Using jQuery and Ajax

Table of contents

Using jQuery and Ajax

About jQuery

jQuery is a solid JavaScript library. Among its many advantages:

- Works as expected with numerous web browsers.
- Clear, concise and intuitive architecture.
- Many available plugins.

You can learn more about jQuery on the official website: http://jquery.com/

jQuery's $

jQuery has the `$()` function, and the $ namespace, which contains all the other functions.

The `$()` function can be used in many ways:

- When using a function as its parameter, `$()` will execute the function once the page's DOM is fully loaded. For instance: `$(function(){ /* do something */ });`
- When passing a string containing a CSS selector, `$()` will return all the HTML nodes which match the selector. For instance: `$('ul#nav');`

  That set of nodes can then be assigned to any of jQuery's methods. For instance, if you want to hide the navigation element returned by the above selector: `$('ul#nav').slideUp('fast');`
- When using pure HTML code as its parameter, `$()` will create the node (DOM element). That node can, again, be used with any of jQuery's methods: `$("<li>Sign Off</li>").appendTo("ul#nav");`

> ⚠ `$()` is merely a shortcut to the library-specific `jQuery()` function. It exists in order to accelerate coding. Many other JavaScript libraries use `$()` for their shortcut.
>
> If you are mixing JavaScript libraries in your theme, you might be better off by telling jQuery to free `$()`. Simply call `$.noConflict()`, and from then on, any call to jQuery will require you to use `jQuery()` instead of `$()`.

You can learn more about `jQuery()` and `$()` on the official website: http://api.jquery.com/jQuery/

Functions in jQuery

jQuery's $ namespace contains all the functions that are not attached to a specific node.

Among these are a few Ajax and utilitarian functions. For instance:

```
$.post('/handler.php', {'action': 'purchase', 'product': 434}, function(data){/* do something */} );
```

You can attach events to a set of nodes returned by `$()`. One of the advantages of using jQuery, once again, is that when handling events, it helps you by harmonizing so that you do not have to cater for each browser specifics.

In order to associate a function to a click event, simple add `.click(function)`.

In order to generate a click event, you can also use `.click()` with any parameter. For instance: `$('#button').click(function(){/* do something */});`

You can learn more about jQuery's Ajax and utilitarian functions on the official website:

- http://api.jquery.com/category/ajax/ (for instance, http://api.jquery.com/jQuery.post/)
- http://api.jquery.com/category/utilities/

Selectors

jQuery offers two ways to select page elements:

- Combine CSS and XPath selectors in a string, which is used with the `$()` function. For instance: `$("div > ul a")`
- Using one of the several methods already available in the `jQuery` namespace.

These two ways can be combined.

```
var my_jQuery_object = $("#my_image");
var my_jQuery_object = $("#menu a");
var my_jQuery_object = $("#id > .classe, #id td:last-child");

/* returns the 'td' elements within the odd 'tr'. */
var my_jQuery_object = $('tr:odd td');

/* returns the fourth paragraph. */
var my_jQuery_object = $("p:eq(4)");

/* returns the 7 first paragraphs. */
var my_jQuery_object = $("p:lt(8)");
```

You can learn more about jQuery's selectors on the official website: http://api.jquery.com/category/selectors/

Methods and callbacks

A whole set of methods are available in the standard API: DOM manipulation, CSS manipulation, event management, visual effects, etc.

For instance, if you wish to have all the paragraphs in a page slowly disappears, use this:

```
$("p").fadeOut();
```

Some methods (such as the `fadeOut()` one) accept another method as a parameter. Such a method will be executed once the first one is done. That is called a callback.

For instance:

```
$(".test").fadeOut("slow",function(){
  $(this).fadeIn("slow");
});
```

All of jQuery's methods return a `jQuery` object. This makes it possible to chain methods, with no limit. You can even write your code so that it reads just like a function block.

For instance, this code works perfectly well, and is easy to read and to update:

```
$(".emptyContent").append("This is a test")
  .css("border", "1px solid red")
  .addClass("fullContent")
  .removeClass("emptyContent");
```

Available plugins in PrestaShop

Here is a list of the jQuery plugins that are available in a default installation of PrestaShop:

Plugin file name	Plugin description
jquery.colorpicker	Photoshop-style color selector.
jquery.cookie-plugin	Read, write and delete cookies.
jquery.dimensions	Manage an element's dimensions.
jquery.easing	Manage the speed of an animation.
jquery.excanvas	Change the canvas of an element (rounded corners / gradients / opacity / draw line, arc, etc.)
jquery.fieldselection	Use and replace the text selected within a zone.
jquery.flip	Flip an element.
jquery.flot	Create graphs presenting data as curves, bars, etc.
jquery.highlight	Add syntax colorization.
jquery.hoverIntent	Add a prediction effect on JavaScript's hover event.
jquery.idTabs	Manage tabs.
jquery.ifxtransfer	Animate an element with a transfer from one container to another.

jquery.jqminmax	Add min-width, max-width, min-height and max-height on all browsers.
jquery.pngFix	Manage transparency in IE 5.5 and IE 6.
jquery.scrollTo	Make the page or element scroll to a certain position.
jquery. serialScroll	Make a series of element scroll to a certain position.
jquery.tablednd	Drag and drop a table's rows.
jquery.typewatch	Execute a function when the user has typed some text in a zone and has stopped typing after a certain amount of time.
jquery.validate- creditcard	Validate a credit card number depending on its type.

A couple other functions

jQuery's API is incredibly complete, and you will spend hours finding new possibilities.

Here is a couple of function that can be tremendously useful when creating a theme.

First, `each()`, which makes it possible to loop through a list of elements:

```
$("img").each(function(){ console.log($(this).attr("src")); });
```

Second, `browser`, which is an object that helps you know which browser you are working with:

```
if($.browser.msie) {
  if($.browser.version == 6) {
    // Your IE6 specific code.
  } else {
    // Code for any other IE browser.
  }
}
```

Making Ajax calls with jQuery

About Ajax

The term "Ajax" is really an acronym for Asynchronous JavaScript and XML. It describes the use of JavaScript to load new content into the current page without having to reload the whole page. The process is called asynchronous because once the Ajax request has been sent to the web server, the browser does not have to wait for the answer in order to perform other tasks. The transferred content does not have to be formatted in XML: it can use JSON, HTML or plain text.

In effect, using Ajax makes it possible to build very dynamic websites

You can learn about the Ajax technique on the following sites:

- Wikipedia: http://en.wikipedia.org/wiki/Ajax_%28programming%29
- Mozilla Developer Network: https://developer.mozilla.org/en/docs/AJAX

About JSON

JavaScript Object Notation is the most used format when transferring data using the Ajax technique, for two main reasons: it is considered lighter than XML, and it can very easily be used by JavaScript, as it resembles a subset of that language. You can learn more about JSON on the following sites :

- http://json.org/
- http://en.wikipedia.org/wiki/JSON
- https://developer.mozilla.org/en-US/docs/JSON.

How to use Ajax in PrestaShop

By default, PrestaShop's controllers use the standard full-page-reload process.

Once PrestaShop detects the "ajax" parameter in a GET or POST query, the controller's `ajax` property is switched to true: `$this->ajax = true;`.

For instance, if your code triggers a URL like such: `http://...&ajax&action=updatelist` ...the controller will then execute the `displayAjaxUpdateList()` method if it exists. If it doesn't exist, it will execute the `displayAjax()` method (by default).

You therefore have to include the code that will take the Ajax call into account. Here is how you would write a simple Ajax query using jQuery:

```
var query = $.ajax({
  type: 'POST',
  url: baseDir + 'modules/mymodule/ajax.php',
  data: 'method=myMethod&id_data=' + $('#id_data').val(),
  dataType: 'json',
  success: function(json) {
    // ....
  }
});
```

And here is how the `ajax.php` script would work:

```
// Located in /modules/mymodule/ajax.php
require_once(dirname(__FILE__).'../../../config/config.inc.php');
require_once(dirname(__FILE__).'../../../init.php');
switch (Tools::getValue('method')) {
  case 'myMethod' :
    die( Tools::jsonEncode( array('result'=>'my_value'));
    break;
  default:
    exit;
}
exit;
```

As you can see in the code sample above, PrestaShop's `Tools` object contains `jsonEncode()`, a method that makes it easy to turn a PHP array into a JSON object:

```
public function displayAjax() {
  $return = array(
    'hasError' => true,
    'errors' => 'Ceci est le message'
  );
  die(Tools::jsonEncode($return));
}
```

jQuery's Ajax methods

jQuery.ajax(url [, settings])

Parameters:

- url: the URL to which the query should be sent
- settings: options and functions

Some options:

- `async` (boolean): default is true
- `type`: default is 'GET'
- `cache` (boolean): default is true
- `data`: GET array sent to the server
- `dataType`: either `xml`, `json`, `script` or `html`

Some functions:

- `beforeSend`: triggered before the Ajax call
- `error`: in case of error
- `success`: in case of success
- `timeout`: in case of timeout
- `complete`: triggered after the call's success or error, whatever the result

You can learn more about `ajax()` here: http://api.jquery.com/jQuery.ajax/

Alternative: jQuery.load(url [, data] [, complete(responseText, textStatus, XMLHttpRequest)])

This method is used to directly load the HTML result to an Ajax call, right within an element on the current page.

For instance:

```
$('#result').load('ajax/test.html', function() {
  alert('Load was performed.');
});
```

Alternative: jQuery.get(url [, data] [, success(data, textStatus, jqXHR)] [, dataType])

This method enables you to call an Ajax script with an HTTP GET query. It is equivalent to the following Ajax call:

```
$.ajax({
  url: url,
  data: data,
  success: success,
  dataType: dataType
});
```

Return values on a jQuery Ajax call

All these jQuery methods return a `jqHXR` object, which is an extension of the `XMLHTTPRequest` object. That object makes it possible to trigger various functions depending on the result of the call:

- Success: `jqXHR.done(function(data, textStatus, jqXHR) {});`
- Error: `jqXHR.fail(function(jqXHR, textStatus, errorThrown) {});`
- Success or Error: `jqXHR.always(function(data|jqXHR, textStatus, jqXHR|errorThrown){ });`
- Success AND Error: `jqXHR.then(function(data, textStatus, jqXHR) {}, function(jqXHR, textStatus, errorThrown) {});`

Here is an example with functions calls depending on the query's result:

```
// Assign handlers immediately after making the request,
// and remember the jqxhr object for this request
var jqxhr = $.get("example.php", function() {
  alert("success");
})
.done(function() { alert("second success"); })
.fail(function() { alert("error"); })
.always(function() { alert("finished"); });

// perform other work here ...
// Set another completion function for the request above
jqxhr.always(function(){ alert("second finished"); });
```