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 [1] to post comments

Source URL: http://www.blackhillsystems.com/?q=node/11

Links
[1] http://www.blackhillsystems.com/?q=user/login&destination=node/11%23comment-form