Lasso Developer Conference

Chicago, September 18-21, 2008

LassoSoft

Boost customer confidence with our **highly reliable** and **secure** solutions.

anu internet services
hosting infrastructure specialists

visit us at: **www.anu.net**

STABLE - FAST - SECURE

Benefit from our new range of *Flexible Managed Web Services*

- New: higher capacity packages
- Dedicated hardware resources
- Increased package flexibility
- Full administrative access
- Secure firewalled Linux OS
- MySQL 5 database server
- Web based administration
- Cost effective: scale as you grow

Also available:

- Managed email hosting
- Domain portfolio management
- Scalable hosting
- Disaster recovery solutions

ANU Internet Services are your hosting infrastructure specialists.

We provide fast, stable and secure Lasso and PHP hosting services on our Managed Web Services plans. Our personal approach results in lasting customer relationships. Our Scalable Hosting and Disaster Recovery solutions enable your business to grow.

Our services are competitively priced for both European and North American markets.

Visit us at www.anu.net

Copyright

Trademarks

Disclaimer

The feature set and details about the features of Lasso 9 are subject to change. LassoSoft is presenting information about this future product at this conference in order to provide you with advance notice of the anticipated feature set. However, changes may be required once feedback is received from beta testers and the quality assurance team.

Non-Disclosure Agreement

Most of the information about Lasso 9 is not yet publicly available. We ask that details about Lasso 9 be kept under non-disclosure until LassoSoft, LLC is able to make a formal public announcement about the software.

Welcome

We would like to welcome you to the Lasso Developer Conference for 2008. We are excited to see both new and familiar developers coming together for a few days to learn more about web programming, build new business connections, and find out what's new in the world of Lasso.

We hope that the conference provides you with valuable new ideas about how you can get more out of Lasso and related technologies. And, we hope to provide you with an in-depth look at the future of Lasso.

This binder includes the papers for each of the talks that will be given during the conference as well as an overview of the presentations which LassoSoft will be providing. Please find the schedule on the next page. Most of the speakers will also be hosting a "round table" on the last day of the conference. This will provide you with an opportunity to speak in person about the topics that were presented in the main sessions.

We would like to thank all of the sponsors of this event. You will find advertisements for them in this binder, on the t-shirts and around the conference center. Please consider supporting these sponsors if you have a chance. We could not put on the developer conference without them.

We encourage you to get the most out of the conference by asking questions and talking with all of the developers and LassoSoft employees present. We have a great collection of talented speakers this year presenting papers on a range of topics, but some of the greatest resources in the Lasso community will be sitting side by side with you in the audience.

Thank you for supporting Lasso. We look forward to many more years powering some of the best web sites around.

Kerry Adams, Kyle Jessup, and Fletcher Sandbeck
Owners, LassoSoft, LLC

Lasso Developer Conference
Chicago, September 18-21, 2008

All of the Lasso Developer Conference events will be held at the University Center Conference Chicago unless otherwise noted. This page provides an overview of the schedule. The following two pages provide details for each day of the conference.

Daily Schedule

Breakfast – Breakfast will be provided in the Lake Room each morning at 8:00 a.m. All attendees are invited to join the speakers and LassoSoft employees for a working breakfast.

Morning Sessions – Morning sessions will be in the Lake Room starting at 9:00 a.m. A snack will be provided at the 10:00 a.m. break.

Lunch – Lunch will be provided for all attendees in The Lake Room each day from 11:30 a.m. to 1:00 p.m. We would like to thank our lunch sponsor Anu Internet Services.

Afternoon Sessions – Afternoon sessions will be in the Lake Room starting at 1:00 p.m. A snack will be provided at the 3:30 p.m. break.

Sunday Round Tables

Round Tables – Round tables will be held on Sunday in the Lake Room from 9:00 a.m until lunch at 11:30 a.m.

Closing – The conference will be closed Sunday afternoon at 1:00 p.m.

Special Events

Keynote – The conference keynote will be Thursday morning at 9:00 a.m. Kerry Adams of LassoSoft will give the keynote address.

LassoSoft Reception – LassoSoft is hosting a reception for all attendees Thursday night at 5:30 p.m. in The Great Room. The reception will include appetizers and a cash bar.

LPA Event – LassoSoft is hosting an event for LPA members on Saturday night at 7:00 p.m. Registration for this event has ended. If you are an LPA member please contact Kerry Adams or write to info@lassosoft.com for more information.

Lasso Developer Conference
Chicago, September 18-21, 2008

All events in the Lake Room at University Center Conference Chicago unless otherwise noted.

Thursday, September 18, 2008

8:00 - 9:00	Breakfast
9:00 - 10:00	**LassoSoft Keynote – Kerry Adams with Fletcher Sandbeck and Kyle Jessup**
10:00 - 10:30	Snack Break
10:30 - 11:30	**LassoSoft – Lasso 9 Introduction – Kyle Jessup and Fletcher Sandbeck**
11:30 - 11:45	**Sponsor – Anu Internet Services – Chris Wik**
11:45 - 1:00	Lunch
1:00 - 2:00	**Knop – Johan Solve**
2:00 - 2:30	Break
2:30 - 3:30	**LassoFusebox – Tami Williams**
3:30 - 4:00	Snack Break
4:00 - 5:00	**All Your Base Are Belong To Us – Bil Corry**
5:30 - 9:30	**LassoSoft Reception (The Great Room)**

Friday, September 19, 2008

8:00 - 9:00	Breakfast
09:00 - 10:00	**LassoSoft – Lasso 9 for Beginners – Fletcher Sandbeck**
10:00 - 10:30	Snack Break
10:30 - 11:30	**L-Migrator – Brian Loomis**
11:30 - 1:00	Lunch – Sponsored by **Anu Internet Services**
1:00 - 2:00	**Server Side / Client Optimization – Jason Huck**
2:00 - 2:30	Break
2:30 - 3:30	**Using Lasso to Create Dynamic PDF documents – Jolle Carlestam**
3:30 - 4:00	Snack Break
4:00 - 5:00	LassoSoft – **Lasso 9 Applications – Kyle Jessup and Fletcher Sandbeck**

Lasso Developer Conference
Chicago, September 18-21, 2008

All events in the Lake Room at University Center Conference Chicago unless otherwise noted.

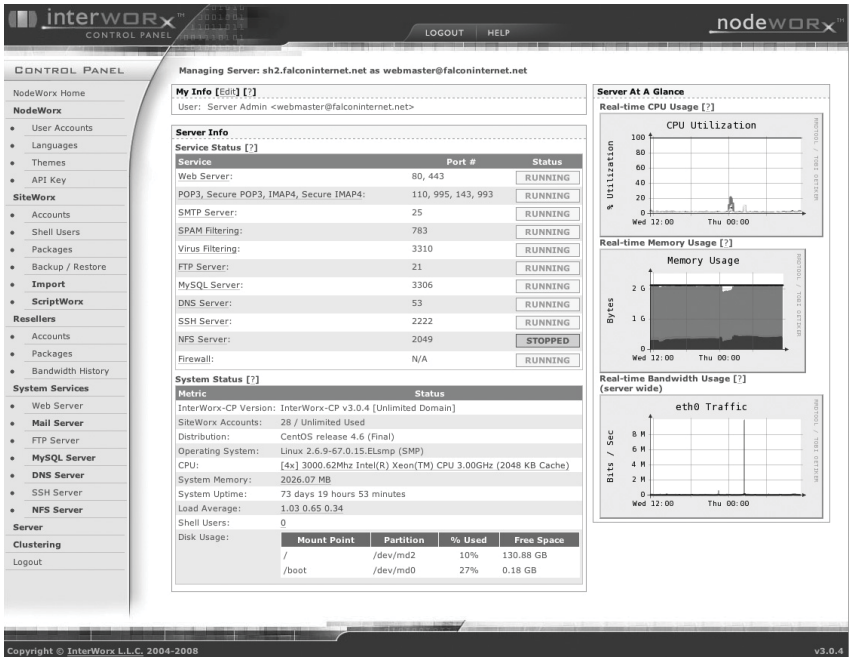
Saturday, September 20, 2008

8:00 - 9:00	Breakfast
9:00 - 10:00	**LassoSoft – Lasso 9 Backward Compatibility – Fletcher Sandbeck**
10:00 - 10:30	Snack Break
10:30 - 11:30	**Future Proof Web Design – Chris Corwin**
11:30 - 1:00	Lunch – Sponsored by **Anu Internet Services**
1:00 - 2:00	**PageBlocks Framework – Nikolaj de Fine Licht**
2:00 - 2:30	Break
2:30 - 3:30	**SVN for FTP Recidivists – Jonathan Guthrie**
3:30 - 4:00	Snack Break
4:00 - 5:00	**LassoSoft – Lasso 9 Data Types – Kyle Jessup and Fletcher Sandbeck**
7:00 - 10:00	LPA Event

Sunday, September 21, 2008

8:00 - 9:00	Breakfast
09:00 - 11:30	**Round Tables**
11:30 - 1:00	Lunch – Sponsored by **Anu Internet Services**
1:00 - 1:15	**Closing**

Hosting that puts YOU in charge!



If you've always had to wait on your internet host to make a change to your hosting, let us introduce you to Interworx Control Panel. Finally, you can have full control over your hosting or dedicated server and setup your sites quickly and easily. From DNS, Email, FTP Users, Server Settings, Bandwidth Controls are all in one easy-to-use package. Best of all Interworx is low cost, yet has all features enabled. Ask us today about Interworx!

Falcon Internet offers dedicated servers, co-location and shared web hosting with the experience to keep your sites up with friendly support and fast response.

Special offer for Lasso Summit Attendees:
Mention this ad and get 30% off any hosting package or free setup on any dedicated or co-located server.

Call us today for your hosting needs!
1-800-984-0793 or (336) 249-7333

FALCON INTERNET
www.falconinternet.net

Celebrating Over a Decade of Quality Hosting for Customers World-wide!

Keynote Address

Kerry Adams — **LassoSoft, LLC**
http://www.lassosoft.com/

Keynote

Kerry Adams will open the Lasso Developer Conference 2008 with a keynote address welcoming all attendees. He will discuss the current state of the Lasso community, LassoSoft's accomplishments over the past year, and LassoSoft's plans for the future.

Kyle Jessup and Fletcher Sandbeck will follow the keynote with an introduction to the new version of Lasso. Lasso 9 promises to be one of the most significant upgrades in Lasso's history bring more speed and capability than every before and providing Lasso with a new foundation that can be used for future growth.

Biography

Kerry Adams is one of the owners of LassoSoft, LLC and the director of sales. Kerry joined OmniPilot Software shortly before the acquisition of Lasso from Blue World, and was involved in sales and customer service, eventually becoming the Sales Manager for OmniPilot. Kerry continues to utilize his skills and passion for the Lasso product, now that LassoSoft has acquired OmniPilot. He is a business professional who utilizes his experience with business management, local and international business partnerships, and enterprise level organizations, to grow LassoSoft and increase it's product exposure on a global scale.

Lasso Developer Conference Materials Download
Server: **http://dowloads.lassosoft.com/beta/** Username: **LDC**
Path: **Lasso Developer Conference** Password: **LCROSS**

Lasso 9 Introduction

Kyle Jessup and
Fletcher Sandbeck

LassoSoft, LLC
http://www.lassosoft.com/

Abstract

Details about the forthcoming version of Lasso will be presented including the design goals for this significant upgrade and an update on the product development roadmap. Many additional details about Lasso 9 will be presented at the other LassoSoft presentations through the Lasso Developer Conference.

Biography

Kyle Jessup is one of the owners of LassoSoft, LLC and the director of software development. Kyle has been developing software professionally for over 10 years. He started working on Lasso at version 1.5 as Blue World's Senior Software Architect. Kyle brings his extensive knowledge of networking, database design and integration, Internet protocols and years of experience developing on a variety of computer systems to the Lasso Professional platform.

Fletcher Sandbeck is one of the owners of LassoSoft, LLC and the author of Lasso's documentation and the popular tip of the week. Fletcher has been designing Web sites for 10 years. He joined Blue World eight years ago working on Lasso 3.6 and helping to author a complete rewrite of Lasso's documentation. Fletcher has a Bachelor of Science in Mathematics with Computer Science from the Massachusetts Institute of Technology.

Lasso Developer Conference Materials Download
Server: **http://dowloads.lassosoft.com/beta/** Username: **LDC**
Path: **Lasso Developer Conference** Password: **LCROSS**

Lasso 9 for Beginners

Fletcher Sandbeck
LassoSoft, LLC
http://www.lassosoft.com/

Abstract

This talk will introduce the Lasso programing language with an emphasis on features of the forthcoming Lasso 9 product. The talk should be of interest to developers who are not yet skilled with the language, but also to skilled developers who want to know more about Lasso 9.

Topics

Topics in this talk will include:

- Scripts and Pages
- FastCGI Web Serving
- Functional Page Design
- Tag Signatures and Dispatch
- Objects and Member Tags
- Captures and Code Blocks
- Loops and New Loop Syntax
- New Foreach Functionality
- Define Tags

Lasso Developer Conference Materials Download
Server: **http://dowloads.lassosoft.com/beta/** Username: **LDC**
Path: **Lasso Developer Conference** Password: **LCROSS**

Lasso 9 Applications

Kyle Jessup and
Fletcher Sandbeck

LassoSoft, LLC
http://www.lassosoft.com/

Abstract

This talk will introduce the new application features in Lasso 9. Lasso 9 will allow "LassoApps" to be built using more powerful tools and runtime support.

Topics

Topics in this talk will include:

- LassoApps
- Initialization
- Programmatic Page Design
- AJAX and XHR
- Runtime Support

Lasso Developer Conference Materials Download
Server: **http://dowloads.lassosoft.com/beta/** Username: **LDC**
Path: **Lasso Developer Conference** Password: **LCROSS**

Lasso 9 Backward Compatibility

Fletcher Sandbeck LassoSoft, LLC
http://www.lassosoft.com/

Abstract

This talk will discuss backward compatibility in Lasso 9 with an emphasis on changes that may be required to get existing solutions running in the new version.

Topics

Topics in this talk will include:

- Lasso 9 Layers: Core, Library, Compatibility
- Lasso as a Scripting Language
- Signatures and Multiple Dispatch
- Reserved Words
- Date Changes
- FastCGI / Site Changes
- Syntax Changes

Lasso Developer Conference Materials Download
Server: **http://dowloads.lassosoft.com/beta/** Username: **LDC**
Path: **Lasso Developer Conference** Password: **LCROSS**

Lasso 9 Data Types

Kyle Jessup and
Fletcher Sandbeck

LassoSoft, LLC
http://www.lassosoft.com/

Abstract

This talk will introduce the new data types in Lasso 9 including a discussion of how to define new types, traits and how to use them, protected members, and more.

Topics

Topics in this talk will include:

- New Types in Lasso 9
- Interogating Types
- Defining Types
- Public, Protected, and Private
- Traits Model
- Defining Traits
- Examples of Traits
- Member Containers and Foreach

Lasso Developer Conference Materials Download
Server: **http://dowloads.lassosoft.com/beta/** Username: **LDC**
Path: **Lasso Developer Conference** Password: **LCROSS**

Knop - Lightweight Application Framework

Johan Sölve Montania System AB
http://www.montania.se/

Abstract

Knop is an open source web application framework for Lasso Professional. Knop provides modules make it easier to handle web forms, database interaction, record listings, site navigation, user authentication and other common tasks. Knop also defines a file structure an application flow. The goal with Knop is to be lightweight and flexible so it is helpful without becoming an obstacle. You can use all of Knop or just selected components. This presentation introduces Knop and shows with a few different examples how Knop can be used to increase your productivity and the quality of your web applications.

Biography

Johan Sölve works as a lead web developer of Montania System AB in Sweden, where he is also a partner. Montania is a software consulting company offering business applications and various custom solutions with a strong focus on Lasso, MySQL and FileMaker.

Johan has been coding Lasso for over 10 years. He is a long time generous contributor to the Lasso community and has have provided numerous improvements in critical parts of Lasso Professional. He is particularly interested in innovative solutions and analytic problem solving. He was a speaker at Lasso Summit 2001 and 2006 and round table host 2007. Johan holds a university degree in Innovation Engineering. Johan lives in Halmstad, southwestern Sweden with wife and two sons. Personal interests include snowboarding and sailing.

Materials

Includes several examples of using the Knop framework for simple forms, database configuration, navigations, and a grid control.

Lasso Developer Conference Materials Download
Server: **http://dowloads.lassosoft.com/beta/** Username: **LDC**
Path: **Lasso Developer Conference** Password: **LCROSS**

Knop Web Application framework

Why framework

Advantages

Higher Productivity

Higher Quality

Features

Disadvantages

Less flexibility

Performance hit

Introducing Knop

Goals with Knop

Be flexible

Be lean

Be focused

Be helpful

Follow standards

Encourage a rich user experience

What is Knop?

Example 1 - a simple form

1-simple-form.lasso

1-simple-form-response.lasso

Example 2 - a form talks to a database

2-form-and-database-config.lasso

2-form-and-database.lasso

2-form-and-database-response.lasso

The Knop Modules

Knop_nav

Example of a virtual URL (path based navigation)

Example of a parameter based URL

3-nav/index.lasso

Knop_database

Knop_form

Knop_grid

Knop_lang

Knop_user

The Knop file structure

Knop application flow

Handle a form submission

Other Knop features

Debugging

Knop

Knop Web Application Framework
For Lasso Professional

Why Framework

Advantages

Higher Productivity
Using a framework lets you focus more on the core functionality of the web site or application, and worry less about the nitty details. It becomes easier to reuse code and modules. You get more done in less time.
Higher Quality
A framework becomes increasingly tested over time. The risk for silly and basic errors is reduced since the framework takes care of the basic functions. You can focus on the important things. You get less bugs to chase.
Features
A framework already has much of those bells and whistles you want to add to the web site but don't have time or budget to even think about. You get many things for free.

Disadvantages

Less flexibility
By using a framework you are more or less bound to the framework's view of the world. It can be problematic to do things in other ways than the framework has intended, or things that go beyond what the framework offers.
Performance hit
A framework puts you a little bit further up above the low level code. There is always some degree of tradeoff between efficient development and raw performance. Higher abstraction level costs CPU cycles.

Introducing Knop

Goals With Knop

Be flexible

Let the developer use the bits they want, allow for customizations and special needs.

Be lean

Stay lightweight, beware of feature bloat, take advantage of native Lasso features as far as possible.
`http://gettingreal.37signals.com/ch15_Beware_the_Bloat_Monster.php`

Be focused

Cover a few main areas where a framework is the most useful, and do it well, Don't try to solve every need.

Be helpful

Don't get in the way.

Follow standards

Encourage the use of modern standards-based and semantically correct HTML and CSS for presentation.

Encourage a rich user experience

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

What Is Knop?

Knop is mainly two things:

1. **A set of custom types** that implements the core functionality of the Knop modules.
2. **A defined application flow** and file structure to support the site's functionality and application logic.

You can choose to only use selected parts of Knop. For example you can use just knop_form to handle forms, or you can use just knop_database to get a richer database abstraction than what Lasso's native inlines offer. But if you use all of Knop, this is what you get:

- A number of modules implemented as custom types that handle the core functions of the framework.
- A defined structure to handle the logic of a web site, to handle page requests, act on form submissions and show the resulting page. A Knop site or application is handled through a single control file, similar to the Onefile structure.
- A defined folder and file structure for the includes needed to handle and support the application logic.
- Support for a modular architecture, where a self contained module easily can be plugged into an existing site.
- Basic templates for HTML and css.
- Support for URL handling with virtual (abstracted) URLs.

Example 1 - A Simple Form

Before we get into the details of Knop it could be useful with a couple of demonstrations on how Knop can be used. One of the Knop modules is a forms generator. Knop_form is a custom type, just as the other Knop modules. It is used to create HTML forms and to help processing the form submission.

This page shows an HTML form.

1-simple-form.lasso

```
[
// Create a new form object
var: 'form'=knop_form;

// Add text fields and a submit button
$form -> (addfield: -type='text', -name='firstname', -label='First name');
$form -> (addfield: -type='text', -name='lastname', -label='Last name');
$form -> (addfield: -type='textarea', -name='message', -label='Message');
$form -> (addfield: -type='submit', -name='button_send', -value='Send message');

// Show the form on the page
]

<form action="1-simple-form-response.lasso">
[$form -> renderform]
</form>
```

To take care of the form input the response page also needs to know about the form, so we first have to define the same form object again (of course we normally define the form object in an include but for this demonstration we just repeat the form definition in the response file).

1-simple-form-response.lasso

```
[
// Create the same form object again
```

```
var: 'form'=knop_form;
$form -> (addfield: -type='text', -name='firstname', -label='First name');
$form -> (addfield: -type='text', -name='lastname', -label='Last name');
$form -> (addfield: -type='textarea', -name='message', -label='Message');
$form -> (addfield: -type='submit', -name='button_send', -value='Send message');

// Load field values from the submission
$form -> loadfields;

// Look at the fielvalues
$form -> updatefields;

]
```

The output from `$form -> updatefields` is a pair array, and one of the nice things with a pair array is that it can be used as dynamic input in an inline. This is very handy and can be used for example like this:

```
inline: -database=...,
      $form -> updatefields,
      -add;
/inline;
```

Example 2 - A Form Talks To A Database

Now we are going to combine two Knop modules to see how they can interact. We will create a database object that the form will interact with. We will also specify an HTML form action in the form object to make it self contained so a complete HTML form can be rendered easily. Finally we'll hook the database object up to the form object. This is where the fun begins.

In this example we move the configuration to an include file that can be reused.

2-form-and-database-config.lasso

```
[
// Create a database object
var: 'db'=(knop_database: -database='knopdemo', -table='customer', -keyfield='id');

// Create a form object
var: 'form'=(knop_form: -formaction='2-form-and-database-response.lasso', -database=$db);

// Add text fields and a submit button
$form -> (addfield: -type='text', -name='firstname', -label='First name');
$form -> (addfield: -type='text', -name='lastname', -label='Last name');
$form -> (addfield: -type='textarea', -name='message', -label='Message');
$form -> (addfield: -type='addbutton', -value='Send message');

]
```

The input page is a simplified version of example 1, since we've moved the configuration to an include and since weve made the form object self contained.

2-form-and-database.lasso

```
[
// Example 2 - A form talks to a database
```


```
// Configure database and form objects in an include
include: '2-form-and-database-config.lasso';

// Show the form on the page, this time the form object is a complete html form
$form -> renderform;
]
```

The response page is really simple. It actually just takes one single line of code to do what it should.

2-form-and-database-response.lasso

```
[
// Configure database and form objects in an include
include: '2-form-and-database-config.lasso';

// Handle the form submission
$form -> process;

// Show the result
]

[$form -> (renderhtml: -excludetype='submit')]
Adding record [$db -> error_msg]
```

Now we have seen the basic principles of a few common Knop operations.

The Knop Modules

The Knop modules are implemented as a number of custom types supported by a few custom tags. Some of the custom types can interact with each other.

The Knop custom types are described below.

Knop_nav

Knop_nav is the heart of a Knop site. It defines the site's structure and navigation. It also keeps track of and validates the visitors current location on a site. It is used to control the application logic of the website or application, keeps track of and processes all include files, generates the navigation menu (a nested ul/li list as default) and breadcrumb, parses the current URL, generates URLs for site internal href links. Knop_nav is the engine room for a site and acts as the main dispatcher for requests and actions.

Knop_nav supports both fully virtual URLs (using an atbegin-based URL handler) as well as parameter based URLs for situations where virtual URLs can't be used.

Example of a virtual URL (path based navigation)

```
http://myhost.com/mypath/tothe/page/
```

Example of a parameter based URL

```
http://myhost.com/?mypath/tothe/page/
```

The two navigation methods result in URLs that look almost the same, the only difference is the "?" after the hostname. Switching navigation method is just a matter of changing a parameter of the navigation object so it's very easy to deploy a Knop based site no matter if it will be hosted on a server that supports virtual URLs or not.

The visitor's current location is called "path" and the current action (if any) is identified by "actionpath".

- Knop path = "where we are" (the page we are coming to).
- Knop actionpath = "what to do" (the page we came from).

Knop_nav can interact with knop_grid.

3-nav/index.lasso

```
[
//Example 3 - Knop_nav

// Create the parent nav object.
var: 'nav'=(knop_nav: -navmethod='param', -currentmarker=' »');

// Define the site structure
$nav -> (insert: -key='home', -label='Home Page');

// Create a child nav object
var: 'subnav'=knop_nav;
$subnav -> (insert: -key='latest', -label='Latest News');
$subnav -> (insert: -key='archive', -label='News Archive');

$nav -> (insert: -key='news', -label='News', -children=$subnav);

// Determine current location so the nav object knows where we are
$nav -> getlocation;

// Generate navigation menu
$nav -> renderhtml;
```


```
// Generate a breadcrumb
$nav -> renderbreadcrumb;

]

<h1>The current page is [$nav -> label]</h1>
<p>The current framework path is [$nav -> path]</p>
```

Knop_database

This is a database abstraction layer that sits on top of Lasso's own database abstraction. It supports both regular Lasso inlines and SQL syntax. MySQL and FileMaker databases are supported currently.
Knop_database provides convenient access to basic CRUD operations (Create, Read, Update, Delete) and has built-in support for record locking, safe random keyvalues and duplicate prevention. A found set of records can either be iterated, or Lasso's native records tag can be used to access the found set. Knop_database can maintain a persistent pointer to a specific record, much like Active Record.
Knop_database primarily uses pair arrays as field specifications, which makes it easy to integrate with standard Lasso inlines, but can also use SQL statements for some of the operations. When interacting with knop_form and knop_grid, pair arrays are normally used to exchange field data and other search parameters. The use of pair arrays for standard inlines is one way to provide greater flexibility.
Knop_database can interact with knop_form, knop_grid and knop_user (for record locking).

Here's an example to demonstrate how to use knop_database to output some fields from a specific database record.

```
// initiate the database object (normally in a config file)
var: 'db_news'=(knop_database: -database='acme', -table='news',
      -username='*****', -password='*****',
      -keyfield='id');

// perform a database search to grab the record (normally in a lib file)
$db_news -> (getrecord: -keyvalue=185);

// show some fields from the database record (normally in a content file)
<h3>[$db_news -> (field: 'title')] </h3>
<p>
[encode_break: ($db_news -> (field: 'text'))]
</p>

----

// The getrecord statement can be simplified slightly since the first parameter
// is the keyvalue
$db_news -> (getrecord: 185);

// The field calls can be simplified using a shortcut that maps unknown
// member tags to field names
<h3>[$db_news -> title] </h3>
<p>
[encode_break: ($db_news -> text)]
</p>

// If a more complex query is needed to get the record, an SQL statement can be used.
```

```
$db_news -> (getrecord: -sql='SELECT * FROM news LEFT JOIN ...');

// A general select can be used as well.
// The data from the first found record will be available as ->field.
$db_news -> (select: -sql='SELECT * FROM news LEFT JOIN ...');


// To output a record listing from a database search, different methods can be used.
// 1. A standard records loop (fastest)
      records: -inlinename=($db_news -> inlinename);
            field: 'title';'<br>';
      /records;
// 2. Iterate the database object
      iterate: $db_news, var: 'record';
            $record -> (field: 'title');'<br>';
      /iterate;
// 3. Use the record pointer
      while: $db -> nextrecord; // increment the record pointer
            // (nextrecord returns true as long as there are more records to show)
            // fetch data from the record the record pointer currently points at
            $db -> field('title');'<br>';
      /while;
```

Knop_form

Forms are one of the most tedious things to handle manually in a web application. First the form fields should be shown on the edit page. They need labels, proper styling and different properties. They may also need initial values to show in the form fields. The values can either be static, come from a database lookup, or from a previous submission of the same form if there was an input error that needs to be corrected, and in that case the erroneous fields or labels needs some highlighting to guide the user. Finally the form submission must be handled by validating the user's inputs and then storing the form data in a database.

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

First we define the form. We give it a form action, and we add the fields and other elements such as submit buttons that the form should contain. The fields can have the same properties as regular HTML form fields do, and they have additional properties to define the options of a select menu, the checkbox options of a checkbox field set, for interaction with databases and other purposes.
Then we fill the form fields with data. It can either come from a form submission or from a database lookup. In the case of a database lookup, the corresponding database field has been declared as property for each form field.
If we want we can set a template for the form to define how the form should be presented in HTML, or just let it use the default template.
Finally we render the form on the page. We can render the entire form at once, or specific fields at a time (we can even set different templates for every field), to have the flexibility needed to be able to accommodate the form in just about any HTML context.
The form object even generates some javascript for us that will warn the user if he navigates away from a "dirty" page (a page that has unsaved changes) and other useful features.
The next step is that the user submits the form. Now the form object makes its second entry by taking care of the form submission. Since all form fields are defined in the form object, it knows where to put each field when we tell it to load data from the form. Since it knows what kind of data is allowed in each field the form object can validate itself with a single call.

If the validation comes across an input error, the form objects prepares to show itself again but this time with the erroneous inputs highlighted.
If the validation passed, the form object comes back to our help once again and provides us with a complete pair array with field name and value pairs (the form fields knew what database fields they correspond to, remember?) that we can feed right into an inline to add or update a database record, or we can get an SQL string that we can put in an SQL statement of our liking.
Knop_form can interact with knop_database.

Knop_grid

This custom type is used to display record listings with sortable columns, pagination, detail link to edit a record, filtering/quicksearch, etc. It requires a reference to a knop_database object because they are so tightly related. It can highlight the affected record when returning to the listing after adding or editing a record.
We can also give it a reference to a knop_nav object, to get the right pagination links and other things. It can also provide a basic "Quicksearch" functionality integrated with the record listing.
Quicksearch and the sort headings generate pair arrays or SQL snippets to interact with knop_database. Sort parameters and the quicksearch query is automatically propagated through a knop_form, so the same set of records is selected after editing a record.
Knop_grid must interact with Knop_database and can optionally interact with Knop_nav.

Knop_lang

This custom type handles language strings for multilingual presentation of the user interface. A knop_lang object holds the language strings for all supported languages. Strings are stored under a unique text key, but the same key is of course used for the different language versions of the same string.
Language strings can be grouped into different knop_lang object instances (variables) if it helps managing them.

When the language of a knop_lang object is set, that language is used for all subsequent requests for strings until another language is set. The selected language is shared between all knop_lang objects on the same page for that visitor, unless another language has been set specifically for an individual knop_lang object.

If no specific language is set on the page, knop_lang uses the browser's most preferred language if it's available in the knop_lang object, otherwise it defaults to the first language (unless a default language has been set for the knop_lang object).

The strings in a knop_lang object can contain replacement placeholders, to be able to insert dynamic text when retrieving a string. The strings can also be a Lasso compound expression which will be evaluated at runtime when the string is retrieved.

Examples

```
var: 'lang_messages'=(knop_lang: -default='en');
$lang_messages -> (addstring: -key='welcome', -value='Welcome to the home page', -
language='en');
$lang_messages -> (addstring: -key='welcome', -value='Välkommen till hemsidan', -
language='sv');
$lang_messages -> (addstring: -key='loggedin', -value='You are logged in as #1# #2#', -
language='en');
$lang_messages -> (addstring: -key='loggedin', -value='Du är inloggad som #1# #2#', -
language='sv');
```

```
// proper call, defaults to the browser's preferred language
$lang_messages -> (getstring: 'welcome');
// shorthand call
$lang_messages -> welcome;

// change language
$lang_messages -> (setlanguage: 'sv');
$lang_messages -> welcome;

// proper call with replacements
$lang_messages -> (getstring: -key='loggedin': -replace=(array: (field: 'firstname'),
(field: 'lastname')));

// shorthand call with replacements
$lang_messages -> (loggedin: -replace=(array: (field: 'firstname'), (field: 'lastname')));
```

-> addlanguage is suitable if you use config files to configure strings, like the strings in knop_grid. It looks like this:

```
#lang -> (addlanguage: -language='en', -strings=(map:
      'quicksearch_showall' = 'Show all',
      'quicksearch_search' = 'Search',
      'linktext_edit' = '(edit)',
      'linktitle_showunsorted' = 'Show unsorted',
      'linktitle_changesort' = 'Change sort order to',
      ...
```

Knop uses knop_lang internally to handle text strings. By providing access to the internal lang object that a Knop module uses, it is ieasy to add custom localizations or modified strings also to the core Knop modules without actually altering Knop itself. As an example if you want to localize an instance of knop_grid to another language on the fly, you can first find out what strings that need to be localized by calling $grid -> lang -> keys. This gives you an array of all string keys that are used across all defined languages.

Then you can just add the new language like this (for quasi Danish), since the ->lang member tag returns a reference to the internal knop_lang object:

```
$grid -> lang -> (addlanguage: -language='da', -strings=(map:
      'quicksearch_showall' = 'Finn alt',
      'quicksearch_search' = 'Søk',
      ...
```

Knop_user

The Knop_user custom type handles user authentication, maintains information about the user and keeps track of permissions for the user.
Authenticating a user checks the login credentials against a specified table, with support for one way encrypted passwords (with salt) and delays between repeated login attempts to prevent brute force attacks. User authentication can also be performed through custom code outside of Knop_user.
Knop_user prevents session sidejacking by comparing a client fingerprint between each page request.
Knop_user is the only Knop custom type that is intended to be stored in a session variable, and actually relies on this.

When a user is being authenticated, all available fields from the user table are stored in the Knop_user variable so user information can be retrieved easily throughout the session. Any additional custom data for the user can also be stored manually in the Knop_user variable.
Knop_user can keep track of user permission by storing arbitrary permission information in the Knop_user variable. Knop_user enhances Knop_database objects by keeping track of record locks set by the user, and releasing record locks for example when navigating to a list of records without saving an edited record.

A basic example

```
var: 'session_user'=(knop_user: -userdb=$users);
session_start: -name='test';
session_addvar: -name='test', 'session_user';

$session_user -> (login: -username=(action_param: 'u'), -password=(action_param: 'p'));

if: $session_user -> auth;
      if: $session_user -> groups >> 'admin';
            $session_user -> (setpermission: 'candelete');
      /if;
else;
      'Authentication failed, ' + ($session_user -> error_msg);
      abort;
/if;

'Welcome, ' + ($session_user -> firstname) + '! ';
if: $session_user -> (getpermission: 'candelete');
      'You are allowed to delete records. ';
/if;
```

The Knop File Structure

A Knop site is built around a single file (for example index.lasso) that acts as a "control center" or main dispatcher, similar to the "Onefile" concept. The main files are of the following types:

- Config - (page specific configuration) configures the request handler, configures the business logic
- Action - request handler, manipulates data
- Library - user interface logic, prepares information to display to the user
- Content - display the information to the user

Files are named with a prefix that tells what kind of file it is, then the Knop path with “/” replaced by “_”. A few special files are named with a double underscore after the prefix. Files are grouped by their type into folders named _config, _action, _library and _content.

Example file structure:

```
_config/
      cfg__global.inc
      cfg__nav.inc
      cfg_customer_edit.inc
      cfg_customer_list.inc
      cfg_news_archive.inc
      cfg_news_latest.inc
_action/
      act_customer_edit.inc
      act_customer_list.inc
      act_news_archive.inc
      act_news_latest.inc
_library/
      lib_customer_edit.inc
      lib_customer_list.inc
      lib_news_archive.inc
      lib_news_latest.inc
_content/
      cnt_customer_edit.inc
      cnt_customer_list.inc
      cnt_news_archive.inc
      cnt_news_latest.inc
index.lasso
```


Knop Application Flow

To explain the application flow of Knop, let's assume we have a web application where the user submits a form and we will walk through the processing of the form submission.

Every page request has one or two vital parameters:

- path (required)

This is the visitor's current location in the application. The path tells the application “where we are”.

- actionpath (optional)

If the current page request is the result of a form submission, the application needs to know what to do with the input. The actionpath tells the application “what do to”.

Don't mix up the actionpath with the 'action' HTML attribute of the form tag itself!

Handle A Form Submission

A) Take care of the input (controlled by actionpath)

1. Find out the actionpath, which is where the submission comes from (this is determined by knop_nav -> getlocation)
2. Load the config for actionpath to define the forms etc for the page we came from. This is crucial to be able to handle the form submission.
3. Perform the actual action by loading form data, validate input and execute the logic needed in response to the form submission.

B) Prepare the output (controlled by path, which was also determined by knop_nav -> getlocation)

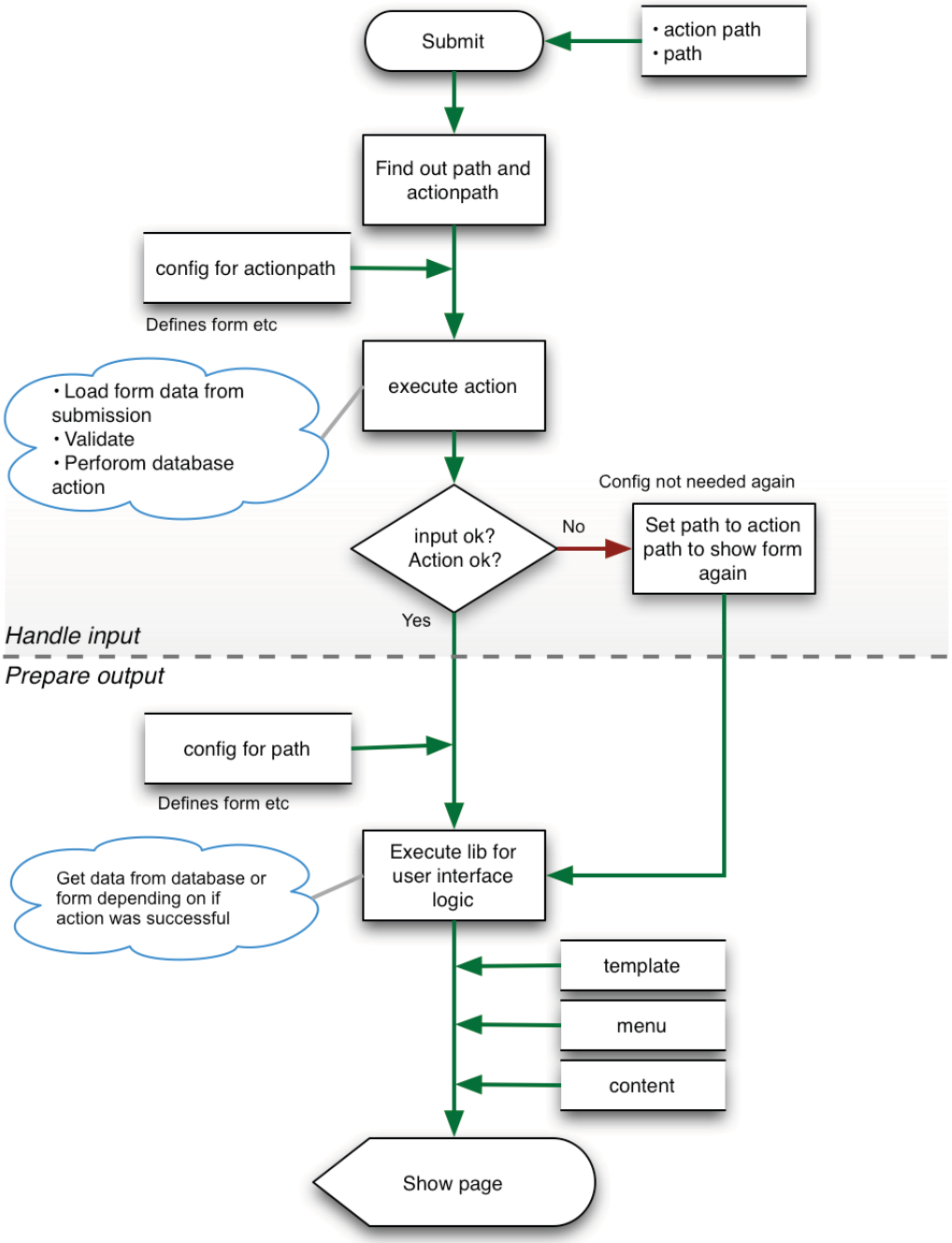
4. Was action successful? (form validation ok, database action wihtout errors etc) ->
    4a. Load config for 'path' to define form, grid etc for display

Action not ok? ->

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

5. Execute the library file for 'path' to prepare the page output
6. Include page template to build the page HTML framework with navigation menu, content area, sidebars etc.
7. The template includes the content for 'path' to generate the actual page contents
8. The finished page is served to browser.

All the include files that are needed to handle the application flow are chosen automatically by Knop_nav.


Submit
• action path
• path
Find out path and actionpath
config for actionpath
Defines form etc
execute action
• Load form data from submission
• Validate
• Perforom database action
Config not needed again
input ok?
Action ok?
No
Set path to action path to show form again
Yes
Handle input
Prepare output
config for path
Defines form etc
Execute lib for user interface logic
Get data from database or form depending on if action was successful
template
menu
content
Show page

Other Knop Features

Debugging

All Knop modules log debug information internally, which can be output easily to be able to track what's happening inside each module.

Caching

Caching can be used to reduce the overheads from configuration that is mostly static, used globally and normally doesn't change such as navigation, database objects, language strings , etc. Configuring all of this repeatedly on every page load is a waste of resources.
The Knop cache consists of two custom tags that can be used like in this example:

```
if: !(knop_cachefetch: -type='knop_lang');
      var: 'lang_nav_key'=(knop_lang: -default='sv');
      $lang_nav_key -> addlanguage(-language='sv', -strings=(map:
            'hem'='hem',
            'loggain'='loggain',
            ...
      knop_cachestore: -type='knop_lang';
/if;
```

`knop_cachestore` stores all page variables of the specified type in a global variable. It does this by iterating through all page variables and checking their type, copying the variables that have a matching type. This way the cache is populated mostly automatically and transparently. There is almost nothing to configure and it just works.
`knop_cachefetch` tries to recreate all the cached page variables of the specified type from the global variable, if there's a cached version available. It returns true if it was successful and false if there wasn't a cached copy available. If there wasn't a cache available, no page variables are created and the configuration needs to be set up from scratch. This is easy to do by using knop_cachefetch in a condition around the normal configuration.

The idea is to call knop_cachefetch first to try to have all knop object instances recreated as page vars, and if knop_cachefetch returns false then configure the knop objects the normal way and call knop_cachestore to cache them for the next page load.
The default cache expiration is 10 minutes (600 seconds). This means that instead of setting up the configuration repeatedly for every single page load for every user, the configuration only needs to be loaded once every 10 minutes by a single user for the benefit of all others.

Some example timings just to give an idea of how much it can help:
Without caching:

- Created langauge strings 23 ms
- Created database objects 73 ms
- Created navigation 142 ms

With caching:

- Created langauge strings 5 ms
- Created database objects 8 ms
- Created navigation 7 ms

The cache can be forced to refresh simply by adding a condition to cache_fetch:

```
      if: $cache_refresh || !(knop_cachefetch: -type='knop_database');
```

```
...
```

If $cache_refresh is true in this example then thew cache will be ignored so the configuration will be set up again.

The caching can also be done per visitor by specifying a session name to use for cache storage. The specified session must be started before using the knop_cache tags with session. knop_cachestore adds the session variable $_knop_cache to the session. Using session to store the cached data is useful for example for navigation, where the configuration can be different for each visitor.

Example:

```
if: !(knop_cachefetch: -type='knop_nav', -session=$session_name);
      var: 'nav'=(knop_nav:
            -default=($lang_nav_key -> hem),
            -root=$siteroot,
            -navmethod=$navmethod);
      $nav -> insert(-key=($lang_nav_key -> hem), -label=($lang_nav_label -> hem), -
url='/');
      ...
      knop_cachestore: -type='knop_nav', -expires=1200, -session=$session_name;
/if;
```

The global variable used for caching is named uniquely for the current site (based on server_name and response_localpath - response_filepath) and it's also possible to specify a -name to further isolate the cache storage if needed (for example if multiple sites are running in the same virtual root and hostname). The global variable is accessed using thread locking to provide a thread safe caching mechanism.

Multiple Ways To Work With Site Modules

Knop's framework folder structure is actually quite liberal. It lets you collect all files in a central _knop directory to be able to centralize modules so they can be shared between different sites. It also lets you modularize parts of a solution in separate _mod directories.

The defined Knop directory tree consists of folders with name _knop, _config, _action, _library, _content or with names that begin with _mod_

Knop looks for framework include files in no less than 10 locations for each file naming convention you specify. For the framework path customer/edit, the actual name and location of the library file for that path can be any of the following:

A) -filenaming='prefix' (this is the default if -filenaming is not specified)

1. _mod_customer/lib_customer_edit.inc // modular prefixed with module name
2. _mod_customer/lib_edit.inc // modular
3. _mod_customer/_library/lib_customer_edit.inc // modular separated, prefixed with module name
4. _mod_customer/_library/lib_edit.inc // modular separated
5. _library/lib_customer_edit.inc // collective ("all modules together") separated
6. _knop/_mod_customer/lib_customer_edit.inc
7. _knop/_mod_customer/lib_edit.inc
8. _knop/_mod_customer/_library/lib_customer_edit.incname
9. _knop/_mod_customer/_library/lib_edit.inc
10. _knop/_library/lib_customer_edit.inc

B) -filenaming ='suffix'

example: _library/customer_edit_lib.inc

C) -filenaming='extension'
example: _library/customer_edit.lib

Knop And MVC

Knop translates to the Model-View-Controller pattern in the following way:

Model

The domain-specific representation of the information on which the application operates

- Config and Library, together with a database

View

Renders the model into a form suitable for interaction, typically a user interface element

- Content

Controller

Processes and responds to events, typically user actions, and may invoke changes on the model

- Config and Action

Why “Knop”?

“Knop” is Swedish for knot, and a knot is what keeps a lasso together. A good knot makes a good lasso experience. The meaning is the same as English knot, which is both used for the speed of boats or airplanes (one nautical mile, 1852 meters, per hour), or a knot on a rope. The speed measurement comes from the rope knot meaning, where they measured how many knots on a rope passed in a given time when they measured the speed of ships in the old days. The word stems from the Dutch word knoop with the same meaning, which is also related to knopp (knob in English).
Knop is pronounced with a sounding “k” and a long “o” just as in groove.

Knop is created and maintained by Johan Sölve, Montania System AB

Credits

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

Introduction to LassoFusebox

Tami Williams **Creative Computing**
http://www.asktami.com/

Abstract

This presentation will provide a brief overview of the Fusebox framework using Lasso Professional 8.5. Fusebox is the most popular framework for building ColdFusion and PHP web applications and due to popularity its been ported to Lasso as well as JSP and even ASP .net. LassoFusebox is a great tool for demonstrating the benefits of a standardized methodology.

Biography

Tami Williams has been teaching Lasso development since version 3. An experienced trainer and professional database and web application developer, she started her Lasso and FileMaker consultancy Creative Computing in 1990. The Lasso community knows her best for her Lasso Fusebox core files, and for the many Lasso developer tools and custom tags available at her web site.

Among the many Lasso projects she has developed are sites for Cornell University, FIRST (For Inspiration and Recognition of Science and Technology), Chapin Hall Center for Children at the University of Chicago, Macy's and Apple Computer, Inc.

Materials

Includes several examples of using LassoFusebox including a contact application.

Lasso Developer Conference Materials Download
Server: **http://dowloads.lassosoft.com/beta/** Username: **LDC**
Path: **Lasso Developer Conference** Password: **LCROSS**

LassoFusebox 3.0 Specification

Original by Hal Helms
Adapted for Lasso by Rich Tretola, April 5, 2003
Updated by Tami Williams, June 26, 2008

This document also contains an edited (by Tami Williams) version of:
*"**Fusebox 3 improves on its predecessor**"*
by Brian Kotek | May 01, 2002 7:00:00 AM
http://articles.techrepublic.com.com/5100-10878_11-1044897.html

*Also look at the "**Fusebox 3.0 Newbie Guide - PHP edition (DOC)**"*
http://bombusbee.com/downloads/files/FuseboxNewbieGuideV3PHP.zip

Adapted from:

Fusebox 3 improves on its predecessor

by Brian Kotek | May 01, 2002 7:00:00 AM

Source: http://articles.techrepublic.com.com/5100-10878_11-1044897.html

The flow of things

Here's a step-by-step description of what happens in a Fusebox application.

1. A user visits your site by going to http://www.mysite.com/.
2. Your web server's default page, index.lasso, is called.
3. Index.lasso includes the core frozen LassoFusebox file (fbx_LassoFusebox3).
4. The core frozen LassoFusebox file includes the fbx_Circuits file, which provides aliases and directory paths to all the circuits (subdirectories) nested beneath it.
5. The core frozen Fusebox file includes the fbx_Global file, which contains our global application settings. One of these settings defines a default fuseaction to use if none is specified (as in this case). It also includes the root fbx_Settings file, which contains other application variables .
6. The core frozen Fusebox file is given the default fuseaction of ContactApp.list_Contacts by the fbx_Global file.
7. The core frozen Fusebox file dissects the fuseaction. It sees that the first part (the circuit alias) is ContactApp. From previously parsing fbx_Circuits, it knows that ContactApp is the alias for the root, so it includes the fbx_Switch file in the root directory (the same directory as the core fbx_LassoFusebox3 file).
8. The fbx_Switch file looks through its case block for a match to the action "list_Contacts" (the second part of the default fuseaction).
9. After finding the match for the action "list_Contacts", the fbx_Switch file includes the following fuses:
   - qry_getAllContacts: finds all the contacts in the database.
   - dsp_AllContacts: displays a list view of the found records. It has links for adding, editing and deleting records.
10. The core frozen Fusebox file grabs the HTML output generated by the two fuses but does not output this to the user yet. Instead, it temporarily stores this output in a variable.
11. The core frozen Fusebox file looks in the fbx_Layouts file and includes a layout file containing a header, footer and a placeholder for any content generated by the circuits. It inserts the variable that is holding the output from the two fuses into this placeholder spot, essentially wrapping the header and footer around the content that has been generated.
12. The user is shown the final page output consisting of a header, a table of found contact records with links to add, edit and delete, and a footer, as one coherent page.

In an application with more than one circuit that also has nested layouts, here's what happens after step 12:

After the output from the qry_getAllContacts and dsp_AllContacts fuses is captured and temporarily stored in a variable, the core frozen Fusebox file looks for an fbx_Layout file in the child circuit's directory. If one exists, the core frozen Fusebox file uses it to find the child circuit's layout file, and inserts the output here, inside this layout. But it still doesn't output anything; it keeps all this data stored in a variable. Then the core frozen Fusebox file grabs the fbx_Layout file from the root directory (the one that provides the global header and footer) and wraps it around the content that was generated by this nested circuit. In the end, we would get a final page that has a header, the secondary layout elements, a table of found contact records with links to add, edit and delete, and a footer.

LassoFusebox

Introduction

Fusebox is a development methodology for web applications. Its name derives from its basic premise: a well-designed application should be similar to the fusebox in a house. In a fusebox, if one fuse is blown, the rest of the fuses continue to work. Each fuse has its own defined job, and Fuse A does its job without any help from Fuses B, C, or D. Similarly, in a Fusebox application, if one section of the application "breaks", the rest of the application should continue to work.

Fuseactions

In a Fusebox application, every action that an application must handle is referred to as a Fuseaction (aka Exit Fuseactions or XFAs). For example, an e-commerce application would include the following possible Fuseactions:

- viewCart
- viewCatalog
- addItem
- checkout

Rather than having fuseactions hard coded into exit points, like this:

```
<form name="contact" action="index.lasso?fuseaction=home.delete"  method="post">
```

Fusebox 3 replaces these hard coded references with XFAs:

```
<form name="contact" action="[$XFA_Delete]" method="post">
```

Naming Fuseactions

Each XFA is a compound fuseaction, consisting of the circuit name (the directory alias defined in fbx_Circuits), a dot separator, and the circuit request (instruction). The value of these fuseactions is set in the fbx_Switch file:

```
[Select: $fuseaction]
	[Case:'Welcome']
		[Var: 'XFA_Edit' = ($self) + 'home.edit']
		[Var: 'XFA_Delete' = ($self) + 'home.delete']
		[Var: 'XFA_New' = ($self) + 'home.new']
		[FBX_Include: 'qry_getContacts.inc]
		[FBX_Include: 'dsp_Contacts.inc']
	[Case]
		<p>This is the default case tag. I received a fuseaction called
		"[$fuseaction]" and I don't know what to do with it.</p>
[/Select]
```

Note: The variable, self, is not part of the Fusebox 3 specification, but is used by many developers to refer to the root circuit's default page and is set in the fbx_Settings file.

snippet from fbx_Settings.inc

```
[var: 'self' = 'index.lasso?fuseaction=' ]
```

The references to XFAs are resolved at run time, allowing for ease of reuse of both fuses and entire circuits.

Fuses

To perform each Fuseaction, a Fusebox application uses a series of Lasso pages. Each page is referred to as a Fuse. To perform the Fuseaction “viewCart” mentioned above, the user's cart must first be queried and then the results must be displayed. Therefore, the Fuses called for the Fuseaction “viewCart” would be:

- qry_getCart.inc
- dsp_Cart.inc

Types of fuses

Fuses can be grouped into a finite number of types. The most common are:

- Display fuses, prefixed with dsp_, which are used to show information to the user.
- Query fuses, prefixed with qry_, which contain SELECT, INSERT, UPDATE, DELETE or other types of queries.
- Action fuses, prefixed with act_, which contain any other type of activity (like sending email).
- Form fuses, prefixed with frm_, which contain forms.
- Validation fuses, prefixed with val_, which are used for form validation.
- Value List fuses, prefixed with vl_, which contain value list items.
- Ajax fuses, prefixed with ajax_, which are used for Ajax–specific code.

Frm_, val_, vl_, and ajax_ are not official Fusebox prefixes.

Additionally, layout files, which are not technically fuses, have a naming convention; they are prefixed with lay_.

Sample fbx_Switch file

We have used many new terms, which can be confusing. Before looking at the files used in a Fusebox application, here is a short sample of code from an fbx_Switch file, showing Fuseactions and Fuses in use. After reviewing, you should have a better idea of what Fuseactions and Fuses are.

```
[Select: $fuseaction]
        [Case: 'showInputForm']
                [Var: 'XFA_add' = ($self) + 'admin.add']
                [Var: 'XFA_edit' = ($self) + 'admin.edit']
                [Var: 'XFA_delete' = ($self) + 'admin.delete']
                [FBX_Include: 'frm_InputForm.inc']
        [Case: 'showEmployeeList']
                [FBX_Include: ' qry_EmployeeList.inc']
                [FBX_Include: ' dsp_EmployeeList.inc']
        [Case]
                <p>This is the default case tag. I received a fuseaction called
                "[$fuseaction]" and I don't know what to do with it.</p>
[/Select]
```


Explanation of the sample fbx_Switch file

In the above file, the Fuseactions are “showInputForm” and “showEmployeeList”. (Don't worry about the [Var: 'XFA_… = …] statements in the “showInputForm” fuseaction, they will be explained later.) The FBX_Include statements are the Fuses for the Fuseactions. The Fuseaction that is required can be passed to the application through a URL variable.

Core Files

Fusebox 3 is built around a library of core files, prefixed with fbx_. These include:

- fbx_LassoFusebox3.inc: A non-editable file which contains the main Fusebox code.
- fbx_Library.inc: Adds custom tags that are needed by the Fusebox core file. It is included within the core file. This file is unique to the Lasso port of Fusebox 3.0.
- fbx_CustomTags.inc: This file is provided for the user to add their own custom tags. This file is unique to the Lasso port of Fusebox 3.0.
- fbx_Sessions.inc: Extends the Fusebox core file to allow Lasso sessions to operate properly.
- fbx_Circuits.inc: This file provides circuit-to-directory mappings.
- fbx_Globals.inc: This file allows default, application-wide variables to be set in the root circuit.
- fbx_Settings.inc: This file allows variables to be set at each circuit. Variables set by parent circuits are inherited by their descendants and may be overridden.
- fbx_Switch.inc: This file processes the fuseaction sent to the application.
- fbx_Layouts.inc: This file determines the layout file to be used for a particular circuit.

fbx_LassoFusebox3.inc

This file should be called by your root circuit's default file (index.lasso, default.lasso, etc.) like this:

```
[Include: 'fbx_LassoFusebox3.inc']
```

The most important thing to keep in mind when using this file is that it is the engine of the application. It controls the entire application, based on the values you set in other files.

At a very high-level, this file does the following:

- Sets a number of variables and initializes some structures.
- Includes the fbx_Library file and the fbx_CustomTags file.
- Converts all action_params to variables.
- Includes the fbx_Sessions file.
- Includes the fbx_Circuits file, which translates the folder structure of the application to a simple structure named Circuits. This makes referring to folders, subfolders and sub-subfolders very easy.
- Creates a reverse lookup of the circuits just defined.
- Includes the fbx_Globals file from the root application folder.
- Gets the Fuseaction and the circuit that was sent to the application.
- Includes the fbx_Settings file, from the root application folder, then includes the fbx_Settings files from all the subfolders in top to bottom order. This allows child fbx_Settings files to overwrite values set in the parent fbx_Settings file.
- Executes the Fuseaction in the correct circuit's fbx_Switch file using the values stored in the $fuseaction and $_Circuits variables. All output is stored in the $layout variable.
- Includes the fbx_Layouts files from all the subfolders in bottom to top order. These files call the individual layout files, which output the contents of the $layout variable.

fbx_Library.inc

The fbx_Library file adds custom tags that are needed by the Fusebox core file. It is included within the core file. This file is unique to the Lasso port of Fusebox 3.0.

fbx_CustomTags.inc

The fbx_CustomTags file was added to allow users to add their own custom tags. This file is unique to the Lasso port of Fusebox 3.0.

fbx_Sessions.inc

The fbx_Sessions file was added to allow Lasso sessions to work properly within the Fusebox architecture. This file should be located within the root directory and all sessions should be declared within this file.

fbx_Circuits.inc

This file provides directory to circuits mapping. It aliases all the subfolders and sub-subfolders in the application. The reserved structure $_Circuits contains key/value pairs where the key is the circuit alias and the value is the directory path beginning with the root circuit.

The fbx_LassoFusebox3 file uses the fbx_Circuits file to resolve compound fuseaction names. Note that the circuit names do not need to be the same as the directory names. There is only one fbx_Circuits file in an application.

The following is a sample of the code in the fbx_Circuits file:

```
[$_Circuits->(Insert:'Home'='myApp')]
[$_Circuits->(Insert:'Admin'='myApp/Administration')]
[$_Circuits->(Insert:'Rules'='myApp/Administration/Rules')]
[$_Circuits->(Insert:'Front' = 'myApp/FrontEnd')]
```

This allows all references to specific folders to be made using aliases. To call the Fuseaction named “myFuseaction” in the folder myApp/Administration/Rules, you would call the index.lasso file as follows:

```
index.lasso?fuseaction=Rules.myFuseaction
```

fbx_Globals.inc

The fbx_Globals file is used by the Fusebox application to set the default fuseaction and other application-wide default values. There must be only one fbx_Globals file and it must reside in the root folder of the application.

fbx_Settings.inc

The fbx_Settings file is used by the Fusebox application to set default values. There must be one fbx_Settings file in the root folder of the application. An individual circuit can have it’s own fbx_Settings file if default values need to be set for files in that circuit, or if a child circuit needs to overwrite a default value in a parent circuit.

fbx_Switch.inc

The fbx_Switch file is probably the simplest of the core files. It decides what files to include on the page, based on the Fuseaction.

Here is a sample of an fbx_Switch file:

```
[Select:$fuseaction]
	[Case:'Welcome']
		[FBX_Include: ($self) + 'qry_getContacts.inc']
		[FBX_Include: ($self) + 'dsp_Contacts.inc']
	[Case]
	<p>This is the default case tag. I received a fuseaction called "[$fuseaction]"
	and I don't know what to do with it.</p>
[/Select]
```

fbx_Layouts.inc

It is often desirable to have a different 'look' for each section of a Web application. Controlling the look of each circuit is the purpose of the fbx_Layouts file. There should be at least one fbx_Layouts file in an application. However, it can be safely omitted, in which case, no layout will be applied. Any circuit that has its own layout requirements should have its own fbx_Layouts file.

An fbx_Layouts file is responsible for setting two variables, $layoutDir and $layoutFile.

$layoutDir points to a directory (if it exists) in which layout files are kept. $layoutFile points to the file to be used for the layout. If you create a layout file, it must, at minimum, reference the variable, $layout.

LassoFusebox 3 API

Fusebox 3 exposes a series of variables to help in writing code:

Public API Variable	Type	May be set?	Description
Fusebox.isRootCircuit	BOOLEAN	NO	Is the directory of the file currently being operated on by fbx_LassoFusebox3 the same as the root circuit of the app?
Fusebox.isTargetCircuit	BOOLEAN	NO	Is the directory of the file currently being operated on by fbx_LassoFusebox3 the same as the target circuit of the app?
Fusebox.fuseaction	STRING	NO	The simple fuseaction extracted from the attributes.fuseaction of form circuit.fuseaction passed in
Fusebox.circuit	STRING	NO	The simple circuit extracted from the attributes.fuseaction of form circuit.fuseaction passed in
Fusebox.rootCircuit	STRING	NO	The circuit alias of the root circuit of the app
Fusebox.targetCircuit	STRING	NO	The circuit alias of the target circuit of the app. Usually this is the same as Fusebox.circuit ($circuit) unless you've made a circuits definition error
Fusebox.thisCircuit	STRING	NO	The circuit alias of the directory of the file currently being operated on by fbx_LassoFusebox3
Fusebox.LayoutFile	STRING	YES	The layout file to be applied to this circuit after the fuseaction and its fuse(s) are done creating their content
Fusebox.LayoutDir	STRING	YES	The relative path from the target circuit to where the layout file to be applied to this circuit is located. If no special layout directory has been created, this should be set to an empty string
Fusebox.CurrentPath	STRING	NO	The relative directory path of the file currently being operated on by fbx_LassoFusebox3, relative from the root of the application (i.e. the root circuit). Example: dir1/dir2/dir3/
Fusebox.RootPath	STRING	NO	The relative directory path of the file currently being operated on by the core frozen Fusebox code, relative to the root of the application (i.e. the root circuit). Example: ../../../
Fusebox.layout	STRING	NO	This variable captures the content generated to that point in preparation for wrapping a layout file around it (as defined by Fusebox.layoutFile). This variable must be inside each layout file in order for content to be passed up to the next level of nested layouts

Fusedoc

Fusedoc is a documentation system/program definition language for documenting fuses. The Fusedoc uses XML syntax wrapped in a Lasso comment and is normally place on top of the fuse file itself. Here is a sample Fusedoc for a display login file:

```
<fusedoc fuse="frm_login.lasso" language="Lasso" version="3.0">
    <responsibilities>I am a form that lets users log in.</responsibilities>
    <properties>
        <history author="Rich Tretola" date="12/10/2002" role="Architect" type="Create" />
    </properties>
    <io>
        <in>
            <string name="self" />
            <string name="XFA_submitForm" />
        </in>
        <out>
            <string name="fuseaction" scope="formOrUrl" />
            <string name="userName" scope="formOrUrl" />
            <string name="password" scope="formOrUrl" comments="password field" />
        </out>
    </io>
</fusedoc>
```

The Fusedoc XML root element has three sub-elements: responsibilities, properties, and io. Of these, only responsibilities is required.

Responsibilities use the first person to have the fuse describe what it is responsible for. Properties is a more generic, catchall section that has three possible sub-elements: history, property, and note. IO (short for input/output) contains in and out sub-elements.

Fusedoc: responsibilities

Each Fusedoc will have a plain English explanation as to what the Fuse is meant to do. This tag does not have any attributes, the English explanation goes in between the opening and closing tags.

Fusedoc: properties

1. Fusedoc: properties: history
The history element has the following attributes:

- author: free text
- date: free text
- email: free text
- role: free text
- type: create | update

The history tag can be used as a tagset with free text between the tags:

```
<history author="Rich Tretola">
This is just a sample valid history element.
</history>
```

2. Fusedoc: properties: property

The property element has the following attributes:

- name: free text
- value: free text

The property tag is an empty tag set.

3. Fusedoc: properties: note

The note element has the following attributes:

- author: free text
- date: free text

The note tag can be used as a tagset with free text between the tags:

```
<note author="Rich Tretola">
This is just a sample valid note element.
</note>
```

Fusedoc: io

4. Fusedoc: io: in/out

Both in and out elements have no attributes. Both elements accept the following sub-elements:

- string
- number
- boolean
- list
- structure
- array
- recordset
- cookie
- datetime
- file

a) Fusedoc: io: in/out: string

The string element has the following attributes:

- name: free text
- scope: application | attributes | caller | cgi | client | form | formorurl | request | server | session | url | variables
- comments: free text
- default: free text
- mask: free text
- oncondition: free text
- optional: true | false

b) Fusedoc: io: in/out: number

The number element has the following attributes:

- name: free text
- scope: application | attributes | caller | cgi | client | form | formorurl | request | server | session | url | variables

- comments: free text
- default: free text
- precision: integer | decimal
- oncondition: free text
- optional: true | false

c) Fusedoc: io: in/out: boolean
The boolean element has the following attributes:
- name: free text
- scope: application | attributes | caller | cgi | client | form | formorurl | request | server | session | url | variables
- comments: free text
- default: free text
- oncondition: free text
- optional: true | false

d) Fusedoc: io: in/out: list
The list element has the following attributes:
- name: free text
- scope: application | attributes | caller | cgi | client | form | formorurl | request | server | session | url | variables
- delims: comma | free text
- comments: free text
- oncondition: free text
- optional: true | false
- default: free text

e) Fusedoc: io: in/out: structure
The structure element has the following attributes:
- name: free text
- scope: application | attributes | caller | cgi | client | form | formorurl | request | server | session | url | variables
- comments: free text
- oncondition: free text
- optional: true | false

The structure element contains one or more of the following sub-elements:
- string
- number
- boolean
- datetime
- array
- structure
- recordset
- list

f) Fusedoc: io: in/out: array
The array element has the following attributes:
- name: free text
- scope: application | attributes | caller | cgi | client | form | formorurl | request | server | session | url | variables
- comments: free text
- oncondition: free text
- optional: true | false

The array element contains one or more of the following sub-elements:

- string
- number
- boolean
- datetime
- array
- structure
- recordset
- list

g) Fusedoc: io: in/out: recordset

The recordset element has the following attributes:

- name: free text
- scope: application | attributes | caller | cgi | client | form | formorurl | request | server | session | url | variables
- primarykeys: free text
- mask: free text |
- comments: free text
- oncondition: free text
- optional: true | false

h) Fusedoc: io: in/out: cookie

The cookie element has the following attributes:

- name: free text
- domain: free text
- expires: free text
- path: free text
- secure: true | false
- value: free text
- comments: free text
- oncondition: free text
- optional: true | false

i) Fusedoc: io: in/out: datetime

The datetime element has the following attributes:

- name: free text
- scope: application | attributes | caller | cgi | client | form | formorurl | request | server | session | url | variables
- mask: free text |
- comments: free text
- default: free text
- oncondition: free text
- optional: true | false

j) Fusedoc: io: in/out: file

The file element has the following attributes:

- path: free text
- action: read | write | append | overwrite | delete | exists | module | include
- comments: free text
- oncondition: free text
- optional: true | false

All Your Base Are Belong To Us

Bil Corry **Lasso.Pro**
http://www.asktami.com/

Abstract

This presentation will provide the demonstration and prevention of several common web vulnerabilities that programmers introduce on their websites. The vulnerabilities to be discussed are sql injection, cross-site scripting (XSS), cross-site request forgeries (CSRF), side-jacking, and input validation. The goal of this presentation is to have the participant leave with a strong understanding of each of the vulnerabilities, why they're important to protect against, and how to protect against them. Also discussed will be auditing tools to help discover these vulnerabilities on your site before it's exploited

Biography

Bil Corry has been developing web applications with Lasso for more than 10 years. During this time, he also has been a major contributor to the Lasso community, including participating on LassoTalk (top 5 all-time contributor); code changes for Email_Send, Auth Tags, PDF Tags, Memory Session Manager, File_Serve, and others; code contributions on TagSwap which include more than 100 custom tags; article and edit contributions on LassoTech; and has spoken at a previous Lasso Developers Conference. Bil holds a degree in Computer Science from California State University, Fullerton.

Lasso Developer Conference Materials Download
Server: **http://dowloads.lassosoft.com/beta/** Username: **LDC**
Path: **Lasso Developer Conference** Password: **LCROSS**

All Your Base Are Belong To Us

(Primer on Lasso & WebAppSec)

Bil Corry
Lasso.Pro

Presented at:
Lasso Developer Conference
September 18-21, 2008
Chicago, Illinois, USA

INTRODUCTION

> *"How are you gentlemen !! All your base are belong to us. You are on the way to destruction. You have no chance to survive make your time. Ha Ha Ha Ha ...."*
>
> - CATS in Zero Wing

Web application security (aka WebAppSec) is one of many disciplines[1] a web developer must master in order to create secure, competent websites. This paper will discuss common vulnerabilities found in web applications and the methods you can use, programming in Lasso, to prevent them.

MONETIZING YOUR WEBSITE

Before we get to the "how" of WebAppSec, we're first going to cover the "why"; why is your website a target?

The short answer: money.

There is a belief among some that their little piece of the Internet is far too small and humble to be of any interest to cyber-criminals.[2] And logically, it may be confusing to understand how your local volunteer soccer league website could be of any value to cyber-criminals. Really, how can a site that has no personal information to steal, no financial information to steal, and simply functions to display the results of soccer games generate an income for cyber-criminals? Mostly likely not even the site owner is generating revenue from the site!

There are two forces that have converged that allow monetizing any website a reality: automated penetration tools and diversity of income streams. Automated penetration tools have made it so cyber-criminals do not have to individually and manually crack each website. Instead, they leverage automated tools that can crack tens of thousands of websites.[3] [4] And thanks to Google, they don't even have to crawl your web application, Google already has and can be used to target sites that match specific requirements and vulnerabilities.[5]

Once the website has been cracked, they can then use one or more strategies for monetizing the site. Here are some of the strategies they employ:

1. Link spam – you'll often see this in guest books and blog comments; link spam generates revenue by enticing your site users to visit the attacker's site (which sells products, has advertising, installs malware, etc). An interview with one such link spammer claimed he made over £100,000 per month![6] And the links also raise the relative value of the attacker's website in the eyes of the search engines, giving it a higher rank when doing searches.[7]
2. Hosting – free, anonymous hosting; cyber-criminals can use a cracked server to host their own content, including warez, malware, phishing sites, pirated content, etc.[8]

---

1 Top 10 Concepts That Every Software Engineer Should Know
http://www.readwriteweb.com/archives/top_10_concepts_that_every_software_engineer_should_know.php

2 Here's one example:
http://groups.google.com/group/google-safe-browsing-api/browse_thread/thread/950c15743e0a3c19/7d1f8600b53e1160

3 Hackers Hijack a Half-Million Sites
http://www.pcworld.com/businesscenter/article/145791/hackers_hijack_a_halfmillion_sites.html

4 Massive SQL Injection Attack 600.000++
http://www.0x000000.com/?i=556

5 Google Hacking Database (GHDB)
http://johnny.ihackstuff.com/ghdb.php

6 Interview with a link spammer
http://www.theregister.co.uk/2005/01/31/link_spamer_interview/

7 Spamdexing: Link Spam
http://en.wikipedia.org/wiki/Link_spam#Link_spam

8 Hacked bank server hosts phishing sites
http://www.computerworld.com/hardwaretopics/hardware/server/story/0,10801,109500,00.html

3. Infect users – your visitors represent an income stream by exploiting their computer and adding it to the attacker's botnet. Their computer can then be used to send spam, launch denial-of-service attacks, rented out to others, etc.[9] [10] [11] And your visitors' computer can also be searched for valuable data, and in combination with a keylogger, the attacker can steal personal information, drain bank accounts, and sell this information to others.[12] In fact, your visitor's computer is so valuable that botnet operators have been known to apply security patches and run defenses against malware to prevent other bot herders from taking over their bots.[13]

And should you decided to only allow certain IP addresses or registered users to interact with the site, know that if it sits on a shared server, it still may be vulnerable via a weakness in another webapp on the same server.[14]

GOLDEN RULE OF CLIENT INPUT

Now that you understand the "why" of WebAppSec, we're now going to delve into the "how" of specific web application vulnerabilities. As we do, keep in mind the golden rule of client input:

"All client input (headers and request) is hostile until proven otherwise or sanitized."

To understand how to prove client input isn't hostile, I'll first start with input validation, then follow it with discussions about four common web application vulnerabilities: SQL injection, cross-site scripting, cross-site request forgeries, and side-jacking. By the end of this paper, you should have a firm grasp of common web vulnerabilities.

9 Storm For Rent
http://www.forbes.com/technology/2008/01/09/storm-worm-cybercrime-tech-security-cx_ag_0109storm.html
10 Home PCs rented out in sabotage-for-hire racket
http://www.usatoday.com/tech/news/computersecurity/2004-07-07-zombie-pimps_x.htm
11 Know your Enemy: Tracking Botnets
http://www.honeynet.org/papers/bots/
12 Underground market for stolen IDs thrives
http://www.usatoday.com/money/industries/technology/2005-03-02-datathieves-usat_x.htm
13 Zombie PCs growing quickly online
http://news.bbc.co.uk/2/hi/technology/4685238.stm
14 MSN IP Search
http://ha.ckers.org/blog/20080803/msn-ip-search/

INPUT VALIDATION

Input validation and sanitation are important tools to protecting your web application. Any input coming from the user must be either validated (that is, make sure the data is in a verified, expected format) or sanitized (only accept whitelisted data or transform the data into a safe format). Failure to do either of these will expose your web application to a variety of attacks.

Let's first talk about whitelisting versus blacklisting. Whitelisting means you have a limited set of acceptable inputs; you either reject or remove anything non-valid. Blacklisting means you have a limited set of UNacceptable inputs; again, you either reject or remove anything non-valid. While they don't appear to be much different, whitelisting is much more secure because you're guaranteed to only get what you allow. Blacklisting has the disadvantage in that you must imagine all attack scenarios and try to block each one. A good example of this is trying to blacklist malicious HTML – imagine you wish to block the <script> tag by looking for "<script"; what you fail to realize is there are UTF-8 whitespace characters that can be inserted into the string that will thwart your blacklist, but still allow the <script> tag to be recognized by the browser (e.g. "<scr*ipt" where the asterisk is a Unicode whitespace character).[15] You should use blacklisting sparingly or as a secondary defense.

The types of validations you should perform depends on where and how the user input is used. In the latter sections on the vulnerabilities, I'll discuss the specific validations required for them. But for now, I'll go over generic validations.

Let's first identify the sources of user input[16]:

- Query parameters in the URL
- The path of the URL (may be echoed as part of "Document not found" error messages)
- Form fields (including hidden fields)
- Cookies
- Other parts of the HTTP request header (such as the referrer URL)
- Data that was inserted into a data store in an earlier transaction, possibly by a different user (e.g. messages in Google Groups, Orkut, GMail).
- Data obtained from a datafeed (e.g. merchant feeds in Google Product Search)
- Data crawled from the web (Google Search) or the local disk (Google Desktop)

As you can see, there are a variety of ways user input is received by Lasso. If you don't use the data (such as the referrer), then you don't have to validate it. You only validate the user input that gets used in your web application.

A general approach to validation is to only allow what you're expecting. So if you're expecting an integer, then either validate that it's an integer, or explicitly cast it as an integer. For example:

```
// validate integer
if(string_isDigit(action_param('quantity')));
        // it's an integer!
/if;

// cast to integer to ensure it's an integer
integer(action_param('quantity'));
```

Personally, I prefer to cast it to the type I want, then perform further validation on it (such as making sure it's non-zero for example). You'd do similar validation/casting on decimals and dates too.

15 More about whitespace characters used to thwart blacklisting:
http://www.gnucitizen.org/blog/snippets-of-defense-ptiv/

16 Verbatim from: Introduction to Cross-Site Scripting Vulnerabilities
http://code.google.com/p/doctype/wiki/ArticleIntroductionToXSS#Sources_of_Untrusted_Data

When using HTML forms, be sure checkboxes, selects or anything else with limited choices really only allows the limited choices. For example:

```
var('valid_choices') = (:'Red','Green','Blue');
var('choice') = 'Red'; // default
if($valid_choices->contains(action_param('choice')));
        $choice = action_param('choice');
/if;
```

When using hidden inputs on forms, you either should validate that they contain a value that's acceptable, or better, just encrypt them, then decrypt them on the response page. That will prevent a malicious user from changing them. You can use [lp_var_pack][17] and [lp_var_unpack][18] to perform the encryption/decryption.

Strings can be validated in a variety of ways, for example Lasso can validate email addresses and credit card numbers (validate that their format matches a known good format). You can also build custom validation for phone numbers, identifiers such as Social Security numbers (if in the States), etc.

For URLs, Google offers a "Safe Browsing API[19]" that allows you to verify if the URL is a known phishing/malware site. While a Lasso interface to the API doesn't currently exist, it's on my projects list, so expect one in the future.

Input validation is the corner-stone of web application security; remember the Golden Rule of Client Input:

> **"All client input (headers and request) is hostile until proven otherwise or sanitized."**

Now let's look at specific web vulnerabilities, starting with SQL injection.

17 [lp_var_pack]
http://tagswap.net/lp_var_pack
18 [lp_var_unpack]
http://tagswap.net/lp_var_unpack
19 Google Safe Browsing API
http://code.google.com/apis/safebrowsing/

SQL INJECTION

SQL injection attacks occur when unsanitized client input is allowed to be used in a SQL query. When unmitigated, an attacker can create their own SQL queries against your datasource (or more likely is an automated tool will compromise your web application[20]). This isn't an issue if you use classic inlines[21] as Lasso will sanitize the client input for you, but if you use -sql inlines, then you must understand how to protect against SQL injection.[22]

There are two ways in Lasso to protect against SQL injection when you want to write your own SQL query; one is to use Prepared Statements and the other is to carefully build a SQL query string using sanitize client input.

Prepared Statements provide the best protection because no sanitation is required on the client input yet you still get to use a SQL query string. Note that not all SQL queries are supported as Prepared Statements in MySQL, so be sure to read the MySQL documentation for more details.[23] How Prepared Statements in Lasso works is you simply specify the query you want to execute, provide the data one or more times and Lasso handles the rest.

An example of a Prepared Statement in Lasso looks like this: [24]

```
Inline(-Database='Contacts', -Table='People',
        -Prepare="INSERT INTO people (`first name`, `last name`) VALUES (?, ?)')";

        Inline(-Exec=Array('Bil','Corry'));
        /Inline;
        Inline(-Exec=Array($firstname,$lastname));
        /Inline;
        Inline(-Exec=Array(action_param('firstname'),action_param('lastname'));
        /Inline;
/Inline;
```

I won't delve into specifics about how to use Prepared Statements as it's well covered in the Lasso Language Guide[25] and a Tip of the Week[26] and is beyond the scope of this paper.

Building your own SQL query strings for -sql inlines is where developers get into trouble; all client input must be sanitized before it can be used in a -sql inline. How it gets sanitized depends on where in the SQL query string you're inserting the client input.

Let's walk through the four most common injection points in SQL query strings and how to secure them.[27]

---

20 Mass SQL Attack a Wake-Up Call for Developers
http://www.technewsworld.com/story/Mass-SQL-Attack-a-Wake-Up-Call-for-Developers-62783.html

21 By "classic inlines" I mean inlines where you specify an action other than -sql (e.g. -add, -delete). All Filemaker inlines are classic inlines.

22 Examples of real-world SQL injection is here:
http://www.evilsql.com/

23 MySQL Documentation - 12.7. SQL Syntax for Prepared Statements
http://dev.mysql.com/doc/refman/5.0/en/sql-syntax-prepared-statements.html

24 Adapted from example in Lasso Language Guide:
http://docs.lassosoft.com/Lasso%208.5/003%20Language%20Guide/002%20Database/007%20Database%20Interaction%20Fundamentals/index.lasso#PreparedStatements

25 Prepared Statements – Lasso Language Guide
http://docs.lassosoft.com/Lasso%208.5/003%20Language%20Guide/002%20Database/007%20Database%20Interaction%20Fundamentals/index.lasso#PreparedStatements

26 Lasso 8.5 MySQL Prepared Statements – Tip of the Week for September 8, 2006
http://www.lassosoft.com/Documentation/TotW/index.lasso?9185

27 Much of this is adapted from my 2004 article on SQL Injection:
http://tagswap.net/articles/SQL_Injection/

Strings

SQL injection into SQL strings happens when a user-provided string is merged into a sql inline and the quotes used to delimit the SQL string are also contained within the user-provided string. Here is a SQL query that plugs in the user-provided value for searching on a first name:

```
var('sql')="SELECT name_first, name_last FROM contacts WHERE
name_first='"+action_param('firstname')+"'";
```

Imagine firstname = "Bil" - the sql query becomes this:

```
SELECT name_first, name_last FROM contacts WHERE name_first='Bil'
```

So far, so good, right? Now imagine instead the user entered:

```
crackerboy'; delete from contacts; #
```

Now the query becomes:

```
SELECT name_first, name_last FROM contacts WHERE name_first='crackerboy';
delete from contacts; #'
```

Viola! Your entire contacts table is hosed. (The pound sign # tells MySQL that the rest of the line is a comment). So how do you prevent SQL injection into SQL strings? You must escape all the quotes in the data provided by the client. Fortunately, Lasso has a built-in tag to do it for you, [encode_sql][28].

Here's the original query rewritten to use it:

```
var('sql')="SELECT name_first, name_last FROM contacts WHERE
name_first='"+encode_sql(action_param('firstname'))+"'";
```

So properly encoded, now the query becomes:

```
SELECT name_first, name_last FROM contacts WHERE name_first='crackerboy\';
delete from contacts; #'
```

That slash \ in front of the embedded quote tells MySQL to ignore it, and thus, the entire string becomes the name_first search criteria.

So that's how you protect your strings in queries, but what about numeric values?

Numeric Values

Unfortunately, numeric values can not be protected using encode_sql; reason being you wouldn't (normally) have quotes around the value to begin with, so you wouldn't need to escape any quotes being sent.

So let's take a look at a query that uses a numeric value:

```
var('sql')="SELECT name_first, name_last FROM contacts WHERE
userid = "+action_param('userid');
```

28 [encode_sql] is for MySQL datasources. There's also [encode_sql92] for datasources that require a different type of escaping, such as SQLite.

Imagine userid = "123" - the sql query becomes:

```
SELECT name_first, name_last FROM contacts WHERE userid = 123
```

Already this has a problem, the user could feed sequential numbers to your query and pull out the entire database list. But that's another topic for another time.

Now imagine instead the user entered:

```
123; delete from contacts;
```

Now the query becomes:

```
SELECT name_first, name_last FROM contacts WHERE userid = 123; delete from contacts;
```

Viola! Your entire contacts table is hosed. (Notice the theme here?)

And even if we encode_sql the passed-in value, it wouldn't change anything. So how do you prevent an attacker from hosing your table this time? Make sure that you explicitly pass a numeric if that's what your query is expecting.

While you could validate the passed-in value to ensure it's numeric using something like [string_isdigit], I instead just cast it explicitly to a numeric using [integer] and [decimal] (depending on if the query needs an integer or decimal).

So let's rewrite that query to prevent sql injection for integer:

```
var('sql')="SELECT name_first, name_last FROM contacts WHERE
userid = "+integer(action_param('userid'));
```

So what happens, you ask, if the user provides the following?

```
123; delete from contacts;
```

Well, when Lasso converts a string to an integer or decimal, it takes every digit it can until it hits a non-numeric character. So casting to integer, the query would become:

```
SELECT name_first, name_last FROM contacts WHERE userid = 123
```

Decimals are handled in an identical fashion.

Date Values

Dates in MySQL can either be strings or numeric; MySQL accepts either. But to prevent SQL injection, the trick is to cast the passed-in value to a Lasso date type using [date] and formatting the output for MySQL:

```
var('lassoDate') = lp_date_stringToDate(action_param('date'),-error=date);
var('sql')="SELECT name_first, name_last FROM contacts WHERE
last_login <= "+$lassoDate->format('%Q');
```

Since a Lasso date type can not hold anything other than dates, you are assured that any bogus strings will be rejected when trying to cast it to a Lasso date.

LIKE Queries

LIKE queries are really just strings, but with an extra twist of pattern matching. What that means is LIKE queries will recognize special characters within the string as wildcard characters, namely, "%" and "_". You can read more about what "%" and "_" do in a pattern-matching query in the MySQL documentation.[29]

To show how "%" and "_" can be abused, let's start off with an example query:

```
var('sql')="SELECT name_first, name_last FROM contacts WHERE
name_last LIKE '"+encode_sql(action_param('lastname'))+"%'";
```

This query will allow someone to enter the first letter(s) of a last name, and find all contacts that begin with them. So imagine the user entered "cor" (you plan to require and validate that they entered at least three characters), the query would look like:

```
var('sql')="SELECT name_first, name_last FROM contacts WHERE
name_last LIKE 'cor%'
```

But imagine if the user entered "%%%", that would validate as three characters and the query would become:

```
SELECT name_first, name_last FROM contacts WHERE name_last LIKE '%%%%'
```

That query would find all of your contacts! So how to handle LIKE? You have two options. Either remove all "%" and "_" found in the string or escape them by replacing "%" with "\%" and "_" with "_". Be sure to still encode_sql the string. Note that you only escape "%" and "_" for LIKE queries (or any of the other pattern-matching queries that recognize "%" and "_"), don't go doing it on all your query strings.

29 MySQL Documentation - 3.3.4.7. Pattern Matching
http://dev.mysql.com/doc/refman/5.0/en/pattern-matching.html

CROSS-SITE SCRIPTING (XSS)

Cross-site scripting (XSS) attacks occur when unsanitized client input is used in content served to the client and/or others. When unmitigated, XSS can perform actions in the context of the site serving the page. This attack can steal cookies, modify the web application behavior, or modify the DOM/page content[30], often without the victim knowing the attack had occurred.[31]

To get a sense of how prevalent this attack is, it's interesting to note that financial sites such as Citibank, Barclays, Paypal and HSBC, security vendors such as McAfee, Verisign and Symantec, and large sites such as Google, eBay, Facebook and MySpace currently have, or have had XSS vulnerabilities within the last six months.[32]

When looking at solutions to protecting your web application, it should be noted that it is very difficult to blacklist all bad content – there are many ways to evade a blacklist filter[33]. A better approach is to only allow whitelisted content and/or encode the content to render it safe.

Let's look at a simple example. Imagine you have a page where the user enters a search term, which on the response page shows up as:

```
[var('search') = action_param('search_term')]
Your search for [$search] found [found_count] results.
```

Imagine the user entered:

```
test<script>alert('hello')</script>
```

Now the page, as served to the user, would look like:

```
Your search for test<script>alert('hello')</script> found 0 results.
```

The user would then see a dialog box with “hello” as the dialog alert text (assuming JavaScript was enabled in the user's browser). Now that isn't terribly interesting because most user's are not trying to exploit themselves. But imagine now that we take it one step further and create an URL that will take any user to the page and show them the exploit:

```
http://mysite.tld/response.lasso?search_term=test%3cscript%3ealert(%27hello%27)%3c%2fscript%3e
```

Now a malicious site can redirect a user to the above URL and execute a script of the attacker's choosing in the context of the vulnerable site. And while alerting 'hello' isn't terribly interesting, an attacker is much more likely to script a solution that steals cookies or performs some other maliciousness.

30 Google doctype – Introduction to Cross-Site Scripting Vulnerabilities
http://code.google.com/p/doctype/wiki/ArticleIntroductionToXSS

31 Wikipedia – Cross-site scripting
http://en.wikipedia.org/wiki/Cross-site_scripting

32 </xssed> xss attacks information
http://www.xssed.com/

33 XSS (Cross Site Scripting) Cheat Sheet
http://ha.ckers.org/xss.html

The above example used a request parameter to exploit the site, it is possible to use other client input vectors, such as user-agents[34], referrers[35], file downloads[36], cookies[37], or any other data received from the client that is used by the web application.

Now that we've discussed the issue, let's look at ways to protect against it. The first thing to understand is we're trying to prevent client input from becoming something malicious. How you protect against it depends entirely on what you're doing with the client input. For example, using our vulnerable code above, to protect our application against XSS, we can encode_html the user input:

```
Your search for [encode_html($search)] found [found_count] results.
```

Now the page, as served to the user, would look like:

```
Your search for test&lt;script&gt;alert(&#39;hello&#39;)&lt;/script&gt; found 0 results.
```

That fixes the problem by rending the <script> as plain text instead.

Let's look at protecting against XSS in the three most common situations:

HTML Body[38]

When client input is used within the <body> of your page, you should use [encode_html] to encode it:

```
Your search for [encode_html(action_param('search_term'))] found [found_count] results.
```

HTML Attribute[39]

When client input is used within a HTML tag attribute, again use [encode_html]:

```
<input type="text" name="test" value="[encode_html(action_param('test'))]">
```

In addition, be sure to surround the attribute with quotes to prevent attribute injection attacks, such as vulnerable code looking like:

```
<input type="text" name="test" value=[encode_html(action_param('test'))]>
```

Which when feed this:

```
test onmouseover=evil_script()
```

---

34 Vulnerable Vulnerability Databases
http://www.0x000000.com/?i=546

35 A Second-order of XSS
http://blogs.iss.net/archive/SecondOrderXSS.html

36 Google XSS
http://xs-sniper.com/blog/2008/04/14/google-xss/

37 XSS, Cookies, and Session ID Authentication – Three Ingredients for a Successful Hack
http://www.informit.com/articles/article.aspx?p=603037&rll=1

38 HOWTO filter user input in regular body text
http://code.google.com/p/doctype/wiki/ArticleXSSInBodyText

39 Google doctype – HOWTO filter user input in tag attributes
http://code.google.com/p/doctype/wiki/ArticleXSSInAttributes

Becomes:

```
<input type="text" name="test" value=test onmouseover=evil_script()>
```

That will cause a malicious script to execute when the mouse is moved over the <input>.

URL Attribute[40]

When client input is used within an URL attribute, use [encode_strictURL] to properly encode their input. For example, imagine your code looks like:

```
[var('test') = action_param('test')]
<a href="test.lasso?param=[$test]">Test</a>
```

If the user supplies the following for the "test" parameter:

```
test"<script>alert('ok')</script>
```

Then the page is served as:

```
<a href="test.lasso?param=test"<script>alert('hello')</script>">Test</a>
```

Which triggers the JavaScript. Adding [encode_strictURL] fixes the issue:

```
<a href="test.lasso?param=[encode_stricturl($test)]">Test</a>
```

The browser is served:

```
<a href="test.lasso?param=test%22%3cscript%3ealert(%27hello%27)%3c%2fscript%3e">Test</a>
```

Problem solved.

Other Attack Vectors

For further reading, Google's doctype project does a great job of providing details on protecting against XSS in a variety of scenarios, including JavaScript and Flash.[41] I highly recommend reading through it to better understand how to protect against XSS.

40 Google doctype – HOWTO filter user input in URL attributes
http://code.google.com/p/doctype/wiki/ArticleXSSInUrlAttributes

41 Google doctype – Web security
http://code.google.com/p/doctype/wiki/ArticlesXSS

CROSS-SITE REQUEST FORGERIES (CSRF)

Cross-site request forgery (CSRF aka XSRF) attacks occur when a victim's browser is secretly directed to a target site and as a result, an action is performed on behalf of the victim as if the victim him/herself had requested it.[42] Typically, there isn't any indication to the victim that any such action has even taken place.

A simple example to illustrate. Say you visit a random site and it has the following embedded in it:

```
<img src="http://www.google.com/setprefs?hl=xx-piglatin">
```

Your browser would try to retrieve an image from that URL, and in doing so, would send a request to Google using your credentials (assuming you've visited Google at least once). From Google's point of view, the request appears legitimate because it's coming from your browser with your cookies. If successful, the above attack would change your default language with Google to Pig Latin; fortunately Google has fixed that CSRF vulnerability (so don't try it at home).[43]

I should note here that with pure CSRF, the attacker is guessing you're a user that is authenticated with a particular website and that you're currently "signed in". And know that the attacker does not have access to the content returned by the server. The attacker can only trick your browser into sending a request to a server and that's the extent of the attack. But even with those limitations, there are a lot of interesting attacks that can be done.

Some real-world examples[44]: using CSRF to perform SQL injection against PHPMyAdmin[45], attacking a site that isn't vulnerable to XSS and using it's IMAP server against it in a reflected attack[46], cookie stuffing to defraud affiliate programs[47], and perhaps one of the more devious uses is Cross-Site File Upload attacks[48].

There are two preferred ways to defeat a CSRF attack: use a unique token or re-authenticate the user.

The way a unique token works is a sufficiently random token is added to each request, then the server verifies the unique token and allows the action. If the unique token is not present or incorrect, the server can reject the request and/or ask the user to re-authenticate. This prevents CSRF because the attacker won't know the token, and thus can't create a valid request for the victim.

Re-authenticating the user acts as a simple "Are you sure?" measure. Since the attacker won't know the username and password for the victim, it isn't possible to successfully execute a CSRF against the target.

Beyond those two methods, there are a few others to defeat a CSRF attack, but they have caveats that make them less desirable to implement. I don't recommend using them, but you can read more about using referrers, double-submit cookies, and POST-only requests at Wikipedia if interested.[49]

---

42 A nice CSRF overview is here:
http://www.0x000000.com/index.php?i=309&bin=100110101

43 Security Threat: Cross Site Request Forgery (CSRF)
http://itmanagement.earthweb.com/secu/print.php/3739621

44 CSRF Hacking Database
http://csrf.0x000000.com/csrfdb.php?do=browse

45 SQL Injecting PhpMyAdmin
http://www.0x000000.com/?i=587

46 The Extended HTML Form Attack Revisited
http://enablesecurity.com/2008/06/18/the-extended-html-form-attack-revisited/

47 Affiliate Programs Vulnerable to Cross-site Request Forgery Fraud
http://www.cgisecurity.org/2008/08/affiliate-progr.html

48 Cross-site File Upload Attacks
http://www.gnucitizen.org/blog/cross-site-file-upload-attacks/

49 Wikipedia – Cross-site request forgery
http://en.wikipedia.org/wiki/CSRF

To give a better idea of CSRF works and how to fix it, let's look at a typical Lasso implementation that's vulnerable to CSRF. I'm going to use a very benign example that wouldn't normally be of much concern in the real world, but it'll get the idea across.

Let's say we have a web application that has a simple “Sign Out” link on every page. The “Sign Out” link looks like this:

```
<a href="/auth/signout.lasso">Sign Out</a>
```

The page signout.lasso works by expiring the current Lasso session. So with that knowledge, a malicious site could target ours by adding this to one of its pages:

```
<img src="http://oursite.tld/auth/signout.lasso">
```

By visiting that malicious page, our session would be terminated on our site. If you remember, there are two ways to defend against this CSRF attack. One is to re-authenticate the user. So on the signout.lasso page, it would prompt the user to enter their username and password to authorize the signing out action. In the context of our malicious page, the request generated by the <img> HTML tag would no longer work since a username and password are now expected.

The other way to defend against this CSRF attack is to use tokens. Using this technique our sign out link would then become:

```
<a href="/auth/signout.lasso?token=Yc9QOe1mLNeQ9SOSM05f">Sign Out</a>
```

The page signout.lasso would look at action_param('token') and make sure it matched the value expected. You can store the value expected in the user's session, or in a database, or you could go another route and instead just encrypt_blowfish the current time and some user identifier (user id, IP address, etc) as the token, then on signout.lasso decrypt it making sure that the time is within five minutes of when it was created and the identifier still matches the current user.

SIDE-JACKING

Side-jacking attacks can occur when a user is (1) on a network that is sniffable, (2) the packets are traveling unencrypted, and (3) the user is using a web application that utilizes cookies to maintain user sessions. The attack is carried out by watching for traffic that contains cookies. By capturing the cookie and replicating it on the attacker's own system, the attacker can then pretend to be the victim to the web application.

As you can see from the description, side-jacking isn't so much a server issue as it is a client issue – as a user, you can protect yourself by never using open wi-fi networks (or if you must, use VPN to encrypt all of your traffic); always use HTTPS when possible; and you can mitigate some risk by logging out explicitly from web applications (which renders the session cookie useless).[50]

There are some measures a server can take to minimize the risk for its users. The best solution is to run all authenticated sessions (those in which a user has logged in) exclusively through HTTPS. When doing so, be sure to detect if a user tries to connect via HTTP and redirect to HTTPS if they do. Also, be sure to use the -secure option of [session_start] for the user session; this will prevent the session cookie from being sent on HTTP, which is an important defense to accidentally leaking the cookie data outside the HTTPS connection.

Another good practice is to provide a mechanism for users to log out of the application; as mentioned above, this helps mitigate the risk of side-jacking by invalidating the session. So even if the attacker captures the session cookie data, if the session is invalid, it can't be used. And of course, limiting the session life to something short (under a half-hour) is also good practice as many users never will log out themselves.

If you're not running HTTPS, then beyond the issue of having cookies stolen via side-jacking, you also have the problem of the user's username and password being sent in cleartext when logging into the web application. The best solution for that is to use Digest Authentication[51].

As for preventing side-jacking in a scenario where the packets are sniffable and unencrypted, there is no “bullet-proof” method. The best you can do is to create a “fingerprint” of the browser that successfully logged in, store the fingerprint in the session, then detect when a browser with a different fingerprint tries to use the session. The fingerprint can be anything unique to the request such as user-agent, ip address, language preferences, etc. Just know in the case of IP addresses, users behind a proxy may all share the same IP address (so you won't catch an attacker if he is on the same wi-fi network that is routed through a common proxy), and also one user behind a proxy may have multiple IP addresses, as is the case with AOL[52]. I'd recommend filtering on IP address unless the user requests that the IP check be turned off (make it optional).

Another solution is to rotate session IDs, issuing a new session ID on every page request.[53] The advantage to that is the session ID lives only for as long as the legitimate user stays on the current page, so it forces an attacker to move immediately against the victim. The disadvantage is it has some funky behavior when reloading the page or using the back button if you embed the session ID in the links rather than as a cookie, so experiment with it and see if it will fit your needs.

If you're interested in seeing side-jacking in action, there's a tool you can use called “Hamster[54]”, although I believe it's Windows-only currently.

---

50 'Sidejacking' Tool Unleashed
http://www.darkreading.com/document.asp?doc_id=130692

51 Wikipedia – Digest access authentication
http://en.wikipedia.org/wiki/Digest_access_authentication

52 AOL Proxy Info
http://webmaster.info.aol.com/proxyinfo.html

53 It's experimental in Lasso – Fletcher talks about it in the thread “Kill Session on Close Browser Window” from 2005-09-29:
http://www.listsearch.com/Lasso/Thread/index.lasso?12572#203922

54 SideJacking with Hamster
http://erratasec.blogspot.com/2007/08/sidejacking-with-hamster_05.html

SUMMARY

It is clear that the days of security by obscurity is long dead. Given that the financial rewards for criminal activity has climbed into the billions[55] and the risk of being caught is slight[56] (and even if caught, it might work out favorably anyhow[57]), security must now be an integral part of any web application. It is no longer a question if the financially-motivated bot masters, with hundreds of thousands of computers at their disposal, will find you[58], it's just a matter of when. And when they do, what will they find?

The goal for this paper is to give you insight into the most common vulnerabilities found in web applications. With education and proper planning, security doesn't have to be something that is accounted for after an application has been deployed, but instead can be part of the development and testing cycle.

In the appendices that follow, you'll find a list of auditing tools, helpful Firefox plug-ins, additional topics to explore, and a couple of professional groups to consider joining. These lists are not exhaustive, but rather are a starting point from which you can expand your knowledge and education of WebAppSec.

The last appendix is a list of blogs that I read regularly which has greatly expanded my own personal knowledge of WebAppSec issues. I definitely suggest adding them to your favorite RSS reader; security is an on-going learning process and being exposed to new ideas is a great way to incrementally increase the depth and breadth of your WebAppSec knowledge.

ABOUT THE AUTHOR

Bil Corry is founder of Lasso.Pro, an international web application development and consulting firm specializing in scalable, secure web applications. Bil has been developing web applications using Lasso for more than 10 years. During this time, he also has been a major contributor to the Lasso community, including participating on LassoTalk (top 5 all-time contributor); code contributions for Email_Send, Auth Tags, PDF Tags, Memory Session Manager, File_Serve, LassoWiki and others; code contributions on TagSwap.net which include more than 100 custom tags; article and edit contributions on LassoTech; winner of a Lasso Programming Challenge; and has spoken at a previous Lasso Developer Conference. Bil holds a degree in Computer Science from California State University, Fullerton.

CONTACT INFORMATION

Bil Corry
Lasso.Pro (http://lasso.pro)
Phone: +1 240 337 2514
Email: (my first name)@lasso.pro

55 Online Threats Cost Consumers $8.5 Billion Over Last Two Years
http://news.yahoo.com/s/cmp/20080805/tc_cmp/209901659

56 Why Hackers Are A Step Ahead of the Law
http://news.cnet.com/2009-1017-912708.html?hhTest=1&tag=fd_lede

57 New Zealand Hacker Released As Police, Judge, Prosecutors All Praise His Mad Hacking Skillz
http://techdirt.com/articles/20080716/1236481702.shtml

58 Attacking Around the Globe Around the Clock
http://blog.imperva.com/2008/04/attacking-around-the-globe-aro.html

APPENDIX A – AUDITING TOOLS

Burp – integrated platform for attacking web applications:

http://portswigger.net/suite/

Google Ratproxy – passive web security assessment tool:

http://code.google.com/p/ratproxy/

Hamster/Ferret – tool for testing side-jacking:

http://erratasec.blogspot.com/2007/08/sidejacking-with-hamster_05.html

Metasploit – somewhat of a Swiss Army knife for attacking a target:

http://www.metasploit.org/

Pantera Web Assessment Studio – web application analysis engine:

http://www.owasp.org/index.php/Category:OWASP_Pantera_Web_Assessment_Studio_Project

Paros Proxy – web application security assessment:

http://www.parosproxy.org/

APPENDIX B – FIREFOX PLUG-INS

Firebug – Firefox plug-in that allows viewing XHR requests, script debugging, and much more:

https://addons.mozilla.org/en-US/firefox/addon/1843

Firecookie – Firefox plug-in that allows viewing and managing cookies:

https://addons.mozilla.org/en-US/firefox/addon/6683

Live HTTP Headers – Firefox plug-in for viewing the HTTP headers going between Firefox and the server:

https://addons.mozilla.org/en-US/firefox/addon/3829

NoScript – Firefox plug-in that allows turning JavaScript on/off; must have to protect your browser too:

https://addons.mozilla.org/en-US/firefox/addon/722

TamperData – Firefox plug-in to view and modify HTTP/HTTPS headers and post parameters:

https://addons.mozilla.org/en-US/firefox/addon/966

User Agent Switcher – Firefox plug-in that allows easy switching of the user agent:

https://addons.mozilla.org/en-US/firefox/addon/59

Web Developer – Firefox plug-in that provides a variety of tools for web development:

https://addons.mozilla.org/en-US/firefox/addon/60

XSS-Me & **SQL Inject-Me** – Firefox plug-ins for detecting XSS and SQL injection vulnerabilities:

http://securitycompass.com/exploitme.shtml

APPENDIX C – ADDITIONAL TOPICS

Additional topics to explore (in no particular order):

Timing Attacks – be sure to read Section 6, "Timing and its implications for Privacy" in the PDF:

http://www.sensepost.com/research/squeeza/dc-15-meer_and_slaviero-WP.pdf
http://ha.ckers.org/blog/20080828/more-timing-precision-enhancements/

Client Authentication – Do's and Don'ts:

http://prisms.cs.umass.edu/~kevinfu/papers/webauth_tr.pdf

ZeroSum – create checksums of web directories to detect malicious changes to your web app files:[59]

http://www.0x000000.com/?i=550

Logic flaws – some examples where logic flaws were exploited:

Man Allegedly Bilks E-trade, Schwab of $50,000 by Collecting Lots of Free 'Micro-Deposits'
http://blog.wired.com/27bstroke6/2008/05/man-allegedly-b.html

Youtube's 18+ Filters Don't Work
http://www.darkseoprogramming.com/2008/06/01/youtubes-18-filters-dont-work/

Privacy flaw exposes Paris Hilton and Lindsay Lohan's private MySpace photos
http://blogs.zdnet.com/security/?p=1244

Yahoo SEM Logic Flaw
http://ha.ckers.org/blog/20080616/yahoo-sem-logic-flaw/

AT&T using User-Agent to give free wi-fi to iPhones at Starbucks:
http://blogs.zdnet.com/security/?p=1067
http://www.darkreading.com/blog.asp?blog_sectionid=447&doc_id=153200

Inside Jobs – some examples how people on the inside breach the system:

Hidden Code Costs Poker Players Thousands
http://catless.ncl.ac.uk/Risks/25.20.html#subj3

S.F. officials locked out of computer network
http://www.sfgate.com/cgi-bin/article.cgi?f=/c/a/2008/07/14/BAOS11P1M5.DTL

One in three IT staff snoops on colleagues
http://www.msnbc.msn.com/id/25263009/

Day of Reckoning? Super Rich Tax Cheats Outed by Bank Clerk
http://abcnews.go.com/Blotter/Story?id=5378080&page=1

Data voyeurism is common
http://redtape.msnbc.com/2008/03/surprised-by--1.html

59 On my to-do list is to port this to Lasso.

ModSecurity – an Apache module that acts as a web application firewall:

http://www.modsecurity.org/

Google DocType – articles on web security:

http://code.google.com/p/doctype/wiki/ArticlesXSS

Google Code University – web security courses:

http://code.google.com/edu/security/index.html

Web Application Hackers Handbook – attack checklist from the book

http://portswigger.net/wahh/tasks.html

Internet Jurisdiction – where are your websites physically located?

Do The Good People Of Florida Think Your Website Is Obscene? You Better Hope Not.
http://www.alleyinsider.com/2008/6/do_people_in_florida_think_your_website_is_obscene_

Host-Proof Data Encryption – data stored by the server is encrypted in such a way that only the user can view it:

http://ajaxpatterns.org/Host-Proof_Hosting#Solution

Rich Content Filters – allowing users to upload HTML can be risky:

Bullet-proof rich content filters:
http://www.gnucitizen.org/blog/bulletproof-rich-content-filters

HTML Purifier
http://htmlpurifier.org/

Infected Devices – brand-new devices pre-installed with malware:

HP USB Keys Shipped with Malware for your Proliant Server
http://isc.sans.org/diary.html?storyid=4247&rss

Back doors in embedded devices (printers, routers, etc)
http://blog.washingtonpost.com/securityfix/2008/04/get_paid_to_find_software_hard_1.html

Devices shipping from abroad with malware
http://www.darkreading.com/blog.asp?blog_sectionid=447&doc_id=148583

HTTP Response Splitting - http header injection:

http://www.aspectsecurity.com/documents/Aspect_File_Download_Injection.pdf
http://www.securityfocus.com/archive/1/425593

Open Redirects – spammers using open redirects to make their spam appear more legitimate:

http://blog.washingtonpost.com/securityfix/2008/07/study_site_redirects_abundant_1.html
http://ha.ckers.org/blog/20080716/redirection-report/

Internet Behavior – your online behavior can reveal much about yourself:

Google has patents that can detect your age, ethnicity, reading level, income, etc:
http://yro.slashdot.org/article.pl?sid=08/03/22/1314253

Using your browser URL history to estimate gender
http://www.mikeonads.com/2008/07/13/using-your-browser-url-history-estimate-gender/

Spyjax - :visited spy tool
http://www.merchantos.com/makebeta/tools/the-spy-is-dead/

Know which social sites the visitor uses
http://azarask.in/blog/post/socialhistoryjs/

Track users browsing via :visited link coloring
https://bugzilla.mozilla.org/show_bug.cgi?id=147777#c78

CSS Spying
http://jeremiahgrossman.blogspot.com/2006/08/i-know-where-youve-been.html

Logging – a case against logging:

http://www.0x000000.com/?i=612

CAPTCHA – it will stop amateurs, but not anyone motivated:

Inside Craigslist's Increasingly Complicated Battle Against Spammers
http://techdirt.com/articles/20080523/0327151211.shtml

How CAPTCHA got trashed
http://www.computerworld.com.au/index.php/id;489635775;fp;;fpid;;pf;1

Breaking The Google Audio Captcha.
http://www.0x000000.com/?i=560

Human CAPTCHA Breaking
http://ha.ckers.org/blog/20080311/human-captcha-breaking/

Inside India's CAPTCHA solving economy
http://blogs.zdnet.com/security/?p=1835

Captcha's broken by mules
http://www.theregister.co.uk/2008/04/10/web_mail_throttled/

PWNtcha
http://libcaca.zoy.org/wiki/PWNtcha

File Uploads – allowing files to be uploaded is risky:

Evil GIFs: Partial Same Origin Bypass with Hybrid Files
http://radar.oreilly.com/2008/06/partial-same-origin-bypass-wit.html

GIFAR - A photo that can steal your Facebook account
http://www.computerworld.com/action/article.do?command=viewArticleBasic&articleId=9111298

Backdooring images
http://www.gnucitizen.org/blog/backdooring-images/

New Worm Transcodes MP3s to Try to Infect PCs
http://www.pcworld.com/businesscenter/article/148603/new_worm_transcodes_mp3s_to_try_to_infect_pcs.html

CPU Attacks – targeting errata in processors

Researcher to demonstrate attack code for Intel chips
http://www.infoworld.com/article/08/07/14/Researcher_to_demonstrate_attack_code_for_Intel_chips_1.html

Password Reset – security questions to reset a password are becoming a less secure way to authenticate:

Researcher mines blogs, social networks to access bank accounts
http://www.computerworld.com/action/article.do?command=viewArticleBasic&articleId=9113405

I forgot my password! (Now what?)
http://www.ravenwhite.com/iforgotmypassword.html

'Forgot your password?' may be weakest link
http://redtape.msnbc.com/2008/08/almost-everyone.html

Column truncation & max_packet_size vulnerabilities – interesting attacks:

http://www.suspekt.org/2008/08/18/mysql-and-sql-column-truncation-vulnerabilities/

OWASP TOP 10 – The top ten most critical WebAppSec vulnerabilities for 2007:

http://www.owasp.org/images/e/e8/OWASP_Top_10_2007.pdf

APPENDIX D – PROFESSIONAL GROUPS

There are two web security organizations I recommend you join, or at least become familiar with: OWASP and WASC. Both offer tools and informative articles.

Open Web Application Security Project (OWASP)

http://www.owasp.org/

Consider joining a local chapter and joining that chapter's email list:

http://www.owasp.org/index.php/Category:OWASP_Chapter

Web Application Security Consortium (WASC)

http://www.webappsec.org/

WASC has a great email list that discusses WebAppSec; I highly recommend it:

http://www.webappsec.org/lists/websecurity/

APPENDIX E – SECURITY BLOGS

A Day in the Life of an Information Security Investigator
http://it.toolbox.com/blogs/securitymonkey

Anachronic
http://www.anachronic.com/

Arbor Networks
http://asert.arbornetworks.com/

CERIAS Blog
http://www.cerias.purdue.edu/site/blog

cgisecurity
http://www.cgisecurity.com/

Chris Weber
http://lookout.net/

Dark Reading
http://www.darkreading.com/

Dark SEO Programming
http://www.darkseoprogramming.com/

Emerging Threats
http://www.emergingthreats.net/

Errata Security
http://erratasec.blogspot.com/

GNUCITIZEN
http://www.gnucitizen.org/blog/

Google Online Security Blog
http://googleonlinesecurity.blogspot.com/

ha.ckers
http://ha.ckers.org/blog/

HostExploit
http://hostexploit.blogspot.com/

HP Application Security Center Community
http://www.communities.hp.com/securitysoftware/blogs/

IBM Internet Security Systems
http://blogs.iss.net/

Jeremiah Grossman
http://jeremiahgrossman.blogspot.com/

Mantasano
http://www.matasano.com/log/

RISKS Digest
http://catless.ncl.ac.uk/Risks

Ronald van den Heetkamp
http://www.0x000000.com/

root labs rdist
http://rdist.root.org/

Rootsecure.net
http://www.rootsecure.net/

Schneier on Security
http://www.schneier.com/blog/

Spamhaus (uses Lasso!)
http://www.spamhaus.org/newsindex.lasso

Stefan Esser
http://www.suspekt.org/

StopBadware.org
http://blog.stopbadware.org/

L-Migrator

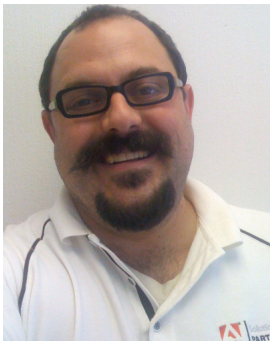
Brian Loomis — **Virtual Relations**
http://www.virtualrelations.us/home/

Abstract

Why are migrations important? This session will discuss how to put your databases on version control. The presentation is based on the assumption that the developer already knows about version control and is using it on their projects in some form. L-Migrator will be discussed in relation to its integration into PageBlocks and its stand alone use. Attendees will come away with a knowledge of the methodology behind migrations, why they are important and when to use them, as well as shortcuts and tips for writing Lasso Migrations and the dangers of performing destructive migrations.

Biography

Brian Loomis of began scripting by automating spreadsheet applications on an Apple IIsi in 1989. He annually attends the Scripting Matters AppleScript Conference, The Lasso Developers Conference, and the FileMaker Developers Conference, where he most recently presented in the ad-hoc sessions on Separation of Data and Interface. Brian is an Apple Consultants Network Member, and Adobe Solution Partner.



Lasso Developer Conference Materials Download
Server: **http://dowloads.lassosoft.com/beta/** Username: **LDC**
Path: **Lasso Developer Conference** Password: **LCROSS**

L-Migrator, Migrations in General, Definition and Scope, Usage within Lasso, Best Practices and Shortcuts.

By Brian Loomis - <www.virtualrelations.us>
PageBlocks: <www.pageblocks.org>

1. Introduction

1.0 Scope

L-Migrator is one of the componenets of PageBlocks, a framework developed by Greg Willits. It is the standard migrations package for Lasso and is similar to many other migrations for other programming languages.

Migrations for database agnostic schema generation are part of the working toward the Agile Manifesto. http://agilemanifesto.org/principles.html

1.1 Who Is This Aimed At?

L-Migrator is for people who are working in a collaborative environment, people who work on multiple stages of development tiers, or anyone that is interested in putting their databases on version control.

This is intended for audiences that are more experienced with programming concepts, advanced strategies, and other methods for standardizing workflow and automating deployment.

This is not intended to replace or augment the L-migrator documentation, its just the conference paper, incidentally there may be content that is similar to the L-Migrator docs.

1.2 What are Migrations?

Dictionary.com defines `migrate` as follows.

mi·grate □verb (used without object), -grat·ed, -grat·ing.

1. to go from one country, region, or place to another.

2. to pass periodically from one region or climate to another, as certain birds, fishes, and animals: The birds migrate southward in the winter.

3. to shift, as from one system, mode of operation, or enterprise to another.

4. Physiology. (of a cell, tissue, etc.) to move from one region of the body to another, as in embryonic development.

5. Chemistry.

   a. (of ions) to move toward an electrode during electrolysis.

   b. (of atoms within a molecule) to change position.

6. (at British universities) to change or transfer from one college to another.

So in this sense, a database migration would be definition number 3, a shift from one system, mode of operation, or enterprise to another. When we think of a database migration we can look at it from several of these perspectives, and in the case of using migrations to automate deployment, we can also see that definition 1 is accurate also, because we can use migrations to push out database schema from one location to another. Well discuss that later in the paper.

In short migrations allow you to make Lasso code that will update and increment or decrement your database schema to keep it inline with the current application requirements, and automate installation or deployment – for example, rolling out multiple Lasso Site instances under a load balancer.

1.3 Why do we want to use migrations?

There are a number of reason s we would want to do this.

Migrations are Database Agnostic – you can write your schema in L-Migrator and deploy it against, SQL-Lite, MySQL, or PosgreSQL.

No need to remember what the syntax was for that SQL syntax.

Migrations allow for different versions of a database. You can easily add the schema to the database and roll up or roll down

your database version. It beats writing alter statements.

Dovetails with iterative development like Agile or Scrum.

Working on a team, when you develop against a local database, once you checkout your code from the repository, all you have to do is run the migrations to bring your copy of the database up to speed.

Allows Adaptive Planning

1.4 How does this factor into the workflow?

Underpinning the Agile development manifesto is the methods which increase effectiveness in self organizing teams. People with different skills and needs need to be able to communicate clearly and effectively.

Evolutionary design techniques recognize that people need to learn things by trying things out. Take a moment if you will and imagine back to when you were first starting to learn Lasso techniques. Remember how it was so important to get code snppets and samples before you were able to code efficiently? For people that are stil learning you recognize the importance of the sites like Lasso Forge, Tagswap, and LassoBin as well as the Lasso Language Guide and more.

Most develops work by coding on a local machine, either a workstation or a laptop and pushing those changes to a server, either over a FPT connection, over ssh or using a deployment methodology using a version control system like git or subversion.

In a tradtional development environment the programmer would work on their own machine developing and modifying the database schema from the command line or a utility such as Navicat or CocoaMySQL. When new code is deployed that would involve new database table s and schema changes to the master server, the devloper would have to perform a schema dump, and then load that SQL schema change on the server or by making the changes to the schema on both systems simultaneously.

In larger organizations with a DBA (Database Administrator), changes must be coordinated with the DBA and this takes time and communiation. Migrations allow developers to submit their changes, and have the DBA review and either commit or modify their changes.

Wouldn't it be great to be able to automate that?

1.5 What is a database migration?

A data base migration is a text file that connects to a database adaptor file and produces database change schema (CREATE TABLE, ALTER TABLE, ADD COLUMN etc.) from the text file. This allows the same text file to create schema by connecting to multiple database types through the adapter to migrate database changes from a laptop ot workstation, using SQLLite, or MySQL to a production system using PostgreSQL, SQL Server, Oracle or MySQL.

2.0 Structure of L-Migrator

The structure of L-Migrator is contained in a zip file which when uncompressed leaves you with a folder called /dbm/ . This is placed at the root of the site. Let's examine the contents:

`/dbm/resources/`

This folder contains the various resources for the package to work:

`/dbm/resources/migrator.lasso`

This is the initial file that is loaded that triggers the migrations and is the main display page for the package.

`/dbm/resources/migrator.cnfg`

This is the config file. It included a directive to create the initial table that tracks the migrations. In future versions we may include this initial config as a migration of its own. Note that it is not encased in LassoScript tags or square brackets.

```
dbName    = l_migrator
migAdaptor  = mysql
migPath   = /dbm/migrations/
messageOn= yes
#---------------------------------------------------------------------
# Before you can use L-Migrator, add this table to your database
#
# CREATE TABLE `_schemaInfo` (
#    `id` int(4) unsigned NOT NULL default '0',
#    `version` smallint(5) unsigned NOT NULL default '0'
```


```
# ) ENGINE=MyISAM DEFAULT CHARSET=utf8
#
# INSERT INTO `_schemaInfo` (`id`,`version`) VALUES ('1','0');
#-----------------------------------------------------------------
```

dbname is obviously the name of the database that need to be migrated and will be connected to. MigAdapter is the name of the database type that is to used. If you are using GIT or subversion or using mirror in an FTP applicaiton you would want to be excluding .cnfg files from those processes, In later versions we may put this variable into some type of environment file so that it is dependent on the production value of the environment, development , test or production, etc. Migration path should bemaintained unless you have changed the way the package is structured. MessageOn is a new variable we have introduced to show the message array at the bottom of the page after the migrations have run.

`/dbm/resources/style.css`

This created the styles that are used in the display of the migrations. Self explanatory.

`/dbm/resources/*.gif`

Graphics for triggering migrations.

`/dbm/resources/schemaView.dsp`

This draws the tables that show the schema on the page when you view the schema.

`/dbm/resources/migratorStartup.lgc`

The migrator startup file contains the logic necessary to prime the pump for the migrations, the configuration file is parsed and

```
<?lassoscript
auth;

var:'configFile' = 'migrator.cnfg';
//--------------------------------------

var:
   'dbName'    = string,
   'migAdaptor'  = string,
   'migPath'     = string,
   'configData'  = null,
   'thisLine'    = string,
   'varPair'   = pair,
   'varName'   = string,
   'message'   = array,
   'varValue'    = string;

if: (file_exists:$configFile);
   $message->Insert:('Config File Exists');
   $configData = string:(file_read: $configFile);
   $configData->(removeleading:bom_utf8);

// split lines

   $configData->trim;
   $configData = (string_replace: $configData, -find='\r\n', -replace='\r');
   $configData = (string_replace: $configData, -find='\n', -replace='\r');
   $configData = ($configData->(split:'\r'));

// remove comments

   if: $configData && ($configData->type == 'array');
     loop: -from=$configData->size, -to=1, -by=(-1);
       $thisLine = $configData->(get:loop_count);
       if: ($thisLine == '')
         || ((string_findregexp:
           $thisLine,
           -find='\\S+')->size == 0)
         || ($thisLine >> 'output_none')
         || ($thisLine->(beginswith:'#'))
         || ($thisLine->(beginswith:'//'));
           $configData->(remove:loop_count);
       /if;
     /loop;
   /if;
// make vars

   iterate: $configData, $varPair;
```

```
    $varName  = ($varPair->(split:'='))->get:1;
    $varValue = ($varPair->(split:'='))->get:2;
    $varName->trim;
    $varValue->trim;
    (var:$varName) = $varValue;
  /iterate;
/if;
//--------------------------------------
library:'/dbm/_resources/migrationEngine.ctyp';
$message->Insert:('End of Migrations Engine');
library:'/dbm/_resources/migrationCommands.ctyp';
$message->Insert:('End of Migration Commands');
library:'/dbm/_resources/migrationAdaptor_MySQL.ctyp';
$message->Insert:('End of Migration Adapters');
library:'/dbm/_resources/migrationSchema.ctyp';
$message->Insert:('End of Schema Reading');
//--------------------------------------
var:
  'migrator'      = (migrationEngine:
               -path    = $migPath,
               -adaptor = $migAdaptor,
               -db      = $dbName),
  'runResult'     = string,
  'tableSchemaData'  = null,
  'tableSchemas'  = map,
  'thisTable'     = string;
$migrator->getCurrentVersion;
$migrator->findMigrationLimits;
?>
```

The next component, the migration adapter is the portion of the system that actually creates the syntax to communicate withthe database. As of now only the MySQL adapter is completed, additional help is needed to create the adaptor files to connect to the other datbases, such as Oracle, SQL Server, SQL Lite and PostGreSQL.

`/dbm/resources/migrationAdaptor_mysql.ctyp`

This file is a custom type that is extremely long, it does not need to be added into here , however it's important that we examine one of the tags that's incuded in thte type, the `convertFieldType` tag

```
define_tag:'convertFieldType',
        -required = 'fieldType', -type = 'string';

//      since my projects and most of the Lasso community's projects
//      are on MySQL, we'll let MySQl column types prevail.
//      so, other than showing an example of how string is converted to varchar
//      this converter pretty much returns the same data type to feed it
//      doing this for all adaptors pretty much eliminates the db agnostic
//      intention of an adaptor, but the architecture is here to support it

        select: #fieldType;
                case:'string';                  return:'varchar';
                case:'fixedString';     return:'char';
                case:'radioBtns';               return:'enum';
                case:'checkBoxes';              return:'set';

                case:'text';                            return:'text';
                case:'tinytext';                return:'tinytext';
                case:'smalltext';               return:'smalltext';
                case:'mediumtext';              return:'mediumtext';
                case:'largetext';               return:'longtext';

                case:'blob';                            return:'blob';
                case:'tinyblob';                return:'tinyblob';
                case:'smallblob';               return:'smallblob';
                case:'mediumblob';              return:'mediumblob';
                case:'largeblob';               return:'longblob';
                case:'binary';                  return:'blob';

                case:'int';                                     return:'int';
                case:'integer';                         return:'int';
                case:'tinyInt';                         return:'tinyint';
                case:'smallInt';                return:'smallint';
                case:'mediumInt';               return:'mediumint';
                case:'largeInt';                return:'bigint';
```


```
            case:'decimal';                    return:'decimal';
            case:'float';                      return:'float';
      case:'double';              return:'double';

            case:'date';                       return:'date';
            case:'time';                       return:'time';
            case:'year';                       return:'year';
            case:'datetime';           return:'datetime';
            case:'timestamp';          return:'timestamp';

      case;                                                 return:#fieldType;
  /select;

/define_tag;
```

Of equal importance is the `setDefaultFor` ctag, it sets the default values for each field that the specific database needs .

```
define_tag:'setDefaultFor',
  -required = 'fieldType', -type = 'string';

  select: #fieldType;
    case:'string';     return:' "" ';
    case:'fixedString';  return:' "" ';
    case:'radioBtns';    return:' "" ';
    case:'checkBoxes';   return:' "" ';

    case:'text';         return:' "" ';
    case:'tinytext';   return:' "" ';
    case:'smalltext';    return:' "" ';
    case:'mediumtext';   return:' "" ';
    case:'largetext';    return:' "" ';

    case:'blob';         return:' "" ';
    case:'tinyblob';   return:' "" ';
    case:'smallblob';    return:' "" ';
    case:'mediumblob';   return:' "" ';
    case:'largeblob';    return:' "" ';
    case:'binary';     return:' "" ';

    case:'int';          return: 0;
    case:'integer';      return: 0;
    case:'tinyInt';      return: 0;
    case:'smallInt';   return: 0;
    case:'mediumInt';    return: 0;
    case:'largeInt';   return: 0;


    case:'decimal';      return: 0.0;
    case:'float';        return: 0;
    case:'double';     return: 0;

    case:'date';         return:'0000-00-00';
    case:'time';         return:'00:00:00';
    case:'year';         return:'0000';
    case:'datetime';   return:'0000-00-00 00:00:00';
    case:'timestamp';    return:'0000-00-00 00:00:00';

    case;                return:' "" ';
  /select;
/define_tag;
```

One thing that is important to note here is the way the L-Migrator commands are mapped to the Database specific commands for their respective syntax. The other commands in this file are are ctags that allow the type to generate and output the syntax necesary to execute the commands.

Lets examine the other files in the dbm/resources as well.

`dbm/resources/migrationCommands.ctyp`

This file is the string and glue that binds the migrations to the database adapter. It's rather long to look at the whole thing but as with the migrations:. Here's the `addTable` ctag from the `migrationCommands.ctyp`

```
define_tag:'addTable',
```

```
  -required = 'name',     -type = string;

  local:
    'thisQueryStatement' = string,
    'thisQueryError'   = false;

  #thisQueryStatement = (self->'queryAdaptor')->(addTable: params);

  self->(execute: #thisQueryStatement,
       -operation = 'addTable',
       -itemName  = #name);

/define_tag;
```

Not real impressive in itself but lets look at the execute tag that is called from within the `addTable` tag. See how the `addTable` tag calls the `queryAdapter` which gets the job done. This is the power of Lasso's custom types and the way the reflexive properties work. Now that the queryAdapter has returned the params for addtable the query is executed by the execute tag.

```
define_tag:'execute',
  -required = 'thisQueryStatement',-type = string,
  -optional = 'table',         -type = string,
  -optional = 'operation',       -type = string,
  -optional = 'itemName',        -type = string;


  !(params >> '-table')
    ? (local:'table') = (self->'tblName');


  !(params >> '-operation')
    ? (local:'operation') = 'custom query';

  !(params >> '-itemName')
    ? (local:'itemName') = '';

  local:'executionError' = string;

  (self->'queryStatements')->(insert:#thisQueryStatement);

  inline:
    -database   = (self->'dbName'),
    -table    = #table,
    -sql      = #thisQueryStatement;

    #executionError = error_currentError;
           $message->Insert:('SQL SChema');
           $message->Insert:('  ' action_statement);
           $message->Insert:('  ' error_currenterror);
  /inline;

  (#executionError == 'No Error')
    ? (self->'queryErrors')->(insert: #operation + ' ' + #itemName + ' succeeded')
    | (self->'queryErrors')->(insert: #operation + ' ' + #itemName + ' failed due to
#executionError);

/define_tag;
```

We've added some hooks here that help us with out debugging in the `migrator.lasso` file. See how our variable `$message` uses the insert statement to add to the array of messages from the migration? We can toggle this with our variable in the `migrator.cnfg` file.

```
           $message->Insert:('SQL Schema');
           $message->Insert:('  ' action_statement);
           $message->Insert:('  ' error_currenterror);
```

The migrationEngine.ctype file is the heart of the system and handles all the transactions to and from the migrations, it writes the changes to the _schmaInfo table and handles the errors produced in the migrations.

`/dbm/resources/migrationEngine.ctyp`

Lets look at a couple key comonents of the `migrationEngine.ctyp`

Update Schema

```
define_tag:'updateSchema';
       local:
              'migrationClassName'          = string,
```


```
                ‘migrationVersion’                  = integer,
                ‘migrationObj’                      = null,
                ‘migrationErrors’                   = array,
                ‘lastMigrationSucceeded’      = true;


        iterate: (self->’migrationClassNames’), #migrationClassName;

                #migrationVersion = integer:((#migrationClassName->(split:’_’))->get:1);

                if: #lastMigrationSucceeded
                        && (#migrationVersion > (self->’currentVersion’))
                        && (#migrationVersion <= (self->’migrateToVersion’));

                        library:((self->’classPath’) + #migrationClassName + ‘.ctyp’);

                        #migrationObj = (\#migrationClassName)->asType;
                        #migrationObj->(init:
                                -adaptor       = (self->’adaptorName’),
                                -db                    = (self->’dbName’));


                        #migrationObj->update;
                        #migrationErrors = #migrationObj->getErrors;

                        (self->’allQueryErrors’)->insert: (#migrationClassName + ‘->update’) =
(#migrationObj->’queryErrors’);
                        (self->’allQueryStatements’)->insert: (#migrationClassName + ‘->update’) =
(#migrationObj->’queryStatements’);


        #lastMigrationSucceeded = self->(analyzeResultOf: #migrationErrors);
        $message->Insert:(‘Last Migration Succeeded ‘ #lastMigrationSucceeded);
        if: #lastMigrationSucceeded;
          inline:
            -database   = (self->’dbName’),
            -table    = ‘_schemaInfo’,
            -keyField = ‘id’,
            -keyValue = ‘1’,
            ‘version’ = #migrationVersion,
            -update;
            $message->Insert:(‘Update Schema’);
            $message->Insert:(‘  ’ action_statement);
            $message->Insert:(‘  ’ error_currenterror);
          /inline;
        /if;
      /if;
    /iterate;
/define_tag;
```

and the rollback portion of this.

```
define_tag:’rollbackSchema’;

  local:
    ‘migrationClassName’    = string,
    ‘migrationVersion’      = integer,
    ‘migrationObj’       = null,
    ‘migrationErrors’       = array,
    ‘lastMigrationSucceeded’  = true;

  iterate: (self->’migrationClassNames’), #migrationClassName;

    #migrationVersion = integer:((#migrationClassName->(split:’_’))->get:1);

    if: #lastMigrationSucceeded
      && (#migrationVersion <= (self->’currentVersion’))
      && (#migrationVersion > (self->’migrateToVersion’));

      library:((self->’classPath’) + #migrationClassName + ‘.ctyp’);

      #migrationObj = (\#migrationClassName)->asType;
      #migrationObj->(init:
        -adaptor  = (self->’adaptorName’),
        -db       = (self->’dbName’));


      #migrationObj->rollback;
      #migrationErrors = #migrationObj->getErrors;
```

```
        (self->'allQueryErrors')->insert: (#migrationClassName + '->rollback') = (#migrationObj-
>'queryErrors');
        (self->'allQueryStatements')->insert: (#migrationClassName + '->rollback') = (#migrationObj-
>'queryStatements');

        #lastMigrationSucceeded = self->(analyzeResultOf: #migrationErrors);

        if: #lastMigrationSucceeded;
          inline:
            -database   = (self->'dbName'),
            -table    = '_schemaInfo',
            -keyField = 'id',
            -keyValue = '1',
            'version' = #migrationVersion-1,
            -update;
            $message->Insert:('RollBack Schema');
            $message->Insert:('  ' action_statement);
            $message->Insert:('  ' error_currenterror);
          /inline;
        /if;
      /if;
  /iterate;
/define_tag;
```

You'll notice that in these two tags also we have added the hooks for the messaging tags.

3.0 Writing Migrations

Writing migrations is as simple as creating a glossary item in BBEdit or snippets in Coda, if anyone is interested in creating syntax snippets. This way you can create migration steps with simple keyboard shortcuts.

Migrations are stored in the `/dbm/migrations/` directory.

Migrations follow a specific naming constraint. Note the examples here:

```
001_Begin.ctyp
002_createTable.ctyp
003_createTable.ctyp
004_createTable.ctyp
005_createTableMonkey.ctyp
006_createMore.ctyp
```

When writing the migrations the naming should be the numeric value of the migration, and the name after that can be anything BUT that name must match the name of the custom type inside that migrations file.

Lets look at some simple migrations:

```
<?LassoScript
        define_type:'001_Begin', 'migrationCommands';

//-------------------------------------------------------------------------------   define_tag:'update';

        /define_tag;


//-------------------------------------------------------------------------------   define_tag:'rollback';

        /define_tag;
/define_type;
?>
```

Notice in the migrations how the enclosing type always matches the filename. This is essential! Also note that the two tags in the migrations file will always be update and rollback.

Here's the syntax for a simple migration:

```
<?lassoscript
define_type:'002_createTable', 'migrationCommands';

//-------------------------------------------------------------------------------
        define_tag:'update';

        //      define tblName once for multiple operations,
        //      or with -table in each operation
        self->(addTable:
```


```
                -table  = 'concerts',
                -name   = 'ID',
                -type   = 'int',
                -size   = '11',
                -autoinc=true,
                -pkey=true);
                self->'tblName' = 'concerts';
                self->(addField:
                        -name   = 'rcrdLock',
                        -type   = 'fixedString',
                        -size   = '1');

                self->(addIndex:
                        -name           = 'rcrdNo',
                        -field          = 'ID');

        /define_tag;

//----------------------------------------------------------------------------
        define_tag:'rollback';

                self->(removeTable: -name='concerts');

        /define_tag;
/define_type;
?>
```

When using the addtable command it requires the option to add the first field, we use an autoincrement with a primary key. This is a new feature that was not available in the original release of PageBlocks or L-Migrator. Please see the L-Migrator docs to explain exaclt how the migrations are written.

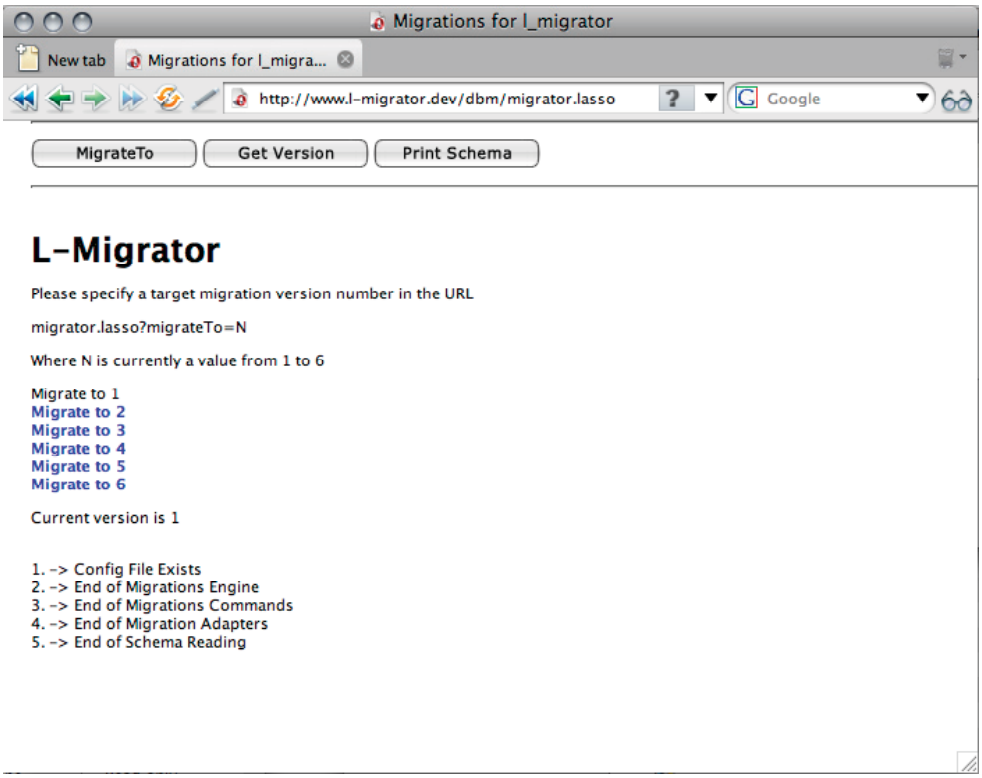
4.0 Executing Migrations

In order to execute the migrations that are in the migrations folder there is a url where the migrations are displayed. I fyou have loaded the standard L-Migrator package outside of PageBlocks type in this address into your browser:

http://www.yoururl.com/dbm/migrator.lasso

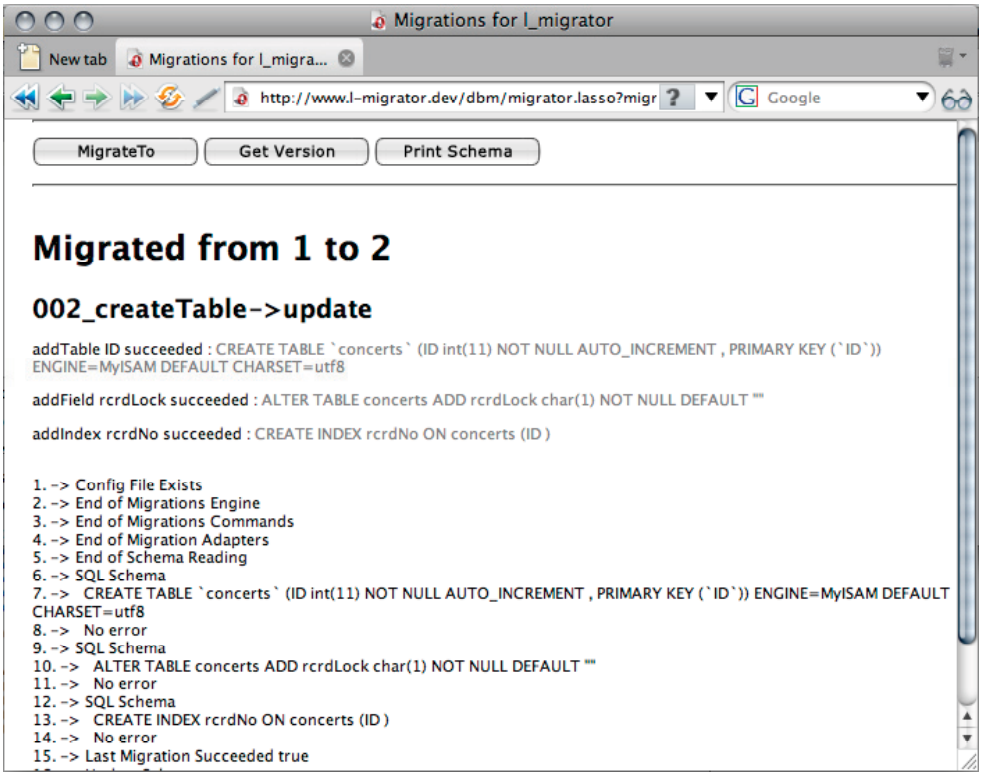
You will see that the standard migrations page loads. L-Migrator will force an authentication session (this is to make sure you have Lasso Site Admin access).

You should now see a window with the following:



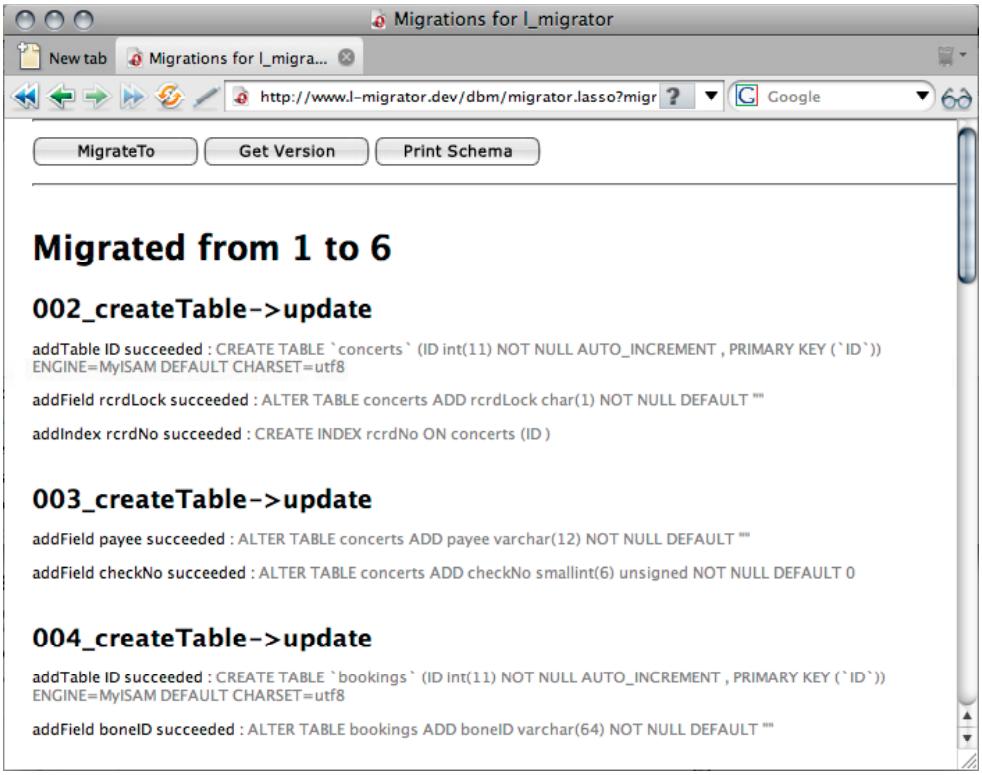
You will have a few migrations loaded, and you will be able to move through them by clicking on the links. Earlier version of L-migrator did not have the feature available, migrations were performed manually by typing the URL with the paramter of the migration you were trying to migrate too – very inconvenient.

Lets run a typical migration:



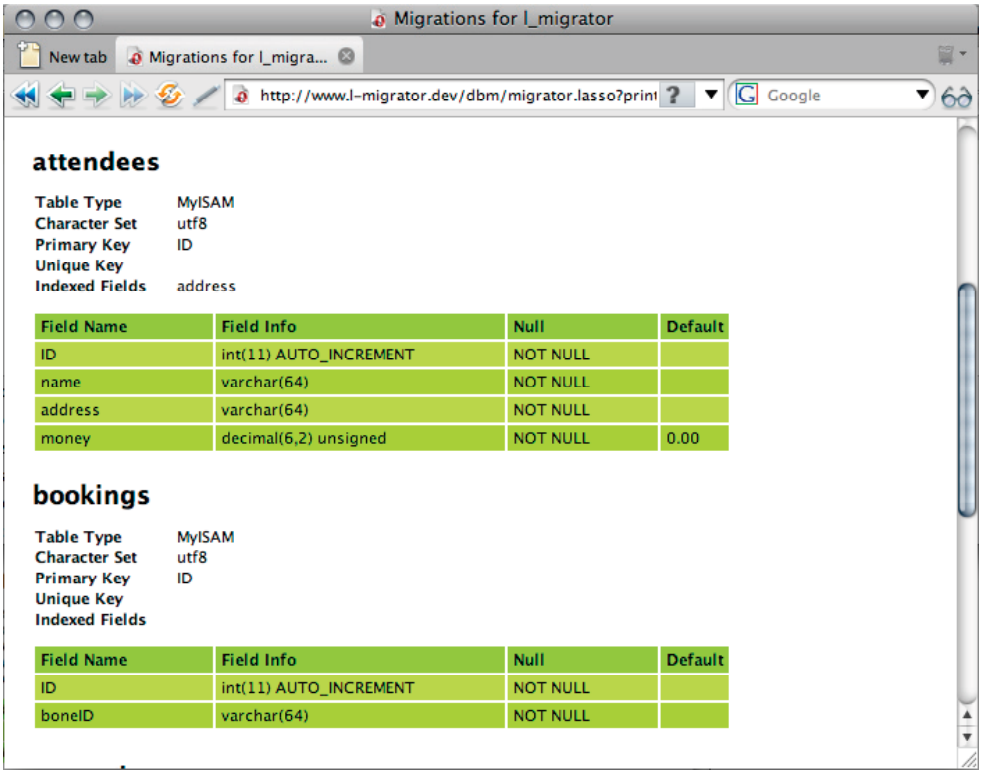
Here you can see that we have migrated from version 1 to version 2. All the syntax needed to perform the alter queries has been generated and run by lasso. Note that you do not need to run up or down only one migration at a time.

Let's do another one and migrate up to version 6:



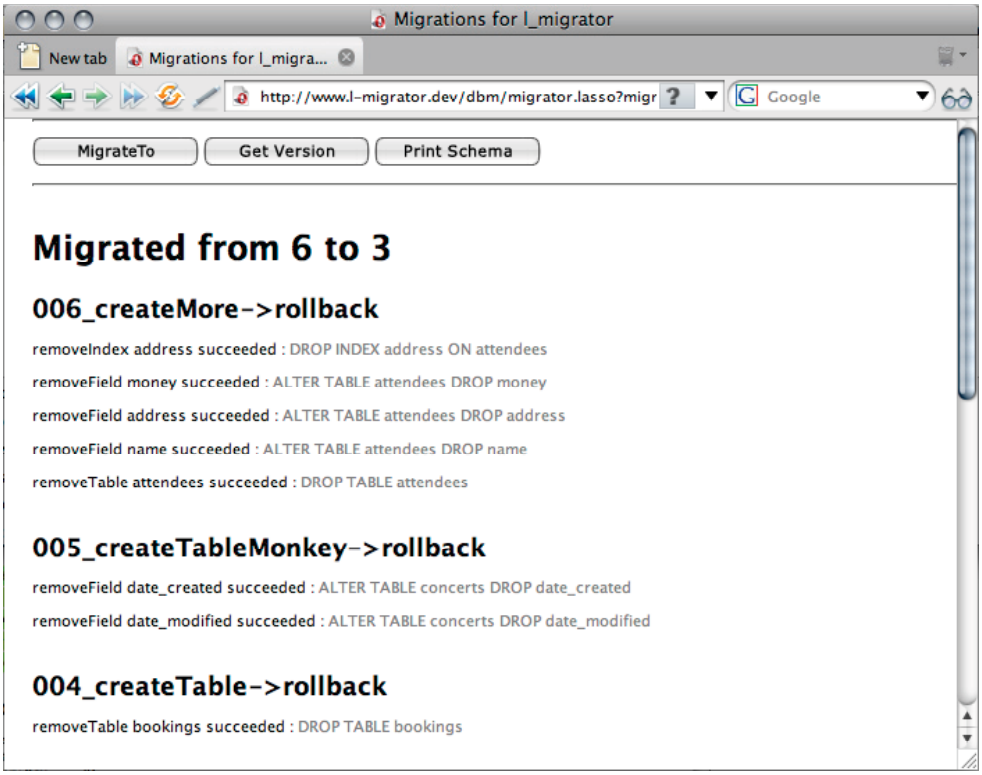
Notice how all the migrations are done version by version. We also have output and descriptions of everything that is happening in the migration process.

Let's examine the structure of the database after the migrations are complete. Click on the Print Schema button.

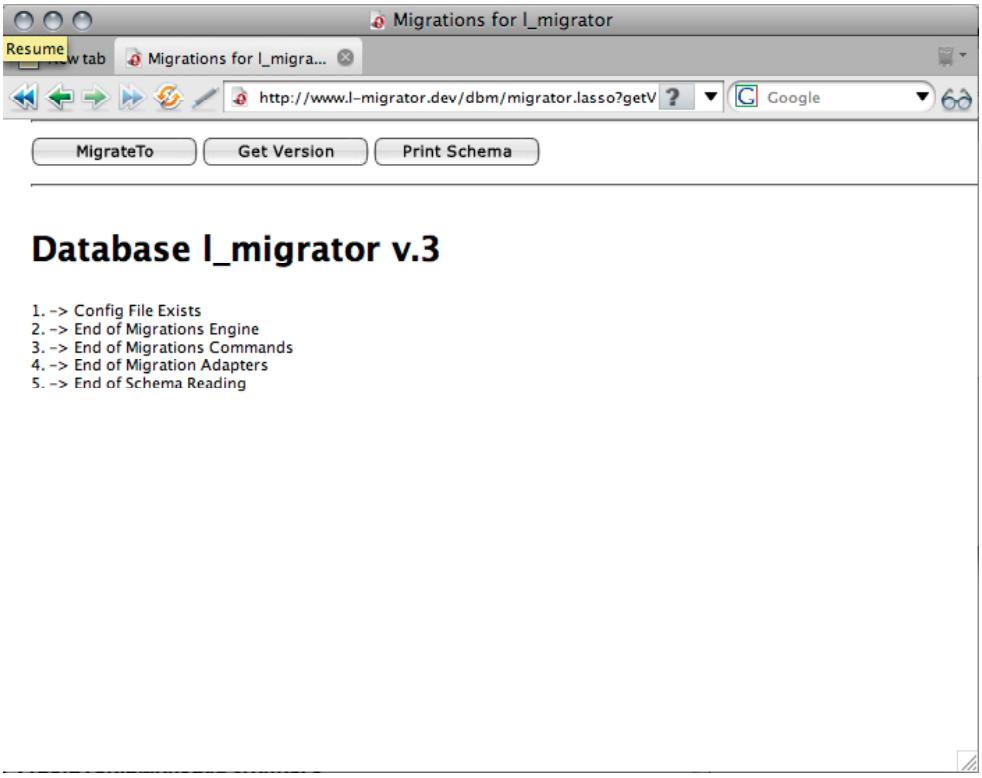


I've scrolled down a little bit so that the schema can be seen. Pretty neat stuff.

Now lets look at rolling back the schema. Click back on the migrate to button and we are going to roll back from version 6 to version 3. Click on the Migrate to 3 Link.



Now we have a successful rollback to version 3. Notice the drop and alter table statements.



You can also immediately pull up the version of the database you are on.

5. Conclusion

Migrations, especially on larger projects or teams can become an essential part of scrum, agile or any other management and development system. When combined with version control, migration files will always produce the schema necessary to run the current version of the application they are bundled with.

For More information:

http://github.com/djdarkbeat/l-migrator/tree/master

Repository for L-Migrator on github, not git users can download a tar file here.

http://www.virtualrelations.us/technology/lasso/l-migrator/

Site of L-Migrator maintainer.

Server Side Techniques for Client Side Optimization

Jason Huck — **Core Five Creative**
http://www.corefive.com/

Abstract

Explore a combination of techniques -- adaptable to any framework or code base -- which help automate the management of images, stylesheets, and javascript, taking advantage of established best practices for client side optimization. See how Lasso can help you organize, combine, optimize, compress, and cache your site's scripts, styles, and other assets to reduce bandwidth usage, cut down on the number of requests per page view, and speed up page rendering.

Biography

Jason Huck is a partner at Core Five Creative, where he's been planning, producing, and managing web-based applications with expertise in information architecture, user experience design, and SEO for over a decade. An active member of the Lasso community, Jason launched tagSwap.net in 2006 to foster code sharing among developers, and manages Lasso integration as part of the FCKEditor team.

Materials

Includes two code samples which demonstrate an asset manager and a pastebin application.

Lasso Developer Conference Materials Download
Server: **http://dowloads.lassosoft.com/beta/** Username: **LDC**
Path: **Lasso Developer Conference** Password: **LCROSS**

Server Side Techniques for Client Side Optimization

Jason Huck, CTO, Core Five Creative

Overview

Web-based applications operate on the venerable client-server model, where the server does the bulk of the heavy lifting, and the client simply makes requests to the server and displays the results. Since most, if not all, of the business logic for an application is expressed in server-side processes, it's not uncommon for optimization efforts to focus on inefficiencies in the middleware, database, and other underlying layers of the server. After all, that's where things are the most complex.

Historically, client-side optimization has amounted to little more than media compression (static images, audio, and video). However, as web browsers have become more sophisticated, and client side scripting has matured, more of the processing burden has shifted to the client side, and thus the amount of data and processing instructions required to fulfill a single page request has grown tremendously. This paper examines the need for optimization of the client side components of a modern web-based application, and explores techniques which can be implemented on the server side to achieve that optimization.

Why Client Side Optimization Is Important

Client-side optimization is important for a number of reasons. First, we as developers must not forget that dial-up is still the prevalent means of accessing the internet in many parts of the world. In 2006, only about 17% of rural U.S. households had broadband access. Secondly, even as broadband availability improves, traditional desktop browsing is forced to share the stage with an increasing number of mobile devices — including the iPhone — which sometimes connect via slower networks such as EDGE, requiring cacheable components to be 25k or less. Finally, even though web apps today may be significantly more complex than they were a couple years ago, users certainly haven't grown more patient, and although your bandwidth may be cheaper, it certainly isn't free. So, if your app consumes less bandwidth and delivers its payloads quicker and more efficiently, everybody wins.

At the forefront of the latest research into client-side optimization is Yahoo!'s Exceptional Performance Team, who published a collection of 13 *Rules for Making Web Sites Fast*, which quickly became the seminal document on the subject. That document has since blossomed into a set of 34 best practices across 7 distinct categories.

This paper explores how Lasso can assist developers in implementing and automating these guidelines, focusing primarily on the original 13 rules. It presents an integrated system — adaptable to any framework or codebase — which will manage the optimization, compression, caching, and delivery of page components.

The Evolution of the Web Page

To fully appreciate the importance of optimization, it's helpful to review the path by which modern web applications became so complex in the first place.

HTML, as originally proposed by Tim Berners-Lee in the early 90's, was extremely basic compared to the options we have today. The first web pages were little more than plain text, with a few simple indications of paragraph breaks and headings, and of course the all-important hyperlinks. The purpose of HTML was to provide structure and hierarchy to a plain text document. The visual presentation (what little there was) was handled entirely by the user agent — and initially there was only one in widespread use: NCSA Mosaic. With only one browser, 22 HTML tags, and no scripts or stylesheets to interpret, early web authors didn't have to worry about compatibility or standards compliance. The only assets one could embed in a page were images. If a page didn't display correctly, it was almost certainly because the author had made a simple mistake.

However, that simplicity didn't last long. As the popularity of the world wide web as a publishing platform grew (thanks to faster modems, and tools like Adobe PageMill and Claris Homepage, which made it easier to author pages) people demanded more control over the presentation of their content. Netscape introduced JavaScript in 1995, and CSS 1.0 became an official recommendation in 1996, though neither would see widespread use for several more years. With the first browser wars well under way, proprietary extensions to HTML began to appear with each new release. Attempts to embed different types of media, each with several competing formats, led to a plethora of incompatible plugins. It was around this time that things started getting really ugly.

Tables, one of the very first additions to HTML, were intended to organize grids of tabular data, never for arranging columns in a page layout. But in the first few years of the 21st century, with CSS still in its infancy, there was little choice. Graphic HTML editors routinely created overly complex layout tables littered with `font` tags and the occasional, primitive inline style. Rudimentary JavaScript was plagued with cross-browser compatibility problems, but nonetheless injected directly into pages with inline event handlers. Consistent rendering across browsers was so difficult that authors began to target just one. The common term for this patchwork mess was *DHTML*.

It took quite a few years to sort everything out. Web development as a trade had to establish itself, and the community had to demand better standards, along with better compliance from browser vendors. Some proprietary concepts became standards (or de-facto standards, such as Flash), while others (`blink`, `marquee`, you know who you are) thankfully fell by the wayside. Since about 2005 or so, *most* web browsers (though not necessarily the most *widely used* ones, unfortunately) have been relatively standards-compliant, such that content authored primarily for one of them would render reasonably well in the others with only minor changes required for full compatibility.

Today, despite a much higher degree of complexity overall, we are seeing an increasing trend back towards the clean, valid, semantically correct markup of the original web, driven by three separate but highly sympathetic motivating forces: standards advocacy (the vision of the semantic web and emerging standards such as microformats), accessibility (ADA and Section 508 compliance), and of course search engine optimization. This trend is supported by an increasingly competitive array of A-Grade browsers, as well as the need to repurpose content for entire new categories of user agents, including mobile browsers, console browsers, and RIA's.

At the same time, DHTML has matured (and we don't call it that any more). AJAX and JavaScript effects are now the norm, and popular JavaScript libraries have, for the most part, solved the browser compatibility problems for us. CSS-based layouts have supplanted table-based layouts for precision and efficiency. And thanks to unobtrusive binding techniques, both scripts and styles can now be completely externalized, just like any other type of asset, cleanly separating the content (markup), presentation (CSS), and behavior (JavaScript) layers.

Challenges of Modern Web Design

Of course, managing all of the components of a modern web application is not without its unique challenges. Even as our markup is getting lighter, our scripts and styles are getting heavier. As projects increase in size and complexity, stylesheets quickly grow long and unwieldy, and can easily become the largest components of a page. It becomes difficult to keep them organized in a consistent fashion.

JavaScript libraries such as jQuery, MooTools, and Script.aculo.us have adopted a modular approach so that developers can include only the parts they need. But in dynamic and decentralized systems where multiple teams are working on the same project, it may be difficult to know whether a given module's dependencies will already be available at the time your component is loaded. Thus, unfortunately, it's not uncommon for core scripts and styles to be included multiple times for a single page request.

Faced with these complications, and in the rush to meet a looming deadline, it can be tempting to add just a little inline javascript or css into the markup. A little style block here, a little `document.write` there...no one will ever know. Except you (and hopefully it's you), a year later, when you're trying to re-skin the project and you suddenly realize

why some parts of the page refuse to render in the new style.

We're faced with a dilemma: keeping all your site assets together in a single file, i.e. a single global stylesheet, is efficient to serve, but difficult to maintain. Splitting assets up into separate files improves maintainability at the expense of performance, since each file requires a separate HTTP request. Frameworks and libraries add to the complexity by introducing dependency trees into the mix.

Fortunately, we can use Lasso to help us strike a balance between maintainability and performance, allowing us to keep our applications fast and efficient, adhering to those aforementioned best practices, without abandoning a dynamic workflow for a build->publish cycle. We'll dissect the components of the system which allows us to do that later. But first, let's get specific about these so-called "best practices." What are they, exactly, and why do we care?

Best Practices (A Subset)

The original 13 best practices for client side optimization, as put forth by the Yahoo! Exceptional Performance Team, are as follows:

(1) Make Fewer HTTP Requests

Although this list is not strictly ordered, this rule is first for a reason: it's one of the most important things to consider when optimizing performance. Why? When a browser requests a page, after the page itself is downloaded, every reference to an asset in that page must be evaluated. It goes something like this:

> *Is it in the cache? No? What domain is it on? Okay. Do we have any cookies for that domain? Okay, load them up and fire off another request. Now let's evaluate the response (more cookies? mmm!) and make sure it's okay. All good? On to the next request!*

This process is repeated for every JavaScript, stylesheet, image, and/or other asset in every `script`, `link`, `anchor`, and `embed` tag in the page. This is often, easily, dozens and dozens of files. The request and response headers sent back and forth can add an extra 1600 or more bytes *per request*, costing bandwidth and time.

The simplest way to reduce the number of requests is to combine files. Put all your stylesheets into one big file. Same with all your scripts. You can even do it with your images, creating what are known as *CSS Sprites*, where CSS controls which portion of a single larger image is shown in different areas of the layout, making it appear as if each clipped area is displaying a completely separate image.

(2) Use A Content Delivery Network

A *Content Delivery Network*, or *CDN*, is a specialized hosting provider which only serves certain portions of your application. Specifically, the completely static portions such as images. Static assets can be served much faster from dedicated CDN's because the requirements for serving are so simple, and the domains from which they are served never set cookies or do anything to inflate the request and response headers used in the handshaking between the server and client.

Also, if you have many assets to serve, splitting the load up between multiple domains can actually work around restrictions in most browsers which limit the number of items that can be simultaneously retrieved from the same host. (The HTTP 1.1 specification recommends no more than two simultaneous downloads per host.) It doesn't take very many assets to realize the performance benefits of moving from sequential to parallel downloading.

Unfortunately, commercial CDN's can be costly, and their low-cost alternatives unreliable. However, we can realize most of the benefits of using a CDN by setting up *asset subdomains*, which are simply additional virtual hosts on your existing server set up to only serve plain, static, cookie-free content. Since the host name changes, browsers will still make parallel requests, even though everything is still being served from the same location.

(3) Add An Expires Header

When assets are sent to a browser, the browser will cache them unless specifically instructed otherwise (in fact, some items will be cached regardless). The web server can indicate within the response header how long a particular asset should be considered "fresh." Adding an Expires Header is important because without it, the browser has to check to see whether the item needs to be refreshed by asking the server. For the smallest assets, the cost of doing that can outweigh the cost of sending the asset itself over the wire.

(4) GZip Components

This one is pretty straightforward. Obviously a compressed asset requires less bandwidth to transfer than its uncompressed counterpart. Most browsers will accept GZip-compressed content, and most web servers will compress content before sending it. We just need to make sure the server is configured to do so.

(5) Put Stylesheets At The Top

Browsers will begin displaying pages as soon as they have enough information about the layout and appearance of the page to do so. This is called *progressive rendering* and can vastly improve the *perceived* performance of larger/longer pages. Since CSS files now contain the bulk of the layout and presentational information about a page, loading the CSS as early as possible, by putting it in the document's head, ensures the browser has all of the information it needs to begin progressive rendering as quickly as possible.

(6) Put Javascripts At The Bottom

JavaScript controls the behavioral layer of a web application. It manipulates the page after its initial internal representation (a.k.a. the *Document Object Model* or *DOM*) has been established. Thus, in the interest of progressive rendering, it's best to defer the loading of JavaScript until as late in the page as possible.

(7) Avoid CSS Expressions

This one's a no-brainer. CSS Expressions are non-standard and supported only in, you guessed it, Internet Explorer. Performance considerations aside, just don't use these. Period.

(8) Make Scripts and Styles External

Purely in terms of performance, the key to this rule is that externalized assets are separately cacheable. Thus, if you use the same assets across multiple pages, it's best to keep those assets external to the page. External files also help keep the content, presentation, and behavior layers distinctly separate, making it easier to repurpose content for other media.

(9) Reduce DNS Lookups

When retrieving a page, if the browser encounters a new hostname, it has to do a DNS lookup to resolve it in order to request whatever assets are being served from the new host. DNS lookups take time which could otherwise be eliminated, so it's best to keep the number of hostnames referenced by a single document as low as possible. This rule conflicts somewhat with the previous rule about using asset subdomains, so it's best to experiment with different combinations to see what provides the best results for your particular project.

(10) Minify Javascript (and CSS)

Minification is a process in which a given section of code is shortened as much as possible without altering its functionality. Long variable and function names are replaced with shorter ones (typically one character), and all extraneous white space, including tabs and newlines, is removed. The result is a file which is smaller (in the case of YUI Compressor, the minifier we'll be using later, 20% smaller on average) even before GZip compression is applied. Smaller files equates to faster load times. Minification usually targets JavaScript, but CSS files can be minified as well.

(11) Avoid Redirects

Any time a redirect is encountered, precious time is wasted spooling up an additional set of requests. Often they are unavoidable, but beyond the obvious META redirects and even Lasso's `redirect_url` tag, you'll want to make sure that links to directories on your site, whether real or virtual, end with a trailing slash, especially when using Apache. Without the trailing slash, Apache has to determine whether you are requesting a directory or an extensionless file, and then, upon discovering that you are indeed requesting a directory, it will redirect you to the same URL with a trailing slash appended. Quicker and simpler to skip all of that and just go straight to the version with the slash already appended.

(12) Remove Duplicate Scripts

When multiple developers are working on separate components of a larger project, they may not know whether some assets will already have been loaded at the time their particular components are included. To be safe, they may include these assets without checking to see if it's even necessary. As a result, these scripts and styles have to be retrieved and evaluated multiple times for a single page view, adding unnecessary overhead and potentially creating additional problems due to load order and cascading.

(13) Configure ETags

An *ETag* (the "E" stands for "Entity" and refers to an individual asset) is simply an additional header containing a checksum which can be used to verify whether a cached item is still "fresh" in the absence of other indications such as an Expires header. Because web servers haven't standardized on how to create ETags so that they are globally unique, and their functionality is essentially moot when Expires headers are used, this is one of the more controversial optimization tips. As long as you aren't running a server cluster, where a browser may attempt to validate an ETag against a different server than the one which created the tag, it won't hurt anything to use them, but take this one with a grain of salt.

An Asset Management System for Lasso-Powered Sites

I wanted to create a system that would automate as many of these optimization techniques as possible, while at the same time not require a drastic change in workflow, and I believe I have struck a reasonable balance, providing intuitive helper functions while still keeping things loosely coupled. This system provides support for rules 1, 2, 4, 5, 6, 8, 10, and 12, but can easily be disabled for debugging.

Getting Organized

Scripts and styles are needed at three distinct levels of a web site, going from general to specific:

- **Global**: These assets are applied to every page within the site.

- **Template or Page**: These assets apply only to a specific page or page type within the site.

- **Sub-Page Components or "Modules"**: These assets apply only to specific areas within a page, though they may be repeated on multiple pages within the site.

Based on this observation, I decided to designate a few special folders to aid in automation. The specific paths are configurable, and you can opt not to use this feature if it doesn't appeal to you, but this is what worked for me:

- **"Base" folders**: One for scripts, and one for styles. Anything placed here is included globally. For CSS, this might be your reset.css file, typography, or a grid framework. For JavaScript, this might be a base library like jQuery or MooTools. Items in base folders are loaded first, in alphabetical order.

• **"Cache" folders**: Again, one each for scripts and styles. This is where the asset management system will store the minified, compressed versions.

• Within my own projects, I also designate folders for **"templates"** and **"modules"**, corresponding to the levels described above, but I decided not to enforce that within the system for fear of making it too specific to my particular habits.

Adding Automation

The basic idea behind the system is simple. At the beginning of each page request, we create two unique arrays. One array is for scripts, and the other for stylesheets. As the page is processed, we insert the paths to the scripts and styles needed for each component at each level, along with the modification date of each file.

After everything else has been processed, we create a unique checksum of that information and look for matching cached files to serve. If matches are found, the appropriate HTML is inserted into the response: a `<link>` tag goes into the document `<head>` and a `<script>` tag goes right before the closing `</body>`.

If no matches are found, depending on what options have been configured, a number of things happen. First, all of the files are concatenated into a single file. Next, that file is optimized and minified using the YUI Compressor. The result is optionally further compressed using GZip compression (only if your web server doesn't do that for you automatically, which is preferable), and the resulting concatenated, minified, compressed file is written to disk. Then, just as if it had already been there, the HTML response is updated to include a link to it.

All of this happens on the fly in a few seconds' time. Once the cached versions are created, that time is reduced to a few milliseconds. Using this "lazy" caching technique, there is no need for a separate build process when deploying sites, and the caches are automatically updated whenever assets are added, edited, or removed. You can programmatically force a refresh of the caches, and if you encounter problems, a single flag tells the system not to combine or compress anything, and it will simply insert all of the `<link>` and `<script>` tags for each resource individually for easier debugging.

With this system in place, there is only one stylesheet and one script linked to each page, regardless of how many you started with, so it helps you reduce the number of HTTP requests (rule 1). These links are automatically inserted in the appropriate places in the HTML response: CSS at the top (rule 5) and JS at the bottom (rule 6). It requires external files to work, so by using it you're forced to use external JS and CSS files (rule 8). With all options enabled, the resulting files are minified (rule 10) and compressed (rule 4). And, even if you insert the same asset into the system multiple times, it will only be included in the final result once, eliminating duplicate scripts (rule 12).

Using Asset Subdomains With Expires Headers

The system also includes primitive support for automating the use of asset subdomains via a series of string replacments performed after the page has been processed. You have the option of supplying a list of paths to asset folders within your web root, along with the subdomain that should be used for each one, as a pair of find/replace regular expressions. So, for example, if you keep all of your images in */assets/images/*, you can set up a subdomain like *images.mydomain.com* to point directly to that folder, and then tell the system to replace all references before serving the page. Thus, markup like this:

```
<img src="/assets/images/logo.gif" />
```

...could become:

```
<img src="http://images.mydomain.com/logo.gif" />
```

In Apache 2.x, the virtual host entry for the above example would be as follows:

```
# Asset Subdomain for Static Content
<VirtualHost *:80>
     DocumentRoot /Library/WebServer/Documents/mydomain.com/assets/images
     ServerName images.mydomain.com

        <Files ~ “^.*$”>
               Order allow,deny
               Deny from all
               Satisfy All
        </Files>
        <Files ~ “\.(gif|jpe?g|png)$”>
               Order allow,deny
               Allow from all
               Satisfy All
        </Files>

        # “Far Future” expires header for static content
        ExpiresActive On
        ExpiresDefault “now plus 3 days”

        # Disable ETags (optional)
        FileETag none
</VirtualHost>
```

Enabling GZip Compression

Although Lasso can compress files on the fly using Fletcher’s [compress_gzip] tag, it’s generally easier and more efficient to enable GZip compression within the web server itself. To enable GZip compression in Apache 2.x, edit your http.conf file by uncommenting the following line:

```
LoadModule deflate_module libexec/apache2/mod_deflate.so
```

...and add the following lines (thanks to Bil Corry for the browser-specific tweaks):

```
# mod_deflate
<Location />
    # Insert filter
    SetOutputFilter DEFLATE

    # Netscape 4.x has some problems...
    BrowserMatch ^Mozilla/4 gzip-only-text/html

    # Netscape 4.06-4.08 have some more problems
    BrowserMatch ^Mozilla/4\.0[678] no-gzip

    # MSIE masquerades as Netscape, but it is fine
    # BrowserMatch \bMSIE !no-gzip !gzip-only-text/html

    # NOTE: Due to a bug in mod_setenvif up to Apache 2.0.48
    # the above regex won’t work. You can use the following
    # workaround to get the desired effect:
    BrowserMatch \bMSI[E] !no-gzip !gzip-only-text/html

    # Don’t compress images
    SetEnvIfNoCase Request_URI \
    \.(?:gif|jpe?g|png)$ no-gzip dont-vary

    # Make sure proxies don’t deliver the wrong content
    Header append Vary User-Agent env=!dont-vary
</Location>
```

Using The System

The interface to the system is primarily through the [asset_manager] custom type, although that type is simply a wrapper for several lower-level tags with more specific functions. If you decide the way I've bundled everything in [asset_manager] is not for you, you can create your own workflow using the other tags.

Initializing The Asset Manager

Near the beginning of the code for a given page request, whether in a global include or just at the top of a page, you must call the ->options member tag to initialize the system. All parameters are optional, so if the default values work for you, all you need to do is this:

```
[asset_manager->options]
```

However, most likely, you'll want to make some changes. The options are:

-usecache: boolean, default true, whether or not to do anything at all that results in cached files. Setting this to false turns off concatenation, minification, compression, and caching, and it just writes all the links to your scripts and styles out individually, though still in the correct places.

-minify: boolean, default true, whether or not to run the concatenated files through the YUI Compressor.

-compress: boolean, default false, whether or not to GZip compress the files. It's off by default because it's better to do this at the web server level.

-refresh: boolean, default false. Setting this to true will cause new files to be generated, even if the checksums haven't changed.

-paths: dictionary, the set of paths which specifies where everything goes. The specific paths are:

yui - The path to yuicompressor.jar. Default: /lib/tools/yuicompressor.jar.
scriptcache - Where scripts are cached. Default: /lib/scripts/cache/.
stylecache - Where styles are cached. Default: /lib/styles/cache/.
scriptbase - Where base scripts are stored. Default: /lib/scripts/base/.
stylebase - Where base styles are stored. Default: /lib/styles/base/.

-subdomains: an array of pairs which will be used in a find/replace regular expression after all other processing has finished. Support for this is experimental. The first value in the pair will be captured for use in the second. For example:

```
-subdomains = array('/lib/[^"]+?' = 'http://static.x.com/\\1')
```

...would cause the following conversions:

```
<img src="/lib/images/logo.gif" />
-> <img src="http://static.x.com/lib/images/logo.gif" />

<img src="/lib/scripts/jquery.js" />
-> <img src="http://static.x.com/lib/scripts/jquery.js" />

<img src="/lib/styles/reset.css" />
-> <img src="http://static.x.com/lib/styles/reset.css" />
```


Once these options have been configured, the simplest usage is to add an individual script or style like so:

```
[asset_manager->add('/path/to/file.js')]
```

The provided path will be added to the correct list based on its file extension. Duplicates and nonexistant paths will be quietly ignored.

Depending on how you like to organize your files, you can also take advantage of a little more automation using the `->loadmodule` tag. Say you have some lasso code which creates a navigation bar at */includes/navigation.inc*. Instead of just including it, if instead you do this:

```
[asset_manager->loadmodule('/includes/navigation.inc')]
```

...then the following two paths will be checked, and automatically added if they exist:

/includes/navigation.css
/includes/navigation.js

I use this extensively in my projects so that I can keep related assets bundled together with the content which references them.

Under normal circumstances, there is no further code required for the asset manager to run. A `[define_atend]` block is created which kicks off the compression and caching routines. However, if you wish to manually trigger the process, you may do so by calling the `->cache` tag:

```
[asset_manager->cache]
```

The following code illustrates the bare minimum steps required to use the asset manager on a basic Lasso page with all options specified:

```
[//lasso
	// must be authorized for file tags and os_process
	// in actual production, use an inline
	auth_admin;

	// make sure shell.lasso is loaded from lassostartup
	// load required tags
	library('tags/array_unique.inc');
	library('tags/asset_manager.inc');
	library('tags/cache_assets.inc');
	library('tags/compress_gzip.inc');
	library('tags/server_webroot.inc');
	library('tags/url_normalize.inc');

	// initialize the asset manager
	asset_manager->options(
		-usecache=true,
		-minify=true,
		-compress=false,
		-refresh=true,
		-paths=map(
			'yui'=server_webroot + '/assetmgr/yuicompressor.jar',
			'scriptcache'='/assetmgr/scripts/cache/',
			'stylecache'='/assetmgr/styles/cache/',
			'scriptbase'='/assetmgr/scripts/base/',
			'stylebase'='/assetmgr/styles/base/'
		),
```

```
                -subdomains=(: '/[^"]+?' = 'http://local.dev\\1')
        );

        // add some assets
        asset_manager->add('scripts/jquery.corner.js');
        asset_manager->add('scripts/global.js');
        asset_manager->add('styles/global.css');
]
<html>
        <head>
                <title>Asset Manager Example</title>
        </head>
        <body>
                <h1>Hello, world.</h1>
        </body>
</html>
```

Creating Your Own Workflow

If you'd rather organize things differently, no problem. The `[asset_manager]` custom type is mostly a wrapper for a custom tag called `[cache_assets]`. Its options are the same, except you pass it an array of filepaths explicitly. Even more basic, if all you want to do is play with YUI Compressor, you can call the `[yui_compress]` tag directly. This tag accepts a source path and an optional target path for output. It requires the `[shell]` tag, and thus also `[os_process]`. There are also tags that convert URL's from relative to absolute. For complete documentation of these tags, visit tagSwap.net.

Caveats

There are a few caveats to using this system. Most notably, since the files that will ultimately be served to the browser will be in a different location from their various source files, it's important that all paths, particularly in the `url()` attributes of CSS files, be absolute instead of relative. Although the system will convert the paths in CSS files which are directly included for you, there is no way to do the same for JavaScript files. Fortunately, it's not uncommon for scripts which include other files to include a base URL variable you can configure as a workaround. Also, you should avoid the use of `@import` to import one stylesheet from within another. Doing so will spawn additional HTTP requests, which partially defeats the purpose of using the system. From a security standpoint, the system requires a Lasso user with permission to use the file tags and `[os_process]`. Finally, it's important to keep the load order in mind. Since the assets in the base folders are loaded in alphabetical order, you'll need to make sure to rename files if that order breaks any dependency chains.

Measuring The Results

In addition to the bare-bones code shown above, another example is included with this paper which is intended to simulate real-world usage. It is a single-page, pastebin-style application called snppt. It's intentionally heavy, including many different scripts and stylesheets (some of which still spawn their own requests), to help illustrate the benefits of using the asset management system. The total client-side response time, number of requests, and bytes transferred when loading snppt were measured using YSlow and Firebug, which are plugins available for Firefox. The table below shows the results with and without the asset manager enabled:

	Disabled	**Enabled**	**Savings**
Response Time*	3.61s	1.90s	47.1%
Transferred	205kb	61kb	70.2%
Requests	20	9	55.0%

** averaged over 10 loads each*

As you can see, the benefits of employing these techniques can be significant. I encourage you to run the provided samples and try incorporating these tools into your own projects.

Tools & Reference

Firebug for Firefox
http://www.getfirebug.com/

YSlow for Firebug
http://developer.yahoo.com/yslow/

Jiffy for Firebug
http://looksgoodworkswell.blogspot.com/2008/06/measuring-user-experience-performance.html

YUI Compressor
http://developer.yahoo.com/yui/compressor/

Writing Efficient CSS
http://developer.mozilla.org/en/docs/Writing_Efficient_CSS

Yahoo! Best Practices for Speeding Up Your Site
http://developer.yahoo.com/performance/rules.html

Using Lasso to Create Dynamic PDF Documents

Jolle Carlestam Cojan

Abstract

No matter how good our information looks on screen, some stuff still need to be printed out on paper. Lassos PDF tag suite offer very powerful and flexible means of producing PDFs for distribution and printing. But the PDF tags can look intimidating and takes some getting used to. This presentation will demonstrate how to use Lasso to produce dynamic multi-page documents, integrating images and other graphics with static and dynamic texts.

Biography

Jolle Carlestam, runs his own business developing web based applications. He's heavily dependent of Lasso, some might call him a true Lasso fan, since version 3. Jolle develops a number of web applications where Lassos PDF capacities is used at it's full limits. He and his company cab be found in Lund, Sweden as most of his clients. Digging into his dark past you could find a university education in theater history and a parallel career in theatre production. This will be Jolle's second presentation held at a Lasso developer conference.

Materials

Includes nine samples of the techniques presented in the paper: Placing text, Placing images, Resizing an image of unknown size, Drawing graphics, Rotation objects, Adding color to objects, handling fonts and font colors, Mixing text of different styles, and From wiki to PDF.

Lasso Developer Conference Materials Download
Server: **http://dowloads.lassosoft.com/beta/** Username: **LDC**
Path: **Lasso Developer Conference** Password: **LCROSS**

Using Lasso to Create Dynamic PDF documents

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

Basic facts

Pixels, Points and halftoning

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

Page sizes

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

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

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

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

Placing text

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

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

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

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

DrawText?

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

Placing images

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

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

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

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

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

Adding images with sufficient print quality

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

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


```
        -width = 100 / 2,
        -left = 100,
        -top = 100);
```

This equals a resolution set in Photoshop to 144 pixels/inch.

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

```
$pdf -> add( $image,
        -height = 100 / 4,
        -width = 100 / 4,
        -left = 100,
        -top = 100);
```

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

Adding images with unknown size

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

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

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

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

// Place the image within the "box".
$pdf -> add( $image,
        -height = $height * $factor,
        -width = $width * $factor,
        -left = 100,
        -top = 300
        );
```

Why multiply with a factor 0.24? Well since 72 (pixels/inch) divided with 300 (pixels per inch) equals 0.24. And 300 pixels/inch gives 4 pixels per line in a 150 LPI.

Adding graphics

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

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

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

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

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

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

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

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

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

Layers

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

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

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

Changing the look of graphics

Line width

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

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


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

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

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

Rounded corners

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

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

Rotate it

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

Images

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

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

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

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

Tilted text

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

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

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

Or when placed using DrawText.

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

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

Defining and using color on graphics and text

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

RGB, CMYK & Spot?

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

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

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

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

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

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

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

Adding color to graphics

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

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

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

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

As of 8.5.5 you can use the new PDF_Color object.

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


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

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

Values can be set separately for lines and fill color.

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

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

PDF_Color

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

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

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

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

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

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

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

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

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

More fun with fonts

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

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

Setting a font variable

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

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

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

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

Standard fonts included

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

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

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

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

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


Using you own fonts

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

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

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

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

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

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

Adding color to fonts and texts

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

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

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

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

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

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

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

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

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

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

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

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

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

A hidden feature!!

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

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

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

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

To do this we first need some font variables:

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

Then we need a color object:

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


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

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

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

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

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

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

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

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

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

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

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

Read more

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

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

Lund July 2008

Jolle Carlestam

Future Proof Web Design

Chris Corwin flickerbulb

Abstract

The web has changed before and will change again, but you can take steps now to make sure your web sites will work in the browsers of 2009 today. We will look at real-world but approachable examples of CSS-based design, enabling your next web applications to be easier to build and maintain, have improved accessibility as well as search engine performance. Along the way we will discuss Doctypes, HTML 5, The Semantic Web, Microformats, and, Webforms.

Biography

Chris Corwin has been developing web sites professionally since 1996 for and for fun since 2002. His major areas of study have been graphic design and typography, and he brings those skills to the web projects he produces, focusing on usability and accessibility.

Lasso Developer Conference Materials Download
Server: **http://dowloads.lassosoft.com/beta/** Username: **LDC**
Path: **Lasso Developer Conference** Password: **LCROSS**

PageBlocks Framework
Introduction, Security, and Localization

Nikolaj de Fine Licht Music&Media Productions
http://musicmedia.dk/

Abstract

Practical examples of how the Lasso-framework PageBlocks handles the following 4 topics: Website security in general; Security concerns when using AJAX and Flash; Localization - both in front end and at CMS level (system language); The use of CSS.

This presentation would shortly introduce the Lasso-framework PageBlocks. Then it would move on to show practical code samples of how PageBlocks handles typical application development issues, each preceded by an ultra-short mention of the issue and the theory behind how PageBlocks handles it.

Biography

Nikolaj de Fine Licht's specialties include management systems for cultural institutions, user-friendly interface design, integration of form and content, design and structure, interaction design and back end programming.

Lasso Developer Conference Materials Download
Server: **http://dowloads.lassosoft.com/beta/** Username: **LDC**
Path: **Lasso Developer Conference** Password: **LCROSS**

PageBlocks Framework For Lasso: Introduction, Security Aspects (Data Validation, AJAX, Flash), Localisation, Handling of CSS and JavaScript

By Nikolaj de Fine Licht - <www.musicmedia.dk>
PageBlocks: <www.pageblocks.org>

1. Introduction

1.0 Scope

The scope of this presentation is to shortly introduce the Lasso-framework PageBlocks and then emphasize on how PageBlocks handles some typical challenges for developers: security and localisation. It also touches on the use of CSS- and Javascript-files.

The presentation doesn't pretend to be exhaustive, neither on PageBlocks in general, nor on the specific areas covered. Its mainly thought of as an "appetizer", an illustration of "how things can be done" if you build web applications with PageBlocks framework. As a consequence, almost all code samples only work in a PageBlocks environment.

1.1 Who Is This Aimed At?

Its my hope that anybody can gather some useful information from this presentation!

Chapter 2 is a short, general introduction to PageBlocks and shows how to download and install it. It should be accessible for anybody.

Chapters 3-5 are meant as demonstrations of how PageBlocks handles some typical challenges in web project development. Those chapters are probably not too accesible for beginners, and the techniques shown are generally not applicable right away outside of a PageBlocks context.

If you are some sort of "on-you-way" into Lasso, then changes are you are approximately where I was a couple of years ago when I started looking seriously at PageBlocks.
In such case I'll say: yes, the learning curve can be somewhat steep (it was for me). But anybody can do what he/she really wants to do, so don't give in! And know that its not either/or! You can set up a PageBlocks project and use it's architecture and some of it's features and tools and continue to do straight forward Lasso programming for the rest. Don't think you need to know the whole thing by heart before writing the first bit of PageBlocks code - I still, two years after, don't know and don't use all aspects of PageBlocks.

If you are new to Lasso but not to web programming, then defenitely my hope you can draw similarities to the knowledge you have and gather some insight in how aspects of beyond-basics web projects are treated with the combination Lasso/ PageBlocks.

1.2 Aknowledgements

Anything "PageBlocks" is by nature in deep debt to it's creator, Greg Willits. Greg developed PageBlocks over some years (formerly FrameWork Pro) as a tool for his own projects, he wrote an extensive Developer Guide for it, and he set up a website and generously put it all at disposal there for the Lasso community and anybody else, free of any charge. When Greg signed out from the Lasso community in November last year, the PageBlocks project was taken over by me.

I'm also in debt to the Lasso Talk community and the fine people there, many of which certainly are more skilled developers than I, always willing to share their findings.

1.3 OK - So What IS A Framework?

The term "framework" meant nothing to me when I started diving into PageBlocks. I believe a word that gets close is tool - a framework is a tool for easier and/or better development of a website project.

But why use a tool? - There is Lasso, that seems to be "a tool", and if you code up your stuff with Lasso, then - given there aren't too many "glitches" - you end up with a working website.

Thats is true, but there are many standard issues and typical routines when making a website project that it soon gets annoying to solve over and over again. At this point you begin re-using code snippets, you start putting common code in a

folder in the website root and include bits of it with `include();`, you maybe start making custom tags (functions) and custom types (methods) - all stuff you can use to solve some of these common tasks in each new project. You have taken your first steps in direction of what eventually could become a framework!

A framework is a standardised set of tools to handle standard issues and typical routines when programming - in our case a web application. A framework takes the language in question and makes ready-to-use building blocks with it. Rails is a framework for the Ruby programming language. Our framework, PageBlocks, takes Lasso and offers at set of structuring schemes, custom tags and -types, and routines to do what you otherwise could end up doing in a verbose, unpratical, not re-usable and - thus - time consuming and not very readable way.

An Example
If you want to look up a single record in a database and assign the field values to a corresponding set of variables you could do something like the following in Lasso:

```
inline(
        -database       = ‘myDb’,
        -table          = ‘myTable’,
        -username       = ‘myUsername’,
        -password       = ‘myPassword’,
        -sql                    = “SELECT * FROM mytable WHERE id = ‘” + $idValue + “’”
);
        // here you would add some code that assigns the field values to a set of vars for later
        // retrieval, could be:
        local(‘mapOfVars’ = map);
        iterate(field_names, local(‘fieldName’));
                #mapOfVars->(insert(#fieldName = field(#fieldName)));
        /iterate;
/inline;
```

The same thing in PageBlocks would look like this:

```
var(‘myDataObject’      = fwp_recordData(‘mytable’));
$myDataObject->(select(
        -select         = ‘*’,
        -keyval         = $idValue,
        -withMakeVars
));
```

Done! You will have each single var value ready at hand, and you will have abstracted var names - that is, var names that differ from the field names. To continue the terminology from above: this is a little tool you can use to do a very common task in a web application.
Of course for this simple code to work there is a number of things that must be in place first; in this case, as the most important, there is a config file that must be set up once. It contains all the information about the table in use, including the field names and the corresponding (differing) var names and more.

1.4 “OK, But What’s In It For Me If I Choose To Build With A Framework Like PageBlocks?”

As we saw in the example above you could not only do a common task with less code than otherwise necessary, you also got vars declared that use names differing from the corresponding field names. To explain why the this is desireable we have to touch an issue that we will deal with again later: security. The short explanation is: it is safer if you can’t deduct the database field names from the var names.

This illustrates a very important point when thinking about using a framework like PageBlocks as opposed to knitting your code up from A-to-Z: not only can you do a lot of things quicker, you also get a lot of added value on top, you simply produce better quality solutions! As we schall see later in this paper you get, among others, built-in handling of security matters and of localisation. If you furthermore care about coding style, about separation of logic and presentation, about emphasis on Object Oriented Programming, on readability for other developers than yourself etc., then you have a number of other good reasons for choosing to build with a framework like PageBlocks.

1.5 “I Guess PageBlocks Makes A Lot Of Assumptions Then...”

As probably any convenient framework, PageBlocks makes a number of assumptions in order to be time-saving and steamlined to work with. I don’t have the background to compare PageBlocks with a variety of other frameworks, all I can say is that the assumptions are there, but they are very easily configurable in case you want to change them, and doing so won’t feel as going against the framework.

I have sometimes come across assumptions that seemed either wrong to me or were simply missing. In such case its important to remember that PageBlocks is not a finished project, it can - and should - continue to be improved.

1.6 PageBlocks Is Overkill For The Little 7-Pages Website

I would say that PageBlocks pays for projects with at least 5-7 2nd-level areas, or some sort of URL abstraction, and/or some sort of access restricted area - a CMS or some other administrative area. It pays a lot if you need localisation at some level. I have happened to develop mainly administrative systems (Internet-based Intranets and CMS'es), and here PageBlocks is a fantastic tool.

1.7 PageBlocks Is Open Source

It is important to stress that PageBlocks is an Open Source project! It is so in the sense that all code is freely accessible and may be used for whatever purpose, also outside of PageBlocks context; it is also so in the sense that anybody can contribute to PageBlocks and have their improvements incorporated into the official release. But it is NOT so, however, in the sense of a public repository or anything else fancy sharing-wise set up. Not because it wouldn't be neat to have, but because of my lack of time and knowledge!

1.8 PageBlocks And The OS'es

PageBlocks is designed to be able to run on OS X, Linux and Windows; it has, however, not been tested on platforms other than OS X. PageBlocks still awaits a knowledgeable Linux- and/or Windows developer to get enthusiastic and dive in, test and apply tweaks possibly necessary to run on those OS'es.

1.9 PageBlocks And The Data Sources

PageBlocks is designed to be able to communicate with any datasource through a data source abstraction layer. Again, PageBlocks hasn't for some time been tested with other data sources than MySQL, simple due to the same facts as mentioned above.

2. Introduction to PageBlocks

2.0 You Want To Install PageBlocks Both Locally And On The Remote Web Server

You must have PageBlocks up and running both on your local development machine and on the web server where you eventually will publish your solution. For setting up a Lasso-enabled web server on a local development machine there are different ways to go depending on your OS and it's version. Mac OS X-instructions are provided in the Read Me accompanying the PageBlocks download package. You can also find such instructions elsewhere on the Internet.
PageBlocks has a feature named Deployment Modes which essentially enables you to run exactly the same version of your project both locally and remotely but still have different names for databases, different levels of debug-info etc.

2.1 Then, What IS PageBlocks Practically Speaking?

Physically PageBlocks consists of a number of files and folders that have to be installed, some in the LassoSite in question and some in the webroot folder; and it consists of a number of tables that must be installed into a MySQL database
Configuration-wise PageBlocks is a set of configurations that has to be made in the LassoSite in question and the Apache webserving configuration file.
Structure-wise PageBlocks is a way of organising the files and folders that make up your project in a certain way which allows the framework to make a number of assumptions about where to find things when serving pages
Programming-wise PageBlocks is a way of coding that, to put it rough, is more about working with the PageBlocks tags and types than writing straight forward Lasso code.

(It would be convenient to be able to build an installer that installs and pre-configures PageBlocks, but it seems Lasso SiteAdmin doesn't currently have an API that allows for this.)

2.2 Got Lost Already?

OK, maybe above standing was enough to turn you off. The Apache webserving configuration file - huh? PageBlocks tags and types - what is a type?
There IS quite a few things to do before even starting to do anything "for real" with PageBlocks, I agree, but it CAN be done :) Below we will look closer at this. But before that I have to point out:

2.3 You Need A Collaborating Hosting Service Provider (HSP)!

Why? Because you need a few things installed and configured that aren't necessarily part of a typical, Lasso-enabled hosting plan. If you host your own solutions this isn't a problem of course, but if - like me - you don't, then you need this

collaborative attitude. Hosting providers I know of that do setup Lasso/PageBlocks-enabled accounts without any extra charge are:

Point In Space <www.pointinspace.com>. Point In Space is the sponsor of the hosting of <www.pageblocks.org>
Flagship Hosting <www.flagshiphosting.com>
Falcon Internet <www.falconinternet.net>

2.4 These Are The Steps

1. Open a Lasso/MySQL-account hosting with your HSP.

2. Currently you will need to ask your HSP to a) put the LassoStartup folder into the account root folder and to b) create a symlink to this folder in it's original place (inside the site folder of the Lasso site in question), so that now you have a folder in your web server root named 'LassoStartup', its empty. By the time you read this, this step may have been eliminated.

3. Download the PageBlocks package from <www.pageblocks.org> and unpack the zipped file. You will now have a folder named 'pageblocks_5XX' (where 5XX represents the version number) wherever you unzipped it to.

4. In this folder grab the file 'error.lasso' in the folder named 'Put_in_LassoAdmin' and mail this file to your HSP, asking to put it in the folder named 'LassoAdmin' inside the site folder of the Lasso site in question.

5. Grab everything in the folder named 'Put_in_LassoStartup' and FTP this to the folder 'LassoStartup' we got in step 1. above. Restart the site. Again, by the time you read this, this step may have been eliminated.

6. Now follow the instructions in the file '__READ_ME_FIRST.txt'. Again, the first time you look at these instructions you may experience an "Oh-My-God"-reaction. Don't let this turn you off! Take a cup of tea or coffee and just follow the indications step by step. Forgetting a few things or running into problems in the beginning is part of the game, don't hesitate to direct any questions to me!

After having setup a PageBlocks-enabled account a couple of times it is actually possible to set up a new project ready for development in less than an hour!

2.5 Now You Have - err... Nothing!

After having followed the instructions mentioned above you will have installed files, you will have configured Lasso SiteAdmin, you will have installed a MySQL database and tables and configured the Apache webserver and a few other things.
What you now have is NOT a full CMS, NOR do you have a sample blog or a sample gallery or a sample something else fancy, you essentially have - nothing! Which isn't correct, because the PageBlocks package does contains a sample website with corresponding database tables, which is basically a copy of <www.pageblocks.org>, as well as a 'blank' project named "pbStart" which despite being 'blank' still includes core PageBlocks functionality such as user management (for access restricted areas) and multi-language string management, which gets us close to a simple CMS (see 5.4 below).

You will always want to install at least the 'blank' project, because this way you have the whole PageBlocks structure in place, including a number of folders and sub-folders, necessary for PageBlocks to find files in the places where it expects to find them.

But it is important to understand, that PageBlocks is a toolset, it doesn't do much until you start using those tools to build a project with.

2.6 What Is Next Then?

At this point you would start building your project, wether tweaking existing pages in one of the sample websites or simply adding new stuff to the 'blank' project. Its beyond the scope of this presentation to guide a way into building a new project. The PageBlocks download package includes a quite extensive Developer Guilde and a complete reference database, also found on <www.pageblocks.com>, for all the tags and types that make up the PageBlocks API (Application Programmar Interface).

3. Security In PageBlocks: Data Validation

3.0 Data Validation As Security Means

Security is an ever increasing concern when making web solutions. While a lot of responsibility for the securing of a server lays with the HSP, the developer is responsible for what the web application does in terms of opening holes into data and

areas not meant for public access.
The problem is that the great power of the Internet and the great impact of good websites lay in their interactive nature, but this interactivity is obtained in your web application by the same means that opens almost all it's potential security risks.

While security is an area which naturally is constantly shifting, then there are basic rules that one can go with in order to get a very long way. PageBlocks is build up around a set of security principles which you find outlined here: <www.pageblocks.org/refc/refc_security>. Building with the PageBlocks framework and with these principles in mind increases chances considerably that your web application will stand well against attacks!

One very important principle is to setup not one but more security layers in order to prevent or decrease impact if one layer is defeated. And one very important such layer is data validation, that is, the checking of all incoming data for maliciously crafted content.

3.1 Data Validation As "Data Cleaner"

Data validation also has another aspect: to check if content conforms to expected formats and content types. We don't want anything else than numbers and maybe a +-sign in phone number fields, so don't let the user get away with typing letters into phone number fields.

3.2 Client Side And Server Side Validation

Data validation can take place directly in the browser (typically by JavaScript) before anything is sent to the server, or it can be taken care of on the server by the code receiving the data. Or both. It is important to know that from a security point of view, only server side validation can prevent intruders, while client side validation can be seen as mainly an interface enhancer.
The validation tools included with PageBlocks are exclusively for server side validation. PageBlocks doesn't currently offer any automisation of client side validation rutines, but this is on the list for future development!
For further introduction to data validation in PageBlocks see <www.pageblocks.org/ftrs/api_validate>.

3.3 Data Validation In PageBlocks

In PageBlocks, validation has been centralized into a single unit (a custom type) which is then used by the database action unit to automate validation of form- and other inputs. The validation unit can also be used to validate inputs outside of database action situations. Essential to the workings is that:

input name <-> database/table field name <-> data type <-> validation instruction

are tied together in a central config file, one config file for each database table (data model). The validation instructions are in the form of validation codes, that refer to a set of built-in validation instructions and/or custom made validation instructions. If you have used AutoValidate you will know this princip.
This enables the database action unit to:

1. find the field names in the given database/table
2. update values from or read values into the corresponding variable/input names
3. underway invoke the validation unit using the validation instruction given for each data

Here is an excerpt from such a config file showing some table fields and their corresponding input names and validation instructions - the {{ and }} are used by the PageBlocks parser to extract the config data:

```
{{tableModel___
#inputName                  fieldName                                  dataType       validation
codes
#-----------          ---------------              ------  ----------------------
m_ctMaTle                          cnctMainTitle                 string  maxlen=30, isAlphaNumeric
m_ctMaNmeF                 cnctMainNameF                 string  maxlen=20, -extn
m_ctMaNmeL                 cnctMainNameL                 string  isRequired, maxlen=20, -extn
m_ctMaIsStd                cnctIsStandard                string  maxlen=1
m_ctMaBirthday       cnctMainBirthday              string  maxlen=10, isDate=euroDate
}}
```

If we take the first line then it instructs that:
- the input name (the input field name in forms, Lasso var name) is 'm_ctMaTle'
- the corresponding field name in the database/table is 'cnctMainTitle'
- the data type is 'string'
- the validation codes are 'maxlen=30' and 'isAlphaNumeric' which means that the string length can't be more than 30 characters and only alphanumeric characters are accepted.

'isAlphaNumeric', 'maxlen=XX' and 'isRequired' are examples of PageBlocks built-in validation codes. For a complete list of built-in validation codes please see the validation section in the Developer Guide that comes included with the PageBlocks

package.

The second line shows, among others, the validation code '-extn'. This code refers to a custom validation instruction written by the developer (it is suggested to make all custom validation codes start with a hyphen to easily distinguish them from the built-in ones). In this case an instruction that allows a set of extended European Unicode characters and a few symbols.

3.4 Before Going On: Separation Of Logic And Display

Before going on with the implemenation of the validation mechanism there is a basic principle in PageBlocks to point out: all code that will eventually create "a page in the browser" is split in two: logic and display. Logic code is the code that for examples issues a database call and retrieves the values, display code is the code that contains or generates HTML and Lasso display code, or in other words, the code that eventuelly will output what we see in the browser.
The principle of "PageBlocks", each page block consisting of a logic block and a display block, that get inserted into a template, is illustrated on the front page of <www.pageblocks.org>.

3.5 Implementation: The Logic

Lets say we are dealing with a simple database update action with values from form fields. Using the table model configs from 3.3 above, the logic in the response file would look like this in PageBlocks:

```
var('myDataObject'     = fwp_recordData('mytable'));
$myDataObject->(update(
        -keyfld        = 'rcrdNo',
        -keyval        = $idValue,
        -inputs        = 'm_ctMaTle, m_ctMaNmeF, m_ctMaNmeL, m_ctMaIsStd, m_ctMaBirthday'
));
```

The neat thing here is, that the ->update member tag (function) of the fwp_recordData object automatically invokes the validation mechanism. The same goes for all other member tags that adds or update values. If one or more validation problems occurs the database action will not take place, instead validation error messages are generated and inserted into the $fw_validator instance of the fwp_validator object.
If you want to trap for no validation errors already in the logic block you would add the following after the update code above:

```
if(!$fw_validator->('errorMsgs')->size);
        redirect_url('/contacts/listing?fw_s=' + $fw_s);
/if;
```

If there were no validation errors the redirect will take place, otherwise we will stay with the file specified in the form submit action. `$fw_s` is the standard session var in PageBlocks.

3.6 Implementation: The Display

The display file accompanying the logic above would contain the form. Something like the following (maxlength and tabindex are left out):

```
<fieldset>
        <legend>Personal Data</legend>
        <form action="[response_filepath]?fw_s=[$fw_s]" method="post" name="form01" accept-
charset="utf-8">
                <input type="hidden" name="idValue" id="idValue" value="[var('idValue')]" />
                <label for="ctMaTle">Title:</label>
                <input type="text" class="size2" name="m_ctMaTle" id="ctMaTle" value="[var('m_ctMaTle')]"
/>
                <label for="ctMaNmeF">First Name:</label>
                <input type="text" class="size2" name="m_ctMaNmeF" id="ctMaNmeF" value="[var('m_
ctMaNmeF')]" />
                <label for="ctMaNmeL">Last Name:</label>
                <input type="text" class="size2" name="m_ctMaNmeL" id="ctMaNmeL" value="[var('m_
ctMaNmeL')]" />
                ...etc
        </form>
</fieldset>
```

If the submition causes validation errors we want to display messages about these problems. One way could be to add the following above each input field (shown only for the Last Name input field):

```
[$fw_validator->('errorMsgs')->find('m_ctMaNmeF') ? $fw_validator->('errorMsgs')->find('m_ctMaNmeF')-
>first->value]
<input type="text" class="size2" name="m_ctMaNmeF" id="ctMaNmeF" value="[var('m_ctMaNmeF')]" />
```

But that soon becomes tedious if there are many fields (and many forms), so maybe we want to display all validation error

messages in one, centralised feedback area somewhere on the page. We can acchieve that this way:

```
<?lassoscript
	if($fw_validator->('errorMsgs'));
		'<p>';
		iterate($fw_validator->('errorMsgs'), local('i'));
			(loop_count) > 1 ? '<p>';
			#i->second + '</p>';
		/iterate;
	/if;
?>
```

Pretty simple! The validation error messages to display all reside in one single config file, where the fwp_validator object pulls them from.

But now we have another problem! A typical error message is "This field can not be left empty". It makes sense when you display the message next to the field causing the problem, but not when we display the messages in one, centralised feedback area as suggested above. We need each message to display the field label it refers to.
To obtain this we add information to the validation codes from the table config file shown in 3.3., the validation codes taken separately would now look like this:

```
#...	validation codes
#...	-----------------------
...	maxlen=30, isAlphaNumeric, haslabel=Title
...	maxlen=20, -extn, haslabel=First Name
...	isRequired, maxlen=20, -extn, haslabel=Last Name
...	maxlen=1
...	maxlen=10, isDate=euroDate, haslabel=Birthday
```

Because of the 'haslabel'-code, the validation error message from above, when referring to the label "Last Name", would now be "The field Last Name can not be left empty" where 'Last Name', furthermore, would be marked up with a class in a span tag `<span class="inputerrfldnm">Last Name</span>`.

But maybe we still want to improve the interface, maybe we also want to highlight the labels of the fields that caused problems. This can be done by doing the following (shown only for the Last Name input field label):

```
<label for="ctMaNmeL"[$fw_validator->('errorMsgs') ? $fw_validator->('errorMsgs') >> 'm_ctMaNmeL' ? '
class="inputerrfldnm"']>Last Name:</label>
```

This code will add the class `inputerrfldnm` to the label tag enabling you to highlight it with CSS.

But there is still more to the display of field labels - see 5.8 below!

3.7 Validation Outside Of Database Actions - Database Actions Without Validation

Sometimes you need to perform validation without database actions, that is, without `fwp_recordData->(add())` or `fwp_recordData->(update())` or similar involved. This can be accomplished with the following:

```
var('myDataObject'	= fwp_recordData('mytable'));
$myDataObject->validateInputs('m_ctMaNmeF, m_ctMaNmeL');
```

`validateInputs()` accepts the keyword `-useGet` which instructs it to validate on GET-params rather than the default POST-params.

And, again, you can check for validation errors by adding this right after:

```
if(!$fw_validator->('errorMsgs'));
	// there was no error so we continue
else;
	// do something else
/if;
```

If you need to do a database action that normally invokes the validation mechanism but want to avoid validation (for example because it was already done by code similar to the one shown above or by the code shown in 3.8 below), you simply add the `-withoutValidate` keyword to the action:

```
var('myDataObject'	= fwp_recordData('mytable'));
var('newIdValue'		= fwpStr_randomID(8)); // PageBlocks random string id generator
$myDataObject->(add(
	-keyval		= $newIdValue,
	-inputs		= 'm_ctMaNmeF, m_ctMaNmeL'
	-withoutValidate
));
```

3.8 Validation Of Data From And To Web Services And Of Cookie Data

PageBlocks offers ways to validate data that is NOT part of the pre-configured data models. It could be validation of data from GET or POST params from a web service, including validation of data from containers such as cookies, XML or JSON; or it could be validation of data before being sent to a web service.

Incoming data could be response to a request in the form of the following JSON string:

```
{“result”: {“shape”: “circle”, “color”: “cc9900”}, “error”: “”, “id”: “1234”}
```

We extract the result and insert into a var:

```
var(‘jsonData’ = map);
$jsonData      = decode_json(‘{“result”: {“shape”: “circle”, “color”: “cc9900”}, “error”: “”, “id”:
“1234”}’)->find(‘result’);
//-> map: (shape)=(circle), (color)=(cc9900)
```

Next, we would have a -formSpec of validation specifications that might look like this:

```
var(‘jsonDataValidations’     = map);
$jsonDataValidations                  = map(
        ‘shape’ = ‘isRequired, isAlpha, hasMaxLength=24’,
        ‘color’ = ‘isRequired, isHTMLColor’
);
```

And before using the JSON data we would perform a validation and act accordingly like this:

```
var(‘dataIsInvalid’    = false);
$dataIsInvalid         = $fw_validator->(validate(
        -usingPairs            = $jsonData,
        -formSpec                     = $jsonDataValidations
));
if($dataIsInvalid);
        // data didn’t pass so do some error reporting
        // you can manually insert validation errors into $fw_validator like this sample:
        // $fw_validator->(insertErrorMsg(-valcode=9999, -input=’jsonData’));
else;
        // proceed to processing
```

Exactly the same procedure could be followed to validate data from a cookie, only would the process of extracting the data in the first place be different.

Similar, say you have a small form that upon submition sends a request to a web service and you want to validate the values before sending them off. It could look like this:

```
var(‘inputValidations’ = map);
var(‘inputsAreInvalid’ = false);
$searchValidations     = map(
        ‘searchString’         = ‘isRequired, hasMaxLength=128’,
        ‘maxResults                   = ‘isPositiveInteger’
);
$inputsAreInvalid      = $fw_validator->(validate(
        -usingPOSTForm,
        -formSpec              = $inputValidations
));
if(!$inputsAreInvalid);
        // the following is in princip but untested
        json_rpccall(
                -method = ‘myMethod’,
                -params = array($searchString,$maxResults),
                -id            = 1234,
                -host   = ‘http://myhost/services/lookup.jsonrpc’
        );
/if;
```

4. Security: Securing AJAX And Flash

4.0 Any File That Receives And Handles Data Must Be Secured

With the increasing use of AJAX-techniques and other extended JavaScript-usages in web applications a whole new attack target area has seen daylight. AJAXified pages typically send requests to helper files on the server, and its a piece of cake for hackers to bypass the original AJAX-calls and issue maliciously crafted requests to those helper files.

Also Flash can be used to communicate with the server, and while it can be a little trickier to figure out the locations of the helper files for a Flash movie it can certainly be done.

I will here look shortly at some securing techniques as applicable in a PageBlocks context. This chapter is by no means exhausting, instead the reader should look into other contributions in the LDC manual focusing entirely on security.
In order to profit from the following you must have a fairly good understanding of how AJAX-calls and -responses and related work.

4.1 Data Validation In AJAX Helper Files

The good news is that the whole validation mechanism described in the previous chapter is at hand for securing AJAX helper files in PageBlocks!

In order to show how this is accomplished I have to shortly describe how a request for a "normal" web page is handled in PageBlocks: PageBlocks works entirely with extensionless URLs. When a request comes in, it passes through config files that will:
- check if the file is listed as existing
- initiate a number of Lasso vars, instantiate a number of objects etc.
- add general template code such as <head>- and <meta>-tags to the HTTP response
- add specific template code for the page requested to the HTTP response and run the code in the blocks
- and more...

But in the case of AJAX helper files you don't want the overhead of general or specific template code. It IS possible to bypass the process entirely by for example calling a .lasso-file which isn't listed as existing in the config files (this is, by the way, the reason you can combine PageBlocks architecure with standard .lasso files and pages). But if you do this, then you won't have the PageBlocks tools at hand, unless you add the following to the very top of your .lasso-file:

```
<?lassoscript
	fwpPage_init;
	// here your code
```

In case of an authenticated page you must be sure the AJAX-request includes the PageBlocks session var, and you add this to the fwpPage_init-tag:

```
<?lassoscript
	fwpPage_init(
		-fw_pgAuthRequired	= true, // means authentication is required for this access
		-fw_pgPrivilege			= 'fw_user->prof.loginValid' // sets the privilege needed
for access
	);
	// here your code
```

But an easier and more conform way is to use the special AJAX- and Service-templates in PageBlocks. They are only "templates" in the sense that they standardize the request processing, they don't add anything to the response. So by using an AJAX-template you can continue to use the whole PageBlocks toolset, the extensionless URLs, the logic/display separation etc. without any downsides. Here is how:

In each subfolder (in PageBlocks terminology a 'module') there is a config file named '_pageConfig.lgc' which, among others, lists the pages available to be requested in that subfolder (module). Each existing page is listed and has some configuration attached, and in that list you configure your AJAX helper file as follows, assuming it's name is 'ajaxHelper':

```
select($fw_requestPage->('name')); // select list of available pages
	case('ajaxHelper');

		$fw_pgTemplate		= fw_kPageIsAjaxHandler;

		$fw_pgPrivilege		= 'fw_user->prvlg.loginValid';
	case(...);
	etc...
/select;
```

The constant 'fw_kPageIsAjaxHandler' used as value for $fw_pgTemplate instructs the PageBlocks template builder unit to proceed without adding anything.
Your AJAX helper file will physically consist of two files, a logic file and a display file. This is, as already mentioned, a basic concept in PageBlocks architecture. If your AJAX helper file doesn't return anything for display, then you just leave the display file empty (but it must be present). So in our case you would have these two files on disk in the current subfolder (modul):

```
ajaxHelper_ajax.lgc
```

ajaxHelper_ajax.dsp

You will notice the endings '_ajax' which instructs PageBlocks of the nature of these files, and you will notice the file extensions .lgc (logic) and .dsp (display) - these extensions are used throughout all PageBlocks to distinguish the task of the page blocks.

Inside the ajaxHelper_ajax.lgc file you can now do validation on the data passed by the AJAX-call exactly the same way we did in 3.5 above:

```
var('myDataObject'     = fwp_recordData('mytable'));
$myDataObject->(update(

        -inputs = 'input1, input2, ...etc,

        -keyfld = 'rcrdNo',

        -keyval = $id,

        -useGet
));
```

However, notice the -useGet keyword. This instructs the validation mechanism to validate GET-params instead of the default POST-params. Most AJAX-calls use the GET-method as default method, and you wouldn't get anything validated if you didn't add this instruction. As long as the param names you pass in your AJAX-call to the helper file are the same as the input names from the data model (see 3.3) in question, then the param values will be processed.

Again, for AJAX requests in an access restricted area you must make sure the PageBlocks session var is sent along with the request.

To resolve the problem of getting possible validation error messages back into the mother page that issued the call the technique will vary a little depending on the AJAX-library you use. But basically, you can add this (same as in 3.6) in your AJAX display page:

```
<?lassoscript
        if($fw_validator->('errorMsgs'));
                '<p>';
                iterate($fw_validator->('errorMsgs'), local('i'));
                        (loop_count) > 1 ? '<p>';
                        #i->second + '</p>';
                /iterate;
        /if;
?>
```

If your AJAX-call script has indicated the id of the -tag that should receive any injected HTML, then the validation error messages will appear in that very -tag. The way validation errors are presented to the user can be refined depending on time and JavaScript-skills!

You may also need to make sure the values in the form fields in the mother page get updated with new values, you can do this by adding some JavaScript to your AJAX-display page:

```
<script type="text/javascript">
        getElementById('ctMaNmeL').value = '[var('m_ctMaNmeL')]';
        ...etc
</script>
```

4.2 Securing Flash File Calls

I will here take a look at one example: a Flash uploader. A Flash uploader can take params from the HTML-environment, it can take values hardcoded into itself, and it submits those together with the file upload to a helper file, which does the post-processing on the server. To prevent a hacker from bypassing the Flash uploader and send directly to the helper file, both the Flash uploader and the helper file must be secured. While the securing of the helper file follows the principles indicated in 4.1 above, there are a few things that can be done to the Flash uploader itself:

Use the SWFObject JavaScript library to add the Flash movie, it can be found here <code.google.com/p/swfobject>. Not so much out of security concerns but because it is the only generally compatible, easily configurable way. With the SWFObject library loaded, the code for adding a Flash movie to your HTML looks something like this:

```
<div id="flashcontent">

        Error during load
</div>
<script type="text/javascript">
```


```
        var so = new SWFObject("illustrationUpload.swf","illustrationUpload","400","140","8","#f7f7ef");
        so.addVariable("fw_s", "[$fw_s]"); // session var
        so.addVariable("lang", "[$fw_client->('language')]"); // language var
        so.addVariable("maxsize", 2000000); // max allowed file size
        so.addVariable("filedest", "illustrationUpload.lasso"); // post processing file
        so.addVariable("fileexts", "*.jpg;*.gif"); // allowed extensions
        so.write("flashcontent");
</script>
```

This example is taken from an access restricted area (a CMS), in a public area you would hardcode the vars maxsize, filedest and fileexts into the movie.
Notice that we pass a value for the PageBlocks session manager, $fw_s, and a value for the PageBlocks client language. The former enables us to check for a valid session in the helper file, the latter enables us to display language dependent text strings in the Flash movie (more on localisation later).

In the ActionScript of the Flash movie, the session var is retrieved and later sent as a GET-param along with the request from the Flash movie to the post processing file (ActionScript):

```
//      declare internal vars and assign values from environment
var fw_s:             String  = fw_s;               // session id
var la:               String  = lang;               // language var
var fl_pp:      String  = filedest;    // destination for post processing file (server script)
 ---
//      build the request string
var my_lv:LoadVars = new LoadVars();
my_lv.fws       = fw_s;
var getparams = (my_lv.toString());
url = fl_pp + "?" + getparams;
```

Now, as for the allowed file extensions (file types), the limitation through Flash is client-side only. In the post processing file there is no way to check the uploaded files based on MIME-type as Flash sends everything as application/octet-stream. As far as I know, the only way to be sure an uploaded file is really what it pretends to be is a) to make sure it wasn't uploaded bypassing the Flash uploader and b) to save it in the post-process file with the extension given.

To make sure a file was really sent to the post-processing file by the Flash uploader, and not by a hacker attempt, you can do a very simple trick: hardcode a var into the ActionScript that is sent along with the request to the helper file by the Flash uploader! Here is an example from a public area not using the PageBlocks session var, $fw_s, but a standard Lasso session. The session id is passed into the Flash uploader the same way as above, except we use a different var name (ActionScript):

```
//      declare internal vars and assign values from environment
var sid:              String  = sid;                // standard Lasso session id
var chk:              String  = "mySecretCheckerValue"; // any alpha-num value
var fl_pp:      String  = filedest;    // destination for post processing file (server script)
```

And the request is now constructed like this:

```
var my_lv:LoadVars = new LoadVars();
my_lv.chk       = chk;
my_lv.sid       = sid;
var getparams = (my_lv.toString());
url = fl_pp + "?" + getparams;
```

If we make the helper file according to the first example shown in 4.1 above, the single .lasso-file, it could look like this:

```
<?lassoscript
        // start session
        client_getparams->(find('sid'))
                ?
                session_start(
                        -id                             = client_getparams->(find('sid'))->first->value,
                        -name                   = 'a',
                        -expires                = 60,
                        -cookieexpires = '',
                        -useauto
                );
        // add some conditional here for session passed or not
        // if session passed, check if the secret checker is present and has the right value
        client_getparams->(find('chk'))
                ? var('chk'     = client_getparams->(find('chk'))->first->value);
        if($chk == 'mySecretCheckerValue');
                // check passed so we start the PageBlocks page initialisation
                fwpPage_init;
                // here post processing code...
        /if;
```

```
?>
```

5. Localisation In PageBlocks

5.0 Localisation On Two Levels

The term *localisation* is here treated as the ability to display in more than one language. Localisation in terms of time zones, date formats, currency, etc. is left out.
We will look at localisation in PageBlocks on two levels: more than one website language, and more than one system language (as in an administrative system such as a CMS for example).

5.1 More Than One Website Language: Pulling The Right Data From The Data Source

There are several ways to acchive language localisation of a website, also within a PageBlocks context. This example uses language specific database tables, one for each language, for the dynamic content, and a mixture of those tables and of CSS for the navigational content. The two languages in this example are English and German, language codes ‘en’ and ‘de’. All way through the code the language defaults to ‘en’ if ‘de’ isn’t explicitly present.

A request for a page in PageBlocks passes the steps outlined in 4.1 above. To determine the language to display in a given webpage, the following code is added in the previously mentioned page config file, ‘_pageConfig.lgc’, that loads before anything else in a given subfolder (module). It uses a standard Lasso session and a cookie for storing the visitors language preference:

```
// if a cookie is present use that language preference
// else try to set cookie with default language ‘en’
if(client_cookies->(find(‘lang’)));
	var(‘lang’	= cookie(‘lang’));
else;
	cookie_set(
		‘lang’		= ‘en’,
		-domain		= $fw_requestPage->(‘domain’),
		-path		= ‘/’,
		-expires	= 525600
	);
/if;

// start session to hold language preference
session_start(
	-name		= ‘a’,
	-expires	= 60,
	-cookieexpires = ‘’,
	-useauto
);

// at this point, if there is no value for var lang yet or if its different from de
// then we define it with default value en
if((!var(‘lang’)) || (var(‘lang’) != ‘de’));
	var(‘lang’		= ‘en’);
/if;

// for var lang we now have either the value retrieved from the cookie or the default value
// however, every page has a language switch link pointing to the page itself,
// so we have to check if that link has been clicked and update values accordingly
protect;
	if(client_getparams->find(‘lang’));
		if(client_getparams->find(‘lang’)->first->value == ‘de’);
			$lang	= ‘de’;
			cookie_set(
				‘lang’		= ‘de’,
				-domain		= $fw_requestPage->(‘domain’),
				-path		= ‘/’,
				-expires	= 525600
			);
		else;
			$lang	= ‘en’;
			cookie_set(
				‘lang’		= ‘en’,
				-domain		= $fw_requestPage->(‘domain’),
				-path		= ‘/’,
				-expires	= 525600
```


```
                    );
                /if;
        /if;
/protect;

// now we have a language var, wether it came from a cookie, was chosen by the user by
// clicking the language switch link, or was the default 'en'.
// so now add resulting value for var lang to the session
session_addvariable(-name='a', 'lang');

// also update the PageBlocks client language var with the current value
// this var is used by PageBlocks for internal localisation procedures, see later
$fw_client->('language') = $lang;

// now we can instantiate the fwp_recordData object based on the correct,
// language specific table
$lang == 'de'
        ? var('scndPages' = fwp_recordData('pages_scnd_de'))
        | var('scndPages' = fwp_recordData('pages_scnd_en'));
```

The two tables, 'pages_scnd_de' and 'pages_scnd_de', are identical in structure and content, except for the variation in language for the stored data.
The `$scndPages` instance can now be used for pulling data for display with the standard PageBlocks code:

```
$scndPages->(select(
        -select         = '*',
        -keyval         = 'myPageId',
        -withMakeVars
));
```

5.2 More Than One Website Language: The Navigational Elements

Continuing the example above, we also want to change the navigational elements based on language. Lets say the main navigation HTML looks like this:

```
<ul id="mainnav">
        <li class="p1">
                <a href="/home"></a>
        </li>
        <li class="p2">
                <a href="/products"[$fw_myUrl->('path') >> 'products' ? ' class="sel"']></a>
        </li>
        ...etc
```

Where `$fw_myUrl->('path')` is a PageBlocks var which returns the path from the current request. The CSS for this navigation section is stored in a separate, language specific CSS. For English the file is named 'nav_en.css' and can look something like this:

```
ul#mainnav {
        list-style-type: none;
        float: left;
        margin-top: 46px;
}
ul#mainnav li {
        display: inline;
        height: 21px;
        text-align: center;
        float: left;
}

ul#mainnav li.p1 {
        width: 84px;
        background: url(/site/media/images/mainnav_01_en-over.gif) no-repeat;
}
ul#mainnav li.p2 {
        width: 89px;
        background: url(/site/media/images/mainnav_02_en-over.gif) no-repeat;}
...etc

ul#mainnav a {
        display: block;
        height: 21px;
}

#mainnav li.p1 a {
        width: 84px;
```

```
        background: url(/site/media/images/mainnav_01_en.gif) no-repeat;
}
#mainnav li.p2 a {
        width: 89px;
        background: url(/site/media/images/mainnav_02_en.gif) no-repeat;
}
...etc.

ul#mainnav a:hover {
        background: transparent;
}

#mainnav li.p1 a.sel {
        background: url(/site/media/images/mainnav_01_en-over.gif) no-repeat;
}
#mainnav li.p2 a.sel {
        background: url(/site/media/images/mainnav_02_en-over.gif) no-repeat;
...etc.
```

Notice the background images that are language specific. There is in this case a similar CSS file for German, referring German background images.
Now, all we have to do is to pick the right CSS file while the HTML markup stays the same. This is done in the page config file, '_pageConfig.lgc', already mentioned above, with a conditional checking upon the language var we established in 5.1 above, and a specific PageBlocks method for adding (or removing) CSS file references in the <head>-tag (see more on this in 6.1 below):

```
$lang == 'de'
        ? $fw_headContent->(addCssFile(($fw_sPath->'css') + 'nav_de.css'))
        | $fw_headContent->(addCssFile(($fw_sPath->'css') + 'nav_en.css'));
```

A website built with PageBlocks which uses these techniques can be seen here: <www.hillenet.net>.

5.3 More Than One Website Language: Error Messages

Eventhough we got a website up and running with two languages here, we still need to think about error messages in case something goes wrong. This is covered in 5.7 below.

5.4 More Than One Website Language: Another Approach!

Another approach one can choose deserves mention here because it makes use of built-in features in PageBlocks that bring us close to a simple CMS - one can actually setup multi-lingual pages in very short time!
PageBlocks has the possibility of assigning socalled 'blockTemplates' in a given sub-directory (module). The feature can be turned on/off in the page config file, '_pageConfig.lgc', with this command (more on $fw_pageModes in 6.1 below):

```
$fw_pageModes->enableBlockTemplates;
```

You would then typically make a combination of one or a few repeating blocks (content that doesn't vary from page to page) and blockTemplate-files that handle the individual page content. The idea in this approach is that:
- each single page does not have a corresponding file but must, however, still be listed in the list of available pages for the subdirectory (100% abstracted URLs takes a little more but is certainly possible)
- the individual content for each page is entered via the PageBlocks -> SiteManager -> SiteStrings (more on this in 5.10 below) which handles an arbitrary number of page string languages

For an example of this approach see <www.pageblocks.org/ftrs> and the corresponding sub-directory (module) in the PageBlocks download package.

5.5 More Than One System Language

By "system language" I mean the language of all static texts in for example a CMS. It includes all navigational texts, headers, subheaders etc., quest-, feedback- and error messages, field labels and button texts in forms, help texts and other documentation etc. If you want to enable more than one system language in an admintrative area, then you are presented with a more complex task than the case above which was only the website.

PageBlocks includes a manager named MVStrings - Multi View Strings - which is the key to the kind of localisation we are speaking of here. MVStrings builts on some elements:
- language specific config files holding all error- and validation error messages
- language specific config files holding all other messages and text fragments
- database tables holding language specific text strings and value list strings
- objects handling string lookups, caching & more

The MVS manager can actually determine which string to display in a given situation based on not only language, but also on media (ie. display on handheld device or in desktop browser) and on an arbitrary *variant*. Its a complete system for very refined string display!

5.6 More Than One System Language: User Language Preference

PageBlocks chooses the language to use for display based on the value of $fw_client->language, which defaults to the core language, which, again, defaults to 'en' for English. You can set a different value for the core language (if you understand the consequences) and/or a different default value for $fw_client->language.

Handling language preference can be done as follows. Somewhere in the top of the '_pageConfig.lgc'-file for the current sub-directory (module) you add:

```
if(var('m_uLang')); // check if user submitted a language choice (see below)
        var('userLanguage'     = $m_uLang);
else;
        !var('userLanguage') ? var('userLanguage' = fw_kCoreLanguage); // default to core language
/if;

$fw_client->(setLanguage($userLanguage));
```

The last line makes sure the language is kept throughout the session.
To make a language select option for the user to choose from you can make a select menu in for example the administration welcome page (simplified):

```
<select name="m_uLang" id="uLang">
        <option value="" selected="true">Choose System Language...</option>
        <option value="da">Dansk</option>
        <option value="en">English</option>
</select>
```

You store the language preference for the current user by adding the following in the logic block of the page with the select menu:

```
$fw_user->(addKeeper('userLanguage'));
$fw_user->(storeKeepers);
```

`$fw_user->(addKeeper())` and `$fw_user->(storeKeepers)` are PageBlocks functions that will store information about the current user for retrieval on next login. The values are restored automatically, meaning that the last stored value of var('userLanguage') is immediatly available after a successful user authentication. That is why we first check if that var is defined before defaulting to fw_kCoreLanguage above in _pageConfig.lgc (remember, _pageConfig.lgc loads before any logic- or whatever file).

5.7 More Than One System Language: Error- And Validation Error Messages

Error- and validation error messages are organised in 2 config files. They reside in a folder named '_pbStrings' in the root folder, their names are:

strings_coreErrors_en.cnfg
strings_coreValErrors_en.cnfg

Notice the language suffix 'en' for English. In the files, the messages are defined with an error number and a string.

The first thing to do when you want to add one or more languages to your website (as in 5.1-3 above) or to your project is to make copies of these files, save them with the appropriate language suffix and translate the messages in there. A discrete number of translations is, however, already available on the PageBlocks website.

Custom error messages and custom validation error messages are added as a combination of the methods for adding custom error routines and custom validation routines and custom string config files. While the methods for adding custom error routines and custom validation routines are outside of the scope of this presentation, the custom error string files need mention: those files are parallel to the core config files listed above, they live on the path /site/strings/ and are named as follows with 'app' for application:

strings_appErrors_en.cnfg
strings_appValErrors_en.cnfg

Those files are empty in a new PageBlocks project, if used they need translated versions exactly as their core counter parts.

5.8 More Than One System Language: Other Messages And Text Fragments - App Strings

You can store messages and text fragments other than the error messages in:

strings_coreStrings_en.cnfg

which lives in the same place as the two core error message files mentioned in 5.7 above, or in a file named:

strings_appStrings_en.cnfg

that lives on the same path as the two application error message files mentioned in 5.7 above.

Again, when doing localisation you would copy those files and save them with the appropriate language suffix. You wouldn't, however, store strings in 'strings_coreStrings_en.cnfg' unless you intentionally make up some core strings you plan on using in several projects.

Strings stored in '/site/strings/strings_appStrings_XX.cnfg' are available throughout the whole application. You can store strings one level deeper, in a PageBlocks *module*, which equals a sub-folder on root level. Such module string config files are named equal to the application wide file from above - the difference is that they live on a different path: '/<module>/_resources/strings/strings_appStrings_XX.cnfg'.

Strings in 'strings_appStrings_XX.cnfg' are defined with a name and a string and maybe media- and variant sub-divisions. Here is an example of a string definition from a module config file, 'strings_appStrings_da.cnfg', for language Danish:

```
{{modHeader:
variant_moduleContacts:
msg___ Kontakter og musikere
/variant;
}}
```

The string name is 'modHeader', the variant is 'moduleContacts' and the string itself is 'Kontakter og musikere'. The variant is here actually only used to keep the PageBlocks' strings caching mechanism on right track.

To apply that string in a display file you simply do:

```
[$fw_appStrings->modHeader]
```

This is possible because the MVS manager catches all application string names and makes them member tags of the `$fw_appStrings` object.

You would typically use the 'strings_appStrings_XX.cnfg' files - wether application-wide or module-wide - to store all text fragments that need localisation: feedback- and warning messages, button value texts, link- and title texts, legend texts etc. etc.

5.9 More Than One System Language: Field Labels, A Special Case

Field labels are typically defined in '/<module>/_resources/strings/strings_appStrings_XX.cnfg' as they usually are module-specific. Defining a string for a field label is done the exact same way as in the example shown above. I tend to use the input names (see 3.3 above) as string names in the 'strings_appStrings_XX.cnfg' files because it decreases the number of things to remember, but their names could be anything. Here the Danish definition for the label 'Last Name' we used in 3.5 and 3.6 above:

```
{{m_ctMaNmeL:
variant_moduleContacts:
msg___ Efternavn
/variant
}}
```

To use in display you do:

```
<label for="ctMaNmeL">[$fw_appStrings->m_ctMaNmeL]:</label>
<input type="text" class="size2" name="m_ctMaNmeL" id="ctMaNmeL" value="[var('m_ctMaNmeL')]" />
```

Now, this creates a problem: As we saw in 3.6 above, the PageBlocks validation mechanism takes the label names for use in it's error messages from the `-haslabel` validation code in the table config files. But now we just made a nice, language dependend solution for the input field labels via the MVS manager - how do we combine those two?

Fortunately, the `-haslabel` validation code inside the table config files can process Lasso! This means that the syntax for retrieving a string value shown in 5.8 above can be used in the table config file as well. If you make comparison to the validation codes shown in 3.3 and 3.6 above, we would now instead write (dataType substituted by <>):

```
{{tableModel___
#inputName             fieldName                        <>             validation codes
#-----------   ---------------       <>            ----------------------
m_ctMaTle                     cnctMainTitle          <>             maxlen=30, isAlphaNumeric,
haslabel=[$fw_appStrings->m_ctMaTle]
m_ctMaNmeF             cnctMainNameF          <>             maxlen=20, -extn, haslabel=[$fw_
appStrings->m_ctMaNmeF]
m_ctMaNmeL             cnctMainNameL          <>             isRequired, maxlen=20, -extn, haslabel=[$fw_
appStrings->m_ctMaNmeL]
```


```
m_ctMaIsStd          cnctIsStandard        <>            maxlen=1
}}
```

Voilà, field label names are now both language dependend in display AND as they appear in the validation error messages!

5.10 More Than One System Language: Strings Longer Than Texts Fragments - Page Strings

The MVS manager in PageBlocks has still more to offer! As touched upon in 5.4 above, language dependend strings can also be stored in a database table, instead of in the config files we looked at so far This is for example useful for texts longer than a few words and for content text on a public website, eventhough there isn't any technical limit to the length of strings defined in the config files. (For the completeness of this presentation it should be mentioned that application strings also can also be stored in a database table, taking application strings even further.)

Page strings stored in the database table are entered and edited in the PageBlocks access restricted administration system. Any PageBlocks installation comes with a ready-to-use Site Manager, which is an administrative interface to the user management, named Administrators, and to the page strings, named Site Strings and a few other things.

To add a page string you need to specify:
- the path to the page where you want to use it
- the language code (if different from core language)
- a name
- a media name (defaults to 'all')
- a variant name (defaults to 'default')
- the string (can be styled with PageBlocks styling markup)

The interface is pretty straight forward. You will always have a core language in which you define your strings in the first place. You can then edit other language versions next to the core version, allowing for a fairly intuitive process.

Strings stored in the config files as shown 5.8-9 are called with `[$fw_appStrings->stringName]` as already mentioned, while strings stored in the database table are called with `[$fw_pageStrings->stringName]`. Contrary to $fw_appStrings, $fw_pageStrings are, among others, defined by the path to the page for which they are to be used, allowing for page strings with the same name but different page paths.

As mentioned in 5.4 above, this is close to be a simply CMS. Actually, the Site Strings system can be taken as inspiration for a CMS of your own. Or you can indeed give access to it as-is for certain users to edit strings, it is, however, maybe not intuitive enough for all kinds of users, and the PageBlocks Site Manager itself isn't localised as of current ;-)

5.11 Value Lists

PageBlocks also has a language dependend value list manager (API). Value lists for select menues, radio buttons and check boxes can be defined with language dependend string values and are stored in a database table. To create value list with a set of values stored in the database you do something like this in the logic-block:

```
var('instrTypes' = fwp_valueList(
  -table    = 'vlists',
  -list     = 'instrumentTypes',
  -scope    = 'site',
  -titleOption= $fw_appStrings->menu04link03a + '...',
  -attributes = map(
    'name' = 'vl_id',
    'id' = 'vl_id',
    'onchange'  = 'if(this.selectedIndex != 0){ this.form.submit(); }'
  )
));
```

The values will be language dependend because `fwp_valueList` uses `$fw_clint->language` to determine which language version to pick. The select menu is then generated in the display-block with this simply line:

```
[$instrTypes->drawAsPopup]
```

5.12 Switching The MVS Manager Off And On

Despite caching of all application strings, the MVS manager causes some overhead. Currently page strings aren't cached (its on the TODO-list), so the use of page strings will cause database lookup on each page.

Its quite a time consuming challenge to convert an already-made project with all strings hardcoded into it to a language sensitive project, so you will always want to apply the techniques described in this entire chapter from day one, except of course if you are 100% sure you will never need localisation. For such cases the MVS manager can be switched either

completely off for the whole project in the master config file, or switched off for a module in the page config file with:

```
$fw_pageModes->disablePageStrings;
```

Actually, 'disabled' IS the default setting for the MVS manager! So what you need to do is really the opposite: if you want to use it - either project-wide or module-wide - you switch it ON with:

```
$fw_pageModes->enablePageStrings;
```

6. Handling Of CSS- and JavaScript Files In PageBlocks

6.0 Customizing automatic processes

As PageBlocks is a tool that takes care of the page assembly process, some areas are "hidden" in automatic procedures. But "hidden" isn't the right word, there is nothing in this process that can't be taken care of on a more individual basis.

Everything inside the <head>-tag is a good example of this. You can set a few vars in the project master config file, and PageBlocks will render a discrete set of tags inside the <head>-tag.

If you need more control (and you usually do), you enable a couple of switches in the application init files, and now you can control basically everything that happens inside the <head>-tag via three config files living in a folder named '_pbLibs':

_defineCSS.lgc
_defineJavaScript.lgc
_definePageHead.lgc

Lets say you want to load a special CSS file based on a condition looking for a certain browser type (did I say MSIE...?) then you would code this conditional in '_defineCSS.lgc'.

6.1 $fw_headContent And $fw_pageModes

But there are some customisations that can be applied very easily with two of those handy tools in PageBlocks:

`$fw_headContent`, already mentioned in 5.2 above
`$fw_pageModes`, already introduced in 5.4 and 5.10 above

$fw_headContent can add and remove certain content from the <head>-tag on a project-wide, a module-wide or a page-wide basis. $fw_pageModes is used to control a number of page assembly options. The different usages of those two controllers are covered in the Developer Guide, here I just wanted to mention a few that are connected with the use of CSS and external Javascript files.

To add a CSS file named 'admin_import.css' living on the PageBlocks standard path for CSS files, '/site/css/', you do:

```
$fw_headContent->(addCssFile($fw_sPath->('css') + 'admin_import.css'));
```

To remove the file you do:

```
$fw_headContent->(removeCssFile($fw_sPath->('css') + 'admin_import.css'));
```

Notice the `$fw_sPath`. The 's' stands for 'site', there is a parallel one, `$fw_mPath`, with 'm' for 'module'. They both return a number of standard paths in PageBlocks and can be used for things like the code example above.
To add or remove a Javascript file named 'scripts_main.js' living on the PageBlocks standard path for Javascript files, '/site/js/', you do:

```
$fw_headContent->(addScriptFile($fw_sPath->('js') + 'scripts_main.js'));
$fw_headContent->(removeScriptFile($fw_sPath->('js') + 'scripts_main.js'));
```

As I said before, this code can be inserted at any level, down to a per-page basis. So you could use it for including page specific Javascript files. PageBlocks has another option for this, however, using `$fw_pageModes`:

```
$fw_pageModes->enableJSPerPage;
```

Again, it can be used on a project-wide, a module-wide or a page-wide basis, but a typical use would be module-wide or page-wide. What it does is that it looks for a Javascript file with the same file name as the page currently being called on the PageBlocks standard path for module-resources: '/<module>/_resources/js/'. So if the page currently being called has the name 'contactEdit' inside the module 'contacts' and it has `$fw_pageModes->enableJSPerPage` specified, then PageBlocks will look for this file: '/contacts/_resources/js/contactsEdit.js'. PageBlocks will add the Javascript file if its in there, otherwise it fails silently.

To make it a little easier to load a few standard AJAX/Javascript libraries, PageBlocks comes with these libraries pre-installed ready to add with $fw_pageModes:

```
$fw_pageModes->enablePrototype;
$fw_pageModes->enableScriptaculous;
$fw_pageModes->enableJquery;
```

They all load a local library included with the PageBlocks package, they don't load an external library resource as often is done in the case of jQuery. This could be added to the TODO-list if wished.

6.2 Dynamically Generated Javascript

PageBlocks doesn't at present have a pre-defined way of adding dynamically generated Javascript. It is my wish to add this functionality at some point; what would be ideal would be to have Javascript validation scripts automatically generated based on the validation codes entered in the table config files (see 3.3 above).

There is, however, a straight forward way to add dynamically generated Javascript based on the values of Lasso variables. Since all Lasso logic code must be executed before starting to generate dynamic Javascript, the method makes use of the fact that you can add Javascript at any point in the page rendering process.
PageBlocks has the ability to enable - or disabling - repeating blocks of code for a any number of pages. Again, $fw_pageModes is used for that:

```
$fw_pageModes->(setRepeatLogic(array('left','main')));
$fw_pageModes->disableRepeatBlocks;
```

This usually serves other purposes such as adding repeating code to a layout - for example a navigation bar.

But the variant

```
$fw_pageModes->(setRepeatBelow(array('main')));
```

has the advantage that it executes after anything else. So if you add a file in your module named 'repeatBelow_main.dsp' and enables it via $fw_pageModes where needed, then you have a perfect place for adding dynamically generated Javascript like the following (based on Prototype):

```
<script type="text/javascript">
        var lang = $F('cl_l');
[iterate($contactObject->getPhones, local('i'))]
        new Form.Observer(
                'phoneform_[#i->(get:2)->(get:1)]',
                0.3,
                function() {
                        $('updatephoneitem_[#i->(get:2)->(get:1)]').setStyle('background: url(/site/media/
images/saveBtn-a_' + lang + '.gif) no-repeat left top;');
                }
        );
        $('updatephoneitem_[#i->(get:2)->(get:1)]').observe('click', updatePhoneItem.
bindAsEventListener(this, [#i->(get:2)->(get:1)]));
        $('delphoneitem_[#i->(get:2)->(get:1)]').observe('click', deleteItem.bindAsEventListener(this,
'phone', [#i->(get:2)->(get:1)]));
[/iterate]
</script>
```

Nikolaj de Fine Licht, Brazil/Denmark, July 2008

SVN for FTP Recidivists

Jonathan Guthrie
xServe, Ltd
http://www.xserve.co.nz/

Abstract

The larger the projects and the tougher the timeframes, the more you and your colleagues will benefit from using Subversion. Actively investing in learning these skills will set you up to win when you or your company lands 'the big one'.

Biography

Jono, as he prefers to be called, is proud of not being 40 yet, proud of his kids, totally into anything on two wheels. He has a passion for working smarter and loves a challenge. Jono heads up 'xServe Limited' a niche development company based in Wellington, New Zealand working for clients across the globe.

Lasso Developer Conference Materials Download
Server: **http://dowloads.lassosoft.com/beta/** Username: **LDC**
Path: **Lasso Developer Conference** Password: **LCROSS**

SVN for FTP Recidivists

Jonathan Guthrie, xServe Ltd; Wellington, New Zealand

The challenge

Imagine the following scenarios.

> You work in a workgroup of developers and you finish some mods on a document and FTP it up to the server, press the appropriate "ok to replace" button, and half an hour later a colleague sends an email to the group angry because his changes have just been overwritten…
>
> You've spent the last few hours writing a script to manage and organize file uploads. The code works, you're happy with your code and you test it. Later that day you open it again and make a quick change to a few variables and close the file. A couple of days later you find that file that you thought you had sorted has an error, and yet you were positive it worked before, but can't remember what it was you changed. You don't really have any way of reverting back to that time when you knew 100% that it worked…
>
> You've worked on a complex page the whole day, and then you find you have accidentally saved over the top, and it's all gone…
>
> You have been working on changes to a client's site for the last week and now you want to FTP them up to the live server – however you don't know exactly which files were changed, and you really don't want to FTP the whole project up afresh because it's huge and you didn't copy down all the files. You have to comb through the whole site comparing last mod dates, hoping you don't miss anything…

Some of these scenarios sound pretty stupid, some achingly familiar, and even if you're still using FTP, AFP, or SMB you're still at risk of needing to use another much ruder acronym.

So if you are currently a frequent user of one of the above TLA's I want to introduce you to an alternative, one that I personally find is better; SVN, or to use it's real name, Subversion[1].

Why am I so 'sold' on Subversion? Because it gives me maximum earning power, with minimum mess ups, and it makes the most of team work. Making team work, effective is a valuable thing as I work in the opposite time-zone to most of my fellow developers. Lets look at how this is done.

Enter the world of Subversion

Subversion is an implementation of a Version Control System (also known as Source Control, Source Code Management or Revision Control).

Revision Control is defined in Wikipedia as:

> … the management of multiple revisions of the same unit of information. It is most commonly used in engineering and software development to manage ongoing development of digital documents like application source code, art resources such as blueprints or electronic models, and other projects that may be worked on by a team of people.[2]

So what does that mean for us practically?

It means that if you are working on a document that a colleague is also working on, SVN will alert you to conflicting changes and you have several ways to resolve the conflicts.

---

1 Subversion is available from http://subversion.tigris.org/

2 Wikipedia, http://en.wikipedia.org/wiki/Revision_control

If you have broken a script or simply killed it by overwriting it, you can either roll it back to any previously committed version, or compare your existing document to a specific prior version and selectively incorporate differences.

But wait! There's more!

How about splitting off a whole new branch of code? By using the SVN's branching feature you can create a whole new project based on an existing one.

Do you like to re-use common chunks of code, like FCK Editor or Custom Tags and Types? No problem: SVN Externals are like directory aliases between projects.

Want to put marker milestones in place so that you can deploy specific point revisions of a project? Use the Tags feature to create a snapshot of your project.

Want to be able to have multiple servers all working off the latest version of the trunk or branch, or all on the same tag? It's easy to script servers to pull the right data from a central SVN server.

Some practical how-to's

So lets look at some practical examples.

For this presentation I will be using Eclipse[3] fitted out with Lasso Studio for Eclipse, and Subclipse[4], a SVN module for Eclipse that integrates SVN function into the Eclipse workflow experience.

Typically, developers working with FTP often write their code on their workstation and upload this to the server to test. Developers using a file share protocol often work directly on the server. This makes sense to some degree in that there's only one lasso setup, and one database setup.

Working with SVN requires a slight mindshift away from central share code access towards a more independent working environment. Tom Wiebe educated me in the term "sandbox" when explaining this concept to me - and this again is found well described in Wikiedia[5].

> A sandbox is a testing (or virtual) environment that isolates untested code changes and outright experimentation from the production environment or repository, in the context of software development including web development and revision control…

So, your own sandbox would typically mean that you have locally:

- Lasso Developer or Lasso Studio for Eclipse
- A datasource such as MySQL or similar
- Apache
- A hosts file modified to suit
- Eclipse / Subclipse

Your setup for the Lasso site will mirror the settings required for deploying the site on the production server.

Your local datasource will have a copy of the database(s) used in the project. You can use a subset of the data if you so wish – there's no sense in having your projects' full database if size and/or confidentiality is an issue.

---

3 Eclipse is an Open Source "Integrated Development Environment". http://www.eclipse.org/

4 Subclipse is an Eclipse Team Provider plug-in giving support for Subversion within the Eclipse IDE . Subclipse is available from http://subclipse.tigris.org/

5 Wikipedia http://en.wikipedia.org/wiki/Sandbox_(software_development)

It makes sense to mirror any custom or special configuration directives from the live sites' Apache config.

Remember, the whole point of this exercise is to duplicate the functionality and the behavior of the live site as much as is practically possible.

However, this is where the full force of the concept of a sandbox comes in: the best thing is that even if you make a mess of your own sandbox, you don't impact the live servers and you don't mess up your colleagues' sandboxes. Most importantly, if you've made a really big mess you can delete your database and restore from the snapshot and delete your project directory and check it out afresh from the SVN server!

This paper is not going to cover the setup of Lasso Developer/Studio, datasource setup, Apache vhost setup or even the install of the SVN server itself – instead this paper is now going on to the business end - some examples of how to manage your code in the Eclipse setup I described earlier.

Working with production and staging servers

Releases to production servers should always be tested thoroughly. However it is impractical, from both a time and cost perspective, to expect development and repository commits to be suspended until that testing is complete and the code is updated on the live servers.

That is why we have **"tags"**.

Tags are a way of marking a specific point in the development of a project – 2.3.4.02 as example. The leading "2" would be the major revision number, the 3 the minor revision, or cycle of releases. The 4 could be the release you're planning on deploying, and the 02 would then be the bug fix revision based on the testing to harden the code prior to deployment.

So you tag a release as 2.3.4.00, test, and discover some minor bugs you need to squash, you fix them, re-tag as 2.3.4.01. You continue until it's solid.

Say 2.3.4.02 is your solid build – that's the one you should export to the production server. You should either issue the command from the terminal, or use a GUI client like SvnX to do this. There are a couple of options here as well. You can either "switch" the link that the production server files are attached to in SVN (ie from tagged release 2.3.3.05 to 2.3.4.02), or you do a new export and change the DocumentRoot directive in your apache virtual host file.

If you do an export the version info is not retained as it strips out all SVN specific info, whereas if you do a checkout, version info can be retrieved using SVN and urgent updates to that tag can still be applied if required.

Either way, where infrequent staged releases are deployed, it is useful on the server side to visually identify which version or tag has been used.

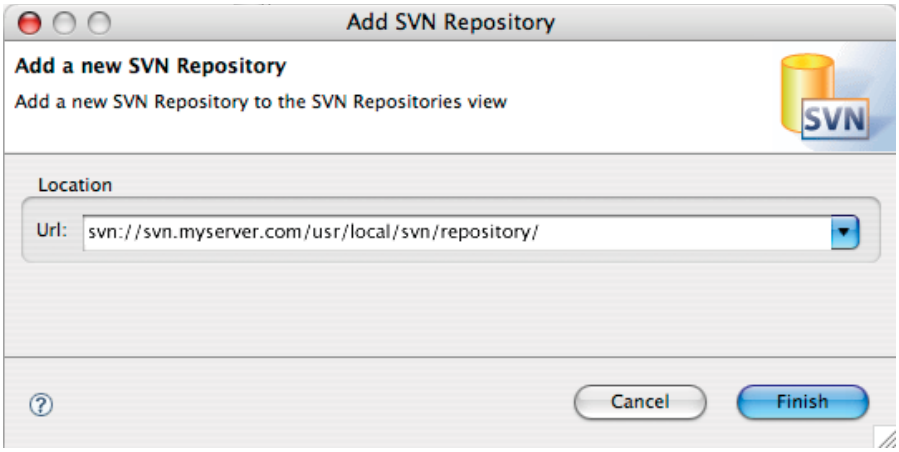
To define the Repository:

Select *Window -> Open Perspective -> SVN Repository Exploring* (You may need to select "other" menu item and choose from the list)

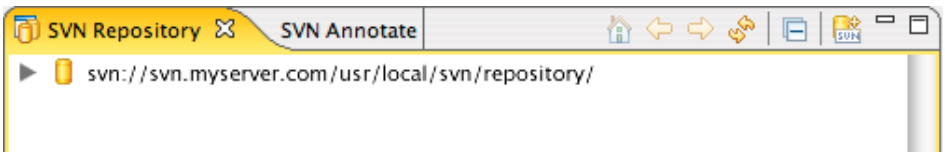
Click the "Add repository" button as below:

Enter the location of the repository, Click finish. The location will be a URL similar to "svn://myserver.com/usr/local/svn/

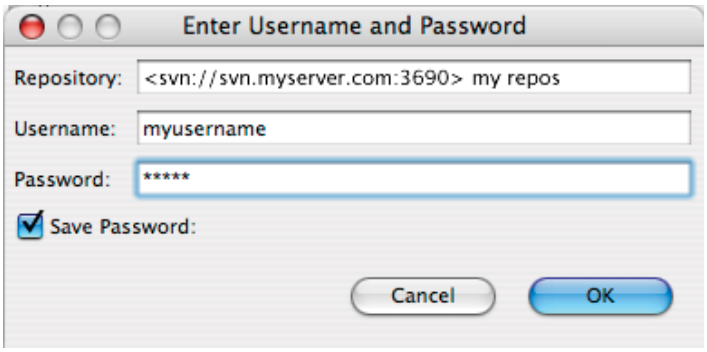
repository/”



You will see the repository list as below.



Click on the triangle to expand, you should see an authentication dialog like the one below appear:



Enter your username and password as supplied, click OK.

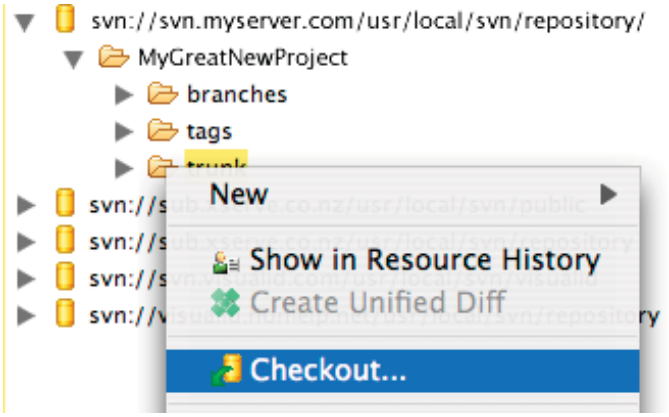
The repository list will be fetched from the server and returned in hierarchical form.

Checking out a new project from the repository

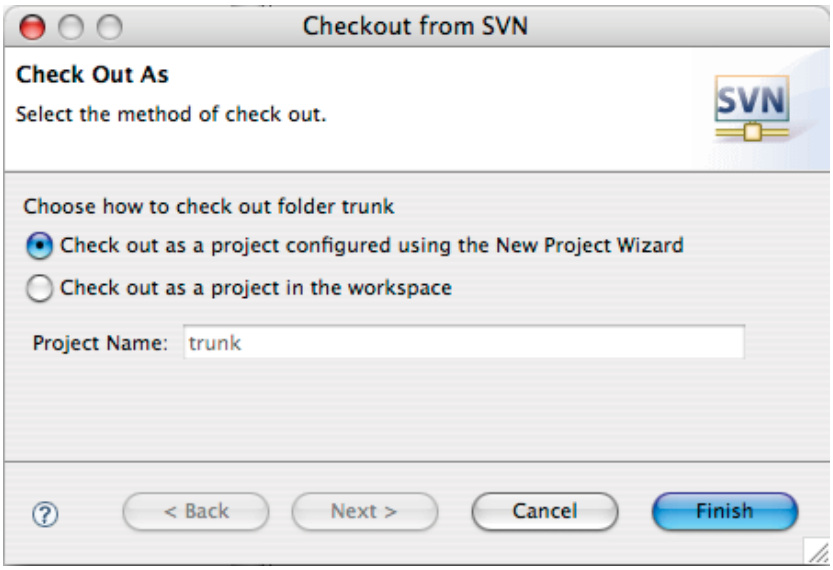
To begin working on the project, you first need to check it out from the repository.

Note: If you are starting a new project instead, you will need to see the *“Creating new projects in the repository”* instructions in the following section.

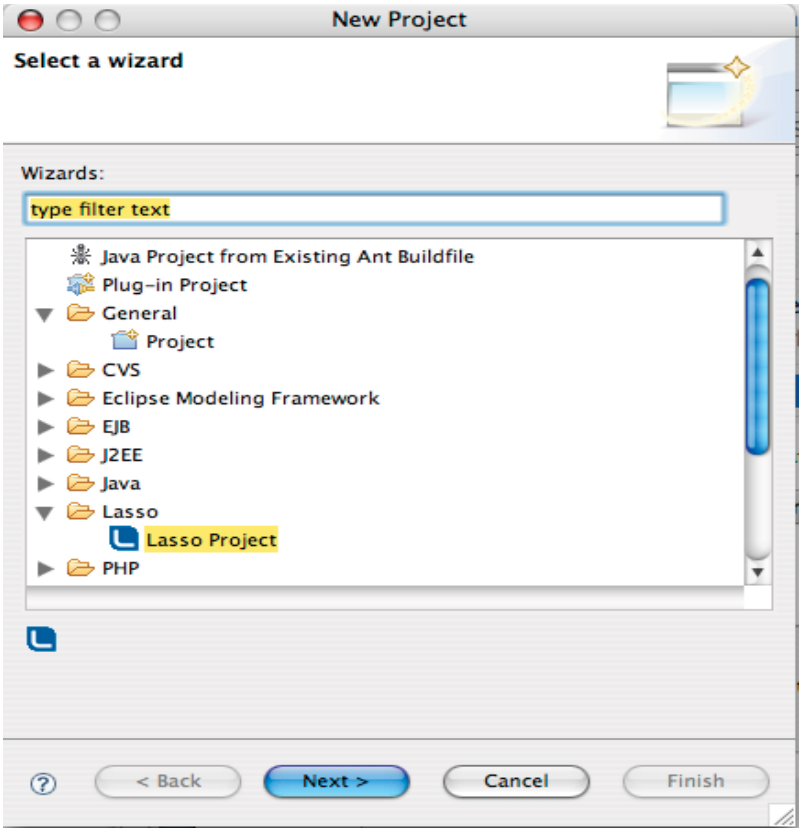
Right-click on the directory you wish to check out as shown below, and choose “Checkout…”



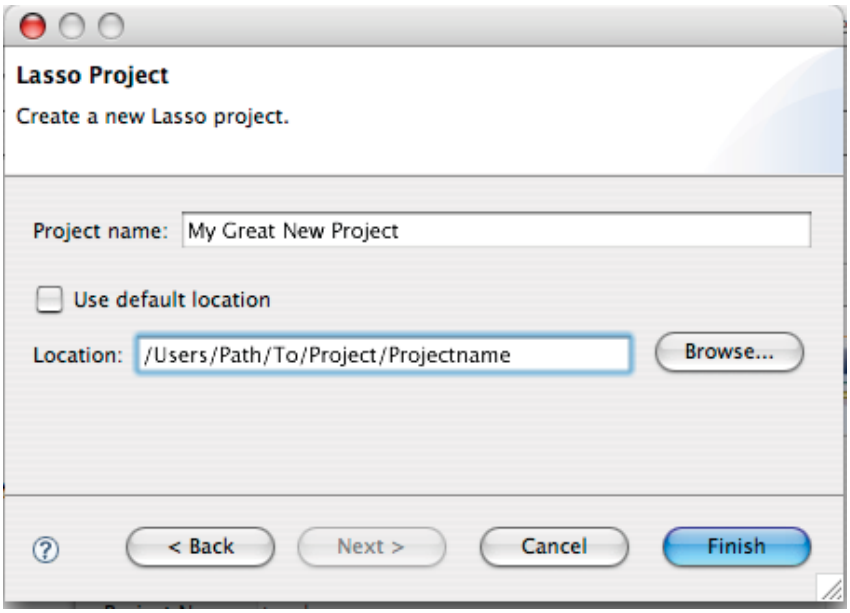
Click “Finish” to use the “New Project Wizard”. You will then be presented with the following screen:



Choose “Lasso Project” and click “Next”



Choose an appropriate name, and if you don't want the project stored in the default location uncheck "Use default location" and navigate to your new directory under "Browse…"



On clicking "Finish" your project will check out from the server.

Your project is now "connected" to the SVN server, and when you make changes Eclipse will show a different icon for the file or directory depending on it's status.

It will signify changes, additions, conflicts.

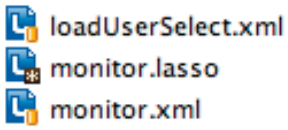
Note: The console pane is also a worthwhile addition to your perspective. It will show you what's been updated, and where

potential problems lie.

Working with Eclipse and SVN

Committing

When you make a change to a file the icon changes as seen in the image below.



The * means it's a local change, and should be committed to SVN.

You don't need to commit a change immediately to the repository but you should get in the habit of doing so regularly. Typically this means committing the changes at each "stage" completion, or when a bug/issue has been resolved involving the files concerned.

At the minimum it is best practice to commit your changes at the end of each day so other developers do not get too many chances for their work to conflict with yours.

When committing, you choose how high or wide to go: you can commit a specific file, a selection of files, the set of changed files in a specific directory, or all changes within the project.

It is important to make sure that you comment your changes on commit. These comments are crucial to other developers (and often yourself!) knowing what was changed and when, and by whom. When needing a rollback or restore, these comments will also aid you in knowing a "last good state".

Updating other users' changes to your own sandbox

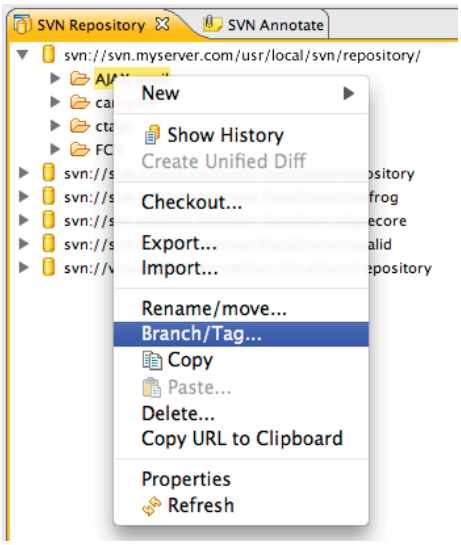
At the beginning of each work session it is advisable to do an update of the entire project to ensure you are incorporating other developers' recent work. This is crucial to help you avoid conflicts (situation where more than one developer works on a given file). If heavy development is underway you may even decide to update your sandbox more frequently to be comfortable that the alterations you are making to the codebase work in harmony with the other concurrent work.

Creating a branch

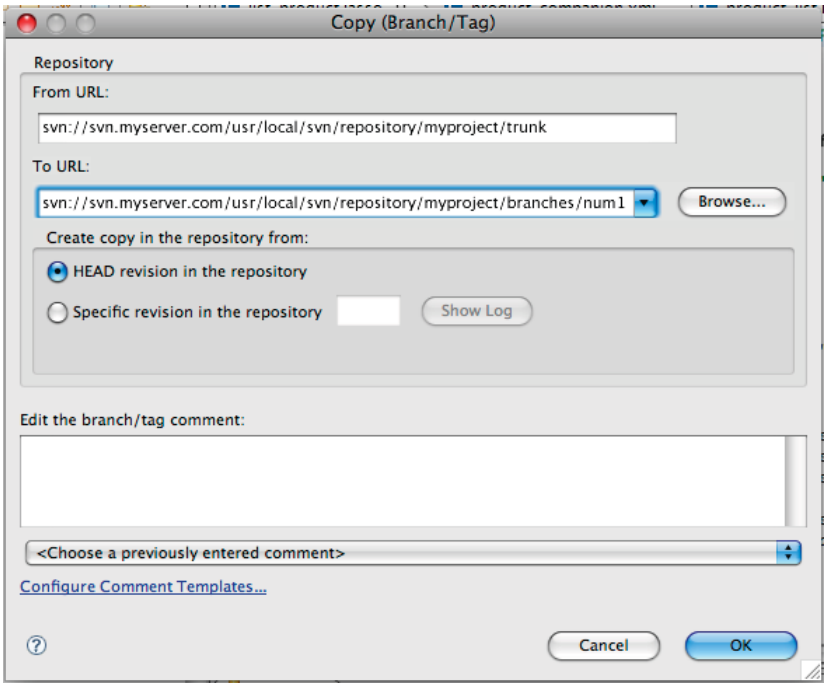
Creating a branch is an easy way to either create an entirely new iteration of a site founded on a "base code", or to create a version of the codebase with which you'd like to do major surgery to without affecting the normal run of bug-fixes and updates to the trunk. This allows parallel development by either yourself or multiple developers.

Select *Window -> Open Perspective -> SVN Repository Exploring*

Right click on the directory you'd like to copy/branch/tag



Select Branch/Tag…

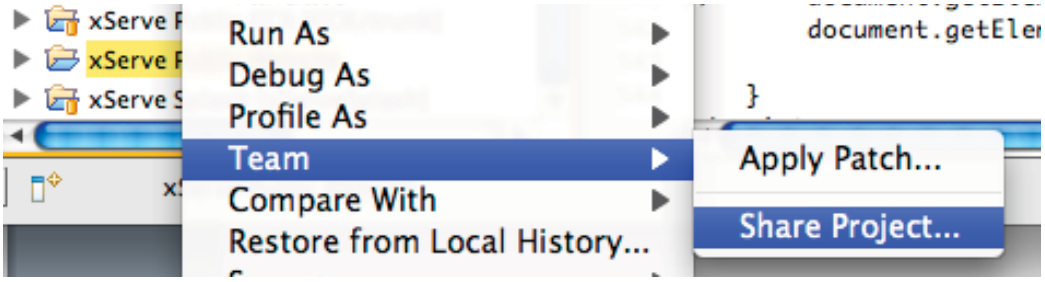


Enter the destination URL, or navigate to it using the “Browse” button. Enter a comment and click OK.

Your branch is done and ready to check out and start using.

Creating new projects in the repository

When you have code you wish to commit afresh to a repository that you have already defined, right click on the project in the resource view navigator, select Team > Share Project…



Select your SVN repository location, specify the folder name, click next and enter an initial commit comment.

The project is then committed and attached locally to your SVN repository.

Some final thoughts

It is good practice to do an update from the repository each morning at least – or more often if frequent work from multiple developers is taking place. This will help minimize conflicts that need to be resolved.

The larger the projects and the tougher the timeframes, the more you and your colleagues will benefit from using this development tool. Actively investing in learning these skills on smaller projects will set you up to win when you or your company lands 'the big one'.

This tool also makes re-use of your code much more efficient, and once in place it is much more easily maintained. This in turn gives a serious boost to your ability to output new sites cleanly and in record time.

Maximum earning power, minimum mess ups, and making the most of team work.

Round Tables

The round tables are your opportunity to find out more about the topics which was discussed earlier in the conference. Most of the speakers will be hosting round tables about the papers they presented. In addition, LassoSoft employees will be available to discuss Lasso 9 and other Lasso issues. Finally, Eric Landmann will be hosting a round table about debugging tips and techniques. The round tables are hands on so please bring your portable computer or code samples.

Under the Hood
Debugging Tips and Techniques

Eric Landmann — **Landmann InterActive**
http://www.landmanninteractive.com/

Abstract

This round table will discuss several methods to get feedback from Lasso to debug applications, to find out what's going on "under the hood." The discussion will include triggering display of variable values, the filename being processed, errors thrown from inlines or file operations, and writing errors to the browser or error logs. Hands-on examples are provided.

Biography

Eric Landmann is the lead developer of two commercial Lasso-based offerings: Graphics Finder, a digital assets management system, and TrackMyBugs, a subscription-based service that helps developers track bugs and feature requests. He has been working with Lasso since 1998, developing a variety of web applications for a diverse commercial and institutional clientele. Eric has contributed to two previous Lasso Summits. His background is in code-based typesetting systems, graphic arts and color separation. His offline passion is ice climbing and high-altitude mountaineering in the high ranges around the world.

Lasso Developer Conference Materials Download
Server: **http://dowloads.lassosoft.com/beta/** Username: **LDC**
Path: **Lasso Developer Conference** Password: **LCROSS**

LDC 2008

"Under the Hood – Debugging Tips and Techniques"

Eric Landmann
Landmann InterActive

INTRODUCTION

In this roundtable we will discuss strategies and methods to get feedback from Lasso to debug applications. The goal is to help the developer understand what's going on "under the hood" with the Lasso page being processed, databases, security, and error logs. The discussion will include triggering display of variable values, the filename being processed, errors thrown from inlines or file operations, and writing errors to the browser or error logs. Code examples are provided. It is assumed that the participant has a very basic knowledge of Lasso functionality. Given that this is a fairly large topic, we will highlight some common problems encountered.

The Problem

Figuring out what is "going wrong" can sometimes be daunting in debugging an application. When the code that looks perfectly reasonable to your eyes does not produce the expected results, it makes you wonder: "What went wrong? How did this happen, I didn't ask for that? Why is there no output at all?"

The primary problem is that, by design and for excellent reasons, middleware applications hide what is happening on the server from the viewer. A secondary problem is that, even if your code is entirely appropriate and works in one context or on one server, layers of security, database connectors, firewalls, browser problems, and other applications or appliances can interfere with the middleware to mess up your world. Your code might be perfectly fine – it might be something else entirely. We will concentrate mainly on coding problems.

THE PROCESS

The debugging process presented here is used to *eliminate possible causes* through a (hopefully) small number of steps to isolate the problem and determine the cause. This process should be useful for many types of problems.

1. IDENTIFY the Origin

First, you need to *identify exactly where the problem is happening.* Sounds easy, yes? Most of the time it is, but there are many times when a problem surfaces downstream of where the actual cause is located. An example of this would be a search returning no records. The query may be in several other includes, a named inline in another file, or rely on a variable that has no value. The primary method to identify where the problem lies is to *simplify your code.*

2. SIMPLIFY the Code

*Reduce the problem to its simplest elements.*This means getting rid of includes, redirects, data lookups, etc. that might interfere. Basically, get rid of everything that has nothing to do with the immediate problem.

3. Get FEEDBACK

The primary way to get feedback during development is to *employ a system that will output debugging information* to the screen and possibly the Lasso errors database and logs. One common system uses a simple debugging variable that acts as a master switch to turn on and off debugging information. Here are some tricks to get feedback from Lasso:

1. Add code to output variable values, including perhaps the type of data.
2. Add code to output errors.
3. If doing file manipulations, add code to output file errors.
4. If in an inline or doing a search, add code to output the number of records found.

4. Create a TEST File

Isolate the test problem by creating a small Lasso file that does only what you want. For example, if you are trying to perform a database search, create a simple file with only the information critical to that search succeeding. If you are using variables that you expect to have a certain value, *hard-code the values* in this file for testing purposes.

5. EXPAND the Test

When you have the basic code working, *expand the test file* to make it more like the real application. This might include, step-by-step, adding back in functionality that was removed. **Note:** It is very important to *test each change* to make sure you don't lose track of what was changed!

6. RESTORE the Functionality
Put the code back into your page, and see if it behaves properly.

USEFUL TIPS

Here are some tips and tricks that can help you through this process. The roundtable materials include sample files (see **"Sample Files"**).

1. Back up your files; this might get messy. When working through a problem, keep incremental files. That way it is easy to back up to a previous version. If you're on a Mac, BBEdit has an essential "Make Backup" feature which is quite handy. Sometimes I keep a backup file that works at a certain level of functionality, and name it "filenameWORKS.lasso" just as a sanity check.
2. Use the vardump.lasso file to see all variables on the page. This file can be tailored to ignore certain types of variable names, which adds extra value when you are using a naming scheme.
3. First, try adding some simple variable output and a variable dump to the page. The answer *might* be obvious.
4. Comment out or delete noncritical code.
5. Hard-code variables with expected values. Output them to the page to be sure they are valid.
6. Pay attention to the HTML source (use two different browsers if you suspect something is not right).
7. As a way to locate the specific line that is outputting, you can put a line number in the file which simply acts as a unique marker so you know precisely which line you are looking at.
8. Use a protect block to isolate errors that will stop Lasso from processing a page, capture the error, and display it.
9. Inspect the data directly with a database client. Verify that what you *think* happened, really *did* happen (or not).
10. If writing custom tags, build in debugging output from the start. That way when you need it, it is there.
11. Use a debug container tag (provided in the materials) to build debugging into your application. It will output debugging messages conditionally.

USEFUL LASSO TAGS

Lasso contains 50 error tags that can be used in many different ways. Here are the most commonly-used ones, plus some others and what they do. Consult the documentation for many other uses.
[Error_CurrentError] – returns the current error code or error message.
[Error_NoError] – is returned when a database action is successful.
[Error_Code] – returns the current Lasso error code (a number).
[Error_Message] – returns the current error message. This tag and [Error_Code] combined are output in [Error_CurrentError].
[Error_Reset] – useful to reset the error to 0 and the message to nothing, if you are not concerned at a particular point that there is an error or don't want a page to stop.
[File_CurrentError] – reports the last error reported by a file tag. A little tricky!
[Protect] – a container tag that is used to "protect" a portion of a page from throwing a Lasso error. Used in conjunction with the [Handle] tag.
[Handle] – Another container tag used in conjunction with [Protect] to output an error captured within the protect block.
[Lasso_ErrorReporting] – Sets the level of error message for the current page. By creatively using this tag, you can conditionally log certain errors.
[Log] – One of many log tags, this one is a container tag that will log the contents between the open and close tags to the specified file.
[Log_Detail], [Log_Critical], [Log_Warning] – These three tags correspond to the various logging levels found in the siteadmin log display.
[Response_Filepath] – Returns the path to the current Lasso file.
[Session_ID], [Session_Result] – [Session_ID] will display the current session ID, [Session_Result] displays whether it is a new session, loaded an existing session, or if the session is expired.

USEFUL CUSTOM TAGS

These custom tags can all be found on tagswap.net (see "Resources").
[Debug] – A container tag that outputs debug messages if $xDebug = Y. Can be customized for your environment.
[LI_URLRedirect] – A custom tag that formats a redirect as a link if $xDebug = Y so the redirect can be viewed.
[ErrorTrack] – A custom tag from Miles that has seven different options for reporting errors or action_params.
[Error_Reset] – A custom tag from Jason Huck that can clear any previously set error code or message or pass your own error codes and messages to it.
[lp_error_clearError] – A custom tag from Bil Corry that resets [Error_CurrentError] to [Error_NoError].

SOME COMMON MISTAKES, PROBLEMS, AND POSSIBLE RESOLUTIONS

1. Assignment Instead of Comparison

The following code is intended to check if $MyValue is equal to the integer 4. But does it? If you do this:

```
If: $MyValue = 4;
```

instead of ...

```
If: $MyValue == 4;
```

... your check will fail (the second equal sign to indicate equality is missing in the first statement). That's because the first usage is assigning 4 to $MyValue. Note that the first will NOT return any sort of error, because it is actually a valid statement!

2. Data is the Wrong Type

One symptom of this is that the "wrong" data is saved to the database. Some examples of this are attempting to save a string to a MySQL integer field, or a date that is incorrectly formatted is saved to a datetime field, yielding a date/time of 0000 00:00:00. The problem here is that MySQL will not accept data that is not of the correct type, and instead will either add a default value or convert the data to the datatype of the field definition. **The Fix:** Create a test file and determine the correct type of data that can be written, then format your data accordingly.

3. Poorly-written SQL Queries That Throw Errors

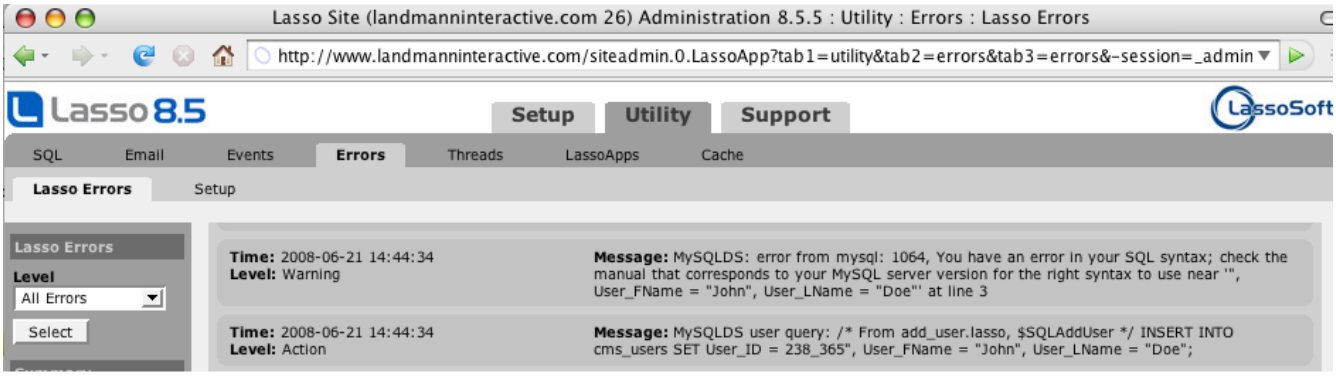
This can be easily diagnosed by writing the query as a variable, outputting the variable to the page, copying the query from the page to a database client, and running the query. You should get immediate feedback about what is wrong. Common problems include an extra comma after fieldnames, mismatched quotation marks for search strings, no WHERE clause specified.

The Fix: If you are writing your own SQL queries, save a valid, working query directly in the code. This makes it easier to troubleshoot if an unexpected situation arises. If you can run the query as rendered on the page and it works, it's not the query. Also, if you turn on logging for Action Statements to the Lasso Errors Database, the first line of the query which contains the "/* From add_user.lasso" line will display, so it can tell you which page is throwing the error.

```
// SAMPLE QUERY
/* From add_user.lasso, $SQLAddUser */
/* INSERT INTO my_users SET
      User_ID = '32F9k99',
      User_FName = 'John',
      User_LName = 'Doe'; */

Var:'SQLAddUser' = '/* From add_user.lasso, $SQLAddUser */
      INSERT INTO ' $xUsersTable' SET
      User_ID = "' $NewUserID ', User_FName = 'John', User_LName = 'Doe';
```

In the above code, there is a missing double quote after $NewUserID. This will create a MySQL error, which can be caught in the Lasso Error database. By adding in the comment on the first line, you can easily see the comment in the Lasso Error database, which gives a clear indication where the problem lies. This can be very useful on busy servers.



4. Inlines Return Some, but not All of the Fields
In Filemaker, the layout being called might not contain the correct fieldname, or the layout might be corrupt. In MySQL or other SQL databases, the query might not have the field stated in the SELECT statement. **The Fix:** In Filemaker, the layout may be corrupt, so create a new layout with only the fields you want. Make sure to change the Inline to use the new layout name. In MySQL, add the fieldname to the query.

5. Inlines Don't Return Any Results
A very common problem. This could be security-related, a bad query, a database not being online, firewalls interfering, or other things.

To test for a security-related problem: Output the [Error_CurrentError] and you will get some feedback.
To test for a database not being available: Use the [Database_HostInfo] tag to check if the database is being reached.
To test for a bad query: See #3 above.
Using SQL Queries: If your query is good, and you still get no results or any feedback, it might be this easy-to-miss problem:

```
Inline: –Database=$xSiteDatabase, –Table=$xUsersTable, $SQLAddUser;
```

Notice the problem? The -SQL parameter to the Inline is missing! The correct Inline command is:

```
Inline: –Database=$xSiteDatabase, –Table=$xUsersTable, -SQL=$SQLAddUser;
```

This would cause the inline to fail, even though the SQL query is perfectly valid.

6. Only 50 Records Returned
A Lasso classic! This "problem" has caught many developers unaware. Lasso will by default return the first 50 records, unless told to do otherwise. **The Fix:** Use "–maxrecords=all" in the inline.

7. Action_Params are "Lost"
This frequently happens when using action_params in the wrong scope. The symptom can be using action_params in an Inline, and seeing blank values.

The Fix: Convert action_params to variables, then use the variables throughout the page, and this problem will be completely avoided. This very simple process allows you to display and re-use the variable throughout a page. Done like so:

```
Var:'AddUserLastname' = (Action_Param:'UserLastname');
Var:'AddUserFirstname' = (Action_Param:'UserFirstname');
```

Note the variables have *somewhat different, but related, names* than the action_param. This is intentional, and using this concept should be part of your naming scheme. To use them:

```
Inline: –Database=$xSiteDatabase, –Table=$xUsersTable,
       'UserLastName' = $AddUserLastname,
       'UserFirstName' = $AddUserFirstname,
       –Add;
```

8. The HTML is Invalid or CSS Hides the Output
Invalid HTML can actually hide results that have been correctly returned. Because the HTML is not properly structured, the browser may not render the page properly. **The Fix:** First: Look at the page source for obvious problems, which might include a missing quote on an href tag, unbalanced table tags, etc. Second, run the page through a validation checker. Also try viewing the page in a different browser. Some browsers are more tolerant of errors than others. (See "Resources" and bad.html).

9. Session-related Problems
These might include the [Session_Start] not being called, the wrong session name, the session being expired, the session variable value being overwritten, or a proxy server interfering with serving unique pages. **The Fix:** Use the session tags to output session values at top and bottom of the page, and compare the ID displayed with the ID on the next page. The ID should be the same. Also compare the [Session_Result] to see if it is "new" or "load."

10. Webserver Caching
Webservers will cache files, sometimes even when they are told not to. This can lead to completely misleading conclusions for the developer, making you think that something did not work when in fact the newly-changed file was not even read! **The Fix:** Delete the file from the webserver, refresh your browser (you will get a "not found" error), then reload the file a second time. This almost always forces a new lookup. Quite annoying!

11. Browser Caching
Modern browsers cache lots of content. To test if your browser is aggressively cacheing a page, enter a simple string or the server time to be output, like this:

```
'238: Hey are we being cached?\n';
```

or

```
'240: The current time is: ' (Server_Time) '\n';
```

If it shows up, you have a "fresh" copy of the page.

12. No Permission – Databases
The infamous -9964 error, means Lasso Security is not set up correctly. Can be a bit of a bugger to troubleshoot. **The Fix:** Create a test page that simply returns a Found_Count, using the same inline parameters as are failing. If necessary, hard-code the variables. See dbconn_test.lasso in the sample files.

13. No Permission – File Operations
Another classic! There are a whole range of file-related security and permissions settings that need to be addressed and set up correctly for everything to work. **The Fix:** See Steve Piercy's super-duper file setup guide (link is in "Resources").

14. Redirects Failing
This could be likely a bad URL or bad logic that redirects multiple times to the same address, sometimes with parameters, sometimes without. **The Fix:** Use the [LI_RedirectURL] tag with debugging on to see the actual tag. This turns the URL into a clickable link, and allows you to pass parameters and arguments.

```
LI_URLRedirect: -Page='edit.lasso',-ExParams=('Aparam=Something'),-UseError='Y',-
Error=$ThisError,-UseArgs='Y';
```

15. File Uploads Failing
This is a particularly troublesome area, due to the number of failure points. At the heart of this is that Lasso accepts file uploads into a temporary folder, then performs actions on the uploaded file. At the end of this, the file is automatically deleted. So presto! There is no file, even though you know it was uploaded. This is a topic in itself, so if you have a particular question, bring in the problem and we can examine it.

16. E-mail Problems
Another area where there are many failure points. The Lasso errors database is essential to troubleshoot. You can add a "Sender" to the e-mail header which is not normally viewed in the e-mail message, but can be helpful to track down exactly which form or bit of code is sending the e-mail. To see the Sender on a successful message, look at the message header. Some sample code might look like this:

```
Email_Send:
       -Host=$SMTP,
       -From=$From,
       -To=$DeveloperEmail,
       -Subject=$EmailSubject,
       -Username=$AuthUsername,
       -Password=$AuthPassword,
       -ReplyTo=$From,
       -Sender=(($Domain)':/admin/frm_support'),
       -Body=(Include:($LibsPath)'email_support.txt'),
       -SimpleForm;
```

A few things to note here:

– nearly every parameter used here is a variable. Putting the data in variables allows you to disable the [Email_Send] tag, and output the values to the screen for inspection as a troubleshooting tactic.

– the –Sender parameter is a hard-coded location of the page or unique identifier that is sending the e-mail. Make sure there is no space in the –Sender.

– the –SimpleForm parameter is optional, but handy; it will output all the form parameters used to generate the e-mail.

See the sample file email_test.lasso in the materials.

CODING PRACTICES THAT AVOID PROBLEMS

Use a Siteconfig File

The siteconfig file should contain information that is constant throughout a site. The siteconfig is typically called at the top of every Lasso file, so you know the variables contained within the siteconfig should always be available to that site. For example, a siteconfig could contain database connection information, e-mail server name, table names, filepaths, etc. Putting this information in one file and loading it into a variables means less failure points and only one place to fix if something needs to be changed. For example:

```
// Session Variables
Var:'xSessionTimeout' = 30,
       'xSessionName' = 'MySessionname',
       'xSessionAdminName' = 'SessionAdmin';
 // Define Databases & User Authentication
Var:'xSiteDatabase' = 'TheDatabase',
       'xSiteUsername' = 'MyUser',
       'xSitePassword' = 'MyTopsecretPW';
 // Table Names
 Var:'xHeirarchyTable' = 'my_heirarchy',
       'xUsersTable' = 'my_users',
       'xErrorsTable' = 'my_Errors';
```

Use Global Variables

If you are writing code that is intended to work with multiple sites, and the sites share some of the same resources (such as a mailserver), put the values of those shared resources in global variables, instead of hard-coding them. This makes it much easier to make a change. The Lasso documentation has examples.

Use a Protect Block when Necessary

A “protect block” is a way to capture errors that may be somehow unavoidable, but capture the condition of the page in a variable and then display that value. In this case we are using the Image tag to get info on a file, and if that fails, setting a handle variable called #HandleOutput. Outside of the protect block, we can then examine that value and set another variable for whether the file is good or not. This method allows us to control display of the error.

```
// This code traps bad files
// Initialize Vars
Local('BadFile') = boolean;
Local('HandleOutput') = null;

// Protect this block from error display
Protect;
       Debug;
              '705: Inside Protect Block<br>\n';
       /Debug;
       Local:'TheImage' = null;

       // This next line fails on bad file
       Local:'TheImage' = (Image:(#pathToImage), -info);
       Handle: !(#TheImage);
              #HandleOutput = 'BAD FILE FORMAT';
       /Handle;
/Protect;

 If:(#HandleOutput) == 'BAD FILE FORMAT';
       // Always log this error
```

```
        Log: $xLogFile;
               '720: Bad file format: ' (#this_FilePath) '\n';
        /Log;
        #BadFile = true;
    Else;
        #BadFile = false;
    /If;

    Debug;
        '730: BadFile = ' #BadFile '<br>\n';
    /Debug;
```

Use Custom Tags
Custom tags (CT) can help you be more consistent and compartmentalize to code. Build debugging into your tags. The [LI_URLRedirect] CT is an example of a tag that changes behavior depending on whether $xDebug is on or not.

Use a Consistent Naming Scheme
In our examples, we are using a simple variable called $xDebug as a toggle to turn on or off debug display. This could be set in a siteconfig file for the entire site, or turned on for a whole page, or just a small portion of a page. Variables starting with an “x” are sitewide variables. Using this naming convention, they can be excluded from debug dumps, because those are known-good values, don't change, and don't need to be listed in variable dumps.

Use Well-Formed HTML
Validate your pages with the W3C validator. You can decide the level to which you need to be compliant.

Test for Situations and Set Your Own Error or Result Codes
When doing database actions, file processes, or e-mail actions, be proactive. Check for a condition, and set a variable one way or the other.

```
// Update the record
Inline: $x_Sys, -SQL=$SQLUpdateSys;
        // If there is an error, dump out error 1012
        // Otherwise dump out error 1014
        If: (Error_CurrentError) != (Error_NoError);
                Var:'Err' = '1012';
                Var:'Option' = (Error_CurrentError);
        Else;
                Var:'Err' = '1014';
        /If;
        Debug;
                '55: Err = ' $Err '<br>\n';
                '55: Option = ' $Option '<br>\n';
                '55: Error_CurrentError = ' (Error_CurrentError) '<br>\n';
                '55: SQLUpdateSys = ' $SQLUpdateSys '<br>\n';
        /Debug;
 /Inline;
```

As mentioned before, the “55:” is simply a marker to where you are in a particular page, and should be changed to indicate the location in your file.

Use a Framework
If it is possible, consider using a framework like Pageblocks, knop, or LassoFusebox. Any of these have debugging built-in as part of the system. See “References” for links. Also explore Jason Huck’s excellent error system (see “Resources”).

TOOLS AVAILABLE

Lasso Error Queue – /siteadmn.lassoapp → Utility → Errors. Used in conjunction with Log tags, you can capture, log, and view various levels of errors.
Variable dump – Displays values of all variables, can be tailored to the user's needs.
The file vardump.lasso is included in the roundtable materials.

CSS class – Use a CSS stylesheet class reserved for error output.
A sample stylesheet debug.css with a class for debug output is included in the roundtable materials.
HTML validation – Use the W3C validator to make sure your page is reasonably compliant
<http://validator.w3.org/>
BBEdit syntax coloring module (Mac only) – helps avoid dumb syntax mistakes
Lasso Studio for Eclipse – Offers advanced features not found in other products, including code folding, matching conditionals, and other cool features. Requires Eclipse (free) to run. Warning: Eclipse is not for the faint-of-heart!
<http://www.lassosoft.com/>
<http://www.eclipse.org/>

RESOURCES

Jason Huck's Error Page – Contains a methodology and some interesting techniques to trap and manage errors
<http://devblog.jasonhuck.com/2007/12/31/error-management-techniques-for-lasso/>
Tagswap – Essential site for the Lasso community's open-source tags <http://tagswap.net/>
Lasso Online Tag Reference – <http://reference.lassosoft.com/>
Steve Piercy's Super Duper File Tag Permissions Page – <http://stevepiercy.com/lasso_stuff/file_perms.lasso>
Pageblocks Framework – <http://www.pageblocks.org/>
knop Framework – <http://montania.se/projects/knop/>
LassoFuseBox Framework

SAMPLE FILES

bad.html – Example of a poorly-structured HTML file
dbconn_test.lasso – Test file to check database connection
debug.css – Sample stylesheet with debug class
debug.lasso – Debug container tag that formats debug output
email_test.lasso – Test file to check e-mail sending
LI_URLRedirect.lasso – Custom tag for redirection
siteconfig.lasso – Sample siteconfig
vardump.lasso – Variable dump file