MOUSE - Application Development Framework

Liomys salvini (Spiny Pocket Mouse)

MOUSE is an open source application development framework created by **Black Hill Systems, LLC** engineers for the rapid development of enterprise software. The framework currently supports:

- User and User Group creation, authentication, and authorization
- Page Creation (Lists, Viewers, Editors) and Handling (JSON Read Services, Printing, Paging)
- Environment Configuration
- Transaction Auditing
- Messaging

MOUSE utilizes proven technologies and has been developed by engineers whose experience includes decades of cumulative experience working on enterprise systems with demanding security requirements. The Framework allows developers to quickly, securely, and cleanly create sophisticated web applications without the need to understand sophisticated programming languages or to find and finance a team of engineers to build from a clean start.

Additional capabilities currently being developed include:

- Enterprise Single Sign On
- Global Search
- Subscriptions
- Workflow Customization

MOUSE has been used in the creation of a diverse number of enterprise applications including a laboratory information management system, a travel agency system, and an entertainment content management system. The concept is loosely based on the Perl MOOSE module and seeks to provide a similar meta-object syntax to support an automated MVC (model-view-controller) implementation.

- Log in [1] to post comments

Framework Installation

[Note: these instructions assume you are using the Apache Web Server. If you are using a different web server, you will need to customize the .htaccess directives to accommodate it.]

Obtain the MOUSE Framework and extract its contents. The FRAMEWORK folder contains the MOUSE infrastructure; the CLEAN_APP folder contains the (mostly empty) folders that you customize for your own purposes. Place the FRAMEWORK and CLEAN_APP contents within your web server's folder so that the final structure looks like:

app-bin -- folder where you will place executable binaries (if any)
app-bkg -- folder where you will place background processes (if any)
app-lib -- folder where you will write your application-specific user interface
app-schema -- folder where you will place your application-specific SQL
cgi-bin
htdocs -- folder where you will write javascript code, place images, etc.
lib
logs
open-bin
perl -- folder where you will write background perl scripts (if any)
schema
htaccess.txt -- change this file to .htaccess on your web server
eula.txt
index.htm

Change the filename of htaccess.txt to .htaccess. Create a new file within the cgi-bin directory called config.php and add the following contents to it:

```
;<?php die();
/*
[db]
conn = DBI:mysql:dbtable:127.0.0.1:3306
db = dbtable
host = 127.0.0.1
pwd = dbpassword
user = dbuser
[construct_db]
conn = DBI:mysql:dbtable:127.0.0.1:3306
db = dbtable
host = 127.0.0.1
pwd = dbpassword
user = dbuser
[lib]
dir = /home/ubuntu/workspace/app-lib
frmwrk = /home/ubuntu/workspace/lib
tools = /home/ubuntu/workspace/app-bkg/bin/
*/
?>
```

Update the contents of this file to reflect your system's configuration. Make sure the permissions on the config.php file are Admin Read Only (chmod 400 config.php). Depending on your configuration, you may need to set this to Admin and Group Read Only.

Import the clean_structure_and_data.sql, lms_functions.sql, and lms_sprocs.sql files into your MySQL instance. Then, do the same for any SQL files contained within the version directories (e.g., v2.0).

The default username and password for the administrative account are "administrator" and "password", respectively.

- Log in [2] to post comments

Global Variables and Functions

Global Variables

$g_obj contains the instantiated Page Object. The *$g_obj* global variable is only used by to add Errors and Messages to the Page Object and must never be exposed to unvalidated user input.

$g_user_object contains the instantiated User Object. The *$g_user_object* global variable is only used to hold user information and must never be exposed to user input.

$DEBUG is a global variable which, when set to 1, emits debugging text to the screen.

Global Functions

lms_get_env_var($key:string) [pull the dot-delimited environment variable from config.php]

lms_get_page_object($appname:string) [retrieves the Page Object if you need a clean copy instead of using the populated $g_obj]

my_redirect($resource:string) [adds an immediate redirect to a page on the local server; useful in pre_render if you need to take someone to a new page after form entry]

lms_get_groups() [returns an array of hashes holding information about all the Luminous user groups]

lms_mimetype($extension:string, $mimetype:string) [given a file extension or a mimetype, return a hash containing both the extension and the mimetype]

lms_get_menu_data() [retrieve all of the information used to generate the menus]

lms_array_match($patterns:array, $target:string, $optional_replace_value:string) [given an array of regex patterns, see if a match occurs in the target; optionally replace the pattern]

hash_search_array($needle:array of hashes, $haystack:array of array of hashes) [search an array with an array of hashes to get the array key for a matching hash in the haystack]

lms_gen_rand_str($length:integer) [generate a random alphanumeric of specific length]

lms_crypt($s:string, $salt:string) [produce a salted crypto string]

lms_uuid() [generates a version 4 Universally Unique Identifier]

- Log in [3] to post comments

URI Modifiers

json=1

When “&json=1” is added to a URL whose code behind executes a Mouse "read_" or "list_" Reflection method, a JSON string representing the corresponding data is presented instead of the usual user interface.

csv=1

When “&csv=1” is added to a URL whose code behind executes a Mouse "read_" or "list_" Reflection method, a pipe ("|") delimited string representing the corresponding data is presented instead of the usual user interface.

print=1

When “&print=1” is added to a URL whose code behind executes a Mouse "read_" or "list_" Reflection method, data paging is disabled in order to permit visualization of all data. Note that this should be used wisely as slow or very large queries could adversely affect rendering of the page or availability of the database. A SQL LIMIT should be used in the object join if this is a possibility.

- Log in [4] to post comments

Core Classes

- Log in [5] to post comments

DB.php

This class extends the Mouse core class. It is extended by any data class object.

It utilizes the mysqli extension for the MySQL Database. It obtains its configuration details from the config.php file and the php.ini configuration file.

$this->mysqli = new mysqli(ini_get("mysqli.default_host"), ini_get("mysqli.default_user"), ini_get("mysqli.default_pw"), lms_get_env_val('db.db'));

Key public methods include:

db_prepare($sql, $bndprms)
db_write($stmt)
commit() --optional
rollback() --optional

- Log in [6] to post comments

FilterPage.php

This class extends the Page core class.

Default attributes are set, but may be overridden by classes extending this class.

$this->total_pages = 0;
$this->page_size = 10;
$this->current_page = 1;
$this->total_rows = 0; --override required

To utilize the features of this class, the total_rows attribute is the only required override. There are no key public methods.

- Log in [7] to post comments

Mouse.php

This class is only extended by the DB core class. It is based on a limited implementation of Perl's Moose module. As such, it utilizes Reflection to catch methods which are undefined in extending classes.

The only public methods are:

has($attributes) --defines the attributes of the Mouse object
get_attribute_names() --gets the "name" of all attributes of the Mouse object

Reflection Methods

Mouse will interpret several Reflection methods. The first set of methods act upon the values of the Mouse Object Attribute; the second set of methods perform database functions; and the last method may be used to modify the Options of a particular Mouse Object Attribute.

Value Methods:
get_ --the Mouse Reader (getter) method
set_ --the Mouse Writer (setter) method
has_ --the Mouse Predicate (isset) method
clear_ --the Mouse Clearer (unset) method

DB Methods:
count_ --count the number of list results returned from the database for the Mouse Object
list_ --select multiple results from the database representing the Mouse Object
read_ --select a single result from the database representing the Mouse Object
delete_ --delete the current Mouse Object from the database
save_ --insert or update the current Mouse Object in the database

Attribute Methods:
modify_ --the Mouse Option editor

Reflection Methods Defined

Reader: For each attribute passed into Mouse via a "has" method, a getter method is created. For example, if "seq" is the attribute passed in, "get_seq" will be generated. This may be overridden by creating a "get_seq" method within the calling Object or within an Object which extends the calling Object.

Writer: For each attribute passed into Mouse via a "has" method, a setter method is created. For example, if "seq" is the attribute passed in, "set_seq" will be generated. This method will not be usable if the attribute's "is" option is set to "ro" (read only). This method may be overridden by

creating a "set_seq" method within the calling Object or within an Object which extends the calling Object.

Predicate: For each attribute passed into Mouse via a "has" method, a predicate method is created. For example, if "seq" is the attribute passed in, "has_seq" will be generated. This may be overridden by creating a "has_seq" method within the calling Object or within an Object which extends the calling Object.

Clearer: For each attribute passed into Mouse via a "has" method, a clearer method is created. For example, if "seq" is the attribute passed in, "clear_seq" will be generated. This method will not be usable if the attribute's "is" option is set to "ro" (read only). This method may be overridden by creating a "clear_seq" method within the calling Object or within an Object which extends the calling Object.

Count: This generates the count of objects returned by the DB List Reader.

DB List Reader: This is a generic list collection obtained from dynamically generated SQL. It is optionally used in conjunction with the auto_render_list Render helper functions. For example, if "list_lms_audit()" is called, Mouse interprets this to be a "list" call of the "lms_audit" database table.

DB Reader: This is a single row DB reader which dynamically generates SQL based upon the Mouse Object and each Attribute's options. It is optionally used in conjunction with the auto_render_view Render helper functions. For example, if "read_lms_audit()" is called, Mouse interprets this to be a "read" of the "lms_audit" database table.

DB Delete: This is a single row DB deletion method for the Mouse object.

DB Save: This is a single row DB save method for the Mouse object.

Modify: In order to modify a Mouse Attribute after it has been instantiated, a variable can be set and passed via the "modify_" Reflection method. For example, in order to add or change the "link" option of the "subject" attribute:

```
$subject->link = function($data) {
  //message_id is in the 0th array position of this MOUSE object
  return('default.php?appname=message-view&message_id=' . $data[0]);};
$this->data->modify_subject($subject);
```

Attribute Options

Each Attribute in a Mouse Object has multiple Options which control how an Attribute is rendered and its specific requirements.

Main Options: *isa, is, name, init, trigger, doc, join, control, required*

List View Relevant Option Only: *link, order*

View and Edit Relevant Options Only: *row, col*

Edit Relevant Option Only: *primary_key*

List and View Relevant Options Only: *style*

Attribute Options Defined

"name": A name to report during errors, to set labels, and to set column names.

"is": Each attribute which is passed into Mouse via a "has" method will be considered readable and writable ("rw") unless otherwise specified with an "ro" (read only).

"isa": A valid PHP Type or a Mouse SubType (eg. boolean, integer, float, string, array, Str32, username, etc.). This is the only required option an Attribute must express.

"init": Default values for attributes may be assigned using the "init" option. If no value is passed in, the default value will be null. If an empty array is desired as the default, the init value will need to be "array()" AND/OR the "isa" value must be 'array'.

"required": If a particular Attribute is required (especially if the DB field cannot be null), this flag will be set to true. If a required Attribute isn't set, an error will be returned to the Page class.

"trigger": If you need to have a method execute any time an attribute's value is set (ie. when the "set_XXX" method is called), provide the name of the method you need to use to the "trigger" option. The "trigger" method will execute AFTER the "set_XXX" method is called. This method needs to be located within the calling Object. This does not occur when the attribute's default value is set via the "init" option.

"doc": Any string you may want to pass along with the attribute. For example, text describing the attribute. This option does not have any programmatic functionality.

"link": An optional Closure Function which permits the rendering of links on associated Attributes of a list view. For example, the Attribute below uses a link function to create a link on the values of the Subject Attribute which point to the message-view page with the message_id equal to the first (or 0th) Mouse Attribute (which, in this case is message_id):

```
subject => array(name => 'Subject', isa => 'Str32',
  link => function($data) {
  return('default.php?appname=message-view&message_id=' . $data[0]);})
```

"primary_key": For Mouse objects which are going to be saved, the table's primary key needs to be identified (boolean).

"join": For some attributes, the actual value may not be what needs to be displayed. For example, a user's name will always be displayed instead of the user_id. In order to achieve this, a table and column join may be used:

```
join => array(table => 'lms_user',
  column => 'lms_user.username',
  where => 'lms_user.user_id = lms_audit.user_id',
  group => 'lms_user.username')
```

If a column => " ' ' " is provided, an empty value is returned for the row. This is useful if you want to show a button, image, or link without data being pulled from the database.
If a column => " " is provided, no value is returned. This is useful if you want to track a value or calculation within a row, but don't want to have the DB SQL query affected.

"order": If there is an "orderable" nature to the attribute, then it may be indicated here with "asc" or "ascending" for sorting A-Z or "desc" or "descending" for sorting Z-A. It is best to arrange the ordered columns at the end of the has() method or all columns will be ordered after it in the SQL call. For example, positioning the ordered "date_time" column before the "user_id" and then sorting by the "user_id" in the auto_render_list method will result in the username being sorted within the date_time sort (that is, ORDER BY date_time, username).

"row": The row that a particular attribute is to be rendered on. This allows for specific arranging of attributes.

"col": The column that a particular attribute is to be rendered on. This allows for specific arranging of attributes.

"control": An optional Closure Function allows the rendering of HTML controls on Edit pages.

```
control => function($args) use ($options) {
  render_multiselect("to_user_id[]", $_POST['to_user_id'], $options, "5");}
```

"style": An optional Closure Function for setting the style of the output results in a List or View page.

```
style => function($data) {
  //$data[5] is the 6th column (a date) of a list collection
  if ($data[5] == "0000-00-00 00:00:00") {
  return("font-weight:bold;");}}
```

Example usage of all Attributes for a single Element:

```
$this->has(array(
  element => array(
  name => 'element name', //the element label/title; leave empty to hide element; empty by default; optional
  is => 'rw', //read-only (ro) or read-writable (rw); 'rw' by default; optional
  isa => 'int' //a valid PHP Type or Mouse SubType; required
  init => 'initial value", //null by default; optional
  required => 0, //set to true (1) if the field cannot be null; 0 by default; optional
  trigger => 'method_name', //execute method_name() existing in this class; null by default; optional
  doc => 'Any text describing what this element is', //null by default; optional
  link => function($data) {return('default.php');}, //turn element output into an HTML link; null by default; optional
  primary_key => 0, //if this is a DB table key, set to true (1); 0 by default; optional
  join => array(table => 'table_name',
  column => 'table_name.column_name',
  where => 'table_name.id = other_table.id',
  group => 'column_name'), //Construct the DB JOIN for complex queries; optional
  order => 'ASC', //'ASC' or 'DESC' sort order for lists; 'ASC' by default; optional
  row => 1, //the row of a table an element will be rendered on; optional
  col => 1, //the column of a table an element will be rendered on; optional
  control => function($args) use ($optional) {do_some_work();}, //render the output as an HTML control; optional
  style => function($data) {do_some_work();}))) //render the output with function-defined CSS; optional
```

Example usage of relevant Attributes for a hypothetical Object:

```
$this->has(array(
  seq => array(isa => 'int'),
  audit_type => array(name => 'Audit Type', isa => 'Str32'),
  entry_type => array(name => 'Entry Type', isa => 'UCStr8'),
  date_time => array(name => 'Timestamp', isa => 'DateTime'),
  user_id => array(name => 'Username', isa => 'int'),
  modified_column => array(name => 'Modified Column', isa => 'Str32'),
  old_value => array(name => 'Old Value', isa => 'str'),
  new_value => array(name => 'New Value', isa => 'str')
));
```

Mouse SubTypes

Mouse SubTypes are children of the standard PHP Types. They can be used to construct commonly used patterns which conform to specific regular expressions (regexes) and post-setter manipulation (eg. capitalization, rounding, etc.).

Mouse SubTypes Defined

"isa": Each SubType must be based upon a valid PHP Type (Boolean, String, Integer, Float, or Array).

"regex": A regular expression used for validating the values passed in.

"where": A function used to modify the value prior to SETTING.

"message": A message to return if the regex doesn't match the value.

Example usage:

```
new SubType('UCStr8',
  array(isa => 'str',
  regex => '/^[A-Z0-9-!"#$%&\(\)*+,.\/:;=?@_~ ]{0,8}$/',
  where => function($val){return(strtoupper($val));},
  message => 'Values must be alphanumeric capitals with a maximum of 8 characters.'));
```

- Log in [8] to post comments

Page.php

This class is extended by any page class object and the FilterPage class directly.

The majority of its methods are overridden by the extending class. The only key public methods include:

addError($e)
addWarning($e)
addMessage($e)
addHeader($e)
addJavascript($e)

The execution sequence of methods which may be overridden is:

1) pre_render()
2) body_onload()
3) titleBar()
4) menuBar()
5) errorList()
6) warningList()
7) messageList()
8) render()
9) footer()

- Log in [9] to post comments

User.php

This file instantiates the User objects (UserGroup and User) and contains a number of non-class

utility methods. These public methods are generally used internally, but may be used externally.

lms_get_user_id()
lms_get_username()
lms_get_current_user()
lms_get_user_object()
lms_get_primary_group_name($user)

- Log in [10] to post comments

Utility Classes

- Log in [11] to post comments

Objects.php

This collection of functions acts to initiate the construction of objects and to regulate the associated privileges. URLs and redirects are managed here as well. It is called by the default files (cgi-bin/default.php and open-bin/default.php) and acts in turn to load the Core Classes.

There are no key public methods.

- Log in [12] to post comments

Render.php

This collection of functions acts to produce output in specific formats or control structures.

Render HTML Controls:

render_input($html_args:array)
render_textarea($html_args:array, $value:string)
render_popup($name:string, $selected_option:string, $options:array of hash, $null_option:boolean, $style_arg:string, $onChange:string, $disabled:boolean)
render_multiselect($name:string, $selected_options:array, $options:array of hash, $number_rows:string, $multiselect:boolean, $style:string, $onchange_args:string, $disable_control:boolean)

Render General Input and Output:

lms_print_r($variable:any) [print the objects in HTML format for debugging purposes]
lmsecho($value:string) [htmlentities a value]
decho($value:string) [debug echo]
null_float($value:any) [cast as a float or NULL]
escape_js($value:string) [replace ', \, and \\ in JS]
lms_trim($value:string) [strip escapes and spaces]

csv_trim($string:string) [strip spaces]
csv_encode($data:array) [render hash as pipe-delimited text]

Page Controls and Views:

set_form($action:string, $encoding:string)
render_delete_button($owner_id:integer, $appname:string, $disable:boolean) [show a delete button if the user has the priv]
render_save_button()
render_button_bar($width:string, $owner_id:integer, $appname:string, $disable:boolean) [if this is an object that is "owned" by a single user, pass that owner_id and the name of the app so that the system can determine if a delete button should be shown]
render_pagination_bar($current_page:integer, $total_pages:integer, $total_rows:integer, $goToPage:boolean, $showGoToPage:boolean)
auto_render_list($data:MouseObject, $list:array of hash, $printable:boolean, $links:array of hash, $title: string)
auto_render_view($data:MouseObject, $title:string, $width:string, $links:array of hash)

- Log in [13] to post comments

System.php

This group of methods acts to execute common non-class based functionality.

The key public methods include:

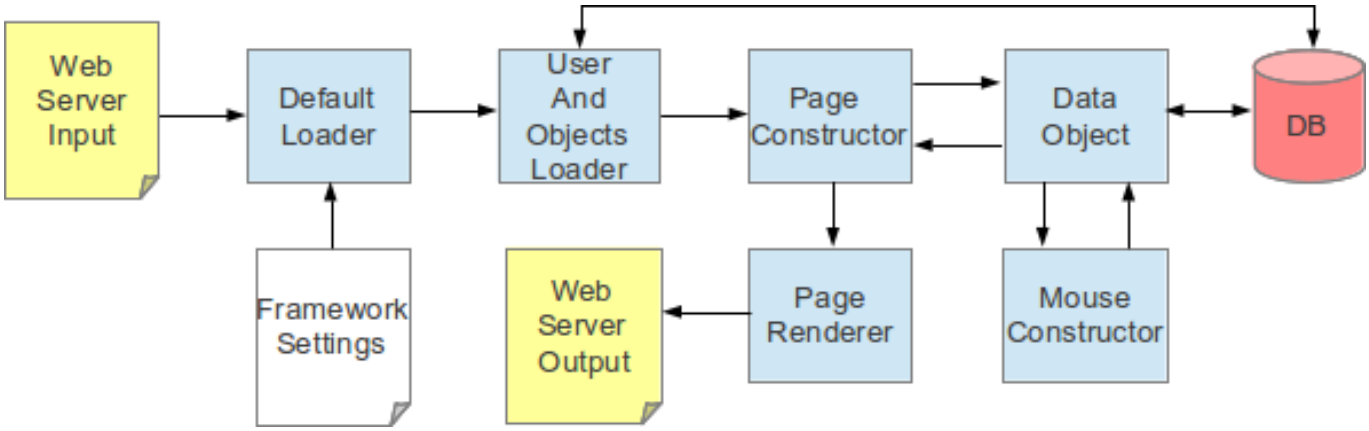
lms_get_groups() [returns all LMS groups]
lms_mimetype($extension:string, $mimetype:string) [returns either extension or mime type for a given $extension or $mimetype]
lms_get_menu_data() [returns the menu list to build the main menu]
lms_array_match($patterns:array, $target:string, $optional_replace_value:$string) [given an array of regex patterns, see if a match occurs in the target]
hash_search_array($needle:hash of arrays, $haystack:array) [search an array with a hash(array) to get the array key]
lms_gen_rand_str($length:integer)
lms_crypt($s:string, $salt:string)
lms_uuid()

- Log in [14] to post comments

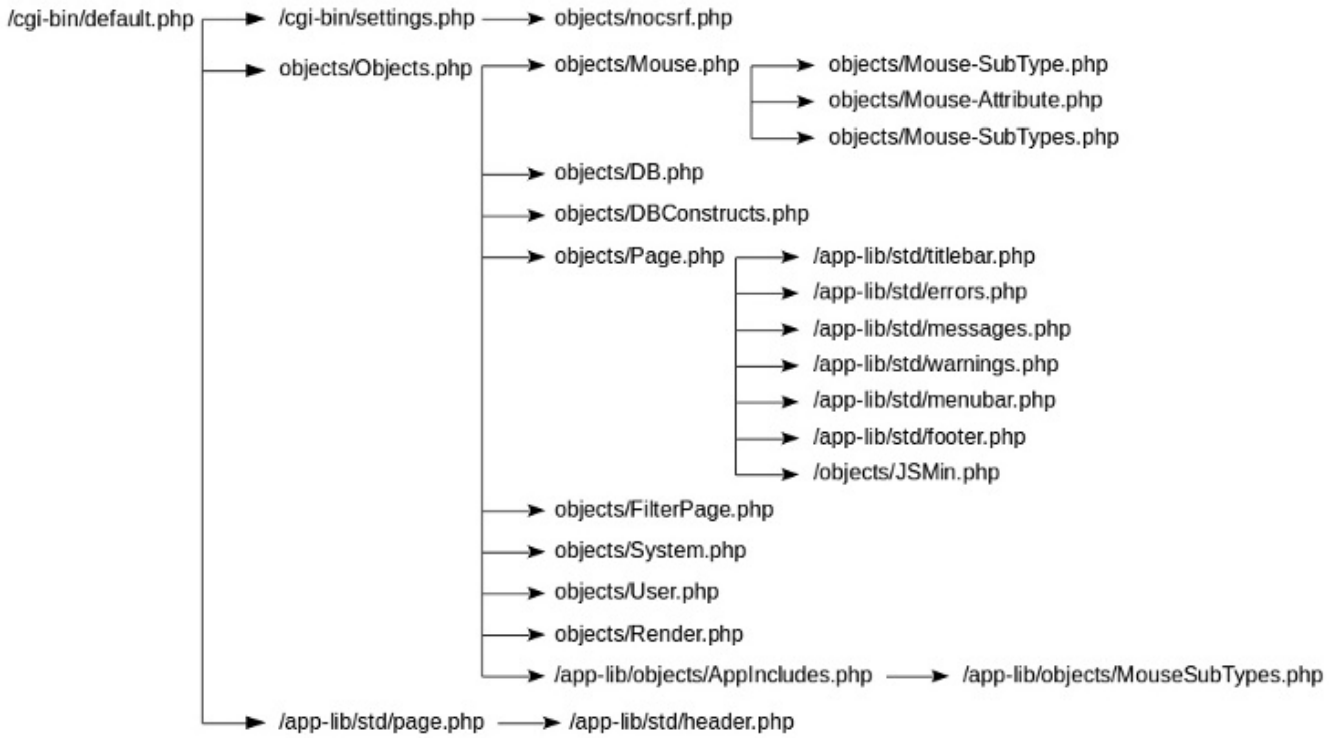
Utilization Samples

The Default Loader takes input from the Web Server as POST/GET parameters and it collects configuration parameters from the Liomys Framework Settings file (config.php). The Default Loader then instantiates the user (if any) based on a browser cookie and loads the Liomys Objects. The appname parameter is checked in the database in order to determine the module loader (in the lib/objects directory), the corresponding constructor, and any global variables. The module loader then loads the corresponding page and data constructors. The Page Constructor instantiates the Data Constructor, collects the appropriate data from the database and returns it to the Page

Constructor. The Data Constructor extends the Mouse Constructor automatically, but a Mouse Object (described below) isn't a required artifact. The Page Constructor then loads the Page Renderer (in the lib/pages directory) unless a Mouse Object exists and the auto-rendering methods are utilized. The rendered page is then output to the web server.



The execution stack is modeled below. The process flows from left to right and then down.



- Log in [15] to post comments

Auditing

Auditing may be achieved in an automated fashion by conforming to a few simple database conventions.

First, the table that you want to audit needs to have a "user_id" which correlates to the user making the change. So, when a user creates, updates, or deletes a row, their user_id is used to track the

change. Then, a few SQL scripts may be executed to generate the triggers necessary to populate the audit table.

To create the audit triggers, the following scripts may be run to generate the corresponding SQL statements.

```
SELECT CONCAT('CREATE TRIGGER ', table_name, '_update_audit AFTER UPDATE ON ', table_name,
' FOR EACH ROW BEGIN DECLARE i_index BIGINT(20); SET i_index = (SELECT MAX(cluster)+1 FROM
lms_audit); ',
audit_column_gen(table_name, 'UPDATE'), ' END$$') FROM information_schema.columns
WHERE table_schema = DATABASE()
AND column_name = 'user_id'
AND table_name NOT LIKE 'lms_post'
AND table_name NOT LIKE 'lms_message'
AND table_name NOT LIKE 'lms_user'
AND table_name NOT LIKE 'lms_user_cookie'
AND table_name NOT LIKE '%audit%'
AND table_name NOT LIKE '%object%';

SELECT CONCAT('CREATE TRIGGER ', table_name, '_insert_audit AFTER INSERT ON ', table_name,
' FOR EACH ROW BEGIN DECLARE i_index BIGINT(20); SET i_index = (SELECT MAX(cluster)+1 FROM
lms_audit); ',
audit_column_gen(table_name, 'INSERT'), ' END$$') FROM information_schema.columns
WHERE table_schema = DATABASE()
AND column_name = 'user_id'
AND table_name NOT LIKE 'lms_post'
AND table_name NOT LIKE 'lms_message'
AND table_name NOT LIKE 'lms_user'
AND table_name NOT LIKE 'lms_user_cookie'
AND table_name NOT LIKE '%audit%'
AND table_name NOT LIKE '%object%';

SELECT CONCAT('CREATE TRIGGER ', table_name, '_delete_audit AFTER DELETE ON ', table_name,
' FOR EACH ROW BEGIN DECLARE i_index BIGINT(20); SET i_index = (SELECT MAX(cluster)+1 FROM
lms_audit); ',
audit_column_gen(table_name, 'DELETE'), ' END$$') FROM information_schema.columns
WHERE table_schema = DATABASE()
AND column_name = 'user_id'
AND table_name NOT LIKE 'lms_post'
AND table_name NOT LIKE 'lms_message'
AND table_name NOT LIKE 'lms_user'
AND table_name NOT LIKE 'lms_user_cookie'
AND table_name NOT LIKE '%audit%'
AND table_name NOT LIKE '%object%';
```

The above scripts require the audit_column_gen function available in the schema/lms_functions.sql file.

To delete the audit triggers, the following scripts may be run to generate the corresponding SQL statements.

```
SELECT CONCAT('DROP TRIGGER IF EXISTS ', table_name, '_update_audit$$') FROM
information_schema.columns
WHERE table_schema = DATABASE()
AND column_name = 'user_id'
AND table_name NOT LIKE 'lms_post'
AND table_name NOT LIKE 'lms_message'
AND table_name NOT LIKE 'lms_user'
AND table_name NOT LIKE 'lms_user_cookie'
```

```
AND table_name NOT LIKE '%audit%'
AND table_name NOT LIKE '%object%';

SELECT CONCAT('DROP TRIGGER IF EXISTS ', table_name, '_insert_audit$$') FROM
information_schema.columns
WHERE table_schema = DATABASE()
AND column_name = 'user_id'
AND table_name NOT LIKE 'lms_post'
AND table_name NOT LIKE 'lms_message'
AND table_name NOT LIKE 'lms_user'
AND table_name NOT LIKE 'lms_user_cookie'
AND table_name NOT LIKE '%audit%'
AND table_name NOT LIKE '%object%';

SELECT CONCAT('DROP TRIGGER IF EXISTS ', table_name, '_delete_audit$$') FROM
information_schema.columns
WHERE table_schema = DATABASE()
AND column_name = 'user_id'
AND table_name NOT LIKE 'lms_post'
AND table_name NOT LIKE 'lms_message'
AND table_name NOT LIKE 'lms_user'
AND table_name NOT LIKE 'lms_user_cookie'
AND table_name NOT LIKE '%audit%'
AND table_name NOT LIKE '%object%';
```

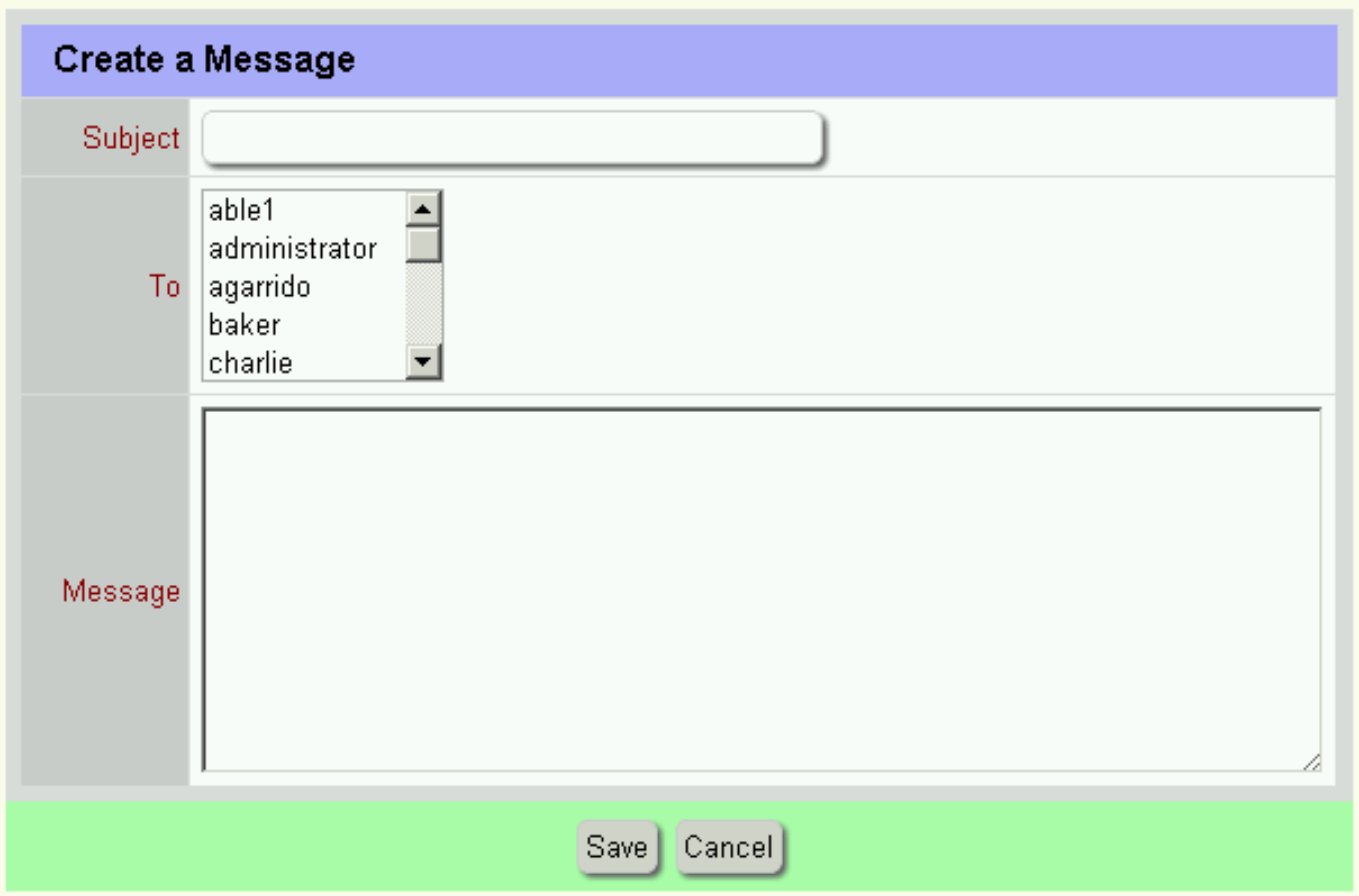
- Log in [16] to post comments

Editor Page Render Example

Using either Mouse Object constructor detailed above, a Message Editor Page Class would look like:

```
class MessageEditor extends Page {
  function __construct() {
  parent::__construct("Message Editor");
  $this->data = new MessageDB();
  $this->list = array();
  }

  function pre_render() {
  parent::pre_render();

  //alternatively, the following may be set as init => $_POST[xxxx] in the MOUSE object
  if (isset($_POST["subject"])) {$this->data->set_subject($_POST["subject"]);}
  if (isset($_POST["message"])) {$this->data->set_message($_POST["message"]);}

  if (isset($_REQUEST['Save'])) {
  if (count($_REQUEST['to_user_id']) > 0) {
  foreach ($_REQUEST['to_user_id'] as $id) {
  $this->data->set_to_user_id($id);
  $this->data->save_lms_message();
  }
  }
  if ($this->error_count() == 0) {
  my_redirect("default.php?appname=message-mgr");
```

```
 }
 }

 if (isset($_REQUEST['Cancel'])) {
 my_redirect("default.php?appname=message-mgr");
 }
 }

 function body() {
 set_form("/cgi-bin/default.php?appname=message-edit");
 auto_render_view($this->data, "Create a Message", "600px");
 render_button_bar("600px");
 echo('</form>');

 }
}
```



- Log in [17] to post comments

General Application Creation

Steps to create a new page:

1) Insert entry into the lms_app table.

INSERT IGNORE INTO lms_app (NAME, module, constructor, proper_name, GLOBAL)

```
  VALUES ('url-ref', 'objects/dir/file.php', 'Class', 'Name to Show on Page', 'global_variable');
```

2) Insert an entry into the lms_menu table.

```
INSERT IGNORE INTO lms_menu (NAME, url, hierarchy, groups, description, visible)
  VALUES ('Audits', '/cgi-bin/default.php?appname=audit', '90.25',
  (SELECT group_id FROM lms_group WHERE groupname IN ('Master Administrator')),
  'The system audit log menu', 1);
```

3) Insert an entry into the lms_app_group table (if the page being created is only available to certain groups)

```
INSERT IGNORE INTO lms_app_group (NAME, group_id)
  VALUES ('audit',
  (SELECT group_id FROM lms_group WHERE groupname IN ('Master Administrator')));
```

4) Add a root .php file to the objects subdirectory (eg. objects/dir/file.php)

```
<?php include_once("objects/admin/data/audit.php"); include_once("objects/admin/view/audit.php");
?>
```

5) Add a data .php file to the data subdirectory (eg. objects/dir/data/file.php)

```
<?php class NameDB extends DB {
  function __construct() {
  parent::__construct();
  $this->has(array( ... ));
  }
  }
?>
```

6) Add a page .php file to the view subdirectory (eg. objects/dir/view/file.php)

```
<?php class Name extends Page {
  function __construct() {
  parent::__construct("Name");
  $this->data = new NameDB();
  }

  function body() {
  $this->auto_render();
  }
  }
?>
```

- Log in [18] to post comments

List (or Manager) Page Render Example

Using either Mouse Object constructor detailed above, a Message Manager Page Class would look like:

```
class MessageManager extends FilterPage {
  function __construct() {
```

```
  parent::__construct("Message Manager");
  $this->data = new MessageDB();
  $this->list = array();
  $this->attributes = array();
  $this->page_size = 20; //overload FilterPage
  $this->total_rows = $this->data->count_lms_message(); //overload FilterPage
  }

  function pre_render() {
  parent::pre_render();
  if ($_REQUEST['print']) {
  $this->page_size = $this->total_rows; //if this is a printable page, show all of the results
  }
  //select from lms_message table
  $this->list = $this->data->list_lms_message(($this->get_current_page()-1) *
$this->get_page_size(),
  $this->get_page_size());
  }

  function body() {
  if (!$_REQUEST['print']) {
  render_pagination_bar($this->current_page, $this->total_pages, $this->total_rows, true);
  }
  auto_render_list($this->data, $this->list, true);
  if (!$_REQUEST['print']) {
  render_pagination_bar($this->current_page, $this->total_pages, $this->total_rows, false);
  }
  }
}
```

Start Prev Next End Goto Page 1 of 1; 13 results

There are 13 results in this page.

Subject	From	To	Time Sent
Can you see me?	administrator	administrator	2012-09-17 22:25:13
Can you see me?	administrator	able1	2012-09-17 22:19:08
Can you see me?	administrator	able1	2012-09-17 22:11:21
Monster test	administrator	administrator	2012-09-15 22:03:17
Monster test	administrator	able1	2012-09-15 22:03:17
Monster test	administrator	agarrido	2012-09-15 22:01:33
Monster test	administrator	administrator	2012-09-15 22:01:33
Monster test	administrator	able1	2012-09-15 22:01:33
test	administrator	administrator	2012-09-15 21:52:56
test	administrator	administrator	2012-09-15 21:51:58
test	administrator	system	2012-09-15 21:12:49
Hello Back	able1	administrator	2012-09-01 20:16:55
Hi there	administrator	able1	2012-09-01 20:16:43

Start Prev Next End Page 1 of 1; 13 results

- Log in [19] to post comments

Mouse Object Construction Options

The following examples take a Mouse class and produce a number of views reflecting standard Create, Read, Update, and Delete (CRUD) functionality.

Generally, Mouse objects can be constructed in one of two ways. The first way is to create a separate Mouse Object for each of the rendering types. The second way is with a single class representing the complete Mouse Object which uses “modify_” Reflection methods to alter the Object based on the type of rendering (eg. list, editor, or viewer) being performed. The first way is cleaner with regard to code clutter; the second, with regard to file clutter.

The first method is preferred as it is easier to read and manage, but the second method could provide a smaller memory footprint and faster memory access should the developer decide to keep the PHP files resident in the web server or compiled to C byte code.

- Log in [20] to post comments

Multiple Mouse Classes (First Method)

The preferred method requires multiple files – one for each Page class – but the Mouse Object constructor contains only those Attributes and Attribute Options necessary for that Page class. As a result, the complete Message Mouse Object for only the Viewer Page file would look like:

```
class MessageDB extends DB {
  function __construct() {
  parent::__construct();
  $g_user_object = lms_get_user_object();

  $this->has(array(
  message_id => array(isa => 'int',
  init => $_REQUEST["message_id"],
  join => array(where => "lms_message.message_id = ? AND " .
  '(lms_message.from_user_id = ' . $g_user_object->user_id .
  ' OR lms_message.to_user_id = ' . $g_user_object->user_id . ')')),
  subject => array(isa => 'Str32'),
  message => array(isa => 'Text',
  name => "Message",
  row => 2,
  col => 1),
  from_user_id => array(name => 'From',
  isa => 'Username',
  join => array(table => 'lms_user lu_from',
  column => 'lu_from.username',
  where => 'lu_from.user_id = lms_message.from_user_id'),
  row => 1,
  col => 1),
  to_user_id => array(name => 'To',
```

```
  isa => 'Username',
  join => array(table => 'lms_user lu_to',
  column => 'lu_to.username',
  where => 'lu_to.user_id = lms_message.to_user_id'),
  row => 1,
  col => 2),
  sent_date_time => array(name => 'Time Sent',
  isa => 'DateTime',
  order => 'desc',
  row => 3,
  col => 1),
  read_date_time => array(isa => 'DateTime',
  name => 'Time Read',
  row => 3,
  col => 2,
  join => array(column => "IF(lms_message.read_date_time=” .
  “0,NOW(),lms_message.read_date_time)"))
  ));
  }

  function update_message_read_time() {
  try {
  $sql = "UPDATE lms_message SET read_date_time = NOW() where message_id = ? and
read_date_time in (0,null)";
  $bind_vars = array(array('value' => $this->get_message_id(), 'type' => 'i'));

  if ($stmt = $this->db_prepare($sql, $bind_vars)) {
  $stmt->execute();
  }
  } catch (Exception $e) {
  print($e->getMessage());
  }
  }
```

- Log in [21] to post comments

Editor Mouse Class Example

```
class MessageDB extends DB {
  function __construct() {
  parent::__construct();
  global $g_user_object;

  $options = $this->get_usernames();

  $this->has(array(
  subject => array(isa => 'Str32',
  name => "Subject",
  row => 1,
  col => 1,
  required => 1,
  control => function($args){
```

```
  $args["size"] = "40";
  $args["maxlength"] = "32";
  render_input($args);}),
  message => array(isa => 'Text',
  name => "Message",
  row => 3,
  col => 1,
  required => 1,
  control => function($args) {
  $args["rows"] = "10";
  $args["cols"] = "60";
  render_textarea($args, $args['value']);}),
  from_user_id => array(isa => 'int',
  required => 1,
  init => $g_user_object->get_user_id()),
  to_user_id => array(name => 'To',
  isa => 'int',
  row => 2,
  col => 1,
  required => 1,
  control => function($args) use ($options) {
  render_multiselect("to_user_id[]", $_POST['to_user_id'], $options, "5");
  })));
  }

  function get_usernames() {
  $options = array();

  $sql = "SELECT user_id, username FROM lms_user ORDER BY username";
  if ($stmt = $this->db_prepare($sql, null)) {
  $stmt->execute();
  $result = $stmt->get_result();
  while ($row = $result->fetch_row()) {
  array_push($options, array('value' => $row[0], 'label' => $row[1]));
  }
  $stmt->close();
  }

  return($options);
  }
}
```

- Log in [22] to post comments

Manager Mouse Class Example

```
class MessageDB extends DB {
  function __construct() {
  parent::__construct();
  $g_user_object = lms_get_user_object();

  $this->has(array(
```

```
  message_id => array(isa => 'int',
  join => array(where => '(lms_message.from_user_id = ' . $g_user_object->user_id . ' OR ' .
'lms_message.to_user_id = ' . $g_user_object->user_id . ')')),
  subject => array(name => 'Subject',
  isa => 'Str32',
  //read_date_time is in the 5th array position of this MOUSE object
  style => function($data) {
  if ($data[5] == "0000-00-00 00:00:00") { return("font-weight:bold;"); }},
  //message_id is in the 0th array position of this MOUSE object
  link => function($data) {
  return('default.php?appname=message-view&message_id=' . $data[0]);}),
  from_user_id => array(name => 'From',
  isa => 'Username',
  join => array(table => 'lms_user lu_from',
  column => 'lu_from.username',
  where => 'lu_from.user_id = lms_message.from_user_id')),
  to_user_id => array(name => 'To',
  isa => 'Username',
  join => array(table => 'lms_user lu_to',
  column => 'lu_to.username',
  where => 'lu_to.user_id = lms_message.to_user_id')),
  sent_date_time => array(name => 'Time Sent',
  isa => 'DateTime',
  order => 'desc'),
  read_date_time => array(isa => 'DateTime')
  ));
  }
}
```

- Log in [23] to post comments

Single Mouse Class (Second Method)

As an example of the second method, the following Mouse Object occupies a single file, but has several methods which are called by the respective Page class in order to render the respective view. When the constructor is called, a basic Mouse Object is instantiated.

```
class MessageDB extends DB {
  function __construct() {
  parent::__construct();

  $this->has(array(
  message_id => array(isa => 'int'),
  subject => array(name => 'Subject',
  isa => 'Str32'),
  message => array(isa => 'Text'),
  from_user_id => array(name => 'From',
  isa => 'int'),
  to_user_id => array(name => 'To',
  isa => 'int'),
  sent_date_time => array(name => 'Time Sent',
  isa => 'DateTime',
  order => 'desc'),
```

```
  read_date_time => array(isa => 'DateTime')
  ));
  }
```

When the Viewer page class calls the set_view_options() method, the Mouse Object is modified to reflect the needs of the Viewer page.

```
function set_view_options() {
  //modify the join option of the message_id attribute
  $message_id->join = array(where => "lms_message.message_id = " . $this->get_message_id());
  $this->modify_message_id($message_id);

  $subject->name = "";
  $this->modify_subject($subject);

  $message->row = 2;
  $message->col = 1;
  $message->name = "Message";
  $this->modify_message($message);

  $from_user_id->join = array(table => 'lms_user lu_from',
  column => 'lu_from.username',
  where => 'lu_from.user_id = lms_message.from_user_id');
  $from_user_id->row = 1;
  $from_user_id->col = 1;
  $this->modify_from_user_id($from_user_id);

  $to_user_id->join = array(table => 'lms_user lu_to',
  column => 'lu_to.username',
  where => 'lu_to.user_id = lms_message.to_user_id');
  $to_user_id->row = 1;
  $to_user_id->col = 2;
  $this->modify_to_user_id($to_user_id);

  $sent_date_time->row = 3;
  $sent_date_time->col = 1;
  $this->modify_sent_date_time($sent_date_time);

  $read_date_time->row = 3;
  $read_date_time->col = 2;
  $read_date_time->name = "Time Read";
  $this->modify_read_date_time($read_date_time);
}

function set_list_options() {
  // modify the Mouse Attribute "subject" to properly render the list
  //message_id is in the 0th array position of this MOUSE object
  $subject->link = function($data) {return('default.php?appname=message-view&message_id=' .
$data[0]);};
  $this->modify_subject($subject);

  // modify the Mouse user_id Attributes in order to render the user lists
  $from_user_id->join = array(table => 'lms_user lu_from',
  column => 'lu_from.username',
  where => 'lu_from.user_id = lms_message.from_user_id');
  $this->modify_from_user_id($from_user_id);

  $to_user_id->join = array(table => 'lms_user lu_to',
  column => 'lu_to.username',
```

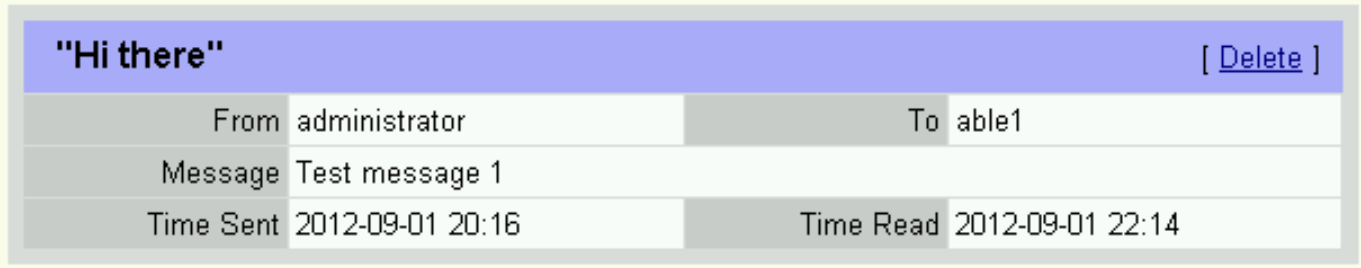
```
  where => 'lu_to.user_id = lms_message.to_user_id');
  $this->modify_to_user_id($to_user_id);
}
```

- Log in [24] to post comments

Viewer Page Render Example

Using either Mouse Object constructor detailed above, a Message Viewer Page Class would look like:

```
class MessageViewer extends Page {
  function __construct() {
  parent::__construct("Message Viewer");
  $this->data = new MessageDB();
  $this->list = array();
  }

  function pre_render() {
  parent::pre_render();
  //handle deletes if a mission_id is presented
  if ($_REQUEST['delete']) { $this->handle_delete(); }

  $this->message = $this->data->read_lms_message();
  if (empty($this->message)) { $this->addError("No message associated with this message ID."); }
  }

  function body() {
  if (!empty($this->message)) {
  $this->data->update_message_read_time();
  $links = array(array(link => "default.php?appname=message-view&delete=1&message_id=" .
  $this->data->get_message_id(), label => "Delete"));
  auto_render_view($this->data, '"' . $this->data->get_subject() . '"', "600px", $links);
  }
  }

  function handle_delete() {
  $this->data->delete_lms_message();
  my_redirect('default.php?appname=message-mgr');
  }
}
```



- Log in [25] to post comments

Debugging

- Log in [26] to post comments

objects/Mouse.php

Setting the global variable $DEBUG above the class declaration will provide significant feedback above the webpage's Title Bar and a small amount of feedback in the Message Area below the Menu Bar. If $DEBUG is set at a higher order (eg. std/page.php), the small amount of feedback in the Message Area will also be generated.

As an example:

ATTRIBUTE: seq= Array ([isa] => int
  Read/Writable? = 1
  Initialize Value =
  Attribute Object ([key] => seq [vals] => Array [name] => [read_writable] => 1 [isa] => integer [trigger] => [doc] => [type] => string [subtype] => Array

Verb is set? 1 (count)
Attribute exists? 0 (lms_audit)

- Log in [27] to post comments

std/page.php

Setting the global variable $DEBUG above the class declaration will provide significant feedback below the webpage's Footer. If $DEBUG is set at this level, any pages loaded after it by the system will also have its debugging activated. $DEBUG may also be set in cgi-bin/settings.php.

As an example:

PHP parsed this page in 0.349 seconds.
PHP used 1,108 KB at peak emalloc usage.
PHP used 1,311 KB at peak memory usage.

PHP has loaded the following files to render this page: Array ([0] => C:\LMS\LIMS\cgi-bin\default.php [1] => C:\LMS\LIMS\cgi-bin\settings.php [2] => C:\LMS\LIMS\lib\objects\Objects.php [3] => C:\LMS\LIMS\lib\objects\Mouse.php [4] => C:\LMS\LIMS\lib\objects\Mouse-SubType.php [5] => C:\LMS\LIMS\lib\objects\Mouse-Attribute.php [6] => C:\LMS\LIMS\lib\objects\Mouse-SubTypes.php [7] => C:\LMS\LIMS\lib\objects\DB.php [8] => C:\LMS\LIMS\lib\objects\Page.php [9] => C:\LMS\LIMS\lib\objects\FilterPage.php [10] => C:\LMS\LIMS\lib\objects\System.php [11] => C:\LMS\LIMS\lib\objects\User.php [12] => C:\LMS\LIMS\lib\objects\Validation.php [13] => C:\LMS\LIMS\lib\objects\Render.php [14] => C:\LMS\LIMS\lib\objects\admin\audit.php [15] => C:\LMS\LIMS\lib\objects\admin\data\audit.php [16] => C:\LMS\LIMS\lib\objects\admin\view\audit.php [17] => C:\LMS\LIMS\lib\std\page.php [18] => C:\LMS\LIMS\lib\std\header.php [19] => C:\LMS\LIMS\lib\std\titlebar.php [20] => C:\LMS\LIMS\lib\std\menubar.php [21] => C:\LMS\LIMS\lib\std\messages.php [22] => C:\LMS\LIMS\lib\std\errors.php [23] =>

C:\LMS\LIMS\lib\std\footer.php

- Log in [28] to post comments

Source URL: https://blackhillsystems.com/?q=node/7

Links
[1] https://blackhillsystems.com/?q=user/login&destination=node/7%23comment-form
[2] https://blackhillsystems.com/?q=user/login&destination=node/63%23comment-form
[3] https://blackhillsystems.com/?q=user/login&destination=node/18%23comment-form
[4] https://blackhillsystems.com/?q=user/login&destination=node/33%23comment-form
[5] https://blackhillsystems.com/?q=user/login&destination=node/8%23comment-form
[6] https://blackhillsystems.com/?q=user/login&destination=node/9%23comment-form
[7] https://blackhillsystems.com/?q=user/login&destination=node/10%23comment-form
[8] https://blackhillsystems.com/?q=user/login&destination=node/11%23comment-form
[9] https://blackhillsystems.com/?q=user/login&destination=node/12%23comment-form
[10] https://blackhillsystems.com/?q=user/login&destination=node/13%23comment-form
[11] https://blackhillsystems.com/?q=user/login&destination=node/14%23comment-form
[12] https://blackhillsystems.com/?q=user/login&destination=node/15%23comment-form
[13] https://blackhillsystems.com/?q=user/login&destination=node/16%23comment-form
[14] https://blackhillsystems.com/?q=user/login&destination=node/17%23comment-form
[15] https://blackhillsystems.com/?q=user/login&destination=node/23%23comment-form
[16] https://blackhillsystems.com/?q=user/login&destination=node/64%23comment-form
[17] https://blackhillsystems.com/?q=user/login&destination=node/29%23comment-form
[18] https://blackhillsystems.com/?q=user/login&destination=node/32%23comment-form
[19] https://blackhillsystems.com/?q=user/login&destination=node/30%23comment-form
[20] https://blackhillsystems.com/?q=user/login&destination=node/24%23comment-form
[21] https://blackhillsystems.com/?q=user/login&destination=node/25%23comment-form
[22] https://blackhillsystems.com/?q=user/login&destination=node/26%23comment-form
[23] https://blackhillsystems.com/?q=user/login&destination=node/27%23comment-form
[24] https://blackhillsystems.com/?q=user/login&destination=node/28%23comment-form
[25] https://blackhillsystems.com/?q=user/login&destination=node/31%23comment-form
[26] https://blackhillsystems.com/?q=user/login&destination=node/19%23comment-form
[27] https://blackhillsystems.com/?q=user/login&destination=node/21%23comment-form
[28] https://blackhillsystems.com/?q=user/login&destination=node/22%23comment-form