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

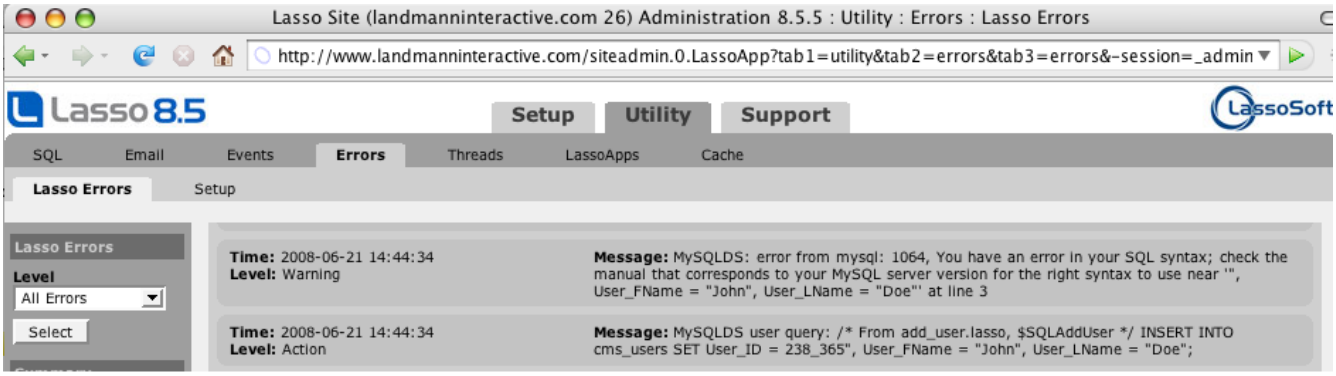
This can be easily diagnosed by writing the query as a variable, outputting the variable to the page, copying the query from the page to a database client, and running the query. You should get immediate feedback about what is wrong. Common problems include an extra comma after fieldnames, mismatched 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