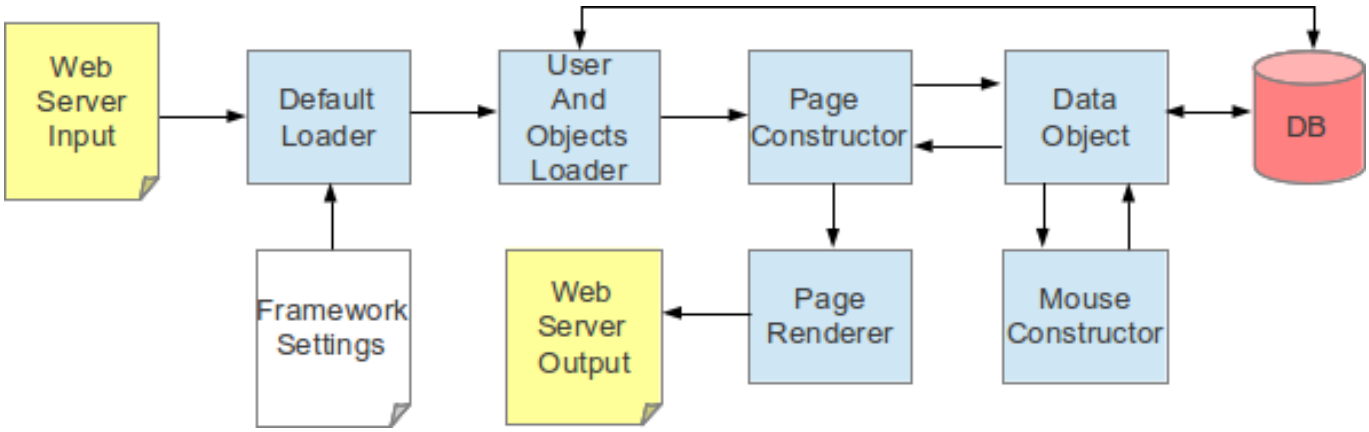
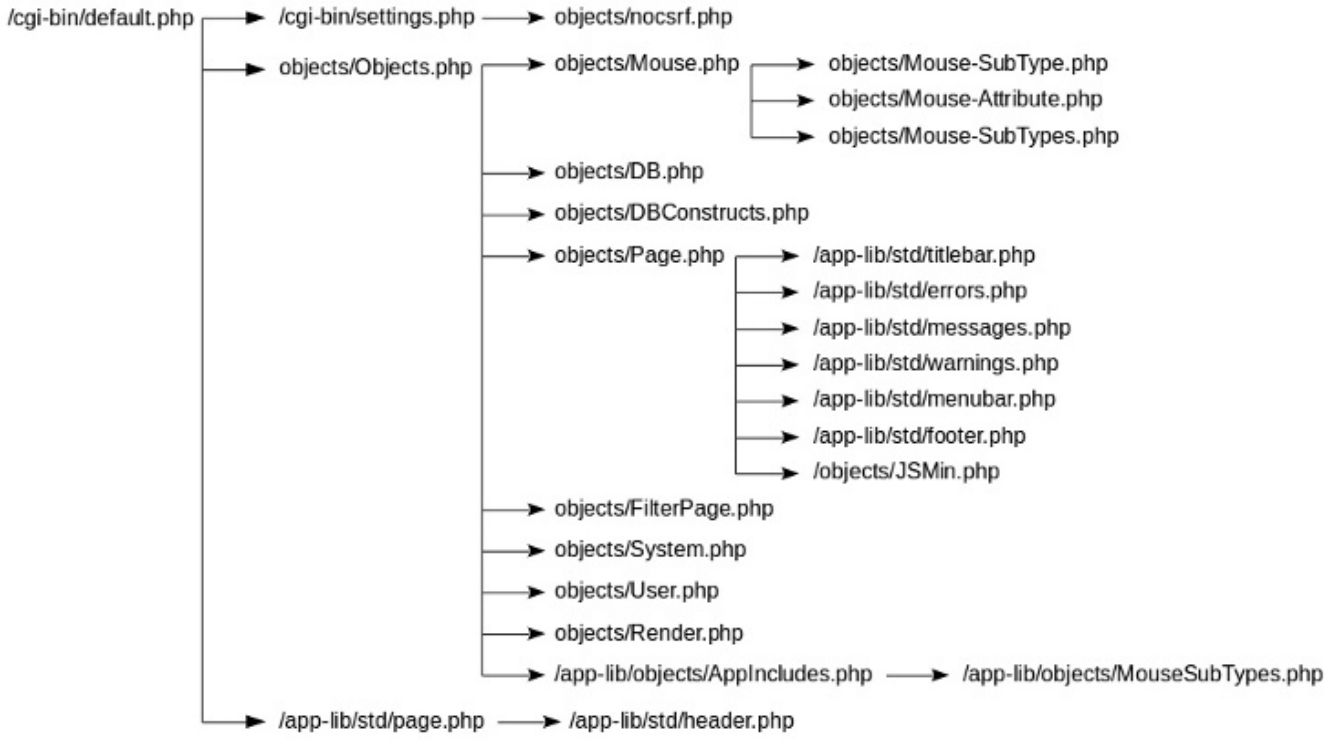
Utilization Samples

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 [1] 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 [2] 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
```

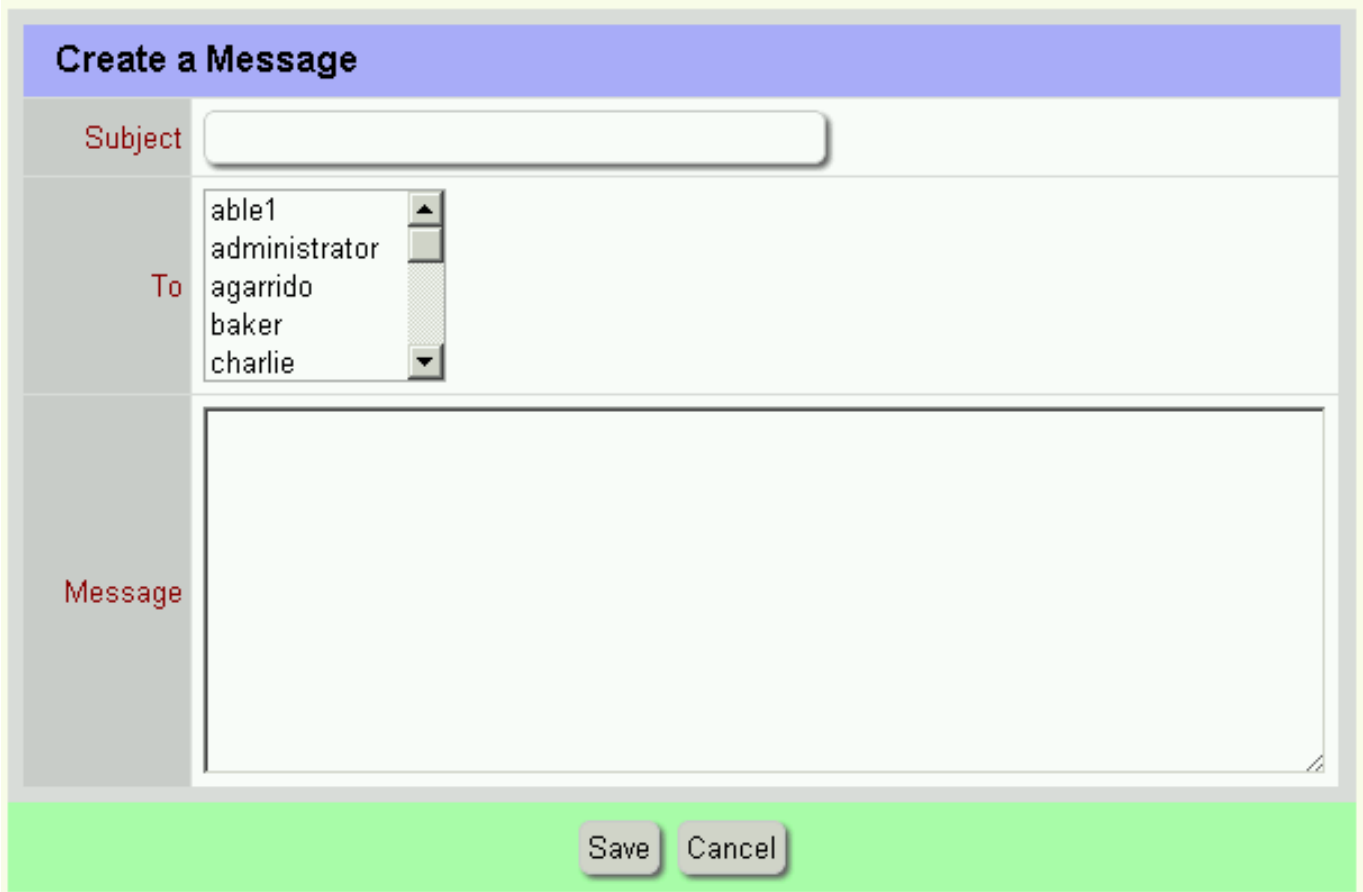
```
 if (isset($_POST["subject"])) {$this->data->set_subject($_POST["subject"]);}
 if (isset($_POST["message"])) {$this->data->set_message($_POST["message"]);}

 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 [3] 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 [4] 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 [5] 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 [6] 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 [7] 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 [8] 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 [9] 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 [10] to post comments

Viewer Page Render Example

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

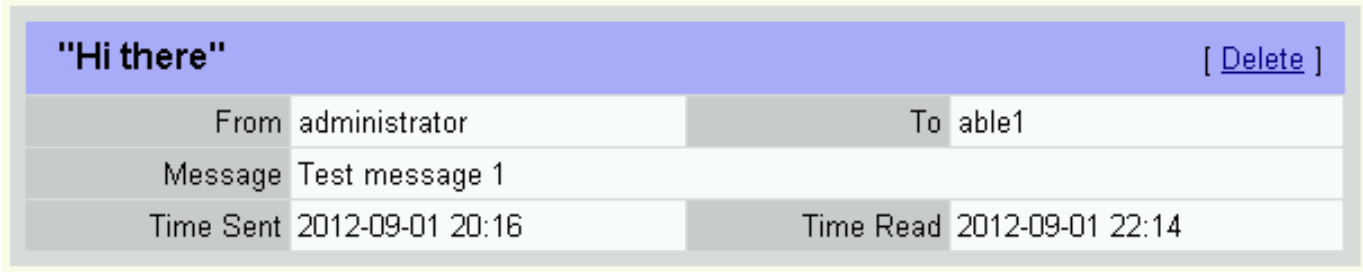
```
class MessageViewer extends Page {
  function __construct() {
  parent::__construct("Message Viewer");
  $this->data = new MessageDB();
  $this->list = array();
  }

  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');
 }
}
```



"Hi there" [Delete]

From	administrator	To	able1
Message	Test message 1		
Time Sent	2012-09-01 20:16	Time Read	2012-09-01 22:14

- Log in [11] to post comments

Source URL: https://blackhillsystems.com/?q=node/23

Links
[1] https://blackhillsystems.com/?q=user/login&destination=node/23%23comment-form
[2] https://blackhillsystems.com/?q=user/login&destination=node/64%23comment-form
[3] https://blackhillsystems.com/?q=user/login&destination=node/29%23comment-form
[4] https://blackhillsystems.com/?q=user/login&destination=node/32%23comment-form
[5] https://blackhillsystems.com/?q=user/login&destination=node/30%23comment-form
[6] https://blackhillsystems.com/?q=user/login&destination=node/24%23comment-form
[7] https://blackhillsystems.com/?q=user/login&destination=node/25%23comment-form
[8] https://blackhillsystems.com/?q=user/login&destination=node/26%23comment-form
[9] https://blackhillsystems.com/?q=user/login&destination=node/27%23comment-form
[10] https://blackhillsystems.com/?q=user/login&destination=node/28%23comment-form
[11] https://blackhillsystems.com/?q=user/login&destination=node/31%23comment-form