWER

Also providing services in FileMaker Consulting and Development and Flash Development

For more information:
iSolutions Inc.
1924 E. McFadden Ave. ~ Santa Ana, CA 92705
Phone ~ 714-245-0311 or 888-212-4620 ~ FAX 714-245-0316
Email ~ pj@isolutions-inc.com ~ Web www.isolutions-inc.com

All products are trademarks of their respected companies.

Getting Started With Lasso mySQL

Table of Contents

Getting Started with Lasso MySQL

by Michael Collins

Lasso MySQL is a version of MySQL that has been integrated into Lasso Professional 5. Lasso MySQL has been designed specifically as a back-end data source for dynamic Web sites. Michael will demonstrate how to get started using Lasso MySQL, some tools that can be used to create database schema, and the LDML tags and administrative application that comes with Lasso 5. He will also discuss techniques for preparing a FileMaker Pro-based Lasso solution for transition to MySQL.

Biography

Michael maintains a well-referenced Web site, LassoDev.com and is known for openly sharing his opinions on the Lasso Talk List. Michael has a Masters degree and a professional computer-related career spanning nearly 10 years. He has been pursing Lasso perfection since he began work with Blue World Communications in 1996. After 14 months as part of the support and development team at Blue World, he struck out on his own to develop Lasso-driven Web sites with Advanced Database Systems. Michael currently provides Web related consultant services with his Seattle based company, Kuwago Web Services.

Notes

Getting Started with Lasso MySQL

Introduction

Lasso Professional 5 installs with a fully integrated MySQL data source known as Lasso MySQL. Lasso 5 utilizes Lasso MySQL for the "site" database that serves as a storage place for site preferences, outgoing email, sessions, and security settings. The Lasso MySQL data source can also be made available for user-defined databases. MySQL is particularly well suited for this purpose because MySQL:

- Is the most popular open source SQL database in use for Web applications.
- Includes the commonly found features of a Relational Database Management System (RDBMS).
- Was designed for its use on the Web.
- Has a reputation as being the fastest back-end database for dynamic Web sites.
- Is one of the easiest SQL database to learn.

Lasso 5 also makes it even easier to get started with MySQL since most of the code written for Web sites integrated with FileMaker Pro can be used with MySQL with relatively few modifications.

Probably the first question that will be asked about the introduction of MySQL is: "Will I have to learn SQL?" The answer is, not as much as you would think. Several tools are available in Lasso 5 to shield you from the details of SQL. A real mastery of SQL takes some time, but one can attain the basic skills quickly. Nevertheless, bear in mind that Lasso makes it possible to use MySQL without any SQL knowledge whatsoever. Techniques for writing LDML will remain very similar to what you may be accustomed to with a FileMaker Pro back-end. The reason for this is that LDML offers a meta-language to perform the same functions as would be carried out in SQL. You will find that you can build quick solutions using standard LDML tags but be able to devise more powerful SQL statements when the need calls for it.

Others will ask: "Is MySQL harder than FileMaker?" I would argue that the answer is no, as long as you are interested strictly in the Web aspects of your database and your GUI will be built as HTML. Once you get used to MySQL, you will find it easier in some ways, yet, it can be challenging in the details. For example:

- What data types do I use for my fields?
- How do I keep referential integrity intact?
- How can I display data from multiple tables?
- Why and where do I apply indexes?

Again, you can grow these skills while utilizing a functional database system.

The purpose of today's talk is not to convince you to abandon FileMaker but to show you how you could off-load your most active database activity to MySQL while retaining other database information in a FileMaker Pro database. Is MySQL integration the death knell for FileMaker Pro when it comes to Lasso? I do not think this is the case. FileMaker Pro will still be the better choice for instances when user interface needs to be done quickly and cheaply for use by a workgroup on the LAN.

My talk today will emphasize Mac OS X, but the same concepts will apply to Windows, so please don't get up and leave if you happen to prefer Windows.

Installation Issues

Lasso MySQL is installed and activated with Lasso 5, in fact, Lasso MySQL is required. Lasso MySQL should be considered part of Lasso. The port and socket for Lasso MySQL should not be changed. Lasso Service gets all of its preferences from Lasso MySQL. It connects using these values as part of its startup sequence and will fail if Lasso MySQL is not active on the proper port and socket. If you need options that can only be achieved using a standard installation of MySQL, you should install a standard installation and configure it in Lasso Administration. There should be no significant disadvantage to running two copies of MySQL on your machine.

After Lasso 5 is installed, there is nothing further to be done to enable Lasso to use MySQL. The initial Lasso MySQL security settings include three user accounts:

- lasso@localhost – Lasso 5 is permitted access to all Lasso MySQL databases with this user account. The Lasso user has a password that is only known to Lasso and cannot (or should not) be altered. Lasso Service will not start unless it can log in to Lasso MySQL on the local host using the pre-defined password.

- root@localhost – This is the main administrator for Lasso MySQL. By default, the root user is assigned an empty string ("") as the password. Note that this root password is the one specific to MySQL, not the one for the OS.

- No user (expressed as an empty string "") - This user account is created for the case in which no user name is used when requesting access. This user account is not given any access privileges whatsoever. It prevents someone from gaining access with no user name and no password (unless the administrator allows it).

The two accounts that do allow access will allow local access only. Thus, in no case is remote access permitted.

Working with the Mac OS X Terminal

Mac OS X comes with a Terminal application that can be used for issuing instructions to the OS from a command line. You will find it necessary at times to perform some tasks using the Terminal application and so should become a bit familiar with its use. When you log in with telnet or with the terminal application, this is referred to as "opening a shell" and the owner of the shell determines what privileges you will have when interacting with files and executing commands. When you open the terminal application it starts a shell specific for your user account. Many of the commands and utilities needed for administrative purposes however require that you open a root shell, in other words, you need to log in as the root administrator.

When you install Mac OS X you set up an Administrator user name and password, this OS user account allows you to perform administrative functions but is not the root account that has complete control over all functions of the system. Apple has elected to not enable this account by default. It must be explicitly enabled for it to be available. In my opinion, when administering Lasso MySQL on the command line using the terminal program on Mac OS X you are best off not enabling the root account. The reason for this is to lessen the chances that you will create files and install programs that can only be used by root, thus not allowing the system to work for a user account with fewer privileges. It is also common practice in UNIX not to use the root account to perform general tasks and it should only be used when it is necessary.

To work with just the Admin user name, prefix any command with "sudo". If a password is required, you will be prompted for it. The sudo program takes as an argument a command line to be executed as root. Your user account must be included in a configuration file to be allowed to utilize this command. This file also specifies the commands you can run, in the case of Mac OS X the user that you set up when installing the OS is included on this list by default and assigned privileges that allow it to perform any task that root is able to. For example, to run the above command use:

```
root# sudo ./mysqladmin -u root password thenewpassword
```

Alternatively, if you will be executing a series of commands as root then start with the "sudo su" or "sudo -s" command to start a new shell as the root user. Use exit to stop issuing commands as root and exit the shell. For example:

```
% sudo su
Password:
root# ./mysqladmin -u root password thenewpassword
root# (another command)
root# (another command)
root# exit
```

While sudo would require you to enter a password each time:

```
% sudo ./mysqladmin -u root password thenewpassword
password:
% sudo (another command)
password:
% sudo (another command)
password:
% sudo etc
password:
```

If you open a root shell to accomplish a series of tasks, there will not be a need to use "sudo" for each command.

Note that in this document, lines that begin with % indicate a UNIX command prompt, it may actually look more like:

```
[localhost:~] mcollins%
```

The command prompt on Mac OS X will appear with the current host followed by the location you are currently in within the UNIX file system (within square brackets), and then followed by the owner of the current shell. However, the prompt can be customized and you should not expect it to always look the same on every system (especially on other UNIX type systems). The text that follows is what would be typed into the terminal. In addition, root# at the start of the line indicates that the command needs to be issued while logged in as the Mac OS X root or with root permissions.

Note that Mac OS X maps its own GUI file system on top of the UNIX file structure so it may appear as though some files do not exist on you system when navigating through folders in the AQUA interface. The terminal application is the best means to tell exactly what files you have and where they are located.

In the simplest of terms, many Unix commands are files that contain a script and have the execute bit turned on (many others are compiled applications). To use a command that is not in the standard location for commands, precede the command with the shorthand "./" to mean "look in this folder for the command". To view the paths to your commands, use the following:

```
% echo $PATH
```

These paths are what Mac OS X will use to search for any command that is specified on the command line. If a command is not found on any of the indicated paths an error occurs. The common location for locating scripts so that they behave like commands is in the directory:

```
/usr/local/bin
```

If you have an executable file in this location you are not required to indicate the path to the

file in order to execute the contents of the file as a script, and you also do not precede the command with a "./". One example use of this is to locate all of the mysql client programs (more to follow on these in the next section) into the /usr/local/bin directory so that you can then simply type the name of the client program to execute it.

MySQL Root Account

If there are any other users with accounts on your server, it is essential that you protect the MySQL root account by creating a password for this account. Even if you are the only one with access to the machine, it is prudent to plug any holes that may be exploited by hackers. The key thing to keep in mind is that the OS root is not the same as the MySQL root. You don't need to have root access to the server to attempt to use the MySQL root account. There might be other user accounts to the server that may be able to navigate to the mysql utility and log in as the MySQL root with no password. This can happen even through a remote telnet session, since even though root is set to localhost only, a telnet session allows one to connect and work as a user would on the localhost. Having no root password would mean this intruder could send any SQL command to the databases, and even delete them. The root account created in MySQL uses the name "root" by convention, it is not required that the account use the name "root", it could be changed to any name you want.

Setting the MySQL root password is an example of when you will need to use the Mac OS Terminal. The task of setting the root password is accomplished using mysqladmin, one of the utility programs installed by MySQL (and Lasso MySQL). The Lasso 5 Setup Guide provides further instructions on how to set the MySQL root password. The command is as follows:

```
root# cd /Applications/LassoProfessional5/LassoMySQL/
root# ./bin/mysqladmin --socket=/tmp/LassoMySQL.sock -u root password thenewpassword
```

Tip: If a password was already set for MySQL root and you want to change the password, you could add the -p option (after root in the above command), you will then be prompted for the existing MySQL root password.

Starting and Stopping Lasso MySQL

When Mac OS X starts up it runs the script located at the following file path:

```
/Library/StartupItems/LassoMySQL/LassoMySQL
```

This script is installed by Lasso. It contains the commands that allow Lasso MySQL to start with the correct configuration. This script contains the following commands:

```
cd /Applications/LassoProfessional5/LassoMySQL/
./bin/safe_mysqld --port=14551 --socket=/tmp/LassoMySQL.sock &
```

The port and socket are different from a standard install of MySQL in order to allow Lasso MySQL to coexist with a standard install of MySQL.
If you need to restart Lasso MySQL for some reason, you can execute the two commands shown above, or execute the startup script as follows:

```
root# cd /Library/StartupItems/LassoMySQL/
root# ./LassoMySQL
```

Note that to stop Lasso a script is provided to stop all associated processes:

```
root# cd /Applications/LassoProfessional5/
root# ./stopmysql --socket=/tmp/LassoMySQL.sock
```

To view Lasso activity and errors in the terminal you need to kill the Lasso process and then start it again using the -a option, as follows:

```
% ps aux
(note the PID for Lasso Service, in this example it is 290)
% sudo -s
root# kill 290
root# cd /Applications/LassoProfessional5/
root# ./LassoService -a
```

Lasso will now initialize and any errors will be reported to this terminal window. Open a new terminal window to execute other commands.

MySQL Client Program Utilities

In addition to the primary MySQL application (the mysqld daemon), MySQL comes with a number of client utility programs. The client programs that would be most relevant to Lasso MySQL administrators include:

- **mysql** - Is an interactive program that allows one to connect to the MySQL server, issue SQL queries or MySQL functions, and then view the results. Note that the mysql client program is always referred to in lower case.

- **mysqlaccess** - Checks the access privileges for a host, user, and database.

- **mysqladmin** - Performs administrative operations, such as creating or dropping databases, reloading the grant tables, flushing tables to disk, and reopening log files. mysqladmin can also be used to retrieve version, process, variables, and status informa-

tion from the server.

- **mysqldump** - Creates a text representation of a MySQL database as SQL statements, this can be put into a file or printed out to the screen.

- **mysqlimport** - Imports text files into their respective tables using the SQL command LOAD DATA INFILE.

- **myisamchk** - For checking, optimizing, and repairing MySQL tables.

- **mysqlshow** - Displays information about databases, tables, columns, and indexes.

The following is a brief look at how some of them actually work.

Each time that you use one of the client programs, a user account and password needs to be specified. In addition, because Lasso MySQL is a non-standard install, the socket must be specified. The port option (--port=14551) is not needed, as this is specified in the socket (but won't hurt if you add it). These are all specified using options on the command line, -u is for the user account name, -p is for the password, and –socket indicates the specific MySQL server that should be used. Here is an example of using all of these options:

```
% cd /Applications/LassoProfessional5/
% ./bin/mysql --socket=/tmp/LassoMySQL.sock –u root –p 'xyz'
```

Actually, it is best if the actual password is not indicated on the command line. Instead, only indicate the –p option with no password. When you do so, a prompt will show after you hit the return key to allow you to enter the password. For example:

```
% ./bin/mysql --socket=/tmp/LassoMySQL.sock –u root –p
Password:
```

The reason the password is entered in the prompt and not on the command line itself is because every command entered into the command line is saved into a file as clear text (your command history file that is retrieved using the arrow keys). Note that if you leave out the user account name, the user name for the owner of the current shell is assumed. For example, if you have opened a root shell (as explained earlier), then executing one of the client programs will use root by default. For Example:

```
root# ./bin/mysql --socket=/tmp/LassoMySQL.sock
```

Thus, if logged in as root you don't need to specify –u root and if using a shell owned by your admin account you use –u root and -p. The following examples leave out these options for brevity sake (besides, while in the root shell the root account is used automatically). See the next section for how to make it a bit easier to log into the mysql client programs.

Here is a short demonstration of using the mysql client programs included in Lasso MySQL. In this I first start a root shell (since I am using the root account to administer MySQL) and then move to the location where the client programs are located:

```
% sudo –s
root# cd /Applications/LassoProfessional5/LassoMySQL/
```

Create a database from an import file located at the root of this computer:

```
root# ./bin/mysql MCExample < /Import_MCExample.mysql
```

Show the currently available databases:

```
root# ./bin/mysqlshow
```

Dump the schema of a database to the terminal (not something you want to do with a large database):

```
root# ./bin/mysqldump --opt MCExample
```

See TABLE 1 to see the output for this command on next page.

To dump the schema of a database to a file use the following:

```
root# ./bin/mysqldump --opt MCExample > /MCExampleCOPY
```

Full details on the options available for the MySQL client utility programs can be found in the MySQL Manual. A copy of the pdf of this document should be included with the Lasso 5 installation package.

Creating Command Aliases

In the current pre-release copy of Lasso 5 on Mac OS X, one has to specify the socket option each time one attempts to use one of the MySQL client programs for databases hosted by

Lasso MySQL. As an example to run these commands they need to use the form:

```
root# ./bin/mysql --socket=/tmp/LassoMySQL.sock MCExample < /Import_MCExample.mysql
```

In short, it becomes burdensome to type --socket=/tmp/LassoMySQL.sock each time.

To change this behavior, you can make aliases of the MySQL supplied client programs and then be able to run these programs as follows:

```
root# lmysqldump --opt lassol_site_1 > /temp.mysql
```

Note that the usual client program command is preceeded by a single letter"l". To modify your command aliases, edit the file located in the following location:

```
/usr/share/init/tcsh/aliases
```

This initialization file sets up the shell environment for all users that log into the system. As an alternate approach you could edit the initialization file that is specific to a user account so that all users do not have these options. Add the following text to the end of the aliases file:

```
# MY ALIASES
alias lmyisamchk '/Applications/LassoProfessional5/LassoMySQL/bin/myisamchk
--socket=/tmp/LassoMySQL.sock'
alias lmyisampack '/Applications/LassoProfessional5/LassoMySQL/bin/myisampack
--socket=/tmp/LassoMySQL.sock'
alias lmysql '/Applications/LassoProfessional5/LassoMySQL/bin/mysql --socket=/tmp/LassoMySQL.sock'
alias lmysqlaccess '/Applications/LassoProfessional5/LassoMySQL/bin/mysqlaccess
--socket=/tmp/LassoMySQL.sock'
alias lmysqladmin '/Applications/LassoProfessional5/LassoMySQL/bin/mysqladmin
--socket=/tmp/LassoMySQL.sock'
alias lmysqldump '/Applications/LassoProfessional5/LassoMySQL/bin/mysqldump
--socket=/tmp/LassoMySQL.sock'
alias lmysqldumpslow '/Applications/LassoProfessional5/LassoMySQL/bin/mysqldumpslow --socket=/tmp/
LassoMySQL.sock'
alias lmysqlhotcopy '/Applications/LassoProfessional5/LassoMySQL/bin/mysqlhotcopy
--socket=/tmp/LassoMySQL.sock'
alias lmysqlimport '/Applications/LassoProfessional5/LassoMySQL/bin/mysqlimport
--socket=/tmp/LassoMySQL.sock'
alias lmysqlshow '/Applications/LassoProfessional5/LassoMySQL/bin/mysqlshow
--socket=/tmp/LassoMySQL.sock'
alias lmysql_zap '/Applications/LassoProfessional5/LassoMySQL/bin/mysql_zap
--socket=/tmp/LassoMySQL.sock'
alias lsafe_mysqld '/Applications/LassoProfessional5/LassoMySQL/bin/safe_mysqld
--socket=/tmp/LassoMySQL.sock --port=14551'
```

Note that each alias definition needs to be on its own line. This list includes the most commonly used items found in the bin directory of the MySQL distribution but not all. All of these should have been installed by Lasso MySQL.

Creating a User Account

You will most likely want to create additional user accounts so that you are not using the root account to administer all databases on the server. You will need to grant a new user account access privileges that allow access to the appropriate databases. You can accomplish this using the GRANT SQL command. This can be accomplished using the mysql client program, but you can also issue a SQL Statement using other methods that will be discussed later in this paper. To open the mysql client program type in the following commands:

```
root# cd /Applications/LassoProfessional5/LassoMySQL/
root# ./bin/mysql
```

Now you are on the command line of the MySQL client program, from here you can issue MySQL commands and SQL statements directly to the MySQL server. The form of the grant command is:

```
mysql> GRANT ALL ON myDBName.myTableName TO myUserName@domain.com IDENTIFIED BY
“myPassword”;
```

There are more specific options than “ALL” which assigns all permissions. To assign privileges to a specific table (which is called “web” in the following example) of a single database (for example, “MCExample”) but allow the user to only connect from a specific domain:

```
mysql> GRANT ALL ON MCExample.web TO john@localhost IDENTIFIED BY “chickensoup”;
```

You can also restrict to a remote host using an asterisk as a wildcard as follows:

```
mysql> GRANT ALL ON MCExample.web TO john@*.MCExample.com IDENTIFIED BY “chickensoup”;
```

To assign privileges to a specific table of a single database, but from any remote location:

```
mysql> GRANT ALL ON MCExample.web TO john@% IDENTIFIED BY “chickensoup”;
```

The % sign is used here as a wildcard, meaning any host. @% is the default if no host is indicated so this is the same as:

```
mysql> GRANT ALL ON MCExample.web TO john IDENTIFIED BY “chickensoup”;
```

To assign privileges to all tables in a single database:

```
mysql> GRANT ALL ON MCExample.* TO john IDENTIFIED BY “chickensoup”;
```

If you want a blanket approach to allow you to have permission to all databases, the following

SQL command can be issued:

```
mysql> GRANT ALL ON *.* TO john IDENTIFIED BY “chickensoup”;
```

Here is an example on assigning privileges that are more specific to a particular user, database, and domain:

```
mysql> GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP ON bernddb.* TO
bernduser@LOCALHOST IDENTIFIED BY “chickensoup”
```

Note: If you grant a user access from the “localhost” host, but would also like that user to be able to connect from any remote location, you need to grant access to that user twice. For example:

```
mysql> GRANT ALL ON *.* TO john@localhost IDENTIFIED BY “chickensoup”;
mysql> GRANT ALL ON *.* TO john IDENTIFIED BY “chickensoup”;
```

Creating Databases, Tables, and Fields

The easy part about creating a set of databases to be used in a SQL based system is the actual creation of databases, tables, and fields. The hard part is to know what to create and defining the entities and relationships that make up the database schema. There is not the space here for even a partial discussion of database design, and so this discussion focuses on the actual mechanisms of creating the database schema.

The LassoMySQL folder uses the same set of directories and files as would be found in a standard MySQL installation. Your databases are located in the data directory (in other words, the DATADIR environment variable for the MySQL binary used with Lasso). For Lasso MySQL the data directory is at:

```
/Applications/LassoProfessional5/LassoMySQL/var/
```

To create a database using the terminal First start a root shell:

```
% sudo su
```

Change your current location to the directory in which the MySQL utility programs are stored:

```
root# cd /Applications/LassoProfessional5/LassoMySQL/
```

Use the mysqladmin command to create a database, indicate a name (in this case it is MCExample):

```
root# ./bin/mysqladmin create MCExample
```

Now display your current list of databases:

```
root# ./bin/mysqlshow
```

A database has now been created but it has no fields or tables. You can now start the mysql client program and start issuing SQL statements to create the database elements. This will be necessary as there are a few GUI options available. In this case, an example database file will be imported. This is accomplished using the mysql client program.

First, you have to create a new database as is shown above. The file containing the SQL commands that construct the database schema are then moved somewhere local to the mysql client program, in this case, in the same directory (the import could happen remotely as well). Execute the command to direct the contents of the file to the mysql client program:

```
root# cd /Applications/LassoProfessional5/LassoMySQL/
root# ./bin/mysqladmin create MCExample
root# ./bin/mysql < Import_MCJobInfo.mysql
```

The database will then have tables and fields created, and the contents of records also added as well. Examine the separately included Import_MCJobInfo.mysql file for details on how this is accomplished.

My personal convention is to use the plural form of a word that describes the data entity. For example, a list of people's names and addresses could be called "Contacts". Next, for each table, I create a key field that is the name of the table (as singular) plus "ID", for example "ContactID". Although, when a table has a generic name, like "Web", "ID" will suffice. In any case, the following properties are used to create the field:

```
ID smallint(6) unsigned NOT NULL auto_increment,
```

This will create a field that is a small integer with a maximum display width of 6 digits (but the full number is retained in storage and also the full value is returned to Lasso after a query is made). Unsigned means that only positive numbers will be used, while NOT NULL means that a value must be present in this field. Finally, auto_increment specifies that a sequential number be automatically generated that will uniquely identify the field. An exception is when a table served as an intersection table is unique by the presence of more than one field, then an ID field would not be needed.

In addition, I typically create a timestamp field to record when the last modification occurred, using the following properties:

```
ModifiedLast timestamp(14) NOT NULL,
```

This field does not need to be updated and when it is not, the current date and time is auto-entered into the field. This will let you know when the last time any change was made to the data stored in that table.

mysqldump could also be used to create a database that is an exact copy of an existing database. First, you need to create that database as follows:

```
root# ./bin/mysqladmin create newDatabaseName
```

Now pipe the results of the dump to the mysql client program to create a new database:

```
root# ./bin/mysqldump --opt MCExample | ./bin/mysql newDatabaseName
```

Note that this could allow you to copy a database from one server to another. To do so, use options to specify the host, username, password, or socket.

Another way to copy a database is to use the UNIX cp (copy) command. This works because MySQL databases are stored as a series of folders and files. This requires that the user associated with mysql is the owner of the current shell, which in the case of Lasso is the root user:

```
root# cd /Applications/LassoProfessional5/LassoMySQL/var/
root# cp -R lasso_site_1 lasso_site_1COPY
```

-R is the option for recursive, meaning copy all files and subfolders.

If you need to add fields or tables to a database, you have several options. You can issue SQL queries to the mysql client program that specify the database schema, use the LDML tags that are available in Lasso 5, use the Lasso Administration app, or use a third-party tool. These various options will be covered next.

Tools Available in Lasso Administration

The Build section of the Lasso Administration interface allows databases, tables and fields to be created using a visual interface. Instructions for using these tools are included in the Lasso 5 Setup Guide.

[A tour of Lasso Administration goes here, however, the final layouts are not available at the time of this writing.]

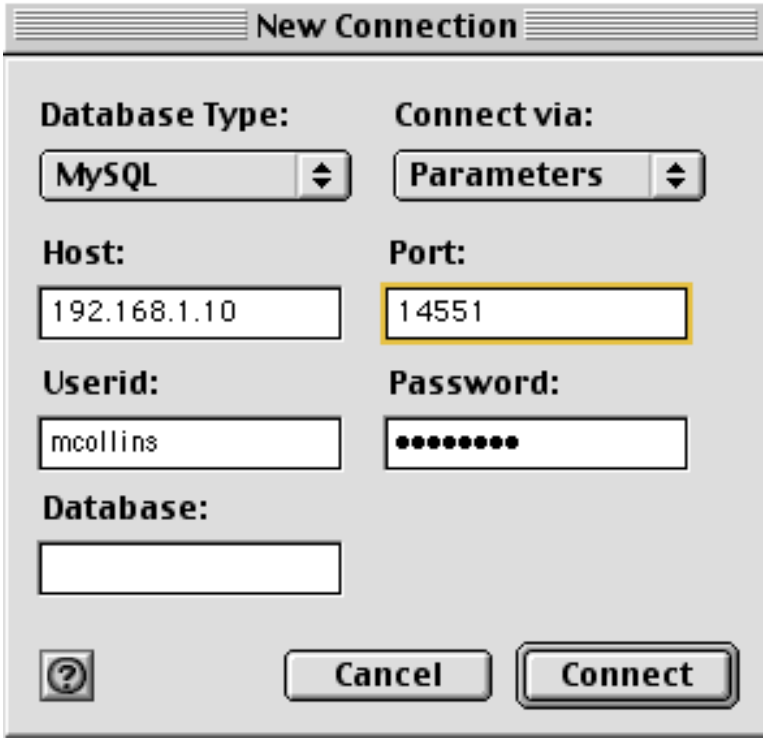
Third-Party Tool: MacSQL Monitor v 2

MacSQL Monitor is a third-party tool for managing SQL databases from the Mac OS. A new version with many great feature enhancements has just been released. Version 2 of MacSQL Monitor provides support for MySQL as well as MS SQL Server, PostgreSQL, ODBC on Mac OS 9, and Oracle via JDBC on Mac OS X (OpenBase and FrontBase are to be added at a later date). MacSQL Monitor allows users to view schema information and issue queries to SQL Databases.

The second version of MacSQL Monitor focused on usability features for those who have limited database experience. The features list for Version 2.0 includes the following:

- Spreadsheet-like viewing/editing of data.
- Automatic generation of Perl scripts and PHP pages that interact with databases.
- Table creation assistant.
- Query builder so you don't have to know SQL to use MacSQL Monitor.
- AppleScript support

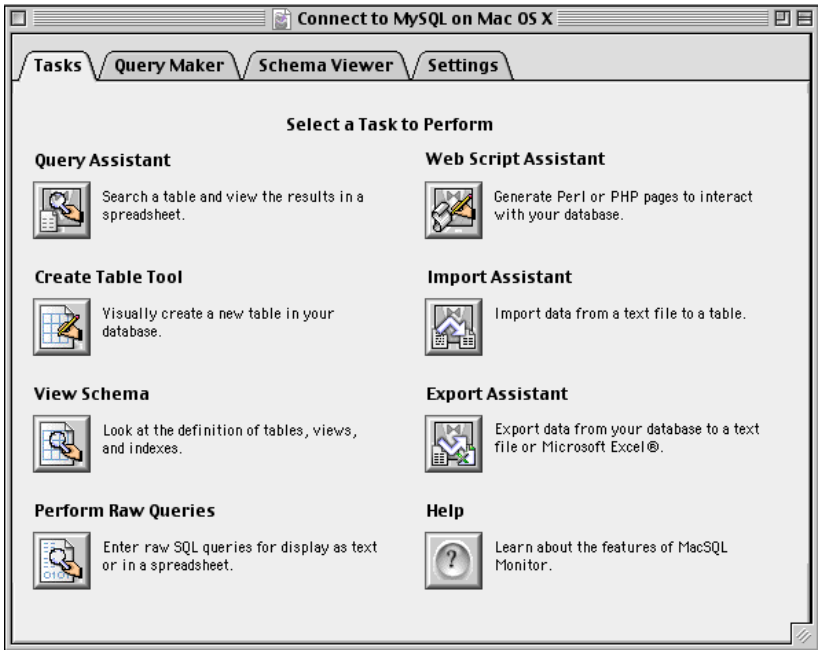
To get started with MacSQL Monitor, launch the application and select "New" from the "File" menu. Then, type your configuration as follows:



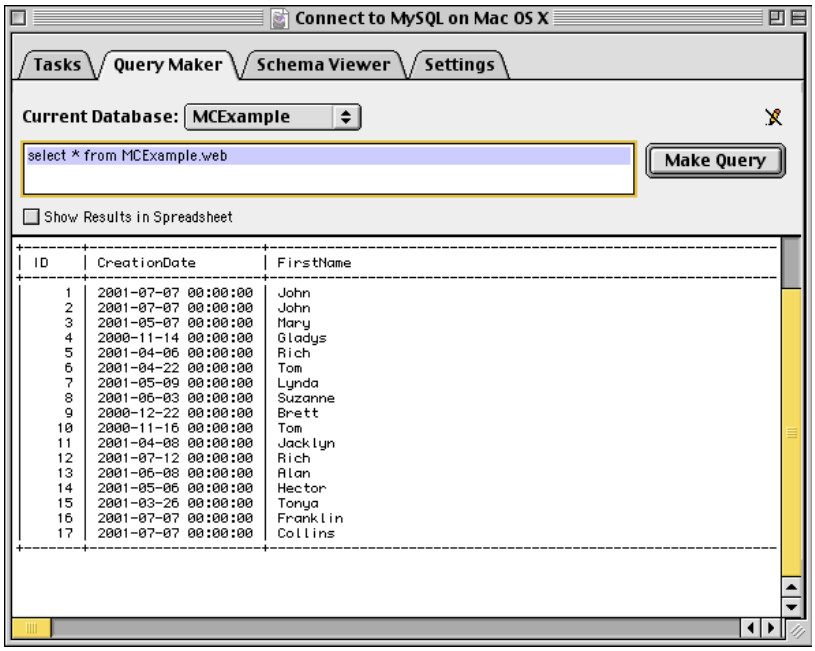
Note that you will need a MySQL user account that allows you to connect from a separate

host, not just “localhost”. MacSQL Monitor makes a direct connection to the MySQL server and thus you are considered as connecting remotely, this is in contrast to first connecting with telnet and then running the mysql client application.

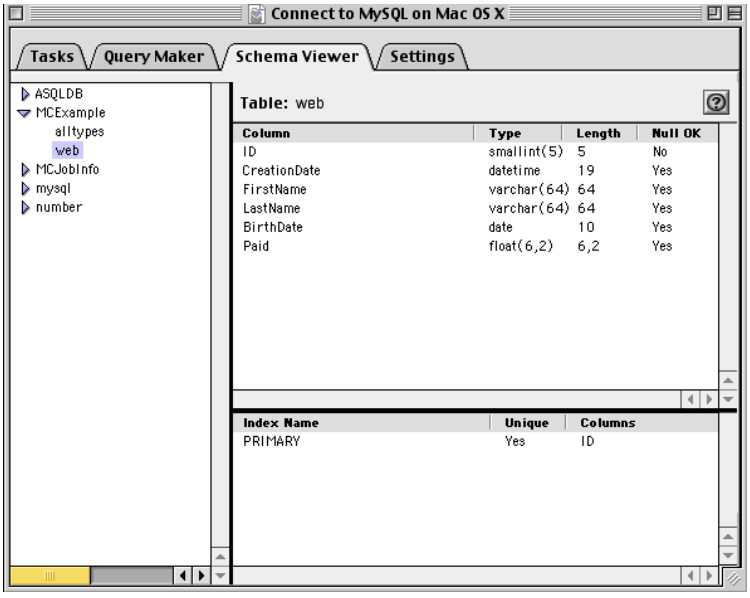
As soon as you connect, perform a save to create a file that encapsulates these settings. Then the next time you can simply open this file to launch MacSQL Monitor and make the connection to the remote server. After you first connect you will be presented with a main menu for the tasks you can perform.



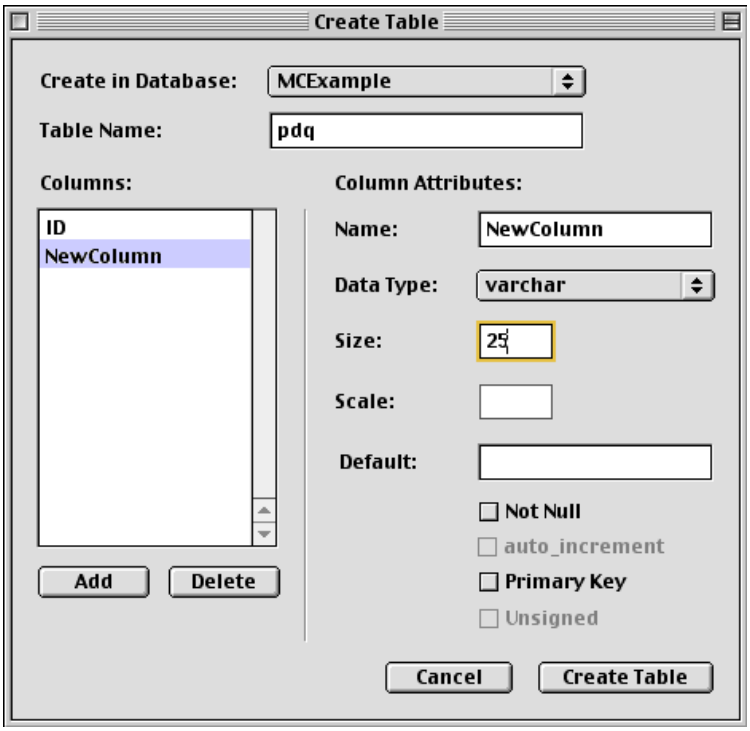
If you then click on “Perform Raw Queries” you are presented with the “Query Maker” tab. You can now type or paste in a SQL statement, execute it, and view the results in the space below. In this manner, you can issue queries directly to the data source for the database indicated.



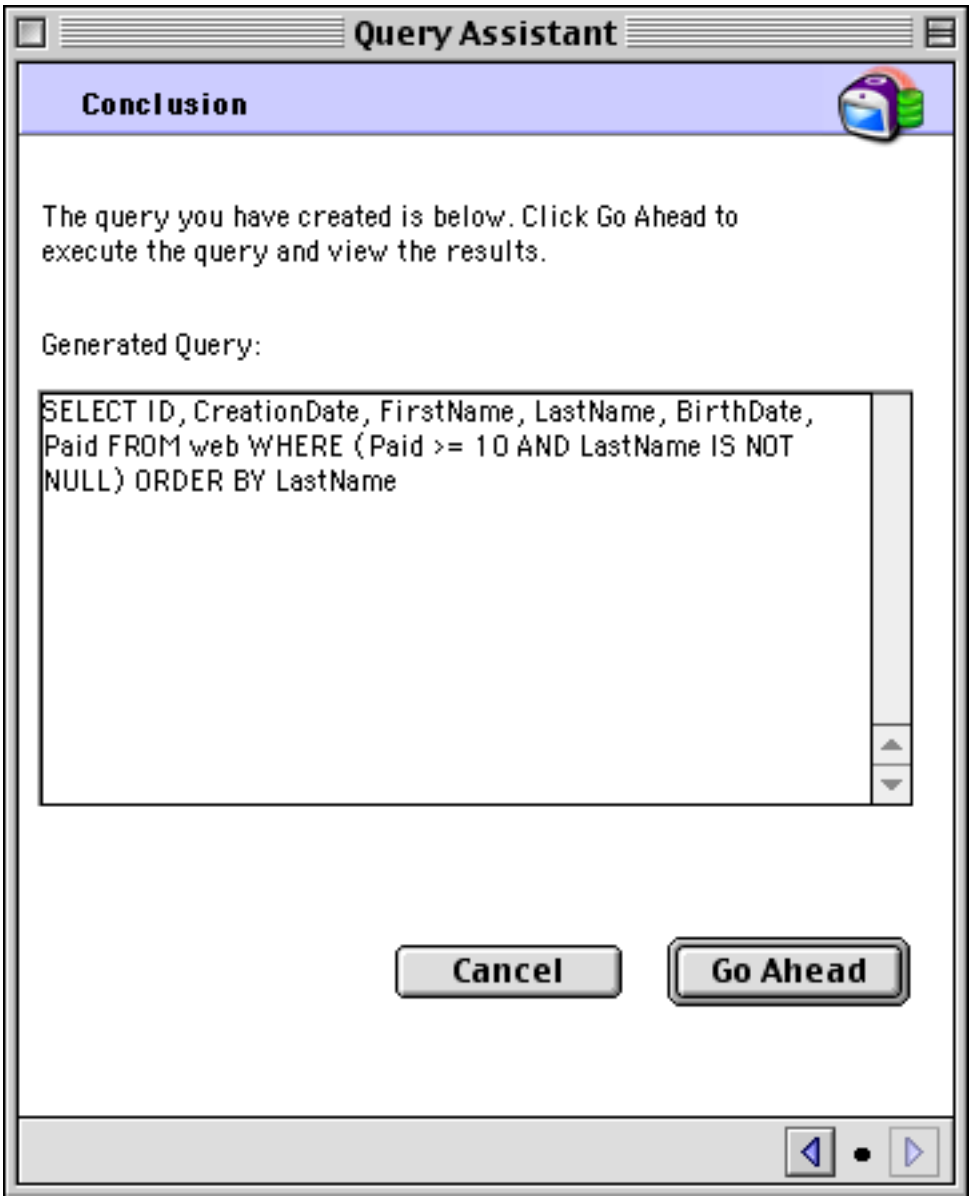
To view the schema for your database, select the "Schema Viewer" tab and click on the triangles next to the database names to reveal the tables available in a particular database. To show the fields for that database double-click on a table name. This will give you the details for each field, its data type, length, and whether nulls are allowed.



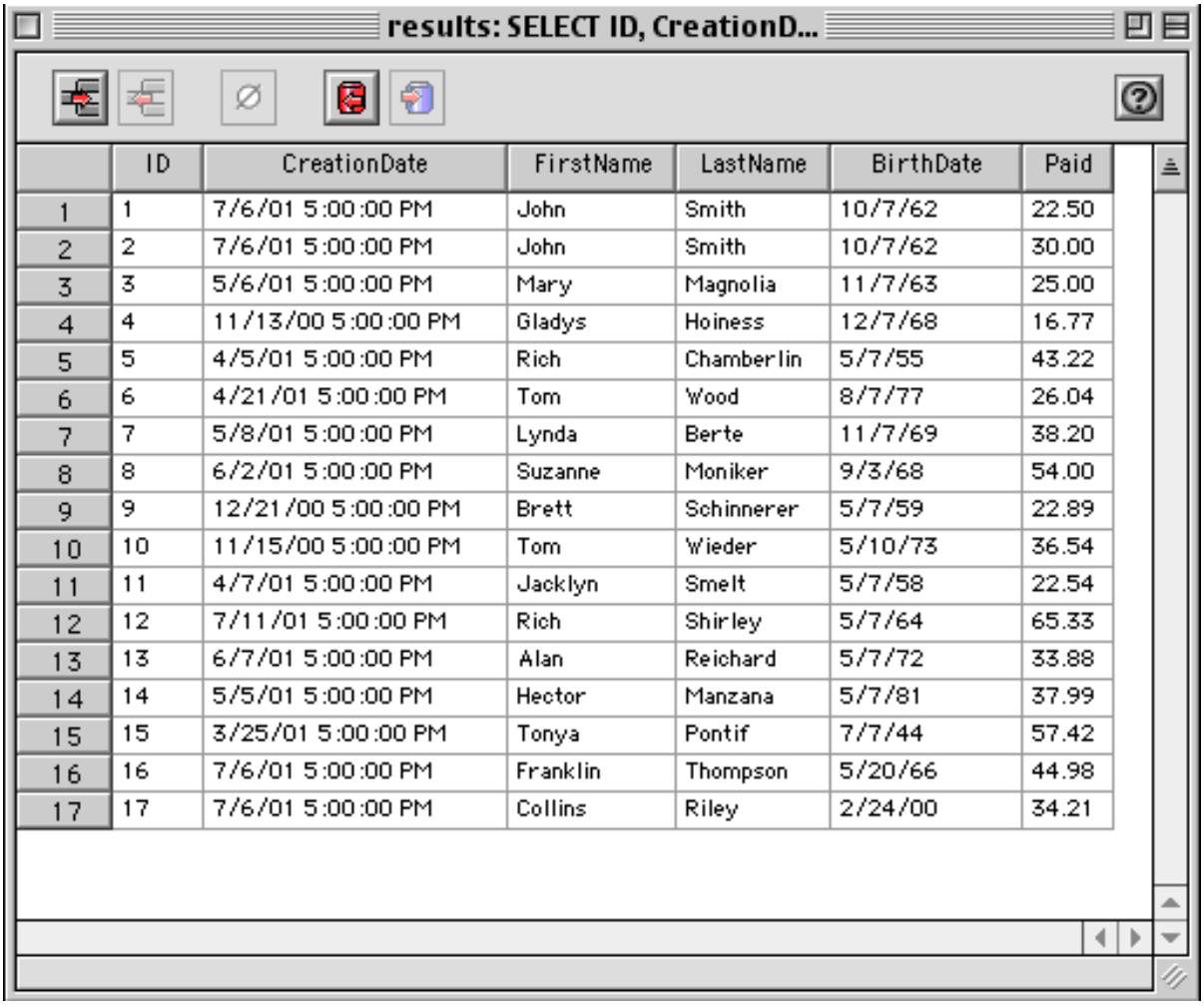
If you return to the main menu you can then select the "Create Table Tool" button and from there create a new table for a database and a series of fields to be included in the table:



There are also a series of Assistants available that work like Wizards and walk you through a series of steps to build the result you are looking for, this includes a Query Assistant, Export Assistant, Import Assistant, and Web Script Assistant. The final step of the “Query Assistant” is to display the query that you can then run to view the results.



To view results in a spreadsheet you can either use the Query Assistant wizard, or check the “Show Results in Spreadsheet” checkbox shown on the “Query Maker” tab and then execute a SQL query for some specified database. An easier way to view the rows from a table is to select a table name on the View Schema tab and then choose Edit Table Data from the Schema menu. The spreadsheet will allow you to view and edit the data found by that query.



results: SELECT ID, CreationD...

	ID	CreationDate	FirstName	LastName	BirthDate	Paid
1	1	7/6/01 5:00:00 PM	John	Smith	10/7/62	22.50
2	2	7/6/01 5:00:00 PM	John	Smith	10/7/62	30.00
3	3	5/6/01 5:00:00 PM	Mary	Magnolia	11/7/63	25.00
4	4	11/13/00 5:00:00 PM	Gladys	Hoiness	12/7/68	16.77
5	5	4/5/01 5:00:00 PM	Rich	Chamberlin	5/7/55	43.22
6	6	4/21/01 5:00:00 PM	Tom	Wood	8/7/77	26.04
7	7	5/8/01 5:00:00 PM	Lynda	Berte	11/7/69	38.20
8	8	6/2/01 5:00:00 PM	Suzanne	Moniker	9/3/68	54.00
9	9	12/21/00 5:00:00 PM	Brett	Schinnerer	5/7/59	22.89
10	10	11/15/00 5:00:00 PM	Tom	Wieder	5/10/73	36.54
11	11	4/7/01 5:00:00 PM	Jacklyn	Smelt	5/7/58	22.54
12	12	7/11/01 5:00:00 PM	Rich	Shirley	5/7/64	65.33
13	13	6/7/01 5:00:00 PM	Alan	Reichard	5/7/72	33.88
14	14	5/5/01 5:00:00 PM	Hector	Manzana	5/7/81	37.99
15	15	3/25/01 5:00:00 PM	Tonya	Pontif	7/7/44	57.42
16	16	7/6/01 5:00:00 PM	Franklin	Thompson	5/20/66	44.98
17	17	7/6/01 5:00:00 PM	Collins	Riley	2/24/00	34.21

While in spreadsheet mode you can make changes to the data displayed and add rows. When your changes are complete, you can then commit the changes so the modifications are made in the database. You also can rollback changes back to when you last executed a commit changes command.

Further instructions for using these tools are included in the MacSQL Monitor Tutorial and Help application.Here is an AppleScript that can be used to grab a selection of text from BBEdit, paste it into MySQL Monitor and then execute it as a query. Obviously, this will only work if the selected text is a valid SQL query. In addition, the script assumes that you have a connection open and a database selected:

```
tell application “BBEdit 6.1”
        set x to contents of selection of front window
end tell
tell application “MacSQL Monitor”
```

```
        activate
        set query text of front window to x
        execute current query of front window
end tell
```

Locate the compiled AppleScript in the BBEdit scripts folder in the OS path (on Mac OS 9):

```
BBEdit 6.1 Folder:BBEdit 6.0:BBEdit Support:Scripts:
```

It will then become available on the BBEdit menu under the AppleScript icon. To assign a key command to the script, choose "Scripts List" from under the AppleScript icon menu item, then select the script you want a key command for from the list in the floating window. Click Set Key to choose a keyboard shortcut for a script.

Lasso MySQL Specific Tags

Lasso 5 offers several tags that are unique to Lasso MySQL. The first two are used to indicate whether the data source of a specified database name is from MySQL or Lasso MySQL:

- [Lasso_DatasourceIsLassoMySQL] Returns True if a database is hosted by Lasso MySQL. The name of a database is used as a parameter to this tag.

- [Lasso_DatasourceIsMySQL] Returns True if a database is hosted by MySQL. The name of a database is used as a parameter to this tag.

Here is an example of how to iterate through all available data sources to display the names of all databases and the tables found in each database:

```
[Database_Names]
[If:(Lasso_DatasourceIsLassoMySQL:(Database_NameItem))==true]

  <p><b>[Database_NameItem]</b>

  [Database_TableNames]

    <br>[Database_TableNameItem]

  [/Database_TableNames]
  <hr>
[/If]
[/Database_Names]
```

Various administrative scripts can be created using these tags to perform actions on the database and tables listed. Some examples of these can be found in the Lasso 5 Setup Guide.

Database Tags

The tags that start with "Database_" are custom tags that help to complete SQL queries based on the parameters supplied in the tag usage. These tags are shortcuts that can be used to execute SQL commands. These tags will come in handy for administration purposes and to script the creation or removal of databases, tables, and fields – for example, to create an initialization script for a solution in which all required databases are automatically created. Following is a brief description of each tag.

[Database_CreateTable] - Creates a table with one field. This one field is named "ID" and is an auto-incremented field that can be used as the primary key for the database.

USAGE:

```
[Database_CreateTable:
        -Database='MyDatabaseName',
        -Table='MyTableName']
```

This tag is equivalent to the following SQL statement:

```
CREATE TABLE MyTableName (id INT PRIMARY KEY AUTO_INCREMENT);
```

[Database_CreateField] - Creates a field in a table.

USAGE:

```
[Database_CreateField:
        -Database='MyDatabaseName',
        -Table='MyTableName',
        -Field='MyFieldName',
        -Type='MyType']
```

The following are optional:

```
        -Default='MyDefaultValue',
        -AutoIncrement,
        -Key,
        -Null,
        -NotNull,
        -AfterField='AnotherFieldName'
        -BeforeFirst
```

This tag is equivalent to the following SQL statement:

```
ALTER TABLE MyTableName ADD COLUMN <details_go_here>;
```

[Database_ChangeField] - Changes a field definition.

USAGE: Same as for [Database_CreateField], with the addition of one parameter:
-Original - specifies the field to be altered.

This tag is equivalent to the following SQL statement:

```
ALTER TABLE MyTableName CHANGE COLUMN MyTableName <details_go_here>;
```

[Database_RemoveTable] - Drops a table from a database. All data in the table will be lost.

USAGE:

```
[Database_RemoveTable:
        -Database='MyDatabaseName',
        -Table='MyTableName']
```

This tag is equivalent to the following SQL statement:

```
DROP TABLE IF EXISTS MyTableName;
```

[Database_RemoveField] - Drops a field from a table. All data in the field will be lost.

USAGE:

```
[Database_RemoveField:
        -Database='MyDatabaseName',
        -Table='MyTableName',
        -Field='MyFieldName']
```

This tag is equivalent to the following SQL statement:

```
ALTER TABLE MyTableName DROP COLUMN MyFieldName;
```

Note there is not a tag that will allow you to create or drop (delete) entire databases.

Inline Tag

Lasso 5 replaces the SQL_Inline tag with the "SQL" command tag that can be used in an inline (only). It is not required to use a SQL inline for Lasso to exchange data with a SQL database. Other actions can be used, however, there are times when a SQL statement will be able to utilize functions or commands not found in LDML. The SQL statement may also be able to accomplish a more complicated request from a database. One example of this is to allow for field level logical operators, i.e. search for this AND that OR the other. Find all Doctors with the last name Perez OR any Doctor that has practiced medicine in Puerto Rico AND went to school at Penn State.

To create a SQL inline, indicate the database to be used with the –Database command tag and pair a SQL statement with the –SQL command tag as follows:

```
[Inline: -Database='MyDatabaseName', -SQL='MySQLStatement']
```

When this inline is processed, Lasso sends the SQL statement to the data source for the indicated database. A few Lasso tags can be used in an inline with a SQL action. MaxRecords and SkipRecords determine how the results are to be displayed. Token tags could also be utilized. Other tags within a SQL inline will have no effect, for example, SortField, SortOrder, ReturnField, username, password, and so forth. This is because values from any other tag are not made part of the specified statement. In other words, Lasso does not change the specified statement, it is simply sends it on to the intended data source.

The results of a query to MySQL are always returned as rows and columns so the following code can be used with any SQL Statement to format the result:

```
[Inline: -Database='MCExample', -SQL='SELECT * FROM web', -MaxRecords=5]

[If:(Error_CurrentError)==(Error_NoError)]
    <table border="1" width="100%" cellspacing="0" cellpadding="2">
    <tr>
        [Loop:(Field_Name:-Count)]
            <td><b>[Field_Name:(Loop_Count)]</b></td>
        [/Loop]
    </tr>
    [records]
    <tr>
        [Loop:(Field_Name:-Count)]
            <td>[Field:(Field_Name:(Loop_Count))]</td>
        [/Loop]
    </tr>
    [/records]
    </table>
[else]
    <p>[Error_CurrentError:-ErrorCode]/[Error_CurrentError]</p>
[/If]

[/Inline]
```

Here is a sample result from using the above to search the MCExample database:

ID	CreationDate	FirstName	LastName	BirthDate	PaidAmount	PaidFlag
1	2000-11-14 12:12:13	Gladys	Hoiness	1968-12-08	16.77	0
2	2000-11-16 04:15:02	Tom	Wieder	1973-05-11	36.54	0
3	2000-12-22 06:32:03	Brett	Schinnerer	1959-05-08	22.89	0
4	2001-02-27 05:57:04	Jake	Barnsworth	1959-11-17	5.02	0
5	2001-03-26 08:38:05	Tonya	Pontif	1979-04-13	57.42	0

Preparing for a Transition from FileMaker to SQL

Suffice to say, as you start out you could implement your database schema much as you would with your FileMaker Pro Web database now, with the same fields*. A good place to start is to simply move the fields in each FileMaker Pro database to a table in a new MySQL database. The primary difference is that instead of a solution consisting of many databases it consists of many tables within one database. In other words, a FileMaker database corresponds to a table in MySQL. The other difference is that FileMaker keeps track of the key field for a database transparently in the record ID. You will have to create your own ID field to use as a key field and use that to uniquely identify records.

One of the biggest issues in converting from FileMaker Pro to MySQL is to create a database schema appropriate for a SQL database. There is a lot to gain by making sure you follow some of the basic principals of database design and normalize the tables by removing redundant or duplicate data. This is usually accomplishing by taking databases with large numbers of fields and decomposing them into smaller tables, all the while maintaining key values that relate the databases together. The art and science involved in designing your databases can be learned over time. For now, start with the principals you may have learned in creating FileMaker Pro databases.

If you plan to migrate an existing solution to MySQL some time in the future, or want to try and gradually make changes to your format files for eventual changeover, there are things you can pay attention to in order to be better prepared. When later you are ready to move to MySQL (or some other SQL based database) there will be fewer changes to be made. Here are some things to keep in mind for that move to MySQL:

- Avoid performing calculations in FileMaker Pro fields. Allow Lasso or JavaScript to calculate values.
- Replace any FileMaker scripts with Lasso based routines.
- Limit the use of techniques that are unique to FileMaker Pro or functionality not found in SQL databases. For example, repeating fields, global fields, lookup fields, or portals.

Nevertheless, most of the same Lasso tags that are used with a FileMaker Pro data source can be used with MySQL. You add, update, delete, and search for records with the same action and other commands tags. Fields and form parameters can be substituted in the same manner. Below are some further details on steps you can take now or when you are migrating to a MySQL database.

- Replace portals by surrounding the portal tags with an inline that searches on the related database. Then change the portal tags to record tags and remove the name of the relationship (and the double colons that follow) from all field references.

- Remove all repeating fields. A repeating field will require a change in your data

structure. You will need to replace the repeating field with a related table.

- A Lasso search on fields in a FileMaker Pro database will have a different behavior than for a SQL database. A FileMaker Pro search will match based on the word, while SQL matches based on the entire field contents.

- Stop using the LDML recid (or recordid) or recid_value (recordid_value) tags. Instead, use KeyField, KeyValue, KeyField_Name, KeyField_Value. KeyValue is a synonym for recid, while KeyField_Value is a synonym for recid_value. In addition, when using FileMaker, KeyField is ignored, it can be specified as nothing or set to (KeyField_Name) as is shown below:

```
[Inline:
   -Database=(Database_Name),
   -Layout=(Layout_Name),
   -KeyField=(KeyField_Name),
   -KeyValue=(KeyField_Value),
      … etc …
   -Update]

[/Inline]
```

- Pay particular attention to data types when adding or retrieving dates from a database. Format date fields when you retrieve or submit dates using the format YYYY-MM-DD, for example 2001-06-09. You can even begin to use this same format to store dates in a FileMaker Pro text type field. The date tags provided by Lasso allow you to accomplish this requirement. To retrieve a field from MySQL you use the following:

```
[Date:(Field:'MyDate'),-DateFormat='%Y-%m-%d %H:%M:%S']
```

Moreover, to submit a date to MySQL you will need to make sure to format it as follows:.

```
[Date_Format:(Var:'MyDate'),-DateFormat='%Y-%m-%d %H:%M:%S']
```

The [Server_Date:] tag with the –Extended parameter will also give you a date in the correct format (for MySQL DATE type fields). –Extended is new to Lasso 5. It appears as:

```
[Server_Date:-Extended]
```

This list is sure to grow as I begin to migrate FileMaker Pro-based solutions to MySQL. Stay tuned to http://www.lassodev.com/ for more details.

* Disclaimer: It is more likely to be easier to convert a FileMaker Pro database that is already being used on the Web with Lasso to a MySQL version that uses a similar data schema. As opposed to converting an existing FileMaker Pro solution that is only being used on the LAN. The reasoning behind this is that in the latter case, the databases may have more complexities such as a dependency on scripts, relationships, lookup fields, repeating fields, portals, and the like.

Inserting a New Record

In SQL, an add action will not show the current field values within the response since the cursor to that record is lost after a record is inserted. This is contrary to how add actions work in FileMaker where all field values are available on the response to an add action. To work around this you can either not show the field values or retrieve a reference to the record that was just added.

One option is to simply populate the response page with the action_param (aka form_param) values as follows:

```
[Inline:
  -Database='Example',
  -Table='web',
  -KeyField='ID',
  'name'=(action_param:'name'),
  'address'=(action_param:'address'),
  'serialNumber'=(action_param:'serialNumber'),
  -Add]
  [If:(Error_CurrentError)==(Error_NoError)]
    [var:'result'=true]
  [else]
    [var:'result'=false]
  [/If]
[/inline]
[If:(var:'result')==true]
  <p><b>Your record was added:</b></p>
  <p>Name: [action_param:'name']<br>
  Address: [action_param:'address']<br>
  Serial Number: [action_param:'serialNumber']</p>
[else]
  <p>Record Not Added Message</p>
[/If]
```

You can also nest a search inline inside of the add action to search for the identifier for that new record. This requires that one use the -KeyField command in the add action. The [KeyField_Value] tag returns the value for the specified field and allows you to find the record that was just added. For example:

```
[Inline:
  -Database='Example',
  -Table='web',
  -KeyField='ID',
  'name'=(action_param:'name'),
  'address'=(action_param:'address'),
  'serialNumber'=(action_param:'serialNumber'),
  -add]
  [If:(Error_CurrentError)==(Error_NoError)]
    [Inline:
      -Database='Example',
      -Table='web',
      -KeyField='ID',
      -KeyValue=(KeyField_Value),
```

```
        -MaxRecords=1,
        -search]
       <p><b>Your record was added:</b></p>
       <p>Name: [field:'name']<br>
       Address: [field:'address']<br>
       Serial Number: [field:'serialNumber']</p>
     [/inline]
   [else]
     <p>Record Not Added Message</p>
   [/If]
[/inline]
```

The AUTO_INCREMENT property can be defined for a key field to generate a sequential number when a new field is added. This allows for all records to be unique. The MySQL last_insert_ID() function returns the AUTO_INCREMENT value that was most recently generated after a record is added to a table (and 0 if no such value has been generated). After an add action, Lasso 5 automatically retrieves the new value for an auto-incremented field using the last_insert_ID() function and returns the value in the [RecordID_Value] tag. For example, [RecordID_Value] will return 25 if the record you created is the twenty-fifth to be created and your table contains an auto-incremented field. The latest auto-incremented value which is retrieved in the RecordID_Value tag can be used as follows:

```
[Inline:
  -Database='Example',
  -Table='web',
  'name'=(action_param:'name'),
  'address'=(action_param:'address'),
  'serialNumber'=(action_param:'serialNumber'),
  -Add]
  [If:(Error_CurrentError)==(Error_NoError)]
    [Inline:
      -Database='Example',
      -Table='web',
      -KeyField='ID',
      -KeyValue=(RecordID_Value),
      -MaxRecords=1,
      -Search]
     <p><b>Your record was added:</b></p>
     <p>Name: [field:'name']<br>
     Address: [field:'address']<br>
     Serial Number: [field:'serialNumber']</p>
    [/inline]
  [else]
    <p>Record Not Added Message</p>
  [/If]
[/inline]
```

The [RecordID_Value] tag is thus used to find a record that was just added, in order to display the current contents of that record. Note that in the case of FileMaker Pro, the value returned by the [KeyField_Value] tag is equal to that for [RecordID_Value], but this is not necessarily the case for SQL databases since the KeyField can differ from a field that has the AUTO_INCREMENT property defined for it.

Moving Data from FileMaker to MySQL

When you are ready to move your data to MySQL, you have several options on how to accomplish this. You can export all of the data to a delimited file and then import into MySQL or use Lasso to find the data in one database, and create records in the new database as you iterate through the records. Here is an example of that technique:

```
[Inline:
  -Database='MCExample.fp3',
  -Layout='Web',
  -MaxRecords='all',
  -FindAll]
  [Records]
    [var:
      'Created'=(Date_Format:(field:'Created'), -DateFormat='%Y-%m-%d %H:%M:%S'),
      'Birth_date'=(Date_Format:(field:'Birth_date'),-DateFormat='%Y-%m-%d'),
      'First_Name'=(field:'First_Name'),
      'Last_Name'=(field:'Last_Name')]
    [$First_Name->Trim]
    [$Last_Name->Trim]
    [Inline:
      -Database='MCExample',
      -Table='Import_Web',
      -KeyField='ID',
      'First_Name'=(var:'First_Name'),
      'Last_Name'=(var:'Last_Name'),
      'Creation_date'=(var:'Creation_date'),
      'Birth_date'=(var:'Birth_date'),
      -Add]
    [/Inline]
  [/Records]
[/Inline]
```

The outer inline finds all records in the FileMaker database. Then for each record, fields are formatted as needed and then a record is added to the SQL database. The real advantage for importing data using an inline tag is to be able to perform data transformation and formatting as the field data is inserted into MySQL rows.

References

Here is a list of references that may be useful to you as you venture into the world of MySQL.

Books

MySQL
Author: Paul DuBois
ISBN: 0735709211
Published: Dec. 1999, New Riders Publishing
Pages: 800
http://www.newriders.com/books/title.cfm?isbn=0735709211
http://www.bookpool.com/.x/SSSSSSCOLL/sm/0735709211

MySQL Manual
Available to download as a PDF or HTML document. Also includes a user comment online version.
http://www.mysql.com/documentation/

SQL in a Nutshell
Author: Kevin Kline with Daniel Kline, Ph.D.
ISBN: 1-56592-744-3
Published: December 2000, O'Reilly & Associates
224 pages
http://www.oreilly.com/catalog/sqlnut/
http://www.bookpool.com/.x/SSSSSSCOLL/sm/1565927443

Linux In a Nutshell, 3rd Edition
Author: Ellen Siever, Stephen Spainhour, Jessica P. Hekman, & Stephen Figgins
ISBN: 0596000251
Published: August 2000, O'Reilly & Associates
http://www.oreilly.com/catalog/linuxnut3/
http://www.bookpool.com/.x/SSSSSSCOLL/sm/0596000251

Unix System Administration Handbook, Third Edition
Author: Evi Nemeth, Garth Snyder, Trent R. Hein, et al.
ISBN: 0130206016
Published: September 2000, Prentice Hall, Paperback, 3rd edition
853 pages
http://www.bookpool.com/.x/SSSSSSCOLL/sm/0130206016

Database Design for Mere Mortals: A Hands-On Guide to Relational Database Design
Author: Michael J. Hernandez
ISBN: 0201694719
Published: December 1996, Addison-Wesley
440 pages
http://www.bookpool.com/.x/SSSSSSCOLL/sm/0201694719

Software

MacSQL Monitor 2.0
Runtime Labs
http://www.rtlabs.com/

SQL Language Module for BBEdit
Runtime Labs
http://www.rtlabs.com/

SQL4X Manager Basic 1.0
MacOS Guru
http://www.macosguru.de/

NiftyTelnet 1.1 SSHr3
Chris Newman
http://andrew2.andrew.cmu.edu/dist/niftytelnet.html

phpMyAdmin 2.1.0
Phpwizard.net
http://www.phpwizard.net/projects/phpMyAdmin/

Articles/ Web Sites

MySQL
http://www.mysql.com

Introduction to Structured Query Language
Author: James Hoffman.
http://w3.one.net/~jhoffman/sqltut.htm

How to Move Your Data from FileMaker Pro on the Macintosh to MySQL on Mac OS X Server
Author: Gavin Clark
http://www.clark-ip.com/docs/FMP_to_mySQL_OSX.html

MySQL Prebuilt Binary for Mac OSX
Author: Gavin Clark
http://miltonshole.thespacemonkey.com/osx/mysql.html

Mac OS X Packages
Author: Marc Liyanage
http://www.entropy.ch/software/macosx/

Enabling MySQL in Mac OS X Server (10.0.4)
Author: Patrick Larkin
http://patricklarkin.com/macosxserver/osxs-mysql-fubar.php

Mac OS X for Web Developers
Author: Wincent Colaiuta
http://hotwired.lycos.com/webmonkey/01/16/index3a.html

Beginning MySQL Tutorial
Author: W.J. Gilmore
http://www.devshed.com/Server_Side/MySQL/Intro/

MySQL Administration
Author: W.J. Gilmore
http://www.devshed.com/Server_Side/MySQL/Administration/

The MySQL Grant Tables
Author: W.J. Gilmore
http://www.devshed.com/Server_Side/MySQL/Grant_Tables/page1.html

Many Web developers prefer MySQL (7/23/01)
Author: Susan Sales Harkins and Martin W. P. Reid
http://builder.cnet.com/webbuilding/0-7537-8-6580620-1.html?tag=sd

MySQL and PostgreSQL Compared
Author: Tim Perdue
http://www.phpbuilder.com/columns/tim20000705.php3

Starmark

EASY, NO SWEAT WEB ADMIN WITH WEB SITE PUBLISHER FOR LASSO™!

* FREE FOR SUMMIT ATTENDEES ONLY*
BULLETIN BOARDS OR ROTATING ADS – $99 VALUES – FREE WHEN YOU PURCHASE WEB SITE PUBLISHER!

HOW COOL IS THIS!

web site publisher for Lasso™

Publishing custom designed web sites quickly and easily is cool. Add a single, Web-based admin for multiple sites and some time-saving modules, and you're talking way cool. That's what happens when you – or your customers use the cutting-edge **Web Site Publisher (WSP) for Lasso.**

WSP is the tightly integrated package that quickly adds, updates and deletes pages from your sites via a secure online administration back office. And it automatically updates navigation to include new sections and pages, and hides sections with no pages.

And now with the purchase of Web Site Publisher, Unlimited Edition at $499,
Summit attendees get Bulletin Boards for Lasso or Rotating Ads for Lasso FREE!

rotating ads for Lasso™

Rotating Ads for Lasso™ – an integrated WSP module or standalone product – lets you quickly add and update banner ads on many Web sites with the single line of code provided. The powerful Web-based administration module tracks all of this information from your Web browser.

FREE with WSP or available for a cool $99.

bulletin boards for Lasso™

Bulletin Boards for Lasso™ – also an integrated WSP module or standalone product – is delivered as an [include] page providing bulletin board functionality for multiple sites and multiple pages within a site.

FREE with WSP or available for a cool $99.

To order any of these really cool solutions – or for more information – please go to www.starmarkintl.com/lasso, enter your Discount Code: "SUMMIT2001", and indicate which product you would like for FREE!

Starmark, developers of Web Site Publisher, Rotating Ads and Bulletin Boards for Lasso, has been developing commercial solutions for Lasso since 1997. All of our solutions have powerful online administration tools and are tightly integrated with each other to comprise an entire suite of solutions for the Lasso community.

©2001 Starmark International. Starmark, the Starmark logo, Web Site Publisher for Lasso, Rotating Ads for Lasso and Bulletin Boards for Lasso are trademarks of Starmark International. All Rights Reserved.

www.starmarkintl.com/lasso

Session Management with Lasso 5

Table of Contents

Session Management with Lasso 5

by Johan Sölve

Each time a visitor on a web site requests another page, he is like a complete stranger to the web server. Everything about the visitor's state is forgotten after each page has been served, and the visitor has to "introduce" himself at each page request. Session Management enables the web server to remember the visitor to keep track of what he is doing and maintain the visitor's state from page to page.

In previous versions of Lasso Web Data Engine the developer had to implement session management himself. Johan will demonstrate how the new built-in session management system in Lasso Professional 5 helps developers create professional and secure dynamic solutions with very little effort.

Biography

Johan Sölve works as a lead web developer at Montania Solutions in Sweden. Montania develops and sells a line of accounting and business administration software on the Swedish market, and combines that with custom FileMaker Pro and web solutions. Johan has worked with FileMaker Pro development since 1993 and with Lasso since 1996. He is a long time generous contributor to Lasso Talk. Prior to his work at Montania he started up and ran a Macintosh reseller and service business for 5 years. He has a Bachelor's Degree in Innovation Engineering.

Notes

Session Management with Lasso 5

Introduction

What Is Session Management?

To describe what session management is, let's begin with looking at how FTP works. When connecting to a server via FTP you remain connected to the server all the time. That way the server can keep track of each user and what users are doing or in what directory users are in every moment. Your FTP client doesn't disconnect until you finish the session.

It's like a phone call. You place a call and talk to the person, and throughout the entire call your phones remain connected to each other, and you don't have to introduce yourself all the time since the ther person knows who he's talking to. When you finish your talk session, you hang up the phone and the line disconnects.
The downside is that while you two are talking with each other, both of your phone lines are busy.

You Are A Stranger On the Web

The web is context-free or stateless, and individual page requests aren't automatically related to each other. Every time a browser requests a web page, the browser connects to the server via the HTTP protocol, asks for the page, the server deliveres the requested page and then the browser close the connection. When you click on a link to go to the next page the same thing repeats itself. The browser sends a request, the server sends a response and then the browser disconnects.

Each page load is a new connection and technically a new session, and once the page is delivered the connection is terminated and the server forgets everything about that visitor. When the visitor requests another web page, the server thinks it is a new visitor that has never been there before so the server needs a new introduction.

This is more like talking over radio intercom. You call the other person by saying his call signal, and when you've said your thing you say "over", release the talk button ("disconnect"), waiting for a reply. Then the other person push his talk button, maybe says his and your call signal to verify the connection, says his thing and then disconnects. From a technical point of view there is nothing that keeps your talk session together, and unless one of you say "over and out", nobody can tell if your talk session has ended or not. On the other hand, the line is available all the time so if someone else wants to talk with you, nothing (except perhaps good manners and radio etiquette) prevents him for breaking into the chat.

Think of a web shop without any kind of session management. You would have to remember or write down the items you wanted to order, and then go to a page with a blank order form where you would enter the items when you're done shopping.

With session management we can store the items on the server as the user browses the shop, and when you're done shopping the server knows what state your shopping cart is in so you only have to fill in payment and shipping details.

So in order to maintain a visitor's state between pages of the site we have to find a way to mimic a persistent session. We need a way to maintain some visitor specific data during his visit at the site, or during the session.

Don't Store Data On the Client

One way we could maintain the state between pages is to simply pass the data from page to page, either as hidden form fields or URL parameters, or as browser cookies. We give the client browser the responsibility to store the data for us.
But storing data on the client side is bad for several reasons.

- It breaks a security rule for web development: "Data coming from the web should not be trusted"
- We loose control of the data. It can be manipulated by the client
- The data is sent in the clear on every request and every page response.
- A hacker can snoop and spoof the data.
- If the client doesn't come back, we loose the data and can't access it if we wanted to.
- It gets difficult to manage because all data must sent from page to page, leading to lots of hidden fields in forms, or very long URLs. The storage is limited.
- If we use cookies to store data on the client browser we face the problem that some clients won't accept cookies for one or other reason.

The Session ID

We should store the session data on the server. Since we can have several visitors at the same time, we use a session identification number to identify each visitor and associate the visitor with his specific data. This session identification is just a pointer but it must be stored on the client side, so it should be a number that is difficult to crack and doesn't carry any information by itself. As we concluded earlier, data coming from the web should not be trusted. By using a very long random number we can make sure of three things:

- The session ID doesn't carry any information other then the session ID itself
- If someone tries to hijack someone else's existing session by guessing a session ID, it

will be extremely unlikely that they will find another existing session.
- If someone increments an existing session ID by just changing the last digit, they won't find another existing session.

It's not only human manipulation we should protect it against, it's also the computers that human crackers tend to use to break in.

But we still can't make sure that someone hasn't captured an existing session ID by "listening" to a visitor's network traffic. To prevent this, we have to use a secure connection (SSL).

Why Not Use the Client's IP Number as Session Identification?

An IP number is simply not necessarily associated with a specific user. The user can sit behind a proxy server or behind a NAT router that shares one IP number, making everyone on his local network appear as if they have the same IP.
He can sit on a dialup modem connection, disconnect his dialup and connect again a little bit later, giving him a new IP address. Another dialup user can get his old IP number a moment later.

Lasso Variables

When setting a variable in Lasso, it is available on the entire page that Lasso is processing. But the moment the page is delivered to the visitor, Lasso variables are cleaned up and forgotten. They will not be available again when the visitor asks for the next page.
If Lasso variables could be globally available somehow, we would come a long way towards session management.

Uses for Session Management - Examples

- User authentication - Store the user ID in the session
- Shopping cart - store the ordered items in the session until the order is submitted
- Multi-page entry form - data is held in the session until all data is commited
- Track a visitor trough a site to see what path a visitor takes through a sitre and how much time he spends on each page
- Give the user a history menu with for example the latest 10 viewed products or news items
- Maintain the state of the user interface (sort order, search request, navigation)
- User preferences (page color etc)

How Does Session Management Work?

Store Data On the Server

We can store the session data in a database, in text files or in memory, it doesn't really matter. The point is that we store the data on the server side.
Each visitor is assigned a unique session ID number. The session ID is used as a key to look up the session data in the session database.

Pass the Session ID From Page to Page

In a cookie
This gives completely clean URLs, for example
http://www.domain.com/page.lasso

+ Simple and reliable
+ Remains available even if user navigates out of the site and comes back
+ Invisible, doesn't interfere with URLs and links
- Not all clients can use them
- Sometimes incompatible

As parameter in a form or URL
A parameter is added to every URL, for example
http://www.domain.com/page.lasso?session=1f1c341fe2b89ac346709e

+ Works with all browsers
- Difficult to implement (the session ID must be added to all links and forms within the site)
- Session ID will show in the location bar and on printouts, and is stored with bookmarks
- Can prevent a page from being indexed on some search engines
- Having a session ID parameter in the URL reduces usability since it makes it harder for the user to manually edit the URL.

We can't always rely only on cookies. A user may have turned off cookies in his browser for whatever reason, or may be using a primitive browser that doesn't even support cookies. The client's system clock can be wrong which can lead to cookies expiring immediately, or the client browser might be incompatible with our server's cookies (like some version of Netscape 6). Some sources claim that 20% of web users don't accept cookies.

Depending on the solution and the target audience it may be advisable to use a fallback of sending the session ID as a hidden form field or URL parameter if cookies aren't available.

There are also other ways to propagate the session ID between pages. For example as part of the path, or part of the hostname. When a session is started, the visitor is redirected to another page with the session ID either in the path or in the hostname.
This gives URLs like

http://www.domain.com/1f1c341fe2b89ac346709e/page.lasso

or

http://1f1c341fe2b89ac346709e.www.domain.com/page.lasso

+ Works with all browsers
+ Doesn't interfere with search engines
+ Sending the session ID is automatic and doesn't require changing any relative links or forms (with hostname it also works with absolute URLs)
- Technically difficult to implement, takes some tweaking of the server (virtual hosting, DNS configuration, request modification)
- Session ID will show in the location bar and on printouts, and is stored with bookmarks
- Less user-friendly URLs

Both these methods takes some extra tweaking of the web server so they are not very easy to implement the first time, but once it is set up they are very easy to use.

Session Expiration and Garbage Collection

As with the radio intercom chat, we don't know for sure when a user ends his session since a user normally doesn't "hang up" when leaving a web site. When the server has delivered a page to the visitor, we don't have a clue about what the visitor is doing, not until he makes the next page request from our server. If he does. We can't tell if he has left our site to surf on to another site, if he's just having a cup of coffee or if he's actually reading something interesting he found on our site.

We have to make an assumption about when a session ends, based on how long a session has been inactive. If a user hasn't requested another page for a certain time, we can assume that the session has ended and should expire.
If a visitor returns with an old session ID, he should be assigned a new session ID instead. That old session ID could come from a cookie in his browser, from a bookmark, or from a link that another visitor sent to him.

Old expired sessions must be deleted from the server to not fill up the server's storage. This is called garbage collection and should be performed by the server at a regular interval.

How To Use Sessions in Lasso 5

So far we have only talked generally about session management. Time to get real with Lasso!

History - In Lasso 3

Lasso 3 has no built in session management
Every aspect of session management had to be implemented manually - generate session ID, store session data, propagate session ID between all pages of the session, make data available on every page of the session, keep track of when a session was last used, handle session expiration and garbage collection.

Lasso could generate a unique session ID for you, but that was it.
Third party LJAPI modules have been developed to provide "global variables" (Detlef Beyer's StaticVar and Stephen Schultz' ESGlobalVariables).
Different Lasso solutions has been made available to help implement session management, but it's still a lot of effort.

Overview

Session management in Lasso 5 is very easy to use. Blue World has done a good job with this. The main principle is very simple: In Lasso 5 you can make variables global so they remember their values between pages for a specific visitor.

Global Variables

Normally, Lasso variables are available throughout an entire page, but they are forgotten between pages.
When using session management, you tell Lasso to remember some variables. When one page has finished processing, Lasso stores the variable values in the internal session database, and when the next page begins processing Lasso restores those variable values so they are available on the next page for that user's visit. In other words, you "link" variable names to a session.

Setting Up Session Management With Lasso 5

To make Lasso remember variables on a page, you initiate a session and link variable names to that session.

1. Initiate a session
`[Session_Start: -name='user_data']` starts a new session.
If the visitor already has a valid session ID for this named session, the existing session is loaded instead.

A new session is started if there is no session ID specified or if the session has expired. Every page that should be able to access that session must have [session_start] in it. After [Session_Start] has loaded an existing session, all saved variable values are available on the page below the [Session_Start] tag.

2. Link variable names to the session
[Session_AddVariable: -name='user_data', 'name', 'color'] tells Lasso to remember the values of the variables 'name' and 'color' when the page has been processed. Linking variables with [Session_AddVariable] only needs to be done on the first page of the session, although it doesn't hurt to use it on all pages of the session.

You can link plain variables to a session, but also the new complex data structures of Lasso 5 such as arrays, maps and pairs.

From now on, the variables you have linked to this session are remembered and accessible on all pages that have [Session_Start: -name='user_data'] with the same session name.

More About the Session Tags

[Session_Start: -name='user_data']
The -Name parameter is required. It lets you use different sessions for the same page. If nothing else is specified, the session ID will be passed from page to page in a cookie and the session will expire after 15 minutes of inactivity. This is good for most uses.
-Expires is an optional parameter that tells Lasso how long time in minutes the session can remain unused before it is expired. The default value is 15 minutes.
-UseCookie is optional. It tells Lasso to pass the session ID between pages using a cookie. This is the default behavior unless -UseLink is specified.
-UseLink is also optional. If you specify -UseLink, Lasso will pass the session ID as parameter in URLs. This is useful if you want to support clients that don't support cookies.

Lasso automatically adds a session ID parameter to all href links that are local, i.e. don't begin with "http://". Note that you still have to add the session ID parameter manually to other kinds of links, for example <FRAME SRC=""> and input forms, so it takes some extra attention and work to use -UseLink.

To end a session, use [Session_End: -name='user_data']. This will delete all data for this session, and it will delete the session cookie if there is one.

There are other session tags as well, please refer to the Lasso 5 Language Guide for more information about the Session tags.

How Sessions Are Stored In Lasso 5

Lasso uses the built-in MySQL database to store session data. If you look at the sessions table with Lasso Instant HTML Publisher, you will find that it looks like the “data” field is empty. This is normal, and it is because the data is stored in a special format in a binary field, which can’t be viewed in Instant Publisher.

The session database can store something like 64 kB of data in each session record. Every named session for every visitor is a separate record, but all variables added to the session are stored together in the same record
If you need to store much data in the session, store it in your own database instead and just put a key value in the sessions database.

Garbage Collection In Lasso 5

Sessions are not deleted from the database exactly when they expire. This is to save processing on the server. Every time a session tag is used there is a certain probability (something like a 1/5 chance) that Lasso will clean up old session data. This means that you can’t rely on session data disappearing from the sessions database exactly when they expire. This doesn’t matter anything since data from an expired session will never be used anyway. If a visitor comes back “too late” with a session ID that has expired, he will be assigned a new session ID instead.

If you don’t specify an -Expires value for [Session_Start], the session data will expire after 15 minutes of inactivity. This means that the visitor hasn’t loaded another page for that time.

Example 1 (beginner): Remembering User Preferences

Demonstration of code and web page

The first page initiates the session and adds the variable names to the session. The second page is a form to enter the user settings. When saving the form, the first page is called again which takes care of setting the preference values when saving the settings. Example code can be found on the Lasso Summit CD at /Presentations/Solve-Sessions/Examples for LP5~/Example 1/.

```
Default.lasso
-------------------------------------------------------------
[Session_Start: -name='prefs']
[Session_AddVar: -name='prefs', 'name', 'color']

[if: (form_param: 'name')!='']
        [var: 'name' = (form_param: 'name')]
[/if]
```

```
[if: (form_param: 'color')!='']
	[var: 'color' = (form_param: 'color')]
[/if]

<html>
<head>
<title>User Preferences</title>
</head>
<body bgcolor="[var: 'color']">
	<h3>Session example: User Preferences</h3>
	Hello, [var: 'name']
	<br>
	Follow this link <a href="page2.lasso">to the next page</a>
	<br>
	Or <a href="userprefs_form.lasso">change your preferences</a>
	<br>
	<br>
	This is the current session id: [session_id: -name='prefs']
</body>
</html>
```

---

userprefs_form.lasso

```
[Session_Start: -name='prefs']

[if: (var: 'color') == '']
	[var: 'color'='#FFFFFF']
[/if]

<html>
<head>
<title>User Preferences</title>
</head>
<body bgcolor="[var: 'color']">
	<h3>Session example: User Preferences</h3>
	Please enter your name and your preferred background color!
	<table border="0" cellpadding="5">
	<form action="default.lasso" method="POST">
	    <tr>
		<td>
			Name:
		</td>
		<td>
			<input type="text" name="name" value="[var: 'name']">
		</td>
	    </tr>
	    <tr>
		<td>
			Color:
		</td>
```

```
                        <td>
                                <select name="color">
                                        <option value="#FFFFFF"
                                                [if: (var: 'color') == '#FFFFFF']SELECTED[/if]>White
                                        <option value="#FFFF66"
                                                [if: (var: 'color') == '#FFFF66']SELECTED[/if]>Yellow
                                        <option value="#99FF99"
                                                [if: (var: 'color') == '#99FF99']SELECTED[/if]>Green
                                        <option value="#CCCCCC"
                                                [if: (var: 'color') == '#CCCCCC']SELECTED[/if]>Gray
                                </select>
                        </td>
                </tr>
                <tr>
                        <td>
                        </td>
                        <td>
                                <input type="submit" name="" value=" Save ">
                        </td>
                </tr>
        </form>
        </table>
        <br>
        <br>
        This is the current session id: [session_id: -name='prefs']
</body>
</html>
----------------------------------------------------------

page2.lasso
----------------------------------------------------------
[Session_Start: -name='prefs']
<html>
<head>
<title>User Preferences</title>
</head>
<body bgcolor="[var: 'color']">
        <h3>Session example: User Preferences</h3>
        Hello, [var: 'name']
        <br>
        This is page 2, independent of page 1.
        <br>
        <br>
        Back to <a href="default.lasso">the home page</a>
        <br>
        Or <a href="userprefs_form.lasso">change your preferences</a>
        <br>
        <br>
        This is the current session id: [session_id: -name='prefs']
</body>
</html>
----------------------------------------------------------
```

Example 2 (intermediate): User Authentication

Demonstration of code and web page

Example code can be found on the Lasso Summit CD at /Presentations/Solve-Sessions/Examples for LP5~/Example 2/.

Example 3 (advanced): "Site History" Popup Menu

Demonstration of code and web page

Example code can be found on the Lasso Summit CD at /Presentations/Solve-Sessions/Examples for LP5~/Example 3/.

Appendix: Mixed Advanced Topics

This section contains a mix of tips and techniques relating to sessions, aimed at advanced users.

Read Only Session Variables

Any change yoy make to a variable that is linked to a session, is saved at the end of the page.

To prevent a session variable from being changed, copy the session variables over to other "local" varibales, and make changes to these local copies.
[var: 'page_local'=(var: 'page')]
When the page has finished processing, the local variable copies are discarded and the original variables remain unchanged.

Uses of Multiple Sessions

You can have multiple sessions on the same page by specifying different names for them. This can be useful if you want some session data to have a longer lifetime than other data. For example, you can use one cookie-propagated session to store some non-critical user interface settings such as hits per page, font settings or background color. This session could have a very high expiration value so it will normally "always" be there. It could be called -Name='site_prefs'.

Then you can have another session to handle mission-critical data on a shorter term, for example user authentication. This session could be cookie-propagated with a fallback to parameter-propagated (-UseLink) if cookies aren't available to ensure functionality, and could have a fairly short lifetime. It could for example be called -Name='authentication'.

Check For External Referrer

For added security, you can check for external referrer when starting a session. Check if the referrer URL is empty or contains another domain than our own. If the request has an external referer, we should give the visitor a new session regardless of the state of the current session. This can prevent session hijacking and inappropriate linking to our site including the session ID. This is not at all bulletproof though since referers can rather easily be suppressed or forged.

Getting More Control Over Session Parameters

Using your own cookie name and parameter name for the session ID

```
[Session_Start: -name='history', -UseNone]
[cookie_set: 'myownsessioncookie'=(session_ID: -name='history')]
```

On the next page, pass the session ID to the [session_start] tag by using an inline wrapper.

```
[inline: -session=('history:' + (cookie: 'myoiwnsessioncookie'))]
        [Session_Start: -name='history', -UseNone]
[/inline]
```

NOTE: The "-UseNone" parameter is not yet supported in Lasso Professional 5. This entire section about "Getting More Control Over Session Parameters" is unsupported by current versions.

Propagating the Session ID Through the Host Name

One way to pass the session ID between pages without using cookies and without having to modify URLs and links on all pages is to send the session ID with the host name. This requires changing the DNS for the domain, and it also requires a webserver with its own IP number so it can accept arbitrary host names (name-based virtual hosts can't be used with this method, unless the virtual hosts scheme can work with wildcards).

First add a wildcard entry in the DNS for the domain:
*.mysite.com. IN A 123.132.45.12

When a user enters the site at www.mysite.com, he is assigned a session ID and immediately

redirected to http://1f1c341fe2b89ac346709e.www.domain.com/page.lasso

On the following pages of the site, have Lasso parse the "Host" field of the client's request header and extract the first part of it (up to the first period). This is the session ID, which will be passed to [session_start] using an inline wrapper.

This session ID is automatically propagated through all local links (both relative and absolute), since local links get their hostname from the current hostname.

This trick doesn't prevent the session ID from being bookmarked or printed, and it doesn't help usability (still unfriendly URLs), but it saves you from the work of adding the URL to all forms and frame calls.

Propagating the Session ID Through the URL Path

Another way to propagate the session ID is to pass it as a part of the URL.
This requires the use of a web server preprocessor to modify incoming requests before they reach the server. Pardeike's Welcome (http://welcome.pardeike.net) is one very versatile tool for WebStar-compatible servers (currently not available for MacOS X). Apache has a mod_rewrite module that also can modify an incoming request based on rules.IIS has similar tools.

It can also be solved without the use of external tools, by building a custom Lasso-based error management using error.lasso. In that case we rely on an error condition to convert the URL to a parameter.

When a user enters the site at www.mysite.com, he is assigned a session ID and immediately redirected to http://www.domain.com/1f1c341fe2b89ac346709e/page.lasso

When the client requests this URL, the preprocessor (Welcome, mod_rewrite or Lasso itself using a custom error.lasso) captures the requested URL, parses it, and converts it to something like
http://www.domain.com/page.lasso?-session=site:1f1c341fe2b89ac346709e
which is the request that the web server and Lasso sees.

This way the session ID is automatically propagated through all relative local links. Absolute local links (those beginning with /) are not helped with this method.

This method doesn't prevent the session ID from being bookmarked or printed, and it doesn't help usability (still unfriendly URLs), but it saves you from the work of adding the URL to all forms and frame calls.

Is Your Site Really Secure?

Table of Contents

Is Your Site Really Secure?

by Jim Van Heule

Is your site really as secure as you think it is? Jim will present techniques for securing web sites from common hacker attacks. Proven methods of protection from URL attacks, database hacking and login intrusions will be discussed. New methods available for Lasso Professional 5 will also be discussed.

Biography

Jim has been working with database design since 1987. He brought these skills to the Internet in 1997 using Lasso technology. He has held senior-level positions at several prominent Internet companies, including product manager for Blue World Communications, Inc. VH Publications, Inc. was founded to bring Lasso solutions to the Internet and continues to work in partnership with Internet firms to help bring their database requirements to the World Wide Web.

Notes

Is Your Site Really Secure?

Introduction

The World Wide Web is quickly becoming as common a communication tool as telephones, televisions and publications. We can now access entire libraries of information in the comfort of our home or office at the touch of a finger. The burden is to provide only the information the client wants others to view or change.

Today's hackers take advantage of the Web's openness of the World Wide Web and find joy in breaking into systems just to prove it can be done. A new breed of professional spies break into rival computer systems to gain advantages over the competition. Database systems are primary targets because they contain vast amounts of information in one structured system.

Though it may not be possible or economical to prevent all intrusion attempts, there are several things that can be done to minimize the chances of intrusion and damage that unauthorized attacks can make. This presentation will review some of the more basic routes hackers can use to gain access to your database information, and will demonstrate some Lasso coding techniques to help minimize the chance of stolen, changed or lost data.

Lasso Classic vs. Lasso Inline Techniques

Lasso Classic

Lasso was founded on what is now commonly called the Lasso Classic coding methodology. This methodology uses a technique that first processes information from the requesting page and then displays that information on the following page.

Database information is passed through the form source, link, or URL to achieve the desired results on the next page.

Lasso Classic Form Example

```
<form action="security_classic.lasso" method="Post">
<input type="hidden" name="-Database" value="wiz_products.fp3">
<input type="hidden" name="-Layout" value="web">
<input type="text" name="Firstname" value="[Form_Param:'Firstname']" size="15">
<br>First Name<br>
<input type="text" name="LastName" value="[Form_Param:'LastName']" size="15">
<br>Last Name<br>
<input type="text" name="Gender" value="[Form_Param:'Gender']" size="15">
<br>Gender<br>
<input type="submit" name="-Search" value="search">
```

Lasso Classic Link Example

```
<a href="security_classic.lasso ?-Database= wiz_products.fp3&-Layout= web
&Firstname=[Var:'Firstname']&Lastname=[Var:'Lastname']
&Gender=[Var:'Gender']&-Search">Search</a>
```

Lasso Classic URL Example

```
http://www.mydomain.com/security_classic.lasso ?-Database= wiz_products.fp3
&-Layout=web&Firstname=[Var:'Firstname']&Lastname=[Var:'Lastname']
&Gender=[Var:'Gender']&-Search
```

Lasso Classic Response Page Example

```
[Records]
[Field:'FirstName'][Field:'LastName']-[Field:'GenderName']<br>
[/Records]
```

The Lasso Classic response easily displays results using the [Records] and [Field] tags. Since all the information is already available, Lasso handles the database query prior to the displaying the response page and then uses the response page to display the results.

Because of the way this information is passed, detailed database structure information, including database, layout, keyfield and field names, are made available to the client. This type of information is a virtual gold mine for hackers wishing to steal, change or destroy your data.

Lasso Inlines

The Inline method is a much more secure method because it does not pass critical database information where hackers may take advantage of it. Database, layout, keyfield and field names can all be hidden which makes it much more difficult to gain access.

This technique typically displays minimal information in the form source, link, or URL to achieve the desired results on the next page.

Lasso Inline Method Form Example

```
<form action="security_classic.lasso" method="Post">
<input type="text" name="Fname" value="[Form_Param:'Fname']" size="15">
<br>First Name<br>
<input type="text" name="LName" value="[Form_Param:'LName']" size="15">
<br>Last Name<br>
<input type="text" name="Gen" value="[Form_Param:'Gen']" size="15">
<br>Gender<br>
<input type="submit" name="-Nothing " value="search">
```

Lasso Inline Method Link Example

```
<a href="security_classic.lasso ?Fname=[Var:'Fname']&Lname=[Var:'Lname']
&Gen=[Var:'Gen']&-Search">Search</a>
```

Lasso Inline Method URL Example

```
http://www.mydomain.com/security_classic.lasso?Fname=[Var:'Fname']
&Lname=[Var:'Lname']&Gen=[Var:'Gender']&-Search
```

Notice how a properly configured Inline passes information to a response page containing no database, layout, keyfield of field name information. Even the field names have been altered so as not to match the true field names in the database. It simply provides the information necessary to perform the database call written into the response page. This provides a much higher level of security than Lasso Classic, since little information about the database structure is being provided.

The response page then uses this information to convert it to a form that can be processed by Lasso and to display the desired results as in the example below.

Lasso Inline Response Page Example

```
[Inline:
    -Database='wiz_products.fp3',
    -Layout='web',
    'Firstname'=(Form_Param:'Fname'),
    'LastName'=(Form_Param:'LName'),
    'Gender'=(Form_Param:'Gen'),
    -RecordID=(Var:'RID'),
    -Search]
[Records]
[Field:'FirstName'][Field:'LastName']-[Field:'GenderName']<br>
[/Records]
```

Both the Lasso Classic and Inline methods provide identical results. The difference is that the Inline method does not provide any database information that hackers can take advantage of.

Sequential ID vs. Unique Non-Sequential Random ID

If record-by-record security is desired, it is imperative not to display a sequential keyfield value used to view, update or remove a record. Sequential keyfields allow the hacker to change one value sequentially in order to gain access to view or change each and every record in the database. This is true when using both the Lasso Classic and Inline methodology.

Hypothetical* Hacker Script Used To Delete Database Information

Lasso Classic URL Sequential ID Update Method

```
[Loop: 100,000]
http://www.mydomain.com/security_classic.lasso ?-Database= wiz_products.fp3
&-Layout=web&Firstname=Hacked&-RecordID=[LoopCount]&-Update
[/Loop]
```

Inline URL Sequential ID Update Method

```
[Loop: 100,000]
http://www.mydomain.com/security_classic.lasso?Fname=Hacked
&ID=[LoopCount]
[/Loop]
```

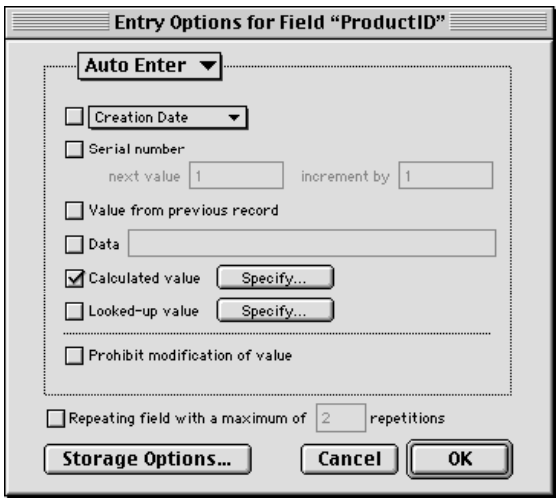
* These scripts are not actually functional. The Lasso [Loop] tags were used for example purposes only.

Both methods achieve the same result of replacing the first-name field in the first 100,000 records with the word "Hacked." This method is extremely easy and has been used against even the largest eCommerce web sites.

Fortunately, with a little extra planning, there is a simple solution that can minimize the damage hackers can make. By changing the sequential keyfield to a unique non-sequential keyfield with trillions of available variations, hackers' sequential attacks become a remote possibility. The following is an example of creating a simple Filemaker database to demonstrate this technique.

Create a Filemaker database from the following table and auto enter calculation:

Name	SequentialNumber	NameID
Bill	1	Auto Enter Calculation
Jane	2	Auto Enter Calculation
Mary	3	Auto Enter Calculation
Steve	4	Auto Enter Calculation



Auto Enter Calculation for NameID

```
"PID" & SerialNumber &
 Middle("ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890",
 Int(Random* 36), 1) &
 Middle("ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890",
 Int(Random* 36), 1) &
 Middle("ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890",
 Int(Random* 36), 1) &
 Middle("ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890",
 Int(Random* 36), 1) &
 Middle("ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890",
 Int(Random* 36), 1) &
 Middle("ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890",
 Int(Random* 36), 1) &
 Middle("ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890",
 Int(Random* 36), 1)
```

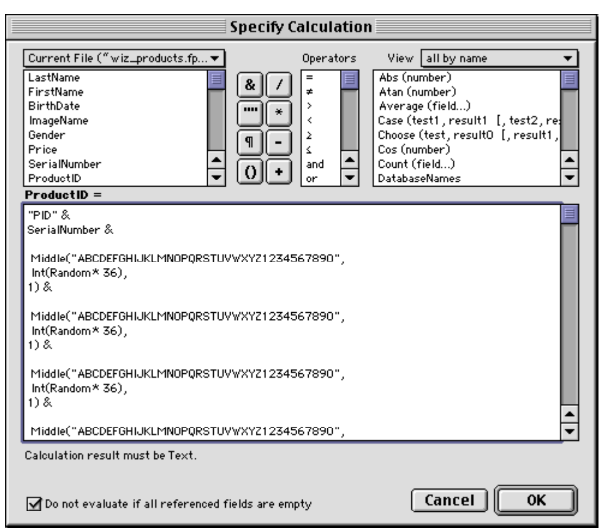
The above NameID calculation provides 78,364,164,096 (78 trillion) possible random combinations. To guarantee the ID remains unique, notice that a unique sequential number is added to the ID string.

Using this method, notice below how a typical URL will look.

Inline URL Sequential ID Update Method

http://www.mydomain.com/security_classic.lasso?Fname=Hacked&ID=PID1EN3J2WR

Now, if hackers try the next ID sequence, there is roughly a one-in-78 trillion chance a second record will be found. These odds can be expanded by increasing the random string, which is desirable for large database sets. However, many databases have a maximum searchable string length that when exceeded, adds no additional random value. The hackers can still use sequential attacks using strings of alpha, numeric and punctuation, but in most cases, it becomes unrealistic to attempt trillions of combinations just to find additional records to manipulate.



Get vs. Post Form Submissions

Using the Post method in a form submission provides minimal security benefits versus using the Get method, because the information hidden in the Post method can easily be seen in the page's source code.

The Post method is still the preferred method, however, because it does clean up the URL and allows for longer strings of information to be passed. Since the Post method does not openly display passed information, it tends to provide some deterrence from casual or novice hackers.

Typical Get Method URL

```
http://www.mydomain.com/security_classic.lasso?Fname=Jill&Lname=Glittery&Gender=Female
```

Typical Post Method URL

```
http://www.mydomain.com/security_classic.lasso
```

Lasso Database Browser

(formerly Lasso Instant Publisher)

Lasso's Database Browser tool is an excellent developers' tool to test database connectivity and to make simple changes to existing databases. Current Lasso versions use a tag named [Auth_Admin] within Database Browser that allows access only via the Lasso administrator's password. This tag can be used with any file for similar purposes and security benefits. Lasso Publisher is very safe to use unless this tag is removed.

The Database Browser can potentially be used as a hacker's tool to gain access to data on a shared server. A person could upload a version of the Lasso Database Browser into their space with the [Auth_Admin] tags removed to gain access to all databases hosted on the system. With this in mind, it is imperative to have a username and password on all databases connected to the Internet. The Database Browser allow hackers to see your database on a shared server, but without the database's password, that is all they get to see.

It should be noted that the Database Browser is not a hackers tool. Anyone with Lasso knowledge could build the same tool and use it for destructive purposes. Lasso's Database Browser is an excellent developers' tool. I use it constantly to test for database connectivity and make minor changes to database records.

Database Password and TCP/IP Access

Any database that has TCP/IP access can be accessed by anyone on the Internet as long as they have access to the IP address. There are a few tools that can minimize who can gain access to the data.

The FileMaker Pro 5 Web Companion has a nice feature that allows access from only one IP address. This is perfect for setting up the Web Companion to be accessed only by the Web server and effectively locks out any requests from other IP addresses. This feature makes upgrading from FMP4 to FMP5 a nice security benefit.

All databases that currently work with Lasso can be set up to require a password and, in some cases, a username as well. By setting up Lasso Security to gain access via the username/password combination, database access via an outside source can be minimized. It is advisable to use a password with at least eight characters that are alpha/numeric in nature and not using any words that can be found in a dictionary.

Cookies, Tokens and Form Parameters

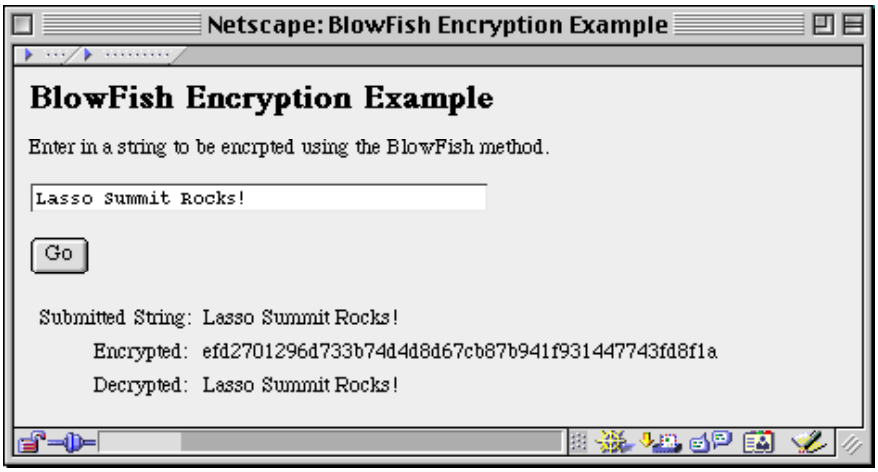
Cookies, tokens and form parameters are all methods for passing information from one web page to another. These are the methods used to keep a clients "state" they can be recognized throughout the site. None of these tools should be considered a security feature since all they are used for is to transport information.

Encryption Methods In Lasso 5

With the release of Lasso Professional 5, three new encryption methods are available for secure transportation or data storage.

BlowFish

BlowFish is a very fast, effective way to encrypt data for storage or transport. It uses a key that the developer creates and uses for encryption and decryption. A different key can be used for different pages or purposes in a Web solution for enhanced security.



BlowFish Encryption Example

```
[Var:'theEncryptedString'=(Encrypt_BlowFish: 'Lasso Summit Rocks!'-Seed='T5Rde3Nbh')]
[Var:'theEncryptedString']
```
Result: efd2701296d733b74d4d8d67cb87b941f931447743fd8f1a

BlowFish Decryption Example

```
[Var:'theDecryptedString'=(Decrypt_BlowFish: (Var:'theEncryptedString'), -Seed='T5Rde3Nbh')]
[Var:'theDecryptedString']
```
Result: Lasso Summit Rocks!

Base64

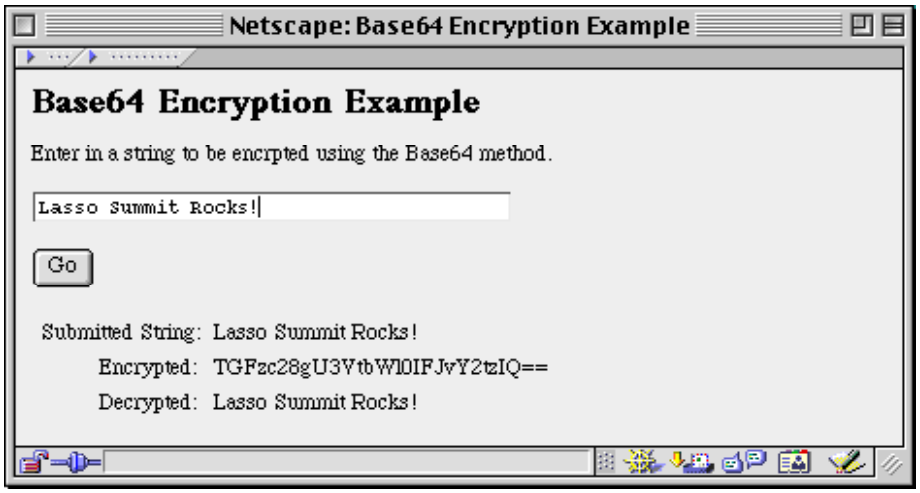
Base64 is a very fast, data-scrambling method ideal for mixing data that does not provide any real degree of security. It helps to obsure data while it is being passed. With the availablility of BlowFish and MD5 in LP5, there is no real reason to use this type of scheme.

Base64 Encryption Example

```
[Var:'theEncryptedString'=(Encrypt_Base64: 'Lasso Summit Rocks!')]
[Var:'theEncryptedString']
```
Result: TGFzc28gU3VtbWl0IFJvY2tzIX==

Base64 Decryption Example

```
[Var:'theDecryptedString'=(Decrypt_Base64: (Var:'theEncryptedString'))]
[Var:'theDecryptedString']
```
Result: Lasso Summit Rocks!



MD5

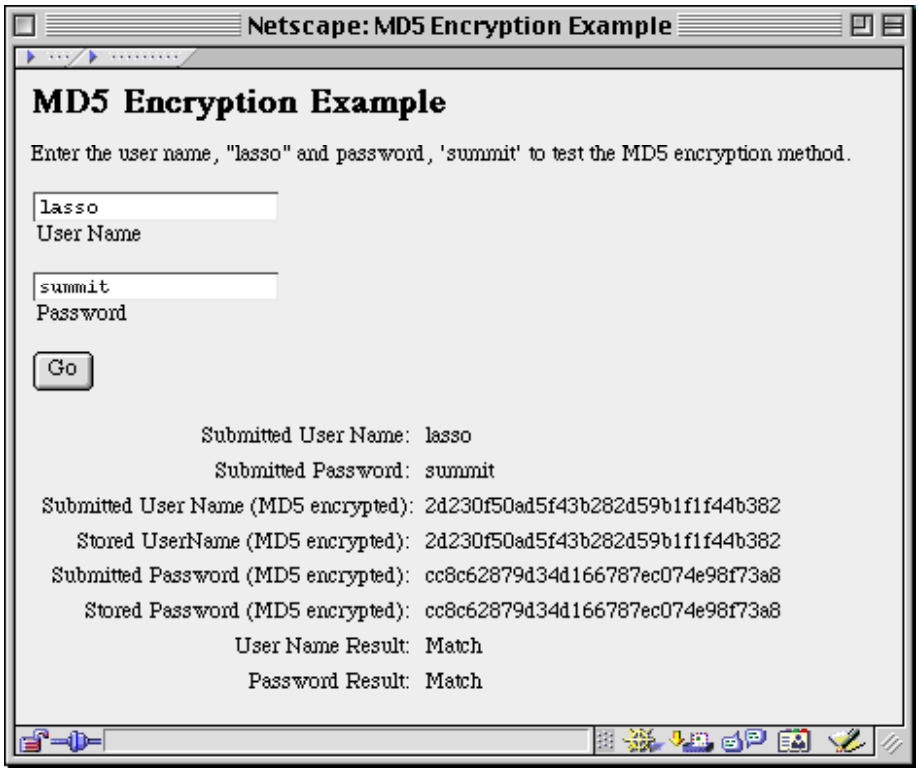
MD5 is a very fast, one-way encryption method ideal for storing encrypted passwords in

a database. When a person enters a password to gain entry, it is encrypted using MD5 and compared to the already encrypted password stored in the database.

MD5 Encryption Example

[Var:'theSecuredPassword'=(Encrypt_MD5:'summit')]
Result: cc8c62879d34d166787ec074e98f73a8

A nice side benefit of this scheme is that it removes the need to do multiple login tests that match the string content and string length to verify a match.



Session Management

All of the methods discussed here deal with hiding key database information, but they do openly pass some type of information to pull information from the database. The methods show how to minimize the threat by cutting down what is passed and using non-sequential and even encryption to thwart potential hackers. Unfortunately, all of these methods still provide some percentage of risk that your data might some day be hacked.

There is a method that provides an even higher level of security by providing a transportation mechanism of secure data, such as keyfields, that reside only on the server side. Session Management is currently the most effective method for controlling hacker attacks available.

It is now built into the new Lasso Professional 5. Johan Sölve will discuss this in detail during his talk on Session Management.

Conclusion

Security must not be taken lightly and should be designed into every Web development project that contains dynamic database connectivity. Several key steps can be taken to help minimize the risk, including the following.

- Use Lasso Inline methodology in place of Lasso Classic to help to hide the database structure.
- Use non-sequential keyfields to minimize the effect of a sequential hacker attack.
- Use the Post method in form submissions to reduce open visibility of passed form parameters.
- Use passwords on databases to minimize access via TCP/IP.
- Use the IP restriction feature in Web Companion if using FleMaker Pro 5.
- Use encryption for data you want to keep secure. This includes data being passed within the Web site and even data stored within the database.
- Use these methods with Session Management for maximum security.

Today's hackers can easily exploit several insecure, common developer techniques that are in use today. The recommended methods described in this paper are designed to achieve the desired increased security level without the need of high-level programming. Though these methods do not guarantee 100% resistance from outside hacker attacks, they do make for a good starting point to help remove the temptation. For those wanting a higher level of security, Session Management is currently one of the best methods available and is covered in detail by Johan Sölve in this manual. But even Session Management needs the techniques described in this paper to achieve increased, desired security levels.

Lasso 3 Hands-On Training

Lasso 3 Hands-on Training

Inline Lasso: The Preferred Method

Part 1

Table of Contents

Lasso 3 Hands-on Training

Inline Lasso: The Preferred Method

by Kirk Bowman

Part 1
Inline Lasso is the preferred programming style for many intermediate to advanced developers. Compared to Classic Lasso, Inline Lasso offers more flexibility and security. This session will demonstrate how to use Inline Lasso to create search, list and detail pages. Lasso tags to be discussed in this session include [Inline], [Form_Param], [ReUseFormParams], [Error_CurrentError] and [Search_Arguments].

Part 2
This session will demonstrate how to update and delete records using Inline Lasso. Lasso tags to be discussed in this session include [Inline], -Token, [Token_Value], [If], [Var_Set], [LoopAbort], [Var_Defined], -RecID, [RecID_Value], [Error_CurrentError], [Include] and [Lasso_CurrentAction].

Biography

Kirk has been developing database solutions since 1991. In 1997 he began developing web solutions with a beta of FileMaker Pro 4.0 and CDML. Shortly thereafter he created his first web solution with Lasso 1.0--a nationwide benefits enrollment system for a forestry company.

Kirk is now president of MightyData, L.L.C. based in Dallas, Texas offering information technology design, development, and training services. He writes a monthly Lasso column for ISO FileMaker Magazine and is the author of several FileMaker and Lasso training courses. He is a jazz music fan and a former professional saxophonist.

Notes

Lasso 3 Hands-on Training, Session 1
Inline Lasso: The Preferred Method (Part 1)

Introduction

Inline Lasso is the preferred method of Lasso programming that has developed since the release of Lasso 2.0. Lasso 2.0 introduced the [Inline] tag. Inline Lasso embeds database actions in LDML format files. Using Inline Lasso several database actions can be executed in one format file. Furthermore, Inline Lasso hides sensitive information such as database name, table (layout) name and passwords from prying eyes.

The opposite of Inline Lasso is Classic Lasso. Classic Lasso uses standard HTML forms and links to pass Lasso commands to the Web server. Classic Lasso requires at least two pages, a call page and a response page. Since Lasso commands are passed to the Web server, the commands can be viewed by looking at the source code in a Web browser.

To use this tutorial, complete the following.

1. Put the Inline_Method folder at the root level of the Web server folder.
2. Configure Lasso and FileMaker 5 to access the Employees database.
3. In Lasso Security, enable the Search permission for All Users for the Employees database.
4. The Inline Method Home Page can be viewed in a Web browser at <http://localhost/inline_method/default.html>.
5. The source code for the demo files is in the folder inline_method/demos/.

Classic Lasso

Prior to learning Inline Lasso, it is important to review Classic Lasso. Run the demo "Classic Lasso" under Exercise 1 on the Inline Method Home Page. The source code for classic1.lasso contains a form to search the Employees database.

```
<FORM ACTION="classic2.lasso" METHOD="post">
    <input type="hidden" name="-Database" value="Employees">
    <input type="hidden" name="-Table" value="web">
    Full Name: <INPUT TYPE="text" NAME="full_name" SIZE="24">
    <INPUT TYPE="submit" NAME="-Search" VALUE="Search">
    <INPUT TYPE="reset">
</FORM>
```

The Action attribute of the Form tag tells the Web server to send classic2.lasso as the response page. The ".lasso" extension tells the Web server to process classic2.lasso using the Lasso Plug-in. Three Input tags tell Lasso the database name, table name, and field name to use for

the search. The Submit button sends the Search command to Lasso.
The response page, classic2.lasso, displays the results of the search using the following code.

```
[Records]
    [Field: 'first_name', EncodeNone] [Field: 'last_name', EncodeNone]<BR>
[/Records]
```

Form Parameter

The concept of form parameters is important to using Inline Lasso. On the current page, any Input tag from the previous page can be displayed using the [Form_Param] tag.

Run the demo "Form Parameters" under Exercise 1 on the Inline Method Home Page. The source code for form_param1.lasso is similar to classic1.lasso.

```
<FORM ACTION="inline1b.lasso" METHOD="post">
    Full Name: <INPUT TYPE="text" NAME="full_name" SIZE="24">
    <INPUT TYPE="submit" NAME="-Nothing" VALUE="Search">
    <INPUT TYPE="reset">
</FORM>
```

The opening Form Tag serves the same purpose--it tells the Web server to send form_param2.lasso as the response page. The Input tag for the field name is also the same.

The Input tags containing the database name and table name have been removed. The Lasso command in the the Submit button has been change to the -Nothing. -Nothing allows Lasso to process a page without accessing a database.

The source code for the response page, form_param2.lasso, displays the "full_name" form parameter.

```
Full Name: [Form_Param: 'full_name', EncodeNone]
```

The [Form_Param] tag contains the parameter 'full_name'. This parameter matches the Name attribute of the Input tag in form_param1.lasso. The [Form_Param] tag displays the contents entered into the Input field on form_param1.lasso.

Inline Lasso

By comparing the code of the response pages, classic2.lasso and inline1a.lasso, the secret of the Inline Lasso is revealed. Run the demo "Inline Lasso 1" under Exercise 1 on the Inline

Method Home Page. The source code in classic2.lasso displays the results of the search.

```
[Records]
    [Field: 'first_name', EncodeNone] [Field: 'last_name', EncodeNone]<BR>
[/Records]
```

This same code is used in inline1a.lasso.

```
[Inline: -Database='Employees', -Table='web', ReUseFormParams, -Search]
    [Records]
        [Field: 'first_name', EncodeNone] [Field: 'last_name', EncodeNone]<BR>
    [/Records]
[/Inline]
```

The difference between classic2.lasso and inline1a.lasso is the Inline tag. The opening [Inline] tag declares the database name, table name and Search action. The ReUseFormParams keyword tells Lasso to use any form parameters on the previous page (inline1a.lasso) as part of the Inline tag. When searching for all names that begin with "j", Lasso replaces the ReUseFormParams keyword with the following code when executing the Inline.

```
-Op='begins with', full_name='j'
```

The closing [/Inline] tag closes the execution of the Inline.

-ClientUsername and -ClientPassword

If a User Name and Password are specified in the Inline tag, Inline Lasso is more secure than Classic Lasso. Run the demo "Inline Lasso 2" under Exercise 1 on the Inline Method Home Page.

The Classic Inline and Inline Lasso 1 demos were performed with Search enabled for All Users for the Employees database. With All Users enabled, a hacker could gain unauthorized access to the database.

The source code for inline2b.lasso uses the -ClientUsername and -ClientPassword tags to specify the User Name 'test' and the Password 'test'.

```
[Inline: -Database='Employees', -Table='web', -ClientUsername='test', -ClientPassword='test',
    ReUseFormParams, -Search]
    [Records]
        [Field: 'first_name', EncodeNone] [Field: 'last_name', EncodeNone]<BR>
    [/Records]
[/Inline]
```

When All Users is disabled, the only way to access the Employees database is through the

inline2b.lasso page. If a hacker attempts unauthorized access, he will not know the User Name and Password and thus not gain access to the database.

Exercise 1 - Basic Inline Lasso

The solution for Exercise 1 can be accessed under Solutions on the Inline Method Home Page.

1. Open search.lasso in the folder inline_method/exercise01/ in a text editor.

2. Insert the following Input tag inside the third cell of the second row of the table. (The other Input tags have already been added for you.)

   ```
   <INPUT TYPE="text" NAME="full_name" VALUE="[Field: 'full_name']">
   ```

3. Insert the -Nothing tag as the Name attribute of the Submit button.

   ```
   <INPUT TYPE="submit" NAME="-Nothing" VALUE="Search">
   ```

4. Save the file.

5. Open listing.lasso in the folder inline_method/exercise01/ in a text editor.

6. Insert the opening Inline tag before the opening Table tag.

   ```
   [Inline: ReUseFormParams, -Database='Employees', -Table='web', -ClientUsername='test',
   -ClientPassword='test', -Search]
   ```

7. Insert the closing Inline tag after the closing Table tag.

   ```
   [/Inline]
   ```

8. Save the file. Open Lasso Security in a Web browser at <http://localhost/lasso/security>.

9. Add a new database "Employees" if it does not already exist.

10. Disable Search for All Users for the Employees database.

11. Add a new User "test" with a Password "test" for the Employees database. Enable the search permission for the "test" User.

12. Load <http://localhost/inline_method/exercise01/search.lasso> in a Web browser and test the solution.

Debugging an Inline

Debugging an Inline tag is difficult because the code for the Inline is contained in one file. Fortunately, the [Error_CurrentError] tag displays the result of an Inline. Run the demo "Inline Success" under Exercise 2 on the Inline Method Home Page. The following code is contained in inline_success.lasso.

```
[Error_CurrentError]
```

The code displays the following message when the Inline search is successful.

```
No Error
```

The ErrorCode keyword inside for the [Error_CurrentError] tag will display the error number instead of the error message. Run the demo "Inline Failure" under Exercise 2 on the Inline Method Home Page. The following code is contained in inline_failure.lasso.

```
[Error_CurrentError, ErrorCode], [Error_CurrentError]
```

The code displays the following message when the Inline search fails due to a bad User Name.

```
-9964, Invalid user name
```

Displaying the form parameters when Lasso processes ReUseFormParams also helps debug an Inline. The Search Argument tags display the form parameters from the previous page. This avoids having to hard code the [Form_Param] tag to view the form parameters.

Run the demo "Displaying Form Parameters" under Exercise 2 on the Inline Method Home Page. The following code in search_args2.lasso displays the search criteria (form parameters) entered on search_args1.lasso.

```
[Search_Arguments]
    [SearchFieldItem] [SearchOperatorItem] [SearchValueItem]<BR>
[/Search_Arguments]
```

The [Search_Arguments] [/Search_Arguments] tags tell Lasso to loop through the form parameters from the previous page. [SearchFieldItem] displays the name of the form parameter. [SearchOperatorItem] displays the search operator for the form parameter. [SearchValueItem] displays the value of the form parameter.

Exercise 2 - Debugging an Inline

The solution for Exercise 2 can be accessed under Solutions on the Inline Method Home Page.

1. Open search.lasso in the folder inline_method/exercise02/ in a text editor.

2. Insert tags to display the error number and error message after the opening [Inline] tag.

```
Inline Error: [Error_CurrentError, ErrorCode], [Error_CurrentError]
```

3. Insert the Search Argument tags to display the form parameters just after the opening Body tag.

```
Search Arguments:<BR>
[Search_Arguments]
    [SearchFieldItem] [SearchOperatorItem] [SearchValueItem]<BR>
[/Search_Arguments]
```

4. Save the file. Load <http://localhost/inline_method/exercise02/search.lasso> in a Web browser and test the solution.

Displaying a Record

The previous demo demonstrated a search page and a list page. The next task is to display a detail page for a specific record using Inline Lasso. Run the demo "Detail Link" under Exercise 3 on the Inline Method Home Page. The following code creates a link to the detail page for each record.

```
<A HREF="detail_link2.lasso?employee_id=[Field: 'employee_id', EncodeNone]">[Field:'employee_number',
EncodeNone]</A>
```

The first part of the HREF attribute specifies the response file "detail_link1b.lasso". The second part of the HREF attribute specifies a form parameter "employee_id" with a value from the field "employee_id". The field tag [Field:'employee_number', EncodeNone] displays the employee number for the link in the browser.

The value of the employee_id field is generated using the following auto-enter calculation in FileMaker.

```
Middle(“ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789”, Int(Random*36), 1) &
Middle(“ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789”, Int(Random*36), 1) &
Middle(“ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789”, Int(Random*36), 1) &
Middle(“ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789”, Int(Random*36), 1) &
```

```
Middle(“ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789”, Int(Random*36), 1) &
Middle(“ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789”, Int(Random*36), 1)
```

This calculation generates a random six-character alphanumeric value. A random value is used to prevent a hacker from trying to gain access to other records by manipulating the employee_id value.

On the detail page, the following Inline searches for the record to display using the "employee_id" form parameter. This is the same Inline we used on the list page in the previous exercise.

```
[Inline: -Database='Employees', -Table='web', -ClientUsername='test', -ClientPassword='test', ReUseForm-
Params, -Search]
```

Another way to display a detail page for a specific record is to use an HTML form instead of link. Run the demo "Detail Form" under Exercise 3 on the Inline Method Home Page. The form below submits the "employee_id" form parameter to the Inline on the response page.

```
<FORM ACTION="detail_form2.lasso" METHOD="post">
    <INPUT TYPE="hidden" NAME="employee_id" VALUE="[Field: 'employee_id', EncodeNone]">
    <INPUT TYPE="submit" NAME="-Nothing" VALUE="View">
</FORM>
```

Exercise 3 - Displaying a Record

The solution for Exercise 3 can be accessed under Solutions on the Inline Method Home Page.

1. Open listing.lasso in the folder inline_method/exercise03/ in a text editor.

2. Insert the following link in the first cell of the second row of the table to create a hyperlink to the detail page.

   ```
   <A HREF="detail.lasso?employee_id=[Field: 'employee_id', EncodeNone]">[Field:'employee_number',
   EncodeNone]</A>
   ```

3. Insert the following form into the last cell of the second row of the table to create a button to the detail page.

   ```
   <FORM ACTION="detail.lasso" METHOD="post">
           <INPUT TYPE="hidden" VALUE="[Field: 'employee_id', EncodeNone]" NAME="employee_id">
           <INPUT TYPE="submit" NAME="-Nothing" VALUE="View">
   </FORM>
   ```

4. Save the file.

5. Open detail.lasso in the folder inline_method/exercise03/ in a text editor.

6. Insert the opening Inline tag before the opening Table tag.

   ```
   [Inline: -Database='Employees', -Table='web', -ClientUsername='test', -ClientPassword='test', ReUse-FormParams, -Search]
   ```

7. Insert the following debugging code after the opening Inline tag.

   ```
   Inline Error: [Error_CurrentError: ErrorCode], [Error_CurrentError]
   ```

8. Insert the closing inline tag after the closing Table tag.

   ```
   [/Inline]
   ```

9. Save the file. Load <http://localhost/inline_method/exercise03/search.lasso> in a Web browser and test the solution.

Lasso 3 Hands-on Training, Session 2
Inline Lasso: The Preferred Method (Part 2)

Updating a Record

After displaying a record using Inline Lasso, the next step is to update a record. Before learning the Inline code, it is important to learn tags to help with flow control.

Setting a Token

The -Token tag is similar to a form parameter. It passes a value from one page to the next. The difference is -Token is not included with the form parameters when using Search_Arguments or ReUseFormParams. Run the demo "Token" under Exercise 4 on the Inline Method Home Page. The following code sets a form parameter and a token on token1.lasso.

```
<INPUT TYPE="hidden" NAME="city" VALUE="Orlando">
<INPUT TYPE="hidden" NAME="-Token.state" VALUE="Florida">
```

The following code displays the form parameter and the token on token2.lasso. Note the token is displayed by the [Token_Value:] tag rather than the Search Argument tags.

```
Form Parameter: [Search_Arguments]
     [SearchFieldItem] = [SearchValueItem]
      [/Search_Arguments]
Token: state = [Token_Value: 'state', EncodeNone]
```

The ability to pass a value from page to page without including it with the form parameters is important for writing optimized code to update a record.

Setting a Token in a Button

A variant of setting a token is to use a Submit button. Run the demo "Button Token" under Exercise 4 on the Inline Method Home Page. The following code sets a token on token_button1.lasso.

```
<INPUT TYPE="submit" NAME="-Token.Demo" VALUE="Next Page">
```

The Name attribute of Input tag contains the -Token tag. The Value attribute of the Input tag contains the value the to which the token is set. The [Token_Value] tag is used to display the token on token_button2.lasso.

```
Token Demo = [Token_Value: 'Demo', EncodeNone]
```

Which Button?

Since a token can be set in a Submit button. We can test if a user has clicked a button by testing the contents of a token. Run the demo "Test for Button Click" under Exercise 4 on the Inline Method Home Page. button_click1.lasso contains the following Submit buttons.

```
<INPUT TYPE="submit" NAME="-Token.Demo" VALUE="Set Token">
<INPUT TYPE="submit" NAME="-Nothing" VALUE="Do No Set Token">
```

The first Submit button sets a token called "Demo" to "Set Token". The second Submit button does not set a token. The following conditional statement tests which button was clicked by testing the contents of the token.

```
[If: (Token_Value: 'Demo') != '']
    The token was set!<BR>
    Token: Demo = [Token_Value: 'Demo', EncodeNone]
[Else]
    The token was not set!
[/If]
```

Update with Inline Lasso

A record is updated by an Inline using a three-step sequence.

1. Did the user click the Update button?
2. If yes, locate the record ID for the record to update.
3. Update the record identified by the record ID.

To prepare for the "Update Demo", enable the Update permission for the "test" user in Lasso Security. Run the "Update Demo" under Exercise 4 on the Inline Method Home Page.

The Action attribute for the Form tag in the source code for update_demo2.lasso is recursive. In other words, it calls itself as the response page. The Update button is a Submit button that sets a token. A conditional statement at the top of the page tests whether the Update button was clicked on the previous page. Below is the code for the first step of the Update sequence.

```
[If: (Token_Value: 'Update') != ''] [/If]

<FORM ACTION="update_demo2.lasso" METHOD="post">
    <INPUT TYPE="submit" NAME="-Token.Update" VALUE="Update">
</FORM>
```

If the user clicked the Update button, the next step is to locate the RecordID_Value or KeyField_Value for the record displayed on the previous page. A hidden Input tag on the previous page passes the employee_id as a form parameter. The following Inline tag searches for the specific record using the "employee_id" form parameter and sets a variable to the record ID.

```
[Inline: -Database='Employees', -Table='web', -ClientUsername='test', -ClientPassword='test',
employee_id=(Form_Param:'employee_id'), -Search]
    [Var_Set: 'theRec'=(RecID_Value)]
    Search Inline: [Error_CurrentError, ErrorCode], [Error_CurrentError]
[/Inline]
```

After "theRec" variable is set to the record ID, the variable is used in an Update inline. The value for the -RecID command is "theRec" variable. The ReUseFormParams keyword submits the field data to the database from the form parameters from the previous page. Below is the code for the Update inline.

```
[Inline: -Database='Employees', -Table='web', -ClientUsername='test', -ClientPassword='test',
-RecID=(Var:'theRec'), ReUseFormParams, -Update]
    Update Inline: [Error_CurrentError, ErrorCode], [Error_CurrentError]
[/Inline]
```

There are two important points to remember to ensure the Update inline is successful. First, the Update inline is not "nested" inside the Search inline. Second, a token rather than a form parameter is used to test if the Update button was clicked. Failure regarding either of these items will create problems using the ReUseFormParams keyword in the Update inline.

Exercise 4 - Updating a Record

The solution for Exercise 4 can be accessed under Solutions on the Inline Method Home Page.

1. Open detail.lasso in the folder inline_method/exercise04/ in a text editor.

2. Insert an Update (submit) button after the closing Table tag to set a token to "Update" when the user clicks it.

   ```
   <INPUT TYPE="submit" NAME="-Token.Update" VALUE="Update">
   ```

3. Insert a Reset button after the Update button.

   ```
   <INPUT TYPE="reset">
   ```

4. Insert a conditional statement after the opening Body tag to test whether

the user clicked the Update button.

```
[If: (Token_Value: 'Update') != ''] [/If]
```

5. Insert an Inline tag after the opening [If] tag to search for the record to update.

```
[Inline: -Database='Employees', -Table='web', -ClientUsername='test', -ClientPassword='test',
employee_id=(Form_Param:'employee_id'), -Search]
[/Inline]
```

6. Set a variable between the Inline tags to the record ID of the record to update.

```
[Var_Set: 'theRec'=(RecID_Value)]
```

7. Insert error tags after the Variable to display the success or failure of the search inline.

   Search Inline: [Error_CurrentError: ErrorCode], [Error_CurrentError]

8. Insert a second Inline after the closing tag of the Search inline to update the record using the record ID stored in the variable.

```
[Inline: -Database='Employees', -Table='web', -ClientUsername='test', -ClientPassword='test',
-RecID=(Var:'theRec'), ReUseFormParams, -Update]
[/Inline]
```

9. Insert error tags inside the Update inline tags to display the success or failure of the Update action.

```
Update Inline: [Error_CurrentError: ErrorCode], [Error_CurrentError]
```

10. Save the file. Load <http://localhost/inline_method/exercise04/search.lasso> in a Web browser and test the solution.

Requiring Search Criteria

One responsibility of a programmer is to know the advantages and disadvantages of the programming techniques used. If a disadvantage is potentially harmful, the programmer should address the issue.

One such disadvantage of using an Inline search with the ReUseFormParams keyword is the user can click the Search button without entering any search criteria and perform a FindAll action. To prevent a Findall, the Search Argument tags can be used to test whether the user has entered any search criteria.

Run the demo "LoopCount" under Exercise 5 on the Inline Method Home Page The Search Argument tags loop through the form parameters from the previous page, executing the enclosed code for each iteration. The following code sets a variable to the number of the iteration using the [LoopCount] tag.

```
[Search_Arguments]
    [Var_Set: 'count'=(LoopCount)]
[/Search_Arguments]

[Var: 'count']
```

Since a form parameter exists only when something is typed in an Input field, the variable is only set when the user enters at least one search criteria. A test whether the variable is set will determine if search criteria where entered. Furthermore, it is only necessary to set the variable one time before performing the test. Run the demo "Variable Defined?" under Exercise 5 on the Inline Method Home Page

```
[Search_Arguments]
    [Var_Set: 'count'=(LoopCount)]
    [LoopAbort]
[/Search_Arguments]

[If:  (Var_Defined: 'count')]
    The variable was set!
[Else]
    The variable was not set!
[/If]
```

The code above executes the code inside the Search Argument tags one time. The [LoopAbort] tag halts the loop after one iteration. The [Var_Defined] tag returns a value of "true" if the variable is set and "false" if it is not. Using an If-Else statement, different HTML is displayed depending whether or not the user entered search criteria.

Exercise 5 - Requiring Search Criteria

The solution for Exercise 5 can be accessed under Solutions on the Inline Method Home Page. In this exercise the search form has been saved in an include file called "search_form.inc" in the Includes folder. The listing form has been saved in an include file called "listing_form.inc" in the Includes folder.

1. Open listing.lasso in the folder inline_method/exercise05/ in a text editor.

2. Insert code after the opening Body tag to set a variable if the user enters at least one search criteria on the search form.

```
[Search_Arguments]
        [Var_Set: 'count'=(LoopCount)]
        [LoopAbort]
[/Search_Arguments]
```

3. Insert an If-Else statement after the closing [Search_Argument] tag to test whether the variable was set.

```
[If:  (Var_Defined: 'temp')]
[Else]
[/If]
```

4. Insert an Include tag after the opening [If] tag to display listing_form.inc if the user entered search criteria.

```
[Include: 'includes/listing_form.inc']
```

5. Insert an Include tag after the [Else] tag to display an error message and include search_form.inc if the user did not enter search criteria.

```
<FONT COLOR="red">You must enter at least one search criteria.</FONT>
[Include: 'includes/search_form.inc']
```

6. Save the file. Load <http://localhost/inline_method/exercise05/search.lasso> in a Web browser and test the solution.

Deleting a Record

The final task in this tutorial is to delete a record. Three new tags will be used in the page flow control. Run the demo "Compound Condition" under Exercise 6 on the Inline Method Home Page. The following code tests whether the user enters a value AND whether the value is a number.

```
[If: (Form_Param:'number') != '' && (String_IsNumeric:(Form_Param:'number'))]
    Thank you for entering a number!
[Else]
    You did not enter a number!
[/If]
```

The double ampersand (&&), called the AND operator, requires both tests to be true for the conditional statement to evaluate as true. The double pipe (||), called the OR operator, requires only one of the tests to be true for the conditional statement to evaluate as true.

Run the demo "Error Trapping" under Exercise 6 on the Inline Method Home Page. The following code test whether the search action was successful and displays the appropriate message.

```
[If: (Error_CurrentError) == (Error_NoError)]
    The search was successful![
Else]
    The search was not successful!<BR>
    Inline Error: [Error_CurrentError: ErrorCode], [Error_CurrentError]
[/If]
```

When an Inline action is successful, Lasso returns the message "No Error". The [Error_NoError] tag is synonymous with the "No Error" message. Comparing [Error_CurrentError] to [Error_NoError] determines whether the database action was successful.

Lasso also contains a tag to determine the type of database action executed. The [Lasso_CurrentAction] can be used in a conditional statement to test the type of action. The following code tests whether the most recent action was a Search.

```
[If: (Lasso_CurrentAction) == 'Search'] The action was Search! [/If]
```

This task combines the tags above with techniques from the previous exercises to delete a record. A record is deleted by an Inline using a three-step sequence.

1. Did the user click the Delete button?
2. If yes, locate the record ID for the record to delete.
3. Delete the record identified by the record ID.

The Delete button is similar to the Update button. A token is set when the user clicks it. The record ID is located by the same Search inline used for the Update. The Delete inline is the same as the Update inline with two exceptions. First, the action tag for the inline is -Delete. Second, the ReUseFormParams keyword is not necessary since the record is being deleted. Below is the logic structure for the page flow.

```
<!-- Code to update or delete record -->
[If: update button was clicked OR delete button was clicked]
    Execute the inline to locate the record ID
    [If: update button was clicked]
        Execute the inline to update the record
    [Else]
        Execute the inline to delete the record
    [/If]
[/If]

<!-- Code to display the correct form -->
[If: the delete action was successful]
    Include the search form
[Else]
    Include the detail form
[/If]
```

Exercise 6 - Deleting a Record

The solution for Exercise 6 can be accessed under Solutions on the Inline Method Home Page. In this exercise the detail form has been saved in an include file called "detail_form.inc" in the Includes folder.

A. Update or Delete Code

1. Open detail_form.inc in the folder inline_method/exercise06/includes/ in a text editor.

2. Insert a Submit button after the Update button to set a token to "Delete" when the user clicks it.

```
<INPUT TYPE="submit" NAME="-Token.Delete" VALUE="Delete">
```

3. Save the file. Open detail.lasso in the folder inline_method/exercise06/ in a text editor.

4. Insert a conditional statement after the opening Body tag to test whether the user clicked the Update or Delete button.

```
[If: (Token_Value: 'Update') !='' || (Token_Value: 'Delete') != '']
[/If]
```

5. Insert a Search inline inside the [If] statement to locate the record ID for the record to update or delete.

```
[Inline: -Database='Employees', -Table='web', -ClientUsername='test', -ClientPassword='test',
employee_id=(Form_Param:'employee_id'), -Search]
        [Var_Set: 'theRec'=(RecID_Value)]
[/Inline]
```

6. Insert a conditional statement after the Search inline to test whether the user clicked the Update button.

```
[If: (Token_Value: 'Update') != '']
[Else]
[/If]
```

7. Insert an Update inline between the [If] and [Else] tags to update the record.

```
[Inline: -Database='Employees', -Table='web', -ClientUsername='test', -ClientPassword='test',
-RecID=(Var:'theRec'), ReUseFormParams, -Update]
        [Include:'includes/inline_result.inc']
[/Inline]
```

8. Insert an Update inline between the [Else] and [/If] tags to delete the record.

```
[Inline: -Database='Employees', -Table='web', -ClientUsername='test', -ClientPassword='test',
-RecID=(Var:'theRec'), -Delete]
        [Include:'includes/inline_result.inc']
[/Inline]
```

9. Insert a conditional statement after the opening [Inline] tag for the Delete inline to set a variable if the Delete action is successful.

```
[If: (Error_CurrentError) == (Error_NoError)]
[Var_Set: 'deleteOK'='1']
[/If]
```

10. The completed code to update or delete a record should be similar to this.

```
<!-- Test if the user clicked the Update or Delete button -->
[If: (Token_Value: 'Update') !='' || (Token_Value: 'Delete') != '']
    <!-- Inline search to locate the record ID -->
    [Inline: -Database='Employees', -Table='web', -ClientUsername='test', -ClientPassword='test',
employee_id=(Form_Param:'employee_id'), -Search]
        [Var_Set: 'theRec'=(RecID_Value)]
    [/Inline]
    <!-- Test if the user clicked the Update button -->
    [If: (Token_Value: 'Update') != '']
        <!-- Inline update -->
        [Inline: -Database='Employees', -Table='web', -ClientUsername='test', -ClientPassword='test',
-RecID=(Var:'theRec'), ReUseFormParams, -Update]
            [Include:'includes/inline_result.inc']
        [/Inline]
    <!-- Else the user clicked the Delete button -->
    [Else]
        <!-- Inline delete -->
        [Inline: -Database='Employees', -Table='web', -ClientUsername='test', -ClientPassword='test',
-RecID=(Var:'theRec'), -Delete]
            <!-- If the Delete action was successful, set a variable to 1 -->
            [If: (Error_CurrentError) == (Error_NoError)]
                [Var_Set: 'deleteOK'='1']
            [/If]
            [Include:'includes/inline_result.inc']
        [/Inline]
    [/If]
[/If]
```

11. Save the file.

B. Error Code and Message

1. Open inline_result.inc in the folder inline_method/exercise06/includes/ in a text editor.

2. Insert two conditional statements to display the correct error message to the user for the update and delete actions.

```
[If: (Error_CurrentError) == (Error_NoError)]
    <FONT COLOR="red">The record was successfully
    [If: (Lasso_CurrentAction) == 'Update']
        updated.
    [Else]
        deleted.
    [/If]
    </FONT>
[Else]
    Inline Error: [Error_CurrentError, ErrorCode], [Error_CurrentError]
[/If]
```

3. Save the file.

C. Including the Forms

1. Open detail.lasso in the folder inline_method/exercise06/ in a text editor.

2. Insert a conditional statement before the closing Body tag to display the search form if a record is deleted. Otherwise, display the detail form.

```
[If: (Var_Defined: 'deleteOK')]
        [Include:'includes/search_form.inc']
[Else]
        [Include:'includes/detail_form.inc']
[/If]
```

3. Save the file. Load <http://localhost/inline_method/exercise06/search.lasso> in a Web browser and test the solution.

Lasso 3 Hands-on Training, Session 3

Secure Login: Usernames and Passwords in FileMaker

Table of Contents

Lasso 3 Hands-on Training Session 3

Secure Login: Usernames and Passwords in FileMaker

by Kirk Bowman

Lasso Security can control access to databases. However, maintaining a large list of users in Lasso Security can be unweilding.
This session will demonstrate how create a secure login routine with usernames and passwords stored in FileMaker. This session will utilize techniques from Inline Lasso: The Preferred Method. Lasso tags to be discussed in this session include [Inline], -Token, [Token_Value], [If], [Var_Set], [Response_FilePath], [String_CountFields], [String_GetFields], [String_RemoveTrailing], [String_Concatenate], [Redirect_URL] and [Cookie_Set].

Biography

Kirk has been developing database solutions since 1991. In 1997 he began developing web solutions with a beta of FileMaker Pro 4.0 and CDML. Shortly thereafter he created his first web solution with Lasso 1.0--a nationwide benefits enrollment system for a forestry company.

Kirk is now president of MightyData, L.L.C. based in Dallas, Texas offering information technology design, development, and training services. He writes a monthly Lasso column for ISO FileMaker Magazine and is the author of several FileMaker and Lasso training courses. He is a jazz music fan and a former professional saxophonist.

Notes

Lasso 3 Hands-on Training, Session 3
Secure Login: Usernames and Passwords in FileMaker

Introduction

Lasso Security can control access to a database. However, sometimes it is necessary to control access to an entire Web site not just a database. This can be accomplished using realms on a Web server. Even so, it is preferable to control access to the Web site and the database using a comprehensive user list. If the user list is stored in a database (like FileMaker), the developer can build a custom security system and administrator interface for the user list.

This tutorial uses techniques from the previous two sessions, Inline Lasso: The Preferred Method, to control access to a Web site using a user list in a FileMaker database. Although the techniques from the previous sessions will be mentioned, refer to the materials for those sessions for a comprehensive overview of the following techniques.

- Setting a token in a button
- Testing whether a button was clicked
- Using inlines to search a database

To prepare for this tutorial, complete the following.

1. Put the Secure_Login folder at the root level of the Web server folder.
2. Configure Lasso and FileMaker 5 to access the Employees database.
3. In Lasso Security, create a user “test” with a password “test” and grant the Search permission.
4. The Secure Login Home Page can be viewed in a Web browser at <http://localhost/secure_login/default.html>.
5. The source code for the demo files is in the folder secure_login/demos/.

Basic Login

The login process begins when the user clicks the Login button to submit their username and password. When the Login button is clicked, two things occur. First, the Login button sets a token. Second, the login form is recursive--it calls itself as the response page. Below is the code for the login form.

```
<FORM ACTION=”index.lasso” METHOD=”post”>
    <TABLE BORDER=”0” CELLPADDING=”2” CELLSPACING=”2”>
        <TR>
            <TD><B>User Name:</B></TD>
```

```
            <TD><INPUT TYPE="text" NAME="username" SIZE="24"></TD>
        </TR>
        <TR>
            <TD><B>Password:</B></TD>
            <TD><INPUT TYPE="password" NAME="password" SIZE="24"></TD>
        </TR>
    </TABLE>
    <INPUT TYPE="submit" NAME="-Token.Login" VALUE="Login"> <INPUT TYPE="reset">
</FORM>
```

A conditional statement at the top of the login page tests whether the user clicked the Login button. If yes, an inline search is performed to verify the username and password. If no, the login form is included in the page.

```
[If: (Token_Value: 'Login') != '']
    <!-- Search the database for the username and password -->
[Else]
    Enter your user name and password.
    [Include: 'includes/login_form.inc']
[/If]
```

If the user clicked the Login button, an inline search is performed using the Username and Password form parameters entered in the login form. A conditional statement inside the Inline tags tests whether the resulting found count is equal to one. If the found count is one, the login is successful. If the found count is zero or greater than one, the login is unsuccessful.

```
[Inline: -Database='Employees', -Table='Web', -Op='equal', username=(Form_Param:'username'), -Op='equal',
password=(Form_Param:'password'), -ClientUsername='test', -ClientPassword='test', -Search]
    [If: (Found_Count) == '1']
        Your login was successful!
    [Else]
        <FONT COLOR="red">Your login was not valid.</FONT>
        [Include: 'includes/login_form.inc']
    [/If]
[/Inline]
```

Exercise 1 - Basic Login

The solution for Exercise 1 can be accessed under Solutions on the File Tags Home Page.

1. Open login_form.inc in the secure_login/exercise01/includes/ folder in a text editor.

2. Insert index.lasso as the Action attribute for the opening Form tag.

   ```
   <FORM ACTION="index.lasso" METHOD="post">
   ```

3. Insert a submit button after the closing Table tag to set a token when the Login button is clicked.

   ```
   <INPUT TYPE="submit" NAME="-Token.Login" VALUE="Login">
   ```

4. Save the file. Open index.lasso in the secure_login/exercise01/ folder in a text editor.

5. Insert a conditional statement after the opening Body tag to test whether the Login button was clicked.

```
[If: (Token_Value: 'Login') != '']
[Else]
Enter your user name and password.
[Include: 'includes/login_form.inc']
[/If]
```

6. Insert an inline after the opening [If] tag to search the Employees database for the username and password.

```
[Inline: -Database='Employees', -Table='Web', -Op='equal', username=(Form_Param:'username'),
-Op='equal', password=(Form_Param:'password'), -ClientUsername='test', -ClientPassword='test',
-Search]
[/Inline]
```

7. Insert a conditional statement after the opening [Inline] tag to test whether the found count form the search is one record.

```
[If: (Found_Count) == '1']
Your login was successful!
[Else]
<FONT COLOR="red">Your login was not valid.</FONT>
[Include: 'includes/login_form.inc']
[/If]
```

8. Save the file. Load <http://localhost/secure_login/exercise01/index.lasso> in a Web browser and test the solution.

Redirect to Another File

Once a user has successfully logged in, the next task if flow control. Where should the user go next? The answer to this question depends on the business requirements for each Web site. The question for this exercise is "How is the user redirected to a new file?"

The Redirect_URL tag in Lasso can redirect the user to a different file. The Redirect_URL tag requires Java to be installed. (For more information on installing Java for Mac OS, refer to Chapter 4: Requirements and Installation in the Lasso Reference Manual. For more information on installing Java for Windows, refer to Chapter 8: Java Installation in the Lasso 3.6.6 Addendum).

The Redirect_URL tag requires one parameter, the web URL to redirect the user to. Run the demo "Redirect" under Exercise 2 on the Secure Login Home Page. The following code redirects the Web browser to the CNN Web site.

```
[Redirect_URL: 'http://www.cnn.com']
```

In the tag above, the redirect path is hard coded. To make a Web site portable from one Web server to another, the redirect path is constructed using variables and string tags. The sequence to build a portable redirect path is below.

1. Set a variable to the current response file path.
2. Set a variable to the number of directories in the current response file path.
3. Set a variable to the name of the current response file.
4. Set a variable to the current response file directory.
5. Set a variable to full redirect path.

Run the demo "Building Redirect Path" under Exercise 2 on the Secure Login Home Page. The following variable is set to the current response file path using the [Response_FilePath] tag.

```
[Var_Set: 'path'=(Response_FilePath)]
```

The next variable is set to the number of directories in the "path" variable using the String_CountFields tag. The String_CountFields tag counts the number of "fields" in a string separated by the delimiter. In the "path" variable, each directory is a "field".

```
[Var_Set: 'count'=(String_CountFields: delimiter='/', (Var:'path'))]
```

The next variable is set to the name of the current response file using the String_GetField tag. The String_GetField tag retrieves a specific "field" by number from a delimited string. Using the "count" variable as the value for FieldNumber retrieves the last field in the "path" variable.

```
[Var_Set: 'remove'=(String_GetField: FieldNumber=(Var:'count'), delimiter='/', (Var:'path'))]
```

Next the path variable is reset to the current response file directory file using the String_RemoveTrailing tag. Since the "remove" variable is the last field in the "path" variable, removing the "remove" variable yields the path to the current directory.

```
[Var_Set: 'path'=(String_RemoveTrailing: Pattern=(Var:'remove'), (Var:'path'))]
```

Finally the "path" variable is reset to the full redirect path using the String_Concatenate tag. The first component is the protocol for the redirect, "http://". The next two components are the Server_Name and the current response file directory "path" variable. The final component is the name of the file.

```
[Var_Set: 'path'=(String_Concatenate: 'http://', (Server_Name), (Var:'path'), 'search.lasso')]
```

Now the path variable can be used as the sole parameter for the Redirect_URL tag.

```
[Redirect_URL: (Var:'path')]
```

Exercise 2 - Redirect to Another File

The solution for Exercise 2 can be accessed under Solutions on the File Tags Home Page.

1. Open index.lasso in the secure_login/exercise02/ folder in a text editor.

2. Set a variable after the opening Body tag to store the current response file path.

   ```
   [Var_Set: 'path'=(Response_FilePath)]
   ```

3. Set a second variable to store the number of directories in the current response file path.

   ```
   [Var_Set: 'count'=(String_CountFields: delimiter='/', (Var:'path'))]
   ```

4. Set a third variable to store to the name of current response file.

   ```
   [Var_Set: 'remove'=(String_GetField: FieldNumber=(Var:'count'), delimiter='/', (Var:'path'))]
   ```

5. Next, reset the "path" variable to store the current response file directory.

   ```
   [Var_Set: 'path'=(String_RemoveTrailing: Pattern=(Var:'remove'), (Var:'path'))]
   ```

6. Next, reset the "path" variable to store the full redirect path.

   ```
   [Var_Set: 'path'=(String_Concatenate: 'http://', (Server_Name), (Var:'path'), 'search.lasso')]
   ```

7. Insert a Redirect_URL tag to redirect the Web browser to the URL stored in the "path" variable.

   ```
   [Redirect_URL: (Var:'path')]
   ```

8. Save the file. Load <http://localhost/secure_login/exercise02/index.lasso> in a Web browser and test the solution.

Requiring a Login

For the login to be effective, the user must be required to login even when loading another page from the site in a Web browser. If this were possible, the user could bypass the login entirely. Since the Web is "stateless", it is necessary to store a piece of information in the user's browser to indicate if the user has successfully logged in. This information is stored in a cookie. The Set_Cookie tag, including all parameters, is below.

```
[Cookie_Set: 'Cookie_Name'='cookie_value', Expires='minutes', Path='path_name',
Domain='domain_name', Secure]
```

By using only the Cookie_Name and Path parameters in the Set_Cookie tag, a RAM cookie is created. A RAM cookie exists only in memory. In other words, it is not a file on the user's hard drive. When the user quits the browser, the cookie disappears. This allows the cookie to verify the user's login only during the current browser session.

Setting a cookie is similar to setting a token. The cookie is set on the first page and can then be retrieved on subsequent pages. However with a cookie, the value does not have to be carried from page to page. Under normal circumstances a RAM cookie remains in the user's browser until the browser is quit. (The user can turn off cookies in a Web browser. More on this issue in the section.)

Run the demo "Setting a Cookie" under Exercise 3 on the Secure Login Home Page. The Set_Cookie tag creates a cookie "test_cookie" with the value "The cookie was set!".

```
[Cookie_Set: Path='/', 'test_cookie'='The cookie was set!']
```

The Cookie tag displays the cookie contents on the next page.

```
[Cookie: 'test_cookie', EncodeNone]
```

The contents of a cookie can be tested using a conditional statement. Run the demo "Testing a Cookie" under Exercise 3 on the Secure Login Home Page. A cookie "sample_cookie" is created with the value "This is your cookie!". A conditional statement on the next page tests whether the cookie was set.

```
[If: (Cookie:'sample_cookie', EncodeNone)!='']
    Cookie succeeded!
[Else]
    Cookie failed!
[/If]
```

By testing the contents of a cookie, the flow of a Web site can be manipulated. By testing for a cookie to verify a successful login, the user can be redirected to the login page if the test fails.

Exercise 3 - Requiring a Login

The solution for Exercise 3 can be accessed under Solutions on the File Tags Home Page.

1. Open index.lasso in the secure_login/exercise03/ folder in a text editor.

2. Set a cookie after the second opening [If] tag to record a successful login.

```
[Cookie_Set: Path='/', 'def456'='Yes']
```

3. Save the file. Open search.lasso in the secure_login/exercise03/ folder in a text editor.

4. Insert a conditional statement after the opening Body tag to test the contents of the login cookie.

```
[If: (Cookie: 'def456') =='Yes']
Your login was successful!
[Else]
<!-- Else the user has not successfully logged in, Redirect to the login page -->
[/If]
```

5. Insert code after the [Else] tag to build a redirect path and then redirect the Web browser back to the login page.

```
[Var_Set: 'path'=(Response_FilePath)]
[Var_Set: 'count'=(String_CountFields: delimiter='/', (Var:'path'))]
[Var_Set: 'remove'=(String_GetField: FieldNumber=(Var:'count'), delimiter='/', (Var:'path'))]
[Var_Set: 'path'=(String_RemoveTrailing: Pattern=(Var:'remove'), (Var:'path'))]
[Var_Set: 'path'=(String_Concatenate: 'http://', (Server_Name), (Var:'path'), 'index.lasso')]
[Redirect_URL: (Var:'path')]
```

6. Save the file. Load <http://localhost/secure_login/exercise03/search.lasso> in a Web browser to try to bypass the login.

7. Load <http://localhost/secure_login/exercise03/index.lasso> in a Web browser and login.

Verifying Cookies are Enabled

Using cookies to verify a login is a good technique, if the user has cookies enabled in the Web browser. If a Web site requires cookies and cookies are not enabled, a potential problem exists.

It takes one page to set a cookie and a second page to test the contents of the cookie. Thus one login attempt is required to verify if cookies are enabled. If after one login attempt cookies are not enabled, an error message should be displayed.

The logic to test if cookies are enabled is below.

```
<!-- Set a RAM cookie to record if cookies are enabled -->
[Cookie_Set: Path='/', 'abc123'='Yes']

<!-- Test whether the user clicked the login button -->
[If: (Token_Value: 'Login') != '']

    <!-- Test whether cookies are enabled in the browser -->
    [If: (Cookie: 'abc123') != 'Yes']

    <!-- Else verify the login -->
    [Else]
        <!-- Inline search for the username and password -->
        <!-- Test whether the found count is one -->
        [If: (Found_Count) == '1']
            <!-- Redirect to search page -->

        <!-- Else the login was invalid -->
        [Else]

        [/If]
    [/If]

<!-- Else the user did not click the login button -->
[Else]
[/If]
```

Exercise 4 - Verifying Cookies are Enabled

The solution for Exercise 4 can be accessed under Solutions on the File Tags Home Page.

1. Open index.lasso in the secure_login/exercise04/ folder in a text editor.

2. Set a cookie after the opening Body tag to record if cookies are enabled.

   ```
   [Cookie_Set: Path='/', 'abc123'='Yes']
   ```

3. Insert a conditional statement before the opening Inline tag to test the contents of the "enabled" cookie.

   ```
   [If: (Cookie: 'abc123') != 'Yes']
   You must enable cookies in your Web browser to use this site.
   Consult your Web browser documentation for more information.
   [Include: 'includes/login_form.inc']
   ```

```
[Else]
```

4. Insert the closing [/If] tag after the closing Inline tag.

   ```
   [/If]
   ```

5. Save the file.

6. Turn off cookies in the Web browser and restart the browser.

7. Load <http://localhost/secure_login/exercise04/search.lasso> in the Web browser and login to test the code to verify cookies.

8. Turn on cookies in the Web browser. Load <http://localhost/secure_login/exercise04/search.lasso> in the Web browser and login again.

Lasso 3 Hands-on Training, Session 4
File Tags: Storing Value Lists in Text Files

Table of Contents

Lasso 3 Hands-on Training, Session 4

File Tags: Storing Value Lists in Text Files

by Kirk Bowman

FileMaker has the ability to store value lists with a database. Most other Lasso data sources do not directly support this feature. This session will demonstrate how to store value lists in text files with code flexible enough for any data source. Lasso tags to be discussed in this session include [Loop], [File_ReadLine], [File_GetLineCount], [File_Control], [File_CurrentError], [If], [File_Exists] and [EncodeURL].

Biography

Kirk has been developing database solutions since 1991. In 1997 he began developing web solutions with a beta of FileMaker Pro 4.0 and CDML. Shortly thereafter he created his first web solution with Lasso 1.0--a nationwide benefits enrollment system for a forestry company.

Kirk is now president of MightyData, L.L.C. based in Dallas, Texas offering information technology design, development, and training services. He writes a monthly Lasso column for ISO FileMaker Magazine and is the author of several FileMaker and Lasso training courses. He is a jazz music fan and a former professional saxophonist.

Notes

Lasso 3 Hands-on Training, Session 4
File Tags: Storing Value Lists in Text Files

Introduction

Lasso File Tags can read, create, edit and delete files. The files can be of any kind although text files are most common. Text files can be used in several ways. One of the most useful is storing value lists for popup menus, radio buttons and checkboxes on HTML forms.

Value lists stored in FileMaker can be accessed by Lasso; however, updating the value lists requires direct access to the database. Value lists in a text file can be updated using a standard FTP client. Other Lasso data sources do not have built-in value lists like FileMaker. Storing the value lists in text files works will all Lasso data sources.

The text files are return-delimited. In other words, each value is on a line by itself followed by a return. This is how value lists are entered in FileMaker. For example, below is a return-delimited value list of colors.

Blue
Green
Red
Purple
Orange

File Tags Module

Functionality in Lasso can be changed by added or removing different Lasso modules from the Lasso Modules folder. The Lasso Modules folder is located at the root level of the Web server folder on Mac OS and at the following path on Windows NT/2000, C://WinNT/system32/.

By default, the File Tags module is not installed in the Lasso Modules folder. To add the File Tags module, stop the Web server. Move the File Tags module from the Optional Modules folder to the Lasso Modules folder. Restart the Web server. The Optional Modules folder is installed in different locations depending on the operating system and Web server. Search the hard drive for "Optional Modules" to locate the folder.

File Tag Security

Once the File Tags module has been installed, File Tags permissions must be enabled for the

specific directory where File Tags will be used.

1. Put the File_Tags folder at the root level of the Web server folder.
2. Open Lasso Security in a Web browser at http://localhost/lasso/security.
3. Click on the File Tags Administration link toward the bottom of the page.
4. Add a new user with the following settings:
   Username: test
   Password: test
   Allowed Path: /file_tags/
   Permissions: Inspect Files, Read Files

The Username and Password will be used in Inline tags to authenticate access to the text files. The Allowed Path is the path from the root level of the Web server folder. The Inspect permission will allow Lasso to determine if a file exists. The Read permission will allow Lasso to read the contents of a text file line by line.

File Tags must be used with care to avoid giving users unauthorized access to files. It is recommended that the administrator create a specific directory for file tags to use and only allow access to that directory under File Tags Administration. Careless use of File Tags can allow users to download the raw code of LDML format files.

Reading a Text File

When using File Tags, optimum performance is achieved when Lasso reads the text file into memory and then accesses the data in memory to process the File Tags. This prevents Lasso from reading the file from disk for each File Tag processed.

The File_Control tags force Lasso to load a text file into memory the first time it is accessed. Each subsequent file access is read from memory. This occurs for all File Tags contained within the File_Control tags.

The simplest example of File Tags is to display the contents of a text file in a Web browser. Run the demo "Read File" under Exercise 1 on the File Tags Home Page. The following code reads the contents of the text file "countries.txt" and displays it in the Web browser.

```
[Var_Set: 'vl_file'='value_lists/countries.txt']
[Inline: -ClientUsername='test', -ClientPassword='test', -Nothing]
    [File_Control]
        [Loop: (File_GetLineCount: (Var:'vl_file', EncodeNone))]
        [File_ReadLine: (Var:'vl_file', EncodeNone), FileLineNumber=(LoopCount)]<BR>
        [/Loop]
    [/File_Control]
[/Inline]
```

The first variable tag stores the path to the text file. This is good programming style. The

code is more readable when referencing the text file using a variable. If the path to the text file changes or the name of the text file changes, the variable only has to be updated once no matter how many times it is used in the code.

The Inline tag authenticates File Tag access to the directory. The Inline tag has only three commands. -ClientUsername, -ClientPassword and -Nothing. -Nothing is used to evoke Lasso without accessing a database.

Next the File_Control tags load any text files into memory. The Loop tag starts a loop based on the number of lines in the file. The File_GetLineCount tag returns the number of lines. The File_GetLineCount tag requires one parameter, the path to the text file. The File_GetLineCount assumes the text file is return-delimited. (See the File_GetLineCount tag in the LDML Reference Database for information on delimiting characters.)

For each iteration of the loop, the File_ReadLine tag displays the contents of the current line. The File_ReadLine takes two parameters, the path to the text file and the line number to read. The LoopCount tag supplies the number of the current line. Finally the Loop, Inline and File_Control tags are closed.

Exercise 1 - Reading a File

The solution for Exercise 1 can be accessed under Solutions on the File Tags Home Page.

1. Create a new text file in a text editor. Type the following values in the text file.

   Red
   Blue
   Green
   Purple
   Orange

2. Save the text file as "colors.txt" in the folder file_tags/values_lists/.

3. Open exercise01.lasso in the file_tags folder in a text editor.

4. Set a variable after the opening Body tag to store the path to the text file.

   ```
   [Var_Set:'vl_file'='value_lists/colors.txt']
   ```

5. Insert Inline tags after the variable to authenticate File Tag access to the text file.

   ```
   [Inline: -ClientUsername='test', -ClientPassword='test', -Nothing]
   [/Inlne]
   ```

6. Insert File_Control tags inside the Inline tags to load the text file into memory.

```
[File_Control]
[/File_Control]
```

7. Insert Loop tags inside the File_Control tags to loop through the lines in the text file.

```
[Loop: (File_GetLineCount: (Var:'vl_file', EncodeNone))]
[/Loop]
```

8. Insert a File_ReadLine tag inside the Loop tags to display a line from the text file for each iteration of the loop.

```
[File_ReadLine: (Var:'vl_file', EncodeNone), FileLineNumber=(LoopCount)]<BR>
```

9. Save the file. Load <http://localhost/file_tags/exercise01.lasso> in a Web browser and test the solution.

Debugging File Tags

Lasso provides tags to help debug file tags. Similar to Error_CurrentError, the File_CurrentError tag returns error messages for file tags. However, File_CurrentError only works inside File_Control tags. As a rule, always place other file tags inside File_Control tags.

Run the demo "File Not Found" under Exercise 2 on the File Tags Home Page. The code below reports "-9984, File not found" because it attempts to read a line from states.txt which does not exist.

```
[File_Control]
    [File_ReadLine: (Var:'vl_file', EncodeNone), FileLineNumber='1']
    File Error: [File_CurrentError, ErrorCode], [File_CurrentError]
[/File_Control]
```

Run the demo "No Permission" under Exercise 2 on the File Tags Home Page. The same code reports "-9961, No permission" because the enclosing Inline uses an invalid user name "demo".

Run the demo "No Error" under Exercise 2 on the File Tags Home Page. The same code reports "0, No Error" because the line was read successfully from countries.txt.

Since Lasso can determine if an error occurs when executing a file tag, it is purdent to test whether Lasso can access a text file before actually executing the code. Run the demo "Error Trap 1" under Exercise 2 on the File Tags Home Page.

```
[File_Control]
[Var: 'temp'=(File_ReadLine: (Var:'vl_file', EncodeNone), FileLineNumber=1)]
[If: (File_CurrentError, ErrorCode) == '0']
    [Loop: (File_GetLineCount: (Var:'vl_file', EncodeNone))]
            [File_ReadLine: (Var:'vl_file', EncodeNone), FileLineNumber=(LoopCount)]<BR>
    [/Loop]
[Else]
    File Error: [File_CurrentError, ErrorCode], [File_CurrentError]
[/If]
[/File_Control]
```

The code above attempts to set a variable to the first line of states.txt. Since an error occurs (the error code does not equal zero), the error code and message are displayed. Run the demo "Error Trap 2" under Exercise 2 on the File Tags Home Page. The same code attempts to set a variable to the first line of countries.txt. Since no error occurs (the error codes equals zero), the contents of the file are displayed line by line.

Exercise 2 - Debugging File Tags

The solution for Exercise 2 can be accessed under Solutions on the File Tags Home Page.

1. Open exercise02.lasso in the file_tags folder in a text editor.

2. Set a variable after the opening [File_Control] tag to the first line of the text file.

   ```
   [Var: 'temp'=(File_ReadLine: (Var:'vl_file', EncodeNone), FileLineNumber=1)]
   ```

3. Insert an If-Else statement after the variable to test whether an error occurred reading the file.

   ```
   [If: (File_CurrentError, ErrorCode) == '0']
   [Else]
   [/If]
   ```

4. Insert a File_ReadLine tag after the opening [If] tag to display the contents the text file, line by line.

   ```
   Loop: (File_GetLineCount: (Var:'vl_file', EncodeNone))]
       [File_ReadLine: (Var:'vl_file', EncodeNone), FileLineNumber=(LoopCount)]<BR>
   [/Loop]
   ```

5. Insert code after the [Else] tag to display the error code and error message.

   ```
   File Error: [File_CurrentError, ErrorCode], [File_CurrentError]
   ```

6. Save the file. Load <http://localhost/file_tags/exercise02.lasso> in a Web browser and test

the solution.

Displaying a Popup Menu

Displaying a popup menu on a Web page requires inserting values from a text file into HTML. Before looking at the Lasso code, it is important to review standard HTML for a popup menu.

```
<SELECT NAME="selectName" SIZE="1">
    <OPTION VALUE="one">first</OPTION>
    <OPTION VALUE="two">second</OPTION>
    <OPTION VALUE="three">third</OPTION>
</SELECT>
```

The Select tags open and close the code for a poup menu. The Name attribute of the opening Select tag is the field name in the database.

The Option tags are used once for each value in the popup menu. The Value attribute of the opening Option tag is the value submitted to the database. The word(s) between the opening and closing Option tags is displayed in the popup menu in the Web browser.

Initializing Variables

Before building a popup menu in HTML, four variables are set. Run the demo "Initialize Variables" under Exercise 3 on the File Tags Home Page. The first two variables are set explicitly in the code.

```
[Var_Set: 'vl_folder'='value_lists']
[Var_Set: 'field_name'='first_name']
```

The "vl_folder" variable contains the path to the folder containing the text file.
The "field_name" variable contains the field name in the database. From the "field_name" variable, two more variables are created. The third variable is the data contained in the field.

```
[Var_Set:'field_value'=(Field:(Var:'field_name', EncodeNone))]
```

The code above uses the "field_name" variable as a parameter for the [Field] tag. In turn, the [Field] tag is a parameter for the [Var_Set] tag. The fourth variable is the path to the text file.

```
[Var_Set:'vl_file'=(String_Concatenate:(Var:'vl_folder', EncodeNone), '/',
 (Var:'field_name', EncodeNone), '.txt')]
```

The code above concatenates the "vl_folder" variable, a forward slash, the "field_name" variable, and the ".txt" extension. This code assumes the name of the text file is the same as

the field name. This is helpful for tracking which text file goes with which field.

Plugging in Variables

To build a popup menu in HTML, three items are needed.

1. The field name is inserted for the Name attribute for the opening Select tag.
2. A line from the text file is inserted for the Value attribute for each opening Option tag.
3. A line from the text file is inserted between the opening and closing Option tags.

Inserting the field name for the Name attribute is easy. Simply reference the variable "field_name" in the code. Inserting a line from the text file requires an extra step. Since the value changes for each line in the text file, it is necessary to set the line to a variable called "vl_item".

```
[Var_Set: 'vl_item'=(File_ReadLine: (Var:'vl_file', EncodeNone), FileLineNumber=(LoopCount))]
```

Run the demo "Plugging in Variables" under Exercise 3 on the File Tags Home Page. The "vl_item" variable is used inside a Loop to build the code for a popup menu.

```
<SELECT NAME="[Var:'field_name', EncodeNone]" SIZE="1">
    [Loop: (File_GetLineCount: (Var:'vl_file', EncodeNone))]
        [Var_Set:'vl_item'=(File_ReadLine: (Var:'vl_file', EncodeNone),  FileLineNumber=(LoopCount))]
        <OPTION VALUE="[Var:'vl_item', EncodeNone]">
        [Var:'vl_item', EncodeNone]
        </OPTION>
    [/Loop]
</SELECT>
```

Displaying Selected Value

The final step in building a popup menu is to show the user which value is currently in the field. Interface-wise, the Web browser displays this value in the popup menu.

Code-wise, HTML places the Selected attribute in the opening Option tag.

```
<OPTION VALUE="Operations" SELECTED>
```

To place the Selected attribute in the correct Option tag, a conditional statement is used. Run the demo "Displaying Selected Value" under Exercise 3 on the File Tags Home Page.

```
[If:(Var:'vl_item', EncodeNone)==(Var:'field_value', EncodeNone)]SELECTED[/If]>
```

The code above compares the value in the field with the value of the current line in the text file. If they match, the Selected attribute is inserted in the opening Option tag. Once the popup menu is displayed properly, the code to update the record is similar to the code in the previous tutorial, Inline Lasso: The Preferred Method.

Exercise 3 - Displaying Popup Menu

The solution for Exercise 3 can be accessed under Solutions on the File Tags Home Page. In this exercise the LDML code for the popup menu will be saved in an include file.

1. Create a new text file in a text editor. Type the following values in the text file.

   Engineering
   Finance
   Marketing
   Operations
   Personnel
   Printing
   Publishing
   Purchasing
   Sales
   Shipping

2. Save the text file as "group.txt" in the folder file_tags/values_lists/.

3. Open exercise03.lasso in the file_tags folder in a text editor.

4. Set a variable after the opening Body tag to store the path to the value list folder.

   ```
   [Var_Set: 'vl_folder'='value_lists']
   ```

5. Set a variable in the second cell of the fourth row of the table to store the name of the field.

   ```
   [Var_Set: 'field_name'='group']
   ```

6. Insert an [Include] tag after the variable to include the LDML code for the poup menu.

   ```
   [Include: 'includes/popup.inc']
   ```

7. Save the file. Open popup.inc in the file_tags/includes/ folder in a text editor.

8. Set a variable at the beginning of the file to store the data from the field.

```
[Var_Set:'field_value'=(Field:(Var:'field_name', EncodeNone))]
```

9. Set a variable after the "field_value" variable to store the path to the text file.

```
[Var_Set:'vl_file'=(String_Concatenate: (Var:'vl_folder', EncodeNone), '/',
(Var:'field_name', EncodeNone), '.txt')]
```

10. Insert the "field_name" variable for the Name attribute of the Select tag.

```
<SELECT NAME="[Var:'field_name', EncodeNone]" SIZE="1">
```

11. Set a variable after the opening [Loop] tag to store the value of the line from the text file.

```
[Var_Set:'vl_item'=(File_ReadLine: (Var:'vl_file', EncodeNone),  FileLineNumber=(LoopCount))]
```

12. Insert the "vl_item" variable for the Value attribute for the Option tag.

```
<OPTION VALUE="[Var:'vl_item', EncodeNone]">
```

13. Insert the "vl_item" variable between the opening and closing Option tags.

```
<OPTION VALUE="[Var:'vl_item', EncodeNone]">
        [Var:'vl_item', EncodeNone]
</OPTION>
```

14. Insert a conditional statement to place the SELECTED attribute in the opening Option tag when necessary.

```
<OPTION VALUE="[Var:'vl_item', EncodeNone]"
[If:(Var:'vl_item', EncodeNone)==(Var:'field_value', EncodeNone)]SELECTED[/If]>
```

15. Save the file. Load <http://localhost/file_tags/exercise03.lasso> in a Web browser and test the solution.

Displaying Radio Buttons

The methodology for building HTML code for radio buttons is the same as a popup menu. LDML is merged with HTML to create the code to display radio buttons. Below is HTML for three standard radio buttons.

```
<INPUT TYPE="radio" NAME="shift" VALUE="1"> 1
<INPUT TYPE="radio" NAME="shift" VALUE="2"> 2
<INPUT TYPE="radio" NAME="shift" VALUE="3"> 3
```

One Input tag is necessary for each radio button. All the radio buttons for the same field have the same Name attribute. The Value attribute is the data submitted to the database. The text following the Input tag is displayed in a Web browser as a label for the radio button.

To build one radio button from a text file, the field name is inserted as the Name attribute of the Input tag. The value of the line from the text file is inserted as the Value attribute. The value of the line is placed after the Input tag as a label.

To build multiple radio buttons, this process is repeated once for each line in the text file. Run the demo "Displaying Radio Buttons" under Exercise 4 on the File Tags Home Page.

```
[Loop: (File_GetLineCount: (Var:'vl_file', EncodeNone), FileEndOfLine='\r')]
    [Var_Set:'vl_item'=(File_ReadLine: (Var:'vl_file', EncodeNone), FileEndOfLine='\r',
FileLineNumber=LoopCount)]
    <INPUT TYPE="radio" NAME="[Var:'field_name', EncodeNone]" VALUE="[Var:'vl_item', Encode-
Name]">
    [Var:'vl_item', EncodeNone]
[/Loop]
```

The code above loops through the lines of the text file. For each line an Input tag is created with the correct Name and Value attributes using the "field_name" and "vl_item" variables. If the field in the database contains the value for one of the radio buttons, the CHECKED attribute is inserted in the Input tag using the following conditional statement.

```
[If:(Var:'field_value', EncodeNone)==(Var:'vl_item',EncodeNone)]CHECKED[/If]
```

Exercise 4 - Displaying Radio Buttons

The solution for Exercise 4 can be accessed under Solutions on the File Tags Home Page. In this exercise the LDML code for the radio button will be saved in an include file.

1. Create a new text file in a text editor. Type the following values in the text file.

   1
   2
   3

2. Save the text file as "shift.txt" in the folder file_tags/values_lists/.

3. Open exercise04.lasso in the file_tags folder in a text editor.

4. Set a variable in the second cell of the fifth row of the table to store the name of the field.

   ```
   [Var_Set: 'field_name'='shift']
   ```

5. Insert an [Include] tag after the variable to include the LDML code for the radio buttons.

   ```
   [Include: 'includes/radiobutton.inc']
   ```

6. Save the file. Open radiobutton.inc in the file_tags/includes/ folder in a text editor.

7. Insert the "field_name" variable for the Name attribute of the Input tag.

   ```
   <INPUT TYPE="radio" NAME="[Var:'field_name', EncodeNome]" VALUE="">
   ```

8. Insert the "vl_item" variable for the Value attribute of the Input tag.

   ```
   <INPUT TYPE="radio" NAME="[Var:'field_name', EncodeNome]" VALUE="[Var:'vl_item', EncodeNone]>
   ```

9. Insert a conditional statement to add the CHECKED attribute into the opening Option tag when necessary.

   ```
   <INPUT TYPE="radio" NAME="[Var:'field_name', EncodeNome]" VALUE="[Var:'vl_item', EncodeNone]"
   [If:(Var:'field_value', EncodeNone)==(Var:'vl_item',EncodeNone)]CHECKED[/If]>
   ```

10. Insert the "vl_item" variable and a Line Break after the Input tag.

    ```
    [Var:'vl_item', EncodeNone]<BR>
    ```

11. Save the file. Load <http://localhost/file_tags/exercise04.lasso> in a Web browser and test the solution.

Displaying Checkboxes

The HTML to display a checkbox is almost identical to the HTML for a radio button. Compare the following Input tags. Only the Type attribute is different.

```
<INPUT TYPE="radio" NAME="nice_traits" VALUE="Helpful"> Helpful
<INPUT TYPE="checkbox" NAME="nice_traits" VALUE="Helpful"> Helpful
```

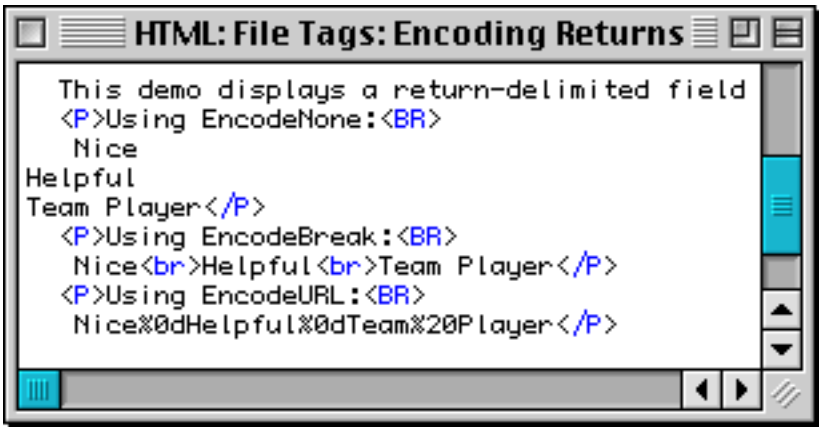
The LDML to build checkboxes is very similar to the LDML for radio buttons. The reason for the difference is a checkbox field (at least in FileMaker) may contain multiple values

delimited by returns.

Encoding Returns

When Lasso displays a checkbox field, the entire contents including the returns are displayed. This is not evident if you display the field without any encoding. (Encoding translates extended characters into character representations. See Chapter 8: Character Sets and Translation in the Lasso Reference Manual for more information on encoding.)

Run the demo "Encoding Returns" under Exercise 5 on the File Tags Home Page. The results of the three types of encoding are evident when viewing the source code of the demo page in a Web browser.



The following code displays the field contents without any encoding.

```
[Field: 'traits', EncodeNone]
```

The following code displays the field contents using EncodeBreak. The EncodeBreak keyword instructs Lasso to translate any returns in the field into Line Breaks. The HTML tag for a Line Break is
.

```
[Field: 'traits', EncodeBreak]
```

The following code displays the field contents using EncodeURL. The EncodeURL keyword instructs Lasso to translate extended characters in the field into URL-safe codes. The URL encoding for a return is %0d.

```
[Field: 'traits', EncodeURL]
```

Using EncodeURL to Insert CHECKED Attribute

Encoding is important because it is necessary to test whether a field contains a value to insert the CHECKED attribute. The CHECKED attribute is required for an Input tag to display the checkbox as "checked" in the Web browser.

Run the demo "Contains Blue" under Exercise 5 on the File Tags Home Page. The following conditional statement evaluates True whether the variable contains Blue or Blue-Green. This is because Lasso recognizes a partial string as a match.

```
[Var_Set:'colors'='Blue']
[If: (Var:'colors')>>'Blue']CHECKED[/If]

[Var_Set:'colors'='Blue-Green']
[If: (Var:'colors')>>'Blue']CHECKED[/If]
```

To avoid checking a checkbox with a partial match, EncodeURL is used to pad values with the URL encoding for a return, %0d. The String_Concatenate tag is used to place %0d before the first value and after the last value. By padding values with %0d, the following conditional statements will evaluate True for Blue but False for Blue-Green.

```
[Var_Set:'colors'=(String_Concatenate:'%0d','Blue','%0d')]
[If: (Var:'colors',EncodeNone)>>(String_Concatenate:'%0d','Blue','%0d')]CHECKED[/If]
[If: (Var:'colors',EncodeNone)>>(String_Concatenate:'%0d','Blue-Green','%0d')]CHECKED[/If]
```

Compare the LDML to insert CHECKED into a radio button versus into a checkbox.

```
[Var_Set:'field_value'=(Field:(Var:'field_name', EncodeNone))]
[If:(Var:'field_value', EncodeNone)==(Var:'vl_item',EncodeNone)]CHECKED[/If]

[Var_Set:'field_value'=(String_Concatenate:'%0D',(Field:(Var:'field_name'),EncodeURL),'%0D')]
[If:(Var:'field_value', EncodeNone)>>(String_Concatenate:'%0D',(Var:'vl_item',EncodeURL),'%0D')]CHECKED[/If]
```

Exercise 5 - Displaying a Checkbox

The solution for Exercise 5 can be accessed under Solutions on the File Tags Home Page. In this exercise the LDML code for the checkbox will be saved in an include file.

1. Create a new text file in a text editor. Type the following values in the text file.

   Nice
   Energetic
   Helpful
   Team Player

2. Save the text file as "traits.txt" in the folder file_tags/values_lists/.

3. Open exercise05.lasso in the file_tags folder in a text editor.

4. Set a variable in the second cell of the fifth row of the table to store the name of the field.

```
[Var_Set: 'field_name'='traits']
```

5. Insert an [Include] tag after the variable to include the LDML code for the radio buttons.

```
[Include: 'includes/radiobutton.inc']
```

6. Save the file. Open checkbox.inc in the file_tags/includes/ folder in a text editor.

7. Set a variable at the beginning of the file to store the data value from the field padded by %0d.

```
[Var_Set:'field_value'=(String_Concatenate:'%0D',(Field:(Var:'field_name'),EncodeURL),'%0D')]
```

8. Insert the "field_name" variable for the Name attribute of the Input tag.

```
<INPUT TYPE="checkbox" NAME="[Var:'field_name', EncodeNone]" VALUE="">
```

9. Insert the "vl_item" variable for the Value attribute of the Input tag.

```
<INPUT TYPE="checkbox" NAME="[Var:'field_name', EncodeNone]"
VALUE="[Var:'vl_item', EncodeNone]>
```

10. Insert a conditional statement to add the CHECKED attribute into the opening Option tag when necessary.

```
<INPUT TYPE="radio" NAME="[Var:'field_name', EncodeNome]" VALUE="[Var:'vl_item', EncodeNone]"
[If:(Var:'field_value',EncodeNone)>>(String_Concatenate:'%0D',(Var:'vl_item',EncodeURL),'%0D')]
CHECKED[/If]>
```

11. Insert the "vl_item" variable and a Line Break after the Input tag.

```
[Var:'vl_item', EncodeNone]<BR>
```

12. Insert an Input tag to allow the user to make the contents of checkbox field "Empty". (To use this checkbox in the Web browser, check only the Empty checkbox. Do not check any other checkboxes.)

```
<INPUT TYPE="checkbox" NAME="[Var:'field_name']" VALUE=""> (Empty)
```

13. Save the file. Load <http://localhost/file_tags/exercise05.lasso> in a Web browser and test the solution.

Lasso 5 Feature Tour

LDML 5 Syntax Tour

Table of Contents

LDML 5 Syntax Tour

by Jim Van Heule

LDML 5 Syntax walks through some of the new tags available in Lasso 5. new string and math and array functionality will be covered with actual syntax of working examples.

Biography

Jim has been working with database design since 1987. He brought his database skills to the Internet in 1997 using Lasso technology. He has held senior level positions at several prominent Internet companies including product manager for Blue World Communications, Inc. VH Publications, Inc. was founded to bring Lasso solutions to the Internet and continues to work in partnership with Internet firms to help bring their database requirements to the world wide web.

Notes

LDML 5 Syntax Tour

Introduction

One of Lasso's strongest features has been its rich tag base that handles and manipulates text and math strings. Lasso Professional 5 (LP5) has added new features that significantly improves upon its base set of tags.

Data can now be handled several ways that may not seem obvious to current Lasso 3 developers. Three constructs are now available when handling string and math data including Symbols, Members along with the familiar Substitution tags. Each of these new constructs will be reviewed.

With the introduction of the new symbol and member tag constructs, it became apparent there needed to be a way to output the results. We will explore the new [Output] along with some new enhancements to the existing [Variable] tags.

This paper will provide a brief overview of the new tags available. Then you will be walked through several exercises to see how each tag works with real code. There is a folder on the Lasso Summit CD that contain a series of Lasso pages providing examples of the new tag syntax and comparing it to existing methods to get the same results using LDML 3 when possible. The folder paths are shown below. To use these example pages, load these files onto your Web server running Lasso 5 once it becomes available.

LDML 5 Syntax Examples
/Hands-On_Training/VanHeule/LDML5/

Data Output

[Output] is a new tag to output the result of any LDML calculation or subtag. There isn't much of a need for this tag using Lasso 3 syntax, but it becomes apparent that this tag will become prevalent when using Lasso 5. Below are some examples of how [Output] will be used later in this paper.

Using LDML 5 Math Symbols
```
[Output:5 + 6]
Result = 11
```

Using LDML 5 String Symbols
```
[Output:'This is ' + 'a string.']
 Result = This is a string.
```

The [Lasso_Comment]...[/Lasso_Comment] tag set has been renamed more appropriately to [Output_None]...[/Output_None]. Anything residing within the opening and closing portions of these tags is not output to the Web page in the client's browser. This is commonly used around areas of code that set variables or do long calculations thus removing a lot of blank space in the viewable source code. Another use it to comment your lasso code within these tags since the commented text is not viewable by the Web client as demonstrated below

```
[Output_None]
Set the color variable
[Var:'Color'='Blue']
[/Output_None]
```

Result: (nothing is displayed)

Variable Operators

Variables are a common way to handle the various data types in Lasso Professional 5. There are new ways to work with and set variables that help to enhance the coding experience. In LDML 3, we needed to use the tag [Var_Set] to set a variable and a second tag [Variable] to display the variable. These tags still are functional in LDML 5, but now you can both set and retrieve variables with new functionality added to the [Variable] tag. Notice below how the new LDML 5 method uses the same [Var] tag to both set and then display the variable. The [Variable] tag uses the equal (=) symbol to set values to the variable.

LDML 3 Method

```
[Var_Set:'Test'='My three sons.']
[Variable:'Test']
```

Result: My three sons.

LDML 5 Method

```
[Variable:'Test'='My three sons.']
[Variable:'Test']
```

Result: My three sons.

The dollar symbol ($) is a second symbol available that returns the value of a variable. By using the new $ symbol to display variables, code length will be reduced considerably. The following examples all output the same result.

```
[Var:'Test'='Today']
[Variable:'Test']
Result: Today

[Var:'Test']
Result: Today

[$Test]
```

Result: Today

String Symbols

String Symbols offer a new method to manipulate string data. Basically, you can now add, subtract and use many commonly used math-type symbols to manipulate text strings. These new symbols allow for much shorter syntax that tends to be more intuitive which will increase coding efficiency. The list displayed below is an excerpt from the LP5 Language Guide providing a snapshot of the newly available symbols.

Symbol	Description
+	Concatenates two strings.
-	Deletes a substring. The first occurrence of the right parameter is deleted from the left parameter.
*	Repeats a string. The right parameter should be a number.
=	Assigns the right parameter to the variable designated by the left parameter.
+=	Concatenates the right parameter to the value of the left parameter and assigns the result to the variable designated by the left parameter.
-=	Deletes the right parameter from the value of the left parameter and assigns the result to the variable designated by the left parameter.
*=	Repeats the value of the left parameter and assigns the result to the variable designated by the left parameter.
>>	Returns True if the left parameter contains the right parameter as a substring.
==	Returns True if the parameters are equal.
!=	Returns True if the parameters are not equal.
<	Returns True if the left parameter comes before the right parameter alphabetically.
<=	Returns True if the left parameter comes before the right parameter alphabetically or if the parameters are equal.
>	Returns True if the left parameter comes after the right parameter alphabetically.
>=	Returns True if the left parameter comes after the right parameter alphabetically or if the parameters are equal.

Concatenating (adding) two strings together is easy using the new (+) symbol. Variables and other LDML string results can also be concatenated together in this manner.

LDML 5 Method
[Output:'This is ' + 'a string.']
Result: This is a string.

LDML 3 Method
[String_Concatenate:'This is ', 'a string.']
Result: This is a string.

Removing a portion of a string is just as simple using the (-) symbol as demonstrated below. This tag only removes the first found matching string. There is no real LDML 3 equivalent, but there is code that can mimic the same results as shown below.

LDML 5 Method
[Output:'This is a string string.' - 'string']
Result: This is a string.

LDML 3 Method
```
[String_Remove: StartPosition=(String_FindPosition='String','This is a string string.'),
EndPosition=(Math_Add:(String_FindPosition: Find='String','This is a string
string.'),(String_Length:'String')),
'This is a string string.']
Result: This is a string.
```

NOTE: [String_RemoveLeading] and [String_RemoveTrailing] also works if the removed string is at the beginning or ending part of the string respectively. In this example [String_Replace] would remove both both instances of "string" providing an incorrect result.

Repeating (multiplying) a string can be accomplished using the (*) symbol. Examples show the ease in LDML 5 versus LDML 3.

LDML 5 Method
```
[Output:'String.' *4]
Result = String. String. String. String.
```

LDML 3 Method
```
[String_Concatenate:'String. ', 'String. ', 'String. ', 'String. ']
Result = String. String. String. String.
```

The equal (=) symbol is familiar to us from LDML 3. It has the added functionality of working on its own or with other LDML tags accepting symbols.

LDML 5 Method
```
[Var:'test'='This is a string'][Var:'test']
Result = This is a string
```

LDML 3 Method
```
[Var_Set:'test'='This is a string'][Var:'test']
Result = This is a string
```

Concatenating a new parameter to an existing variable saves the step of concatenating a nested variable to itself. The (+=) symbol concatenates the right parameter to the value of the left variable.

LDML 5 Method
```
[Var: 'Test'='This is a']
[(Var:'Test')+=' String.']
[Var:'Test']
Result = This is a String.
```

LDML 3 Method
```
[Var_Set: 'Test'='This is a']
[Var_Set: 'Test'=(String_Concatenate:(Var:'Test'), ' String.')]
[Var: 'Test']
Result = This is a String.
```

You can also delete a parameter from an existing variable using the (-=) symbol. The right parameter is deleted from the value of the left variable. A LDML 3 solution equivalent is possible but not very practical as shown below.

LDML 5 Method

```
[Var: 'Test'='This is a string string.']
[(Var:'Test')-='string']
[Var:'Test']
Result = This is a string.
```

LDML 3 Method

```
[Var_Set: 'Test'='This is a string string.']
[Var_Set:'Test'=(String_Remove:
StartPosition=(String_FindPosition:
Find='string',(Var: 'Test')),
EndPosition=EndPosition=(Math_Add:(String_FindPosition:
Find='string',(Var:'Test')),(String_Length:'string')),
(Var:'Test'))][Var: 'Test']
Result = This is a string.
```

Repeating a string to an existing variable is accomplished using the (*=) symbol. It repeats the value of the left parameter and assigns the result to the variable designated by the left parameter.

LDML 5 Method

```
[Var: 'Test'='String. ']
[(Var:'Test')*=4]
[Var:'Test']
Result = String. String. String. String.
```

LDML 3 Method

```
[Loop:4]
[Var_Set:'Test'=(String_Concatenate:(Var:'Test'),'String.')]
[/Loop]
[Var:'Test']
Result = String. String. String. String.
```

By using the (>>) symbol, a string can be tested to see if it contains a specified string. It returns True if the left parameter contains the right parameter as a substring.

LDML 5 Method

```
['This is a string.'>>'string']
Result = true
```

LDML 3 Method

```
[If:(String_FindPosition:Find='string', 'This is a string.')>0]true[Else]false[/If]
Result = true
```

The (==) symbol works the same as what is already available for conditional tags [If], but can be used directly to compare two strings. It returns True if the strings are equal.

LDML 5 Method

```
['This is a string.'=='This is a string.']
Result = true
```

LDML 3 Method

```
[If:'This is a string.'=='This is a string.']true[Else]false[/If]
Result = true
```

The (!=) symbol works the same as what is already available for conditional tags [If], but can be used directly to compare two strings. It returns True if the strings are not equal.

LDML 5 Method
[‘This is a string.’!=’This is a string.’]
Result = false

LDML 3 Method
[If:’This is a string.’!=’This is a string.’]true[Else]false[/If]
Result = false

The (<) symbol works on strings similar to what is already available for math functions compared with conditional tags [If] and can be used directly to compare two strings. It returns True if the left string comes before the right string alphabetically.

LDML 5 Method
[‘Apple’<’Pear’]
Result = true

LDML 3 Method
There is no LDML 3 equivalent.

The (<=) symbol works on strings similar to what is already available for math functions compared with conditional tags [If] and can be used directly to compare two strings. It returns True if the left string comes before or is equal to the right string alphabetically.

LDML 5 Method
[‘Apple’<=’Apple’] and [‘Apple’<=’Pear’]
Result = true and true

LDML 3 Method
There is no LDML 3 equivalent.

The (>) symbol works on strings similar to what is already available for math functions compared with conditional tags [If] and can be used directly to compare two strings. It returns True if the left string comes after the right string alphabetically.

LDML 5 Method
[‘Apple’>’Pear’]
Result = false

LDML 3 Method
There is no LDML 3 equivalent.

The (>=) symbol works on strings similar to what is already available for math functions compared with conditional tags [If] and can be used directly to compare two strings. It returns True if the left string comes after or is equal to the right string alphabetically.

LDML 5 Method
[‘Apple’>=’Apple’] and [‘Apple’>=’Pear’]
Result = true and false

LDML 3 Method
There is no LDML 3 equivalent.

String Member Tags

String Member tags is an additional new method to manipulate string data. Member tags are easily recognized since all begin with the (->) construct. The tags bring new features to LDML many that are not currently available in LDML 3. The list displayed below is an excerpt from the LP5 Language Guide providing a snapshot of the newly available member tags.

Tag	Description
[String->Append]	Casts the parameters to strings and appends them to the string. Modifies the string and returns no value. Requires one string parameter.
[String->BeginsWith]	Returns True if the string begins with the parameter. Requires a single string parameter.
[String->Contains]	Returns True if the string contains the parameter as a substring. Requires a single string parameter.
[String->EndsWith]	Returns True if the string ends with the parameter. Requires a single string parameter.
[String->Find]	Returns the position at which the first parameter is found within the string or 0 if the first parameter is not found within the string. Requires a single string parameter.
[String->Get]	Returns a specific character from the string. Requires a single integer parameter.
[String->Length]	Returns the number of characters in the string.
[String->RemoveLeading]	Removes all instances of the parameter from the beginning of the string. Modifies the string and returns no value. Requires a single string parameter.
[String->RemoveTrailing]	Removes all instances of the parameter from the end of the string. Modifies the string and returns no value. Requires a single string parameter.
[String->Split]	Splits the string into an array of strings based on the delimiter specified in the first parameter. This tag does not modify the string, but returns the new array. Requires a single string parameter.
[String->SubString]	Returns a substring. The start of the substring is defined by the first parameter and the length of the substring is defined by the second parameter. Requires two integer parameters.
[String->Trim]	Removes all white space from the start and end of the string. Modifies the string in place and returns no value.

The (->Append) member tag casts the parameters to strings and appends them to a variable. The modified variable returns no value until is called later.

LDML 5
```
[Var:'Test'='A String']
[(Var:'Test')->append:' along.']
[(Var:'Test')]
Result = String along.
```

LDML 3
```
[Var_Set:'Test'='A String']
[Var_Set:'Test'=(String_Concatenate:(Var:'Test'),' along.')]
[(Var:'Test')]
Result = String along.
```

The (->BeginsWith) member tag returns True if the string begins with the parameter specified.

LDML 5
```
[Output:'String'->BeginsWith:'Str']
Result = true
```

LDML 3
There is no LDML 3 equivalent.

The (->Contains) member tag returns True if the string contains the parameter as a substring.

LDML 5
```
[Output:'String'->Contains:'tri']
Result = true
```

LDML 3
There is no LDML 3 equivalent.

The (->EndsWith) member tag returns True if the string ends with the parameter specified.

LDML 5
```
[Output:'String'->EndsWith:'ing']
Result = true
```

LDML 3
```
[String_EndsWith:'String',Find='ing']
Result = true
```

The (->Find) member tag returns the position at which the first parameter is found within the string or 0 if the first parameter is not found within the string.

LDML 5
```
[Output:'1234567890'->Find:'456']
Result = 4
```

LDML 3
```
[String_FindPosition: Find='456','1234567890']
Result = 4
```

The (->Get) member tag returns a specific character from the string.

LDML 5
```
[Output:'A String'->get:4]
Result = t
```

LDML 3
```
[String_Extract:StartPosition=4,EndPosition=4,'A String']
Result = t
```

The (->Length) member tag returns the number of characters in the string.

LDML 5
```
[Output:'A String'->Length]
Result = 8
```

LDML 3
[String_Length:'A String']
Result = 8

The (->RemoveLeading) member tag returns all instances of the parameter from the beginning of the variable. The variable result is not output until the variable is later called.

LDML 5
[Var:'Test'='xxxString']
[Output:(Var:'Test')->RemoveLeading:'x']
Result = String

LDML 3
[Var_Set:'Test'='xxxString']
[Var_Set:'Test'=(String_RemoveLeading: Pattern='x','xxxString')]
[Var:'Test']
Result = String

The (->RemoveTrailing) member tag removes all instances of the parameter from the end of the variable. The variable result is not output until the variable is later called.

LDML 5
[Var:'Test'='Stringxxx']
[Output:(Var:'Test')->RemoveTrailing:'x']
Result = String

LDML 3
[Var_Set:'Test'='xxxString']
[Var_Set:'Test'=(String_RemoveLeading: Pattern='x','Stringxxx')]
[Var:'Test']
Result = String

The (->Split) member tag splits the string into an array of strings based on the delimiter specified in the first parameter. This tag does not modify the string, but returns the new array. Requires a single string parameter.

LDML 5
[Output:'1/2/3/4/5'->Split:'/']
Result = array: (1), (2), (3), (4), (5)

LDML 3
[List] tags have similar functionality, but I would not consider them a true equivalent.

The (->SubString) member tag returns a substring. The start of the substring is defined by the first parameter and the length of the substring is defined by the second parameter. Two integer parameters are required.

LDML 5
[Var:'test'='A long string']
[Output:(Var:'Test')->SubString:3,4]
Result = long

LDML 3
```
[Var:'test'='A long string']
[String_Extract: StartPosition=3, EndPosition=6,(Var:'Test')]
Result = long
```

The (->Trim) member tag removes all white space from the start and end of the string. Modifies the string in place and returns no value.

LDML 5
```
[Var:'Test'=' A long String ']
[(Var:'Test')->Trim]
-[Var:'Test']-
Result = -A long String-
```

LDML 3
```
[Var_Set:'Test'=(String_RemoveLeading:
Pattern=' ',(String_RemoveTrailing:
Pattern=' ',(Var:'Test')))]
-[Var:'test']-
Result = -A long String-
```

New String Tag

A new string tag has been added that searchs and replaces regular expressions (also called Grep). These tags can be useful for globally finding and replacing common errors during input and replacing old tag syntax with new tags.

LDML 5 Method
```
[Var:'Test'='(Client_Address, EncodeNone)(Client_IP, EncodeNone)']
[String_ReplaceRegExp:(Var:'Test'),-Search='\(([A-Za-z_]+),',-Replace='\1:']
```

This expression looks for any string that starts with (then followed by alpha characters that end with a comma. It then replaces the comma with a colon.

```
Start = (Client_Address, EncodeNone)(Client_IP, EncodeNone)
Result = (Client_Address: EncodeNone)(Client_IP: EncodeNone)
```

LDML 3 Method
There is no LDML 3 tag equivalent.

Math Symbols

Math Symbols offers a new method to manipulate math functions and data. Basically, you can now add, subtract and use many commonly used math-type symbols to manipulate numbers without having to use the [Math_...] type syntax..

These new symbols allow for a much shorter syntax that tends to be more intuitive which will increase coding efficiency. The list displayed below is an excerpt from the LP5 Language Guide providing a snapshot of some newly available symbols.

Symbol	Description
+	Adds two numbers.
-	Subtracts the right parameter from the left parameter.
*	Multiplies two numbers.
/	Divides the left parameter by the right parameter.
=	Assigns the right parameter to the variable designated by the left parameter.
+=	Adds the right parameter to the value of the left parameter and assigns the result to the variable designated by the left parameter.
-=	Subtracts the right parameter from the value of the left parameter and assigns the result to the variable designated by the left parameter.
*=	Multiplies the value of the left parameter by the value of the right parameter and assigns the result to the variable designated by the left parameter.
/=	Divides the value of the left parameter by the value of the right parameter and assigns the result to the variable designated by the left parameter.
==	Returns true if the parameters are equal.
!=	Returns true if the parameters are not equal.
<	Returns true if the left parameter is less than the right parameter.
<=	Returns true if the left parameter is less than or equal to the right parameter.
>	Returns true if the left parameter is greater than the right parameter.
>=	Returns true if the left parameter is greater than or equal to the right parameter.

The (+) symbol adds together two numbers. This tag works on its own or with other LDML tags.

LDML 5 Method
```
[Output:5 + 6]
Result = 11
```

LDML 3 Method
```
[Math_Add:5,6]
Result = 11
```

SPECIAL NOTE: The (+) symbol is used for both string and math manipulation. In LDML 5, it now becomes a necessity to code strings using quotes and numeric values without quotes. Notice that when we take the above LDML 5 code and wrap the numeric values with quotes, the numbers are concatenated together instead of added. This is true for most symbol tags.

```
[Output:'5' + '6']
Result = 56
```

The (-) symbol subtracts the right parameter from the left parameter.

LDML 5 Method
```
[Output:10-6]
Result = 4
```

LDML 3 Method
```
[Math_Sub:10-6]
Result = 4
```

The (*) symbol multiplies two numbers.

LDML 5 Method
```
[Output:2*4]
Result = 8
```

LDML 3 Method
```
[Math_Mult:2,4]
Result = 8
```

The (/) symbol divides the left parameter by the right parameter.

LDML 5 Method
```
[Output:10.0/3]
Result = 3.333333
---------------------------------------------------------------------
[Var:'a'=10.0][Var:'b'=3] [(Var:'a')/(Var:'b')]
Result = 3.333333
```

LDML 3 Method
```
[Math_Div:10.0/3]
Result = 3.333333
---------------------------------------------------------------------
[Var:'a'=10.0][Var:'b'=3] [Math_Div:(Var:'a'),(Var:'b')]
Result = 3.333333
```

The (=) symbol assigns the right parameter to the variable designated by the left parameter. This is already a commonly used symbol in LDML 3.

LDML 5 Method
```
[Var:'test'=10+5][Var:'test']
Result = 15
```

LDML 3 Method
```
[Var_Set:'test'=(Math_Add:10,5)][Var:'test']
Result = 15
```

The (+=) symbol adds the right parameter to the value of the left variable and assigns the result to the variable. There is no output until the variable is later called.

LDML 5 Method
```
[Var: 'Test'=10]
[(Var:'Test')+=5]
[Var:'Test']
Result = 15
```

LDML 3 Method
```
[Var_Set: 'Test'=10]
[Var_Set: 'Test'=(Math_Add:(Var:'Test'),5)]
[Var: 'Test']
Result = 15
```

The (-=) symbol deletes the right parameter from the value of the left variable and assigns the result to the variable. There is no output until the variable is later called.

LDML 5 Method
```
[Var: 'Test'=10]
[(Var:'Test')-=6]
[Var:'Test']
Result = 4
```

LDML 3 Method
```
[Var_Set: 'Test'=10]
[Var_Set:'Test'=(Math_Sub:(Var:'Test'),6)]
[Var: 'Test']
Result = 4
```

The (*=) symbol multiplies the value of the left variable by the right parameter and assigns the result to the variable. There is no output until the variable is later called.

LDML 5 Method
```
[Var: 'Test'=10]
[(Var:'Test')*=4]
[Var:'Test']
Result = 40
```

LDML 3 Method
```
[Var_Set:'Test'=10]
[Var_Set:'Test'=(Math_Mult:(Var:'Test'),4)]
[Var:'Test']
Result = 40
```

The (/=) symbol divides the value of the left variable by the value of the right parameter and assigns the result to the variable. There is no output until the variable is later called.

LDML 5 Method
```
[Var:'Test'=10.0]
[(Var:'Test')/=3]
[Var:'Test']
Result = 3.333333
```

LDML 3 Method
```
[Var_Set:'Test'=10.0]
[Var_Set:'Test'=(Math_Div:(Var:'Test')/3)]
[Var:'Test']
Result = 3.333333
```

The (==) symbol returns True if the parameters are equal.

LDML 5 Method
```
[4+6==10]
Result = true
```

LDML 3 Method
```
[If:(Math_Add:4,6)==10]true[Else]false[/If]
Result = true
```

The (!=) symbol returns True if the parameters are not equal.

LDML 5 Method
```
[10!=10]
Result = false
```

LDML 3 Method
[If:10!=10]true[Else]false[/If]
Result = false

The (<) symbol returns True if the left parameter is less than the right parameter.

LDML 5 Method
[6<10.6]
Result = true

LDML 3 Method
[If:6<10]true[Else]false[/If]
Result = true

The (<=) symbol returns True if the left parameter is less than or equal to the right parameter.

LDML 5 Method
[10<=10] and [10<=9]
Result = true and false

LDML 3 Method
[If:10<=10]true[Else]false[/If] and [If:10<=9]true[Else]false[/If]
Result = true and false

The (> symbol returns True if the left parameter is greater than the right parameter.

LDML 5 Method
[10>9]
Result = true

LDML 3 Method
[If:10>9]true [Else]false [/If]
Result = true

The (>=) symbol returns True if the left parameter is greater than or equal to the right parameter.

LDML 5 Method
[10>=10] and [9>=10]
Result = true and false

LDML 3 Method
[If:10>=10]true[Else]false[/If] and [If:9>=10]true[Else]false[/If]
Result = true and false

Decimal Member Tags

Decimal Member tags are used to manipulate decimal portions of a numeric value. All member tags are easily recognized since all begin with the (->) construct. The tags bring new features to LDML many that are not currently available in LDML 3. The list displayed below is an excerpt from the LP5 Language Guide providing a snapshot of some newly available member tags.

Tag	Description
-Precision	The number of decimal points of precision that should be output. Defaults to 6.
-GroupChar	The character which should be used for thousands grouping. Defaults to empty.
-Scientific	Set to True to force output in exponential notation. Defaults to False so decimals are only output in exponential notation if required.

The (->SetFormat:-Precision) member tag sets the number of decimal points of precision that should be output. It defaults to 6.

LDML 5
```
[Var:'test'=12.3456789] [(Var:'Test')->(SetFormat:-Precision=3)]
Result = 12.346
```

LDML 3
There is no LDML 3 tag equivalent.

The (->SetFormat:-GroupChar) member tag sets character which will be used for thousands grouping. It defaults to empty.

LDML 5
```
[Var:'test'=12345.6789]
[(Var:'Test')->(SetFormat:-GroupChar=',')]
Result = 12,345.678900
```

LDML 3
There is no LDML 3 tag equivalent.

The (->SetFormat:-Scientific) member tag can be set to True to force output in exponential notation. The tag defaults to False so decimals are only output in exponential notation if required.

LDML 5
```
[Var:'test'=12345.6789]
[(Var:'Test')->(SetFormat:-Scientific='True')]
Result = 1.234568e+004
```

LDML 3
There is no LDML 3 tag equivalent.

Array

An array is a sequence of values that are stored and retrieved by numeric index. The values stored in an array can be of any LDML data type. An array can be set in the following manner. Arrays can be output easily using a variable call to allow the developer to view the array in its raw form.

LDML 5
```
[Var:'Days'=(Array:'Sunday','Monday','Tuesday','Wednesday','Thursday','Friday','Saturday')]]
[Var:'Days']
Result = array: (Sunday), (Monday), (Tuesday), (Wednesday), (Thursday), (Friday), (Saturday)
```

The member tag (Array->Get) returns an item from the array. Accepts a single integer parameter identifying the index of the item to be returned. Defaults to the last item in the array. This tag can be used as the left parameter of an assignment operator to set an element of the array.

```
LDML 5
[Var:'Days'=(Array:'Sunday','Monday','Tuesday','Wednesday','Thursday','Friday','Saturday')]]
[(Var:'Days')->(Get:3)]

Result = Tuesday
```

The member tag (Array->Find) returns an array of elements that match the parameter. Accepts a single parameter of any data type.

```
LDML 5
[Var:'Days'=(Array:'Sunday','Monday','Tuesday','Wednesday','Thursday','Friday','Saturday')]]
[(Var:'Days')->(Find:'Wednesday')]
Result = array: (Wednesday)
```

The member tag (Array->Insert) inserts a value into the array. Accepts a single parameter that is the value to be inserted and an optional integer parameter identifying the index of the location where the value should be inserted. Defaults to inserting the parameter at the end of the array. Returns no value.

```
LDML 5
[Var:'Days'=(Array:'Sunday','Monday','Wednesday','Thursday','Friday','Saturday')]]
[(Var:'Days')->(Insert:'Tuesday',3)]
[Var:'Days']
Result = array: (Sunday), (Monday), (Tuesday), (Wednesday), (Thursday), (Friday), (Saturday)
```

The member tag (Array->Last) returns the last item in the array.

```
LDML 5
[Var:'Days'=(Array:'Sunday','Monday','Tuesday','Wednesday','Thursday','Friday','Saturday')]]
[(Var:'Days')->(Last)]
Result = Saturday
```

The member tag (Array->Merge) merges an array parameter into the array. Accepts n array parameter and three integer parameters that identify which items from the array parameter should be inserted into the array. Defaults to inserting the entire array parameter at the end of the array. Returns no value.

```
LDML 5
[Var:'Days'=(Array:'Sunday','Monday','Wednesday','Thursday','Friday','Saturday')]]
[Var:'Dictionary'=(Array:'Tudor','Tuesday','Tuff','Tuft','Tug','Tuition')]]
[(Var:'Days')->(Merge:$Dictionary)] [Var:'Days']
Result = array: (Sunday), (Monday), (Wednesday), (Thursday), (Friday), (Saturday), (Tudor), (Tuesday),
(Tuff), (Tuft), (Tug), (Tuition)
```

Merge Parameters
First The array which is to be merged, the source array.
Second The index in the destination array where the elements of the source array should be inserted. Optional, defaults to the end of the destination array.
Third The index in the source array of the first element which should be inserted into the destination array. Optional, defaults to 1.
Fourth The number of elements from the source array to insert into the destination array. Optional, defaults to all elements from the third parameter to the end of the source array.

LDML 5
```
[Var:'Days'=(Array:'Sunday','Monday','Wednesday','Thursday','Friday','Saturday')]]
[Var:'Dictionary'=(Array:'Tudor','Tuesday','Tuff','Tuft','Tug','Tuition')]]
[(Var:'Days')->(Merge:$Dictionary,3,2,1)] [Var:'Days']
Result = array: (Sunday), (Monday), (Tuesday), (Wednesday), (Thursday), (Friday), (Saturday)
```

The member tag (Array->Remove) removes an item from the array. Accepts a single integer parameter identifying the index of the item to be removed. Defaults to the last item in the array. Returns no value.

LDML 5
```
[Var:'Days'=(Array:'Sunday','Monday','Tuesday','Tuesday','Wednesday','Thursday','Friday','Saturday')]]
[(Var:'Days')->(Remove:4)]
[Var:'Days']
Result = array: (Sunday), (Monday), (Tuesday), (Wednesday), (Thursday), (Friday), (Saturday)
```

The member tag (Array->RemoveAll) removes any elements that match the parameter from the array. Accepts a single parameter of any data type. Returns no value.

LDML 5
```
[Var:'Days'=(Array:'Sunday','Monday','Tuesday','Tuesday','Tuesday','Wednesday','Thursday','Friday','Saturday')]]
[(Var:'Days')->(RemoveAll:'Tuesday')]
[Var:'Days']
Result = array: (Sunday), (Monday), (Wednesday), (Thursday), (Friday), (Saturday)
```

The member tag (Array->Size) returns the number of elements in the array.

LDML 5
```
[Var:'Days'=(Array:'Sunday','Monday','Tuesday','Wednesday','Thursday','Friday','Saturday')]]
[(Var:'Days')->(Size)]
Result = 7
```

The member tag (Array->Sort) reorders the elements of the array in alphabetical or numerical order. Accepts a single boolean parameter. Sorts in ascending order by default or if the parameter is True and in descending order if the parameter is False.

LDML 5
```
[Var:'Days'=(Array:'Sunday','Monday','Tuesday','Wednesday','Thursday','Friday','Saturday')]]
[(Var:'Days')->(Sort:1)]
[Var:'Days']
Result = array: (Friday), (Monday), (Saturday), (Sunday), (Thursday), (Tuesday), (Wednesday)
```

Conclusion

This paper gave a sampling of some of the new tags we can expect from Lasso professional 5. The Symbol and member tags have the greatest potential for changing the fundamental ways we write LDML code. The intuitive nature of the new syntax should allow novice Lasso developers a faster learning curve, while allowing advanced Lasso coders the ability to write faster, cleaner code.

Upgrading Techniques and Considerations

Table of Contents

Upgrading Techniques and Considerations

by Jim Van Heule

Conversion of Lasso 3 code over to Lasso Professional 5 can be a daunting task. This paper will take a look at some of the things that need to be considered and provides some examples on how to deal with some of the syntax issues.

Biography

Jim has been working with database design since 1987. He brought his database skills to the Internet in 1997 using Lasso technology. He has held senior level positions at several prominent Internet companies including product manager for Blue World Communications, Inc. VH Publications, Inc. was founded to bring Lasso solutions to the Internet and continues to work in partnership with Internet firms to help bring their database requirements to the world wide web.

Notes

Upgrading Techniques and Considerations

Introduction

Lasso Professional 5 (LP5) provides significantly more tools and enhancements compared to the current version of Lasso 3. Even with these substantial changes, many of the current Lasso tags also work in LP5. Some changes are necessary to bring Lasso to the next level, such as syntax changes and tag revisions.

Converting current Lasso solutions to LP5 will require careful thought and consideration to achieve the desired working effect. Various techniques will be covered to better prepare developers for the inevitable Web site conversion.

Comma Conversion

The most significant code conversion will require simple syntax changes. This most likely will cause the greatest amount of distress for developers unless a good converter is constructed that takes most syntax changes into consideration.

Commas are no longer allowed after tag names. For example, [Server_Date, Long] is no longer supported and will cause a Lasso error. Tags like this will need to have the comma replaced by a colon to work properly. The most common tags that use this type of syntax are [Server_Date], [Server_Day], [Server_Time] and [Error_CurrentError]. Below are examples of these tags in their current form and then after proper conversion for LP5.

Current LDML 3	**LDML 5**
[Server_Date, ShortY2K]	[Server_Date: ShortY2K]
[Server_Day, Long]	[Server_Day: Long]
[Server_Time, Short]	[Server_Time: Short]
[Error_CurrentError, ErrorCode]	[Error_CurrentError: ErrorCode]

Other tags might also cause a Lasso error when encoding is used. This will only occur with tags that currently require commas to be used after the tag and just before the encoding keyword. A list of tags that will be effected by this type of syntax error is listed below however this list is not all-inclusive. Though many of these tags probably will not use this type of syntax, they must be considered in a conversion solution because it would only take a few of these in a large solution to create a conversion nightmare.

Current LDML 3	**LDML 5**
[Client_Address, EncodeNone]	[Client_Address: -EncodeNone]
[Client_IP, EncodeBreak]	[Client_IP: -EncodeBreak]
[Client_Type, EncodeURL]	[Client_Type: -EncodeURL]
[Date_GetCurrentDate, EncodeRaw]	[Date_GetCurrentDate: -EncodeRaw]

```
[DB_LayoutNameItem, EncodeNone]
[DB_NameItem, EncodeNone]
[DirectoryNameItem, EncodeNone]
[Lasso_DataSourceModuleName, EncodeNone]
[Layout_Name, EncodeNone]
[RepeatingValueItem, EncodeNone]
[SearchFieldItem, EncodeNone]
[SearchOperatorItem, EncodeNone]
[SearchValueItem, EncodeNone]
[SortFieldItem, EncodeNone]
[SortOrderItem, EncodeNone]
[Token_Value, EncodeNone]
[Table_Name, EncodeNone]
```

```
[DB_LayoutNameItem: -EncodeNone]
[DB_NameItem: -EncodeNone]
[DirectoryNameItem: -EncodeNone]
[Lasso_DataSourceModuleName: -EncodeNone]
[Layout_Name: -EncodeNone]
[RepeatingValueItem: -EncodeNone]
[SearchFieldItem: -EncodeNone]
[SearchOperatorItem: -EncodeNone]
[SearchValueItem: -EncodeNone]
[SortFieldItem: -EncodeNone]
[SortOrderItem: -EncodeNone]
[Token_Value: -EncodeNone]
[Table_Name: -EncodeNone]
```

Exercise 1 - Comma Conversion of Lasso Tags

This exercise will use regular expressions (also called Grep) to globally replace improper comma syntax with a colon. This exercise requires BBEdit on the Mac or HomeSite on the PC.

1. With LP5 running, open a browser and load /Hands-On_Training/VanHeule/Upgrading/update_comma.lasso. You should receive a comma syntax error. This page has an example of all the various tags mentioned previously that might contain a comma causing this type of error.

2. Open the file, /Hands-On_Training/VanHeule/Upgrading/update_comma.lasso, in either BBEdit or HomeSite.

3. Mac Only: In BBEdit, enter **APPLE-F** to open the Find Window. Click the **Use Grep** checkbox, enter the following find criteria and then click **Find All**.

   ```
   \[([A-Za-z_]+),
   ```

   PC Only: In HomeSite, enter **Shift-CTRL-F** to open the Extended Find Window. Click the **Regular Expressions** checkbox, enter the following find criteria and click **Find**.

   ```
   \[([A-Za-z_]+),
   ```

   This expression searches for anything that starts with [and is followed by a string of alpha characters that might also contain an underscore and finally ends with a comma. It doesn't care what follows the comma.

4. There should be nineteen in the resulting found set.

5. Next, we will do a global find and replace to convert all of the code on the page.

Mac Only: In BBEdit, enter **APPLE-F** to open the Find Window. Click the **Use Grep** checkbox, enter the following find criteria.

```
\[([A-Za-z_]+),
```

Enter the following replace criteria and click **Replace All**.

```
[\1:\2
```

PC Only: In HomeSite, enter **Shift-CTRL-F** to open the Extended Find Window. Click the **Regular Expressions** checkbox, enter the following find criteria.

```
\[([A-Za-z_]+),
```

Enter the following replace criteria and click **Replace All**.

```
[\1:
```

6. Save this document as /Hands-On_Training/VanHeule/Upgrading/update_comma_converted.lasso and open it up in your browser. You should now be able to view the page with no error messages.

Though this exercise converted only one page, BBEdit and HomeSite both have the ability to globally convert multiple pages and even entire Web sites.

Exercise 2 - Comma Conversion of Nested Lasso Tags

This exercise is very similar to the first, except that it will find and replace any nested Lasso tags with the comma syntax error. It will again use regular expressions (also called Grep) to globally replace improper comma syntax with a colon. This exercise requires BBEdit on the Mac or HomeSite on the PC.

1. With LP5 running, open a browser and load /Hands-On_Training/VanHeule/Upgrading/update_nested.lasso. You should receive a comma syntax error. This page has an example of all the various nested tags mentioned previously that might contain a comma causing this type of error.

2. Open the file, /Hands-On_Training/VanHeule/Upgrading/update_nested.lasso, in either BBEdit or HomeSite.

3. Mac Only: In BBEdit, enter **APPLE-F** to open the Find Window. Click the **Use Grep** checkbox, enter the following find criteria and then click **Find All**.

```
\(([A-Za-z_]+),
```

PC Only: In HomeSite, enter **Shift-CTRL-F** to open the Extended Find Window. Click the **Regular Expressions** checkbox, enter the following find criteria and click **Find**.

```
\(([A-Za-z_]+),
```

This expression searches for anything that starts with (and is followed by a string of alpha characters that might also contain an underscore and ends with a comma. It doesn't care what follows the comma.

4. There should be twelve in the resulting found set.

5. Next, do a global find and replace to convert all code on the page.

   Mac Only: In BBEdit, enter **APPLE-F** to open the Find Window. Click the **Use Grep** checkbox, enter the following find criteria.

   ```
   \(([A-Za-z_]+),
   ```

   Enter the following replace criteria and click **Replace All**.

   ```
   (\1:\2
   ```

   PC Only: In HomeSite, enter **Shift-CTRL-F** to open the Extended Find Window. Click the **Regular Expressions** checkbox, enter the following find criteria.

   ```
   \(([A-Za-z_]+),
   ```

   Enter the following replace criteria and click **Replace All**.

   ```
   (\1:
   ```

6. Save this document as /Hands-On_Training/VanHeule/Upgrading/update_nested_converted.lasso and open it up in your browser. You should now be able to view the page with no error messages.

Exercise 3 - Comma Conversion Using Lasso Regular Expressions

Lasso Professional 5 can also be used to perform the same regular expression functions as what can be found in BBEdit and HomeSite using a new tag like the one in the following example. This is not so much an exercise but more an example of coding possibilities using the new regular expression tag.

```
[String_ReplaceRegExp:(Var:'A'),-Search='search_expression',-Replace='replace_expression')]
```

The following code uses two Lasso regular expression tags to produce the same results as what was originally done in HomeSite or BBEdit.

```
[Var:'theFile'=(String_ReplaceRegExp:(Var:'theFile'),-Search='\[([A-Za-z_]+),',-Replace='\1:')]
[Var:'theFile'=(String_ReplaceRegExp:(Var:'theFile'),-Search='\(([A-Za-z_]+),',-Replace='\1:')]
```

An example of this in action can be tested by pointing your browser at /Hands-On_Training/VanHeule/Upgrading/update_auto.lasso. This page takes the errant code from /Hands-On_Training/VanHeule/Upgrading/update_combined.lasso and converts it using the regular expression tags. It saves the page as /Hands-On_Training/VanHeule/Upgrading/update_combined_converted.lasso and displays the result.

Keyword Hyphens

In LP5, keyword names will always begin with a hyphen. Current solutions that do not use hyphens still work without error, but it would be a good idea to start using hyphens. Future Lasso versions might not support keywords that do not start with a hyphen.

Current Lasso 3

```
[Inline:
    Database-'test',
    Table='web',
    Keyword='Z26n',
    Search]
[/Inline
```

LP5 Converted

```
[Inline:
    -Database='test',
    -Table='web',
    -Keyword='Z26n',
    -Search]
[/Inline]
```

Encoding Keywords

Encoding within nested tags will now default to -EncodeNone that should significantly increase coding efficiency since less code will be necessary to write. In most cases, this change should have no effect on current solutions unless an encoding tag other than -EncodeNone was nested. Coding examples are provided below.

Acceptable Lasso 3 Code That Will Work In LP5

```
[String_Concatenate: (Var:'a', EncodeNone), '[Tag]', (Var:'b', EncodeNone), EncodeNone]
```

Shortened LP5 Code With Identical Results

```
[String_Concatenate: (Var:'a'), '[Tag]', (Var:'b'), -EncodeNone]
```

Acceptable Lasso 3 Code That Will Work In LP5

```
[String_Concatenate: (Var:'a', EncodeURL), '[Tag]', (Var:'b', EncodeNone), EncodeNone]
```

Shortened LP5 Code With Identical Results

```
[String_Concatenate: (Var:'a', -EncodeURL), '[Tag]', (Var:'b'), -EncodeNone]
```
(the nested -EncodeURL still remains.)

Some developers place encoding keywords at the beginning of nested tag statements. Encoding keywords only have an effect on the nested tags if they are placed after the encoding tag. This type of syntax will have to be rewritten to have the same desired result. Notice in the Lasso 3 code below the placement of EncodeNone at the beginning of the tag structure. This would not work in LP5. A sample of corrected code for LP5 follows.

Lasso 3 Code That Will Not Work Properly With LP5

```
[String_Concatenate: EncodeNone, (Var:'a', EncodeNone), '[Tag]', (Var:'b', EncodeNone),]
```

Shortened LP5 Code With Expected Results

```
[String_Concatenate: (Var:'a'), '[Tag]', (Var:'b'), -EncodeNone]
```

Else If

The current [Else:If:] syntax is no longer supported in LP5. The "Else, If" construct itself, though, has not been eliminated. Instead the [Else] tag has been enhanced to handle the old tag requirements.

Common Lasso 3 [If:Else:] Code That Is Non-Functional In LP5

```
[If: (Var:'A'=='Blue')] The color is blue.
[Else:If: (Var:'A')=='Green']    The color is green.
[Else] The color Is neither blue nor green.
[/If]
```

The required conversion is actually fairly simple and can be done by replacing all found [Else:If: with [Else:.

Converted For LP5

```
[If: (Var:'A'=='Blue')] The color is blue.
[Else: (Var:'A')=='Green']    The color is green.
[Else] The color Is neither blue nor green.
[/If]
```

Using what was learned converting comma syntax errors, the Lasso regular expression tag can again be used to globally change [If:Else:] tags to the proper format. The following code can be used to convert existing Lasso 3 code to the new LP5 format. Notice the second tag takes into account the space sometimes used in the [If:Else] statement.

```
[Var:'theFile'=(String_ReplaceRegExp:(Var:'theFile'),-Search='(Else:If:)',-Replace='Else:')]
[Var:'theFile'=(String_ReplaceRegExp:(Var:'theFile'),-Search='(Else: If:)',-Replace='Else:')]
```

Post Inline

The [Post_Inline] tag has been replaced by a more functional tag appropriately named [Schedule_Create]. It might be possible to create global replace schemes for simple [Post_Inline] coding, but, in most instances, it would be worthwhile to convert this tag by hand while taking advantage of the advanced functionality of the [Schedule_Create] tag. This session does not have the time to cover the conversion to this method, but you can view the new syntax code style below.

New LP5 Event Schedule Syntax

```
[Schedule_Create:
    -Start=(Date, Defaults to Today),
    -End=(Defaults to Never),
    -URL=(URL to Execute),
    -Repeat=(True/False, Defaults to True),
    -Restart=(True/False, Defaults to True),
    -Delay=(Minutes),
    -UserName=(Optional),
    -Password-(Optional)]
```

File and Logging Tags

File and logging tags work as expected in Lasso Professional 5, as long as LP5 is running on the same machine as the Web server. At the time of this writing, it is not clear how File and Logging tags will work on a system where LP5 and the Web Server reside on separate machines.

Numbers Versus Text Manipulation

All current Lasso 3 tags work as expected when manipulating numbers and text, and no conversion should be necessary. It should be noted that the new LP5 syntax regards any text to be within quotes and numbers to not be enclosed within quotes. The examples below demonstrate this much easier than it is to describe.

Quoted Numbers Interpreted As Text Concatenation

```
[Output: '1' + '2']
Result: 12
```

Non-Quoted Numbers Treated As Math Addition

[Output: 1 + 2]
Result: 3

Math and Date Precision

The math and date tags now use 64-bit precision. Any solution that might have depended on the lesser precision of Lasso 3 can be effected.

Loop Precision

The [Loop] limitation of 1,000 iterations has been removed making infinate loops possible. Solutions depending on the 1,000-loop limitation will need to be modified.

Macros

Macros are no longer supported in Lasso Professional 5, and there is no easy upgrade path to follow. The most logical approach is to convert existing macros to the new Custom Tags.

Tag Name Changes

There are approximately 40 tags that have had a name change to better promote the consistence of LDML. None of these changes should have an effect on the conversion process because LP5 will support both tag naming conventions. Here is an example of some of the tag name changes.

LDML 3	**LDML 5**
-AddError	-ResponseAddError
-AnyError	-ResponseAnyError
-ClientUsername	-Username
[DB_NameItem]	[Database_NameItem]
[Lasso_Comment]	[Output_None]
[Lasso_SessionID]	[Lasso_UniqueID]

FileMaker Pro

For those developers currently using the FileMaker Pro Remote datasource module, there will be no considerations necessary to make calls to your FMP databases. Those using the local FMP datasource (Mac only) will have to convert their solution to meet the requirements of the FMP Remote datasource. This is covered better in the Lasso 3.6 FMP Modules Guide.pdf, but here is a list of things effected.

- Field-level logical operators are not supported.
- Script command tags using .back are not supported.
- FMP graphic conversion is not supported.

Conclusion

There are many things to consider when upgrading from Lasso 3 to Lasso Professional 5 including syntax changes. As LP5 nears completion, more will be known about the upgrade path and tools available to assist in this process. For now, this paper should help provide a starting point of what to expect and some methods to help ease the conversion process.

Lasso Summit

2001 Developer's Conference

Conference CD

consulting FileMaker & Lasso training

Design and development services:

- Database Design
- Web Site Design
- Project Management
- FileMaker Development
- Lasso Development
- AppleScript Development
- SQL Database Integration
- Palm Handheld Integration

Public and private training classes:

- Introduction to FileMaker
- Intermediate Relational Design
- Philosophy of Scripting I
- Philosophy of Scripting II
- Lasso: Apprentice Wizardry
- Lasso: Security Wizardry
- Lasso: E-commerce Wizardry

"This system works like a dream. I'd say it's cut our time in half. What's more, we were able to eliminate a staff position and reduce our overhead." — *Leiann Staggs, Sunbelt Power Controls*

"What was once a horror for me is now a simple matter with the FileMaker-based system that MightyData shaped for us. It has dramatically improved the quality of service and communication we can provide students, and at a reduced cost to our taxpayers." — *Debbie Marlowe, Independence Public Schools*

"Kirk is awesome. Now I'm ready to make a killer FileMaker solution!" — *Barry Evleth, Ingram Micro*

"You exceeded my expectations in both quality and content!" — *Melyssa St. Michael, UltraFit*

"These guys know their stuff, but more importantly, they know how to teach." — *Bruce Newlin, AFM-Dawn Computers*

"Kirk Bowman is gifted with the art of instructing and teaching." — *Kari Rhame, Clear Creek ISD*

Database Building & Training

Call 800-287-0845

or see our web site: www.mightydata.com

info@mightydata.com • Dallas, Texas