AJAX for CodeIgniter 1.5.1 *v0.1*

AJAX for CodeIgniter is a CodeIgniter library that provides you with AJAX functionality for your CodeIgniter web applications. It comes with simple to use AJAX helpers that you can directly put in use in your apps. AJAX for CodeIgniter includes both Prototype and Scriptaculous helpers.

How to use AJAX for CodeIgniter?

Place the AJAX library in the following directory (default libraries directory):

```
www-root/system/libraries/
```

Place the javascript files in your www root directory. For example:

```
www-root/javascript/
```

If you use mod_rewrite, make sure you allow the javascript directory to be accessed. To do this, set up your *.htaccess* file to look something like this:

```
RewriteEngine on
RewriteCond $1 !^(index\.php|images|stylesheets|javascript)
RewriteRule ^(.*)$ /index.php/$1 [L]
```

Then, simply include whichever javascript files you would like to use in your html file:

```
<script src="javascript/prototype.js" type="text/javascript"></
script>
<script src="javascript/effects.js" type="text/javascript"></script>
<script src="javascript/dragdrop.js" type="text/javascript"></
script>
<script src="javascript/controls.js" type="text/javascript"></
script>
```

Finally, load up the AJAX library as you would with any other CodeIgniter library. There are two possible ways to do this. Directly from within your controller:

```
$this->load->library('ajax');
```

Or, auto-load the AJAX library from within *system/application/config/autoload.php*:

```
$autoload['libraries'] = array('database', 'session', 'ajax');
```

Then all you need to do to use an AJAX helper is to call *$this->ajax*. For example:

```
echo $this->ajax->link_to_remote("Login", array('url' => '/login',
'update' => 'divblock'));
```

JavaScript Class

Provides functionality for working with JavaScript.

button_to_function($name,$function=null)

Returns a link that'll trigger a JavaScript function using the onclick handler.

Example :

```
button_to_function('Greetigs', 'alert("Hello world!")' );
```

escape($javascript)

Escape carrier returns and single and double quotes for JavaScript segments.

tag($content)

Returns a JavaScript tag with the content inside.

Example :

```
tag('alert("All is good")');
```

link_to_function($name,$function,$html_options=null)

Returns a link that'll trigger a JavaScript function using the onclick handler and return false after the fact.

Example :

```
link_to_function("Greeting","alert('Hello world!')");
```

Prototype Class

Provides a set of helpers for calling Prototype JavaScript functions, including functionality to call remote methods using Ajax. This means that you can call actions in your page without reloading the page, but still update certain parts of it using injections into the DOM. The common use case is having a form that adds a new element to a list without reloading the page.

evaluate_remote_response()

Returns 'eval(request.responseText)' which is the JavaScript function that form_remote_tag can call in ['complete'] to evaluate a multiple update return document using update_element_function calls.

form_remote_tag($options)

Returns a form tag that will submit using XMLHttpRequest in the background instead of the regular reloading POST arrangement. Even

though it's using JavaScript to serialize the form elements, the form submission will work just like a regular submission as viewed by the receiving side (all elements available in params). The options for specifying the target with ['url'] and defining callbacks is the same as link_to_remote.

A "fall-through" target for browsers that doesn't do JavaScript can be specified with the ['action']/['method'] options on ['html'].

Example :

```
form_remote_tag( array('url'=>$some_url) );
```

link_to_remote($name,$function,$html_options=null)

Returns a link to a remote action defined by options['url'] that's called in the background using XMLHttpRequest. The result of that request can then be inserted into a DOM object whose id can be specified with options['update']. Usually, the result would be a partial prepared by the controller with either render_partial or render_partial_collection.

Example :

```
link_to_remote("Login", array('url'=> '/login'));
```

You can also specify a hash for options['update'] to allow for easy redirection of output to an other DOM element if a server-side error occurs:

Example:

```
link_to_remote("Login", array('url' => '/login', 'update' = array
('success'=>'dologin','failure'=>'error')));
```

Optionally, you can use the options[:position] parameter to influence how the target DOM element is updated. It must be one of ['before'], ['top'], ['bottom'], or ['after'].

By default, these remote requests are processed asynchronous during which various JavaScript callbacks can be triggered (for progress

indicators and the likes). All callbacks get access to the request object, which holds the underlying XMLHttpRequest.

To access the server response, use request.responseText, to find out the HTTP status, use request.status.

Example:

```
link_to_remote("Login", array('url'=> '/login'
'complete'=>'undoRequestCompleted(request)'));
```

The callbacks that may be specified are (in order):

- **loading**
  Called when the remote document is being loaded with data by the browser.
- **loaded**
  Called when the browser has finished loading the remote document.
- **interactive**
  Called when the user can interact with the remote document, even though it has not finished loading.
- **success**
  Called when the XMLHttpRequest is completed, and the HTTP status code is in the 2XX range.
- **failure**
  Called when the XMLHttpRequest is completed, and the HTTP status code is not in the 2XX range.
- **complete**
  Called when the XMLHttpRequest is complete (fires after success/ failure if they are present).

You can further refine ['success'] and ['failure'] by adding additional callbacks for specific status codes.

If you for some reason or another need synchronous processing (that'll block the browser while the request is happening), you can specify options['type'] = 'synchronous'.

You can customize further browser side call logic by passing in JavaScript code snippets via some optional parameters. In their order of use these are:

- **confirm**
  Adds confirmation dialog.
- **condition**
  Perform remote request conditionally by this expression. Use this to describe browser-side conditions when request should not be initiated.
- **before**
  Called before request is initiated.
- **after**
  Called immediately after request was initiated and before :loading.
- **submit**
  Specifies the DOM element ID that's used as the parent of the form elements. By default this is the current form, but it could just as well be the ID of a table row or any other DOM element.

observe_field(**$field_id**,**$options =null**)

Observes the field with the DOM ID specified by field_id and makes an Ajax call when its contents have changed.

Required options are either of:

- **url**
  url_for-style options for the action to call when the field has changed.
- **function**
  Instead of making a remote call to a URL, you can specify a function to be called instead.

Additional options are:

- **frequency**
  The frequency (in seconds) at which changes to this field will be detected. Not setting this option at all or to a value equal to or less than zero will use event based observation instead of time based observation.
- **update**
  Specifies the DOM ID of the element whose innerHTML should be updated with the XMLHttpRequest response text.
- **with**
  A JavaScript expression specifying the parameters for the XMLHttpRequest. This defaults to 'value', which in the evaluated context refers to the new field value. If you specify a string without a

"=", it'll be extended to mean the form key that the value should be assigned to. So :with => "term" gives "'term'=value". If a "=" is present, no extension will happen.

- **on**
  Specifies which event handler to observe. By default, it's set to "changed" for text fields and areas and "click" for radio buttons and checkboxes. With this, you can specify it instead to be "blur" or "focus" or any other event.

Additionally, you may specify any of the options documented in link_to_remote.

observe_form($form,$options=null)

Like observe_field, but operates on an entire form identified by the DOM ID form_id. options are the same as observe_field, except the default value of the :with option evaluates to the serialized (request string) value of the form.

periodically_call_remote($options=null)

Periodically calls the specified url (options['url']) every options ['frequency'] seconds (default is 10). Usually used to update a specified div (options['update']) with the results of the remote call. The options for specifying the target with :url and defining callbacks is the same as link_to_remote.

remote_function($options)

Returns the JavaScript needed for a remote function. Takes the same arguments as link_to_remote.

Example:

```
<select id="options" onchange="<?= remote_function(array
('update' => 'options', 'url' => $some_url) ? >">
<option value="0">Hello</option>
<option value="1">World</option>
</select>
```

submit_to_remote($name,$value,$options=null)

Returns a button input tag that will submit form using XMLHttpRequest in the background instead of regular reloading POST arrangement. options argument is the same as in form_remote_tag.

> These functions are from JavaScriptGenerator class which was merged into prototype.

dump($javascript)

Writes raw JavaScript to the page.

ID($id,$extend=null)

Returns a element reference by finding it through id in the DOM. This element can then be used for further method calls.

Examples:

```
ID('blank_slate'); // => Will return $('blank_slate');
ID('blank_slate','show'); // => $('blank_slate').show();
```

alert($message)

Displays an alert dialog with the given message.

assign($variable,$value)

Assigns the JavaScript variable the given value.

call($function,$args = null)

Calls the JavaScript function, optionally with the given arguments.

delay($seconds=1,$script='')

Executes the content of the block after a delay of seconds.

hide($ids)

Hides the visible DOM elements with the given ids.

insert_html($position,$id,$options_for_render=null)

Inserts HTML at the specified position relative to the DOM element identified by the given id.

position maybe one of:

- **top**
  HTML is inserted inside the element, before the element's existing content.
- **bottom**
  HTML is inserted inside the element, after the element's existing content.
- **before**
  HTML is inserted immediately preceeding the element.
- **after**
  HTML is inserted immediately following the element.

Example:

```
//Insert the rendered 'navigation' partial just before the DOM
//element with ID 'content'.
insert_html('before','content',array('partial'=>'navigation');
```

redirect_to($location)

Redirects the browser to the given location.

remove($ids)

Removes the DOM elements with the given ids from the page.

replace($id,$options_for_render=null)

Replaces the "outer HTML" (i.e., the entire element, not just its contents) of the DOM element with the given id.

replace_html($id,$options_for_render=null)

Replaces the inner HTML of the DOM element with the given id.

select($pattern)

Returns a collection reference by finding it through a CSS pattern in the DOM. This collection can then be used for further method calls.

Examples:

```
select('p') // => $$('p');
```

show($ids)

Shows hidden DOM elements with the given ids.

toggle($ids)

Toggles the visibility of the DOM elements with the given ids.

Scriptalaculous Class

Provides a set of helpers for calling Scriptaculous JavaScript functions, including those which create Ajax controls and visual effects.

To be able to use these helpers, you must include the Prototype JavaScript framework and the Scriptaculous JavaScript library in your pages.

The Scriptaculous helpers' behavior can be tweaked with various options. See the documentation at script.aculo.us for more information on using these helpers in your application.

dragable_element($element_id,$options=null)

Makes the element with the DOM ID specified by element_id draggable.

Example :

```
draggable_element('my_image',array('revert'=?'true'));
```

drop_receiving_element($element_id,$options=null)

Makes the element with the DOM ID specified by element_id receive dropped draggable elements (created by draggable_element). and make an AJAX call By default, the action called gets the DOM ID of the element as parameter.

Example :

```
drop_receiving_element('my_cart',array('url'=>$update_url));
```

sortable_element($element_id,$options=null)

Makes the element with the DOM ID specified by element_id sortable by drag-and-drop and make an Ajax call whenever the sort order has changed. By default, the action called gets the serialized sortable element as parameters.

Example :

```
sortable_element('my_list',array('url'=>$order_url));
```

visual_effect($name,$element_id=false,$js_options=null)

Returns a JavaScript snippet to be used on the Ajax callbacks for starting visual effects.

Example :

```
sortable_element('my_list',array('url'=>$order_url));
```

These functions are from JavaScriptMacro class which was merged into Scriptalaculous.

auto_complete_field($field_id,$options=null)

Adds AJAX autocomplete functionality to the text input field with the DOM ID specified by field_id.

This function expects that the called action returns a HTML <ul> list, or nothing if no entries should be displayed for autocompletion.

You'll probably want to turn the browser's built-in autocompletion off, so be sure to include a autocomplete="off" attribute with your text input field.

The autocompleter object is assigned to a Javascript variable named field_id_auto_completer. This object is useful if you for example want to trigger the auto-complete suggestions through other means than user input (for that specific case, call the activate method on that object).

Required options are:

- **url**
  URL to call for autocompletion results in url_for format.

Addtional options are:

- **update**
  Specifies the DOM ID of the element whose innerHTML should be updated with the autocomplete entries returned by the AJAX request. Defaults to field_id + '_auto_complete'
- **with**
  A JavaScript expression specifying the parameters for the XMLHttpRequest. This defaults to 'fieldname=value'.

- **frequency**
  Determines the time to wait after the last keystroke for the AJAX request to be initiated.
- **indicator**
  Specifies the DOM ID of an element which will be displayed while autocomplete is running.
- **tokens**
  A string or an array of strings containing separator tokens for tokenized incremental autocompletion. Example: :tokens => ',' would allow multiple autocompletion entries, separated by commas.
- **min_chars**
  The minimum number of characters that should be in the input field before an Ajax call is made to the server.
- **on_hide**
  A Javascript expression that is called when the autocompletion div is hidden. The expression should take two variables: element and update. Element is a DOM element for the field, update is a DOM element for the div from which the innerHTML is replaced.
- **on_show**
  Like on_hide, only now the expression is called then the div is shown.
- **after_update_element**
  A Javascript expression that is called when the user has selected one of the proposed values. The expression should take two variables: element and value. Element is a DOM element for the field, value is the value selected by the user.
- **select**
  Pick the class of the element from which the value for insertion should be extracted. If this is not specified, the entire element is used.

in_place_editor($field_id,$options=null)

Makes an HTML element specified by the DOM ID field_id become an in-place editor of a property.

A form is automatically created and displayed when the user clicks the element.

The form is serialized and sent to the server using an AJAX call, the action on the server should process the value and return the updated value in the body of the reponse. The element will automatically be

updated with the changed value (as returned from the server).

Required options are:

- **url**
  Specifies the url where the updated value should be sent after the user presses "ok".

Addtional options are:

- **rows**
  Number of rows (more than 1 will use a TEXTAREA)
- **cols**
  Number of characters the text input should span (works for both INPUT and TEXTAREA)
- **size**
  Synonym for :cols when using a single line text input.
- **cancel_text**
  The text on the cancel link. (default: "cancel")
- **save_text**
  The text on the save link. (default: "ok")
- **loading_text**
  The text to display when submitting to the server (default: "Saving...")
- **external_control**
  The id of an external control used to enter edit mode.
- **load_text_url**
  URL where initial value of editor (content) is retrieved.
- **options**
  Pass through options to the AJAX call (see prototype's Ajax.Updater)
- **with**
  JavaScript snippet that should return what is to be sent in the AJAX call, form is an implicit parameter
- **script**
  Instructs the in-place editor to evaluate the remote JavaScript response (default: false)

in_place_editor_field(**object**, **$tag_options = null**, **$options = null**)

Renders the value of the specified object and method with in-place

editing capabilities.

text_field_with_auto_complete(**object**, **$tag_options = null**, **$options = null**)

Wrapper for text_field with added AJAX autocompletion functionality.