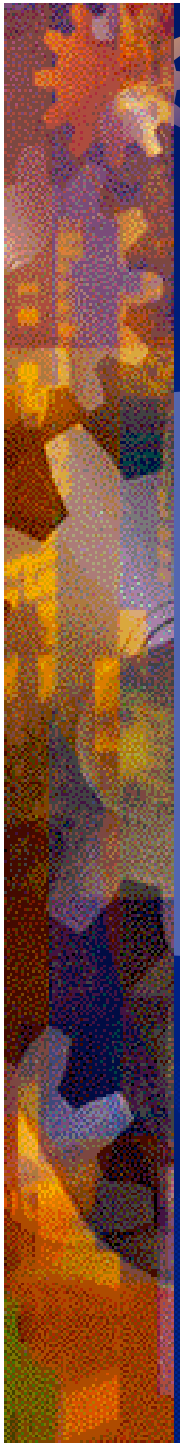
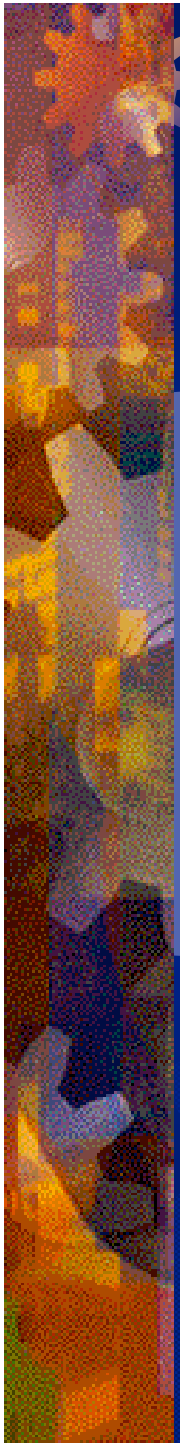


Creating Custom LDML Tags

Presented by:
Bil Corry
Lasso.Biz

LPA Lasso Partner Alliance
blueworld

LassoWARE Solutions Partner



Your life for the next 1.5 hours

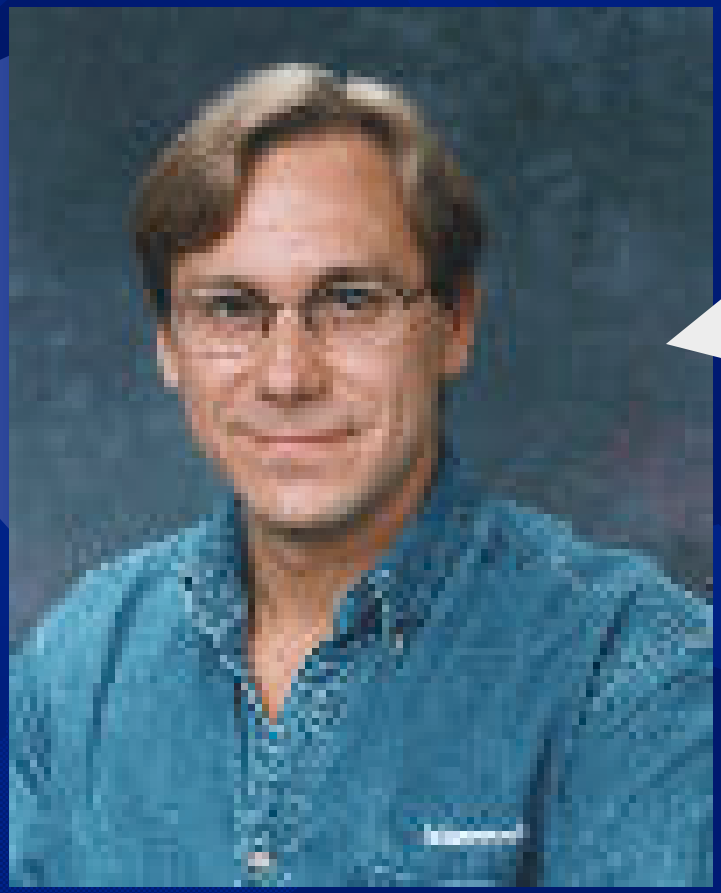
- Who is that Bil guy with just one L?
- What are custom LDML tags?
- Why should you use custom LDML tags?
- When should you use custom LDML tags?
- How do you use custom LDML tags?
- How do you create custom LDML tags?
- How do you share custom LDML tags?
- Power of Custom LDML Tags: AutoValidate
- Q & A
- Autograph signing

You can ask questions at any time!

The Bil with one L

- Lasso developer since v2.0
- Website developer since 1995
- Pig out food is Pizza
- Member of Lasso Partner Alliance
- LassoWare Solutions Provider
- Degree in Computer Science, Minor in Women's Studies
- Voted "Most Sexy" on LassoTalk

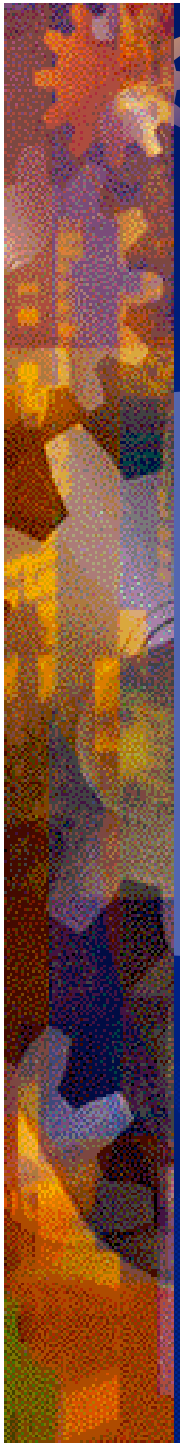
The Bill with two Ls



Not me!

What are custom LDML tags?

- New tags defined and created in LDML
- They can be used just like any built-in Lasso tag.
- If you can program in Lasso you can extended its functionality!
- Other two ways to add new tags, LCAPI and LJAPI (not covered today)



Why should you use custom LDML tags? (i.e. why should you listen to this presentation and not doze off?)

- Code abstraction
- Code re-use
- Code automation
- = LESS WORK MORE PLAY

Code abstraction

- Take complicated code and condense it into a simple tag or group of tags
- Example: [Email_Send] is over 1100 lines of code to send email
- Other developers don’t need to know how it works, just how to use it.

Code re-use

- Write a thousand-line custom tag once and re-use it infinitely
- Maintain the code in one location
- Debug once, use everywhere!

- Write a thousand-line custom tag once and re-use it infinitely
- Maintain the code in one location
- Debug once, use everywhere!

Code automation

- Have Lasso perform tasks so that you don’t have to.
- Less mistakes, less coding, less debugging
- Work smarter, not harder
- Example: AutoValidate

When should you use custom LDML tags?

- Not an exact science!
- Things to consider:
  - Scope of the custom tag
  - Redundancy of the custom tag
  - Portability of the custom tag

Scope of the custom tag

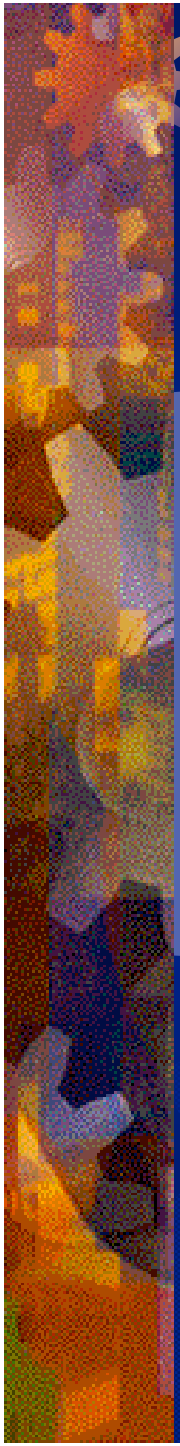
- Avoid tags that do too much or too little
  - Example: Too much = [myentirepage]
  - Example: Too little = <[h][t][m][l]>

Redundancy of the custom tag

- Custom tags should reduce your coding not duplicate existing tags
- Example: [todays_date] returns date_getcurrentdate. Could just as easily have done [date_getcurrentdate]

Portability of the custom tag

- Custom solutions that rely on custom tags must be able to port easily. This may be a problem when moving to a shared hosting solution.
- Know where and how your solutions will be hosted.



How do you use custom LDML tags?

- Custom tags are used identically as the built-in tags, the only difference is you must first define the custom tag before it can be used.
- To define a custom tag so that you can use it, you must either define it at the page level or define it globally.

Page-level Custom Tags

- Place the [define_tag] code on the page above where you are going to use the custom tag.
- Requires that the custom tag be defined on each page that it is used.
- Works in shared environments where you can’t add global custom tags

Global Custom Tags

- Place the [define_tag] code in its own .Lasso file and place the file in the LassoStartup folder. Restart LassoService to load.
- I use the naming convention TagName.Lasso
- The custom tag will be loaded at startup and may be used on any page site-wide (just like the built-in tags).
- You should debug as a page-level custom tag first, otherwise every change to the tag will require a LassoService restart.
- Will not work in shared environments if you can’t add global custom tags

How do you create custom LDML tags?

- Three types of custom LDML tags:
  - Substitution
    - [someTag] substitutes a value into page
  - Container (LP6 Only)
    - [someTag] … [/someTag]
  - Types
    - [var:'myVar' = myCustomType]
    - [$myVar->memberTag]
- Will be focusing on LP6, my paper focuses on LP5

FYI: Naming Conventions

- Custom tags can be named with any combination of letters, numbers and underscores.
- Tags beginning with an underscore are reserved for Blue World's use.
- Blue World recommends naming custom tags beginning with a short author identifier, followed by an underscore, then the name of the custom tag.
  - Example: [bc_myCustomTag]

FYI: Variables vs. Locals

- As you will see, any variables declared within the custom tag becomes available to the calling page
- Use locals for any values you don't want to pass back to the calling page or any variables on the calling page you don't want to accidentally modify

FYI: Documentation within tags

- Be careful to not do this (infinite loop):

```
[define_tag:'myTag']
   [myTag] version 1.0
   [myTag] will return blah to user
   [return:'blah']
[/define_tag]
```

- Instead, use HTML comments or [noprocess]:

```
[define_tag:'myTag']
   <!-- [myTag] version 1.0
   [myTag] will return blah to user -->
   [return:'blah']
[/define_tag]
```

Substitution Tags

- Three steps to success:
  - Get the parameters that were passed in
  - Process
  - Pass the result(s) back out
- Your tag doesn't have to receive parameters or return results.

Passing parameters in

- Two ways to pass parameters in:
  - Explicitly when the tag is called
    - `[myTag: -someParam = 'test123']`
  - Implicitly using a variable
    - `[var:'someParam' = 'test123']`
    - `[myTag]`
- Or you don't have to pass a parameter at all.

Passing the result(s) back out

- Three ways to pass the result(s) back out:
  - Return a single value that is “substituted” into the calling page (can be of type array or map for more than one value)
  - Return a variable or variables
  - Return as if it was an [include] using undocumented –autooutput (tricky advanced feature)
- Or you don’t have to pass a result back at all.

Skeleton Substitution Tag

- The absolute minimum for a custom substitution tag:
  - `[define_tag:'myAlert']`
  - `[/define_tag]`
- Doesn't actually do anything, but would be used like this:
  - `[myAlert]`

Simple tag example

- No parameters passed in, no results passed out:

```
[define_tag:'myAlert']
  [email_send:
   -host='smtp.mindio.com',
   -to='bil@mindio.com',
   -from='lassoServer@mindio.com',
   -subject='myALERT!',
   -body='myAlert was triggered!']
[/define_tag]
```

Simple tag example cont

- Instead of this:
  - `[email_send:`
  - `-host='smtp.mindio.com',`
  - `-to='bil@mindio.com',`
  - `-from='lassoServer@mindio.com',`
  - `-subject='myALERT!',`
  - `-body='myAlert was triggered!']`
- We now do this:
  - `[myAlert]`

Passing a parameter into a tag

- Method #1: With explicit parameter
  - `[myAlert: 'Jim is not wearing the same shirt!']`
- Method #2: With implicit parameter (variable)
  - `[var: 'myAlert_AlertMessage' = 'Jim is not wearing the same shirt!']`
  - `[myAlert]`

Passing a parameter into a tag cont

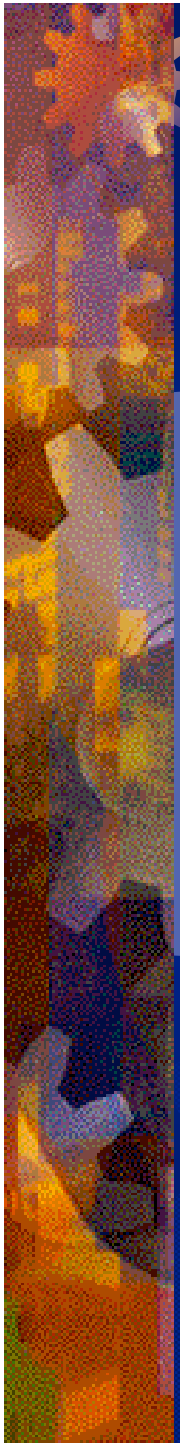
- Method #1: With explicit parameter

```
[define_tag:'myAlert': -required='alert']
   [email_send:
   -host='smtp.mindio.com',
   -to='bil@mindio.com',
   -from='lassoServer@mindio.com',
   -subject='myALERT!',
   -body='myAlert was triggered!  The alert
was ' + #alert]
[/define_tag]
```

Passing a parameter into a tag cont

- Method #2: With implicit parameter

```
[define_tag:'myAlert']
  [email_send:
  -host='smtp.mindio.com',
  -to='bil@mindio.com',
  -from='lassoServer@mindio.com',
  -subject='myALERT!',
  -body='myAlert was triggered!  The
alert was ' + $myAlert_alertMessage]
[/define_tag]
```



More on explicit parameters

- Explicit parameters can be accessed easily using –required or –optional
- Lasso will automatically create locals of your –required or –optional parameters
- Lasso will throw an error if a –required parameter is missing
- Lasso first matches all named parameters to their –required and –optional, then matches all unnamed parameters in order to the specified –required then -optional

Example of –required and -optional

- `[define_tag: 'myAlert', -required='name', -required='email', -optional='subject']`
- Will create two locals within the custom tag, #name and #email, and possibly a third local, #subject.
- Optional parameters must be tested for:
  - `[if: !(local_defined:'subject')]`
  - `  [local:'subject'='']`
  - `[/if]`

Examples of how Lasso assigns –required and -optional

- `[define_tag: 'myAlert',-required='name',`
  `-required='email',`
  `-optional='subject']`
- **Will Work:**
- `[myAlert: 'Bil Corry','bil@bilcorry.com']`
- `[myAlert:`
  `'Bil Corry','bil@bilcorry.com','test']`
- `[myAlert: -name='Bil Corry',`
  `-email='bil@bilcorry.com',-subject='test']`
- `[myAlert: -email='bil@bilcorry.com','Bil Corry']`
- **Won't Work:**
- `[myAlert: -email='bil@bilcorry.com','test','Bil Corry']`

Recap

- So far we now know how to create a basic substitution tag and how to pass parameters into it. Remember, there are three types of tags: substitution, container and type. Right now we're only looking at substitution.
- Now we're going to focus on how to get the results back out of custom substitution tags.

Passing the result(s) back out

- Three ways to pass the result(s) back out:
  - Return a single value that is “substituted” into the calling page (can be of type array or map for more than one value)
  - Return a variable or variables
  - Return as if it was an [include] using undocumented –autooutput (tricky advanced feature)
- Or you don’t have to pass a result back at all.

[Return] Single Value

- [return] is entirely straightforward, simply call [return] from within a custom tag to immediately abort the tag and return a value

- `[return: 15]` returns the integer 15

- `[return: 'doghouse']` returns the string 'doghouse'

- `[return: (array: 15,'doghouse')]` returns an array containing 15 and 'doghouse'

[Return] Example

- A tag defined as this:

```
[define_tag:'html']
[return:'html']
[/define_tag]
```

- Can be called like this:

```
<[html]>
```

- And becomes this:

```
<html>
```

Passing Results Using a Variable

- Very easy for beginners since similar to how you construct a Lasso page
- Can pass as many variables as needed
- Harder to use in a multi-developer environment, not self-contained

Variable Example

- A tag defined as this:

```
[define_tag:'html']
[var:'html'='html']
[/define_tag]
```

- Can be called like this:

```
[html]<[$html]>
```

- And becomes this:

```
<html>
```

Passing Results Using -AutoOutput

- Custom tags, just like built-in tags, are automatically HTML encoded unless another encoding scheme is defined.
- -AutoOutput allows you to override the auto HTML encoding.
- It's undocumented, so it may stop working in future versions of Lasso, but it does work in LP5 and LP6.

Encoding Problem Example

- A tag defined as this:

```
[define_tag:'html']
[return:'<html>']
[/define_tag]
```

- Can be called like this:

```
[html]
```

- And becomes this:

```
&#60;html&#62;
```

- Instead of this:

```
<html>
```

Two solutions:

- Call the tag like this:

```
[html: -encodenone]
```

And becomes this:

```
<html>
```

- Use -AutoOutput

-AutoOutput

- A tag defined as this:

```
[define_tag:'html',-AutoOutput]
<html>[/define_tag]
```

- Can be called like this:

```
[html]
```

- And becomes this:

```
<html>
```

-AutoOutput Gotchas

- Can’t use [return] inside of a custom tag that uses –AutoOutput
- Every whitespace (returns, spaces, tabs) and any HTML comments will be passed back to the calling page – everything between [define_tag] and [/define_tag]

Advance topics to explore

- [Precondition]
  - Execute code before the tag
- [Postcondition]
  - Execute code after the tag
- -Priority
  - Sets up a chain of tags or optionally replaces tags
- -Criteria
  - Specify a condition to be met, if fails, next tag in chain is called
- -Async
  - Run the tag asynchronously

Recap: How do you create custom LDML tags?

- Three types of custom LDML tags:
  - Substitution
    - [someTag] substitutes a value into page
  - Container (LP6 Only)
    - [someTag] … [/someTag]
  - Types
    - [var:'myVar' = myCustomType]
    - [$myVar->memberTag]

Container tags

- Allow you to create tags using an opening and closing tag.
- LP6 only
- Example:

```
[url]www.lasso.biz[/url]
```

- Becomes:

```
<a href="http://www.lasso.biz">www.lasso.biz</a>
```

Container Tags Cont

- By default, no encoding is performed on container tags (similar to substitution tags specifying –AutoOutput)
- Use –container to specify it's a container tag
- Lasso will require a closing tag
- [run_children] will return everything between your opening and closing tags, think of it as an include

Container Tags: Example [url]www.somesite.com[/url]

- Define the tag like this:

```
[define_tag:'url',-container]
[return:'<a href="http://' + run_children + '>' +
  run_children + '</a>']
[/define_tag]
```

- Using it like this:

```
[url]www.lasso.biz[/url]
```

- Becomes:

```
<a href="http://www.lasso.biz">www.lasso.biz</a>
```

Advance topics to explore

- Sub-container tags, like [loop_count]

Custom types

- Allows Lasso to store and manipulate objects using member tags.
- Probably the toughest to understand and code for (at least it was for me).
- It is created by defining the type, then the member tags.

Skeleton custom type

```
[define_type:'myType']
   [define_tag:'memberTag']
   [/define_tag]
   [define_tag:'anotherMemberTag']
   [/define_tag]
[/define_type]
```

- To use the above:

```
[var:'test'=(myType)]
[$test->memberTag]
[$test->anotherMemberTag]
```

Example

```
[define_type:'customer']
  [local:'name'='']
  [local:'email'='']
  [define_tag:'setName',-require='name']
     [self->'name'=#name]
  [/define_tag]
  [define_tag:'getName']
     [return: self->'name']
  [/define_tag]
[/define_type]
```

Example cont

- To use the custom tag:

```
[var:'test'=(customer)]
[$test->setName:'Bil']
[$test->getName]
```

- Outputs

```
Bil
```

Important things to know

- [self->'localVar'] refers to the local variable 'localVar' for that instance of the type
- The local variables are persistent for the life of the variable.

Custom types allow for slick data abstraction

- Using custom types, you can then have the custom type do your database queries, abstracting it from the Lasso page.
- Makes it easier to employ non-technical developers who perhaps don’t know anything about databases

Example of data abstraction

```
[var:'user' =(userType)]
[while: !$user->auth]
   [auth]
   [$user->setUsername: (client_username)]
   [$user->setPassword: (client_password)]
   <!-- authenticate user, make DB call to lookup user
   info -->
   [$user->authenticate]
[/while]
Welcome [$user->getName]!<br>
<br>
You have [$user->getCredit] credits left to spend.
```

Advance topics to explore

- There's a lot more you can do with Custom Types.
- I'd recommend reading the entire chapter on Custom Types in the Extending Lasso Guide.

How do you share custom LDML tags?

- Blue World maintains a directory of resources here:
  `http://www.blueworld.com/blueworld/products/lassosolutions.html`
- Two popular sites: LassoScripts.com and LassoTracker.com
- You can announce on LassoTalk

The power of custom LDML tags: AutoValidate

- AutoValidate is a two-tag custom LDML system for validating HTML forms, AutoValidate_Encode and AutoValidate_Decode
- Lasso does all the work so that you don’t have to.
- Great example of a RAD tool.

How it works

- You create a HTML form with the field names crafted with how you want it validated.
- AutoValidate_Encode processes the forms before sending it on to the user, changing the field names to their “normal” name and inserts an encrypted hidden HTML field containing all the validation information.
- On the response page, AutoValidate_Decode decrypts the hidden field, validates all the fields and alerts you to any validation errors.

An example of the form

- The developer codes the page like this:

```
[handle]
[autoValidate_encode]
[/handle]

<form ...>
   <input type="text"
   name="vf.username.an.req.len<=24">
   <input type="text"
   name="vf.password.req.len>=6.len<=12.an.nospc">
   <input type="hidden" name="region"
   value="northwest">
</form>
```

An example of the form

- The user sees the page like this:

```
<form ...>
  <input type="hidden" name="vfdata"
value="DAFD...blowfish encrypted text...4A4D">
  <input type="text" name="username">
  <input type="text" name="password">
  <input type="hidden" name="region"
  value="northwest">
</form>
```

An example of the response

- The response page looks like:

```
[AutoValidate_Decode]
[if: $vferrmsg->size > 0]
<p class="mystyle">[$vferrmsg]</p>
[else]
No Errors
[/if]
```

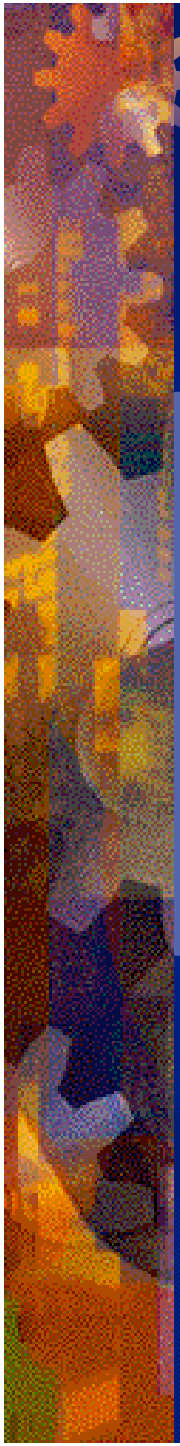
An example of the response

- The response page to the user looks like:

The field **User Name** cannot be empty.
The field **Password** must be at least 6 characters long.
The field **Password** must be comprised of only alphanumeric characters (a...z, A...Z, 0...9).

Other features

- Support for user-entered dates, such as 1-30-2003 or 1/30/2003, and Euro dates such as 30-1-2003 or 30/1/2003
- Auto encrypt all hidden fields
- Support for custom validation
- Support for “friendly names” for the user instead of field names (such as “First Name” instead of “name_first”)
- Many other features
- All done with custom LDML tags!
- Beta version available here: www.lassoware.com



Thank You!

- Questions and Answers
- Autographs and Pictures

Bil Corry

bil@lasso.biz

A copy of this presentation will be made available at: www.lasso.biz