

Search Engine Friendly AJAX

Contributed by Hernâni
Wednesday, 16 April 2008

Lots of SEO experts are concerned about the raising of AJAX in web development. In my point of view, search engine operators should be the ones more concerned about that, because AJAX or the so called web 2.0 gives users a much better experience than with the "regular" web, and since search engines are providing a service to the community (and they earn lots of money with it), they should follow the tendencies and not try to dictate them. But that's a discussion I'll leave for the experts. My purpose with this post is to demonstrate my method.

The theory is, search engines only follow anchor tags, so, we must convert the anchor tags in ajax requests using javascript. Since google and its team mates doesn't read javascript, all it will see are regular links. Now we must find a way to identify each request, is it an ajax request, or a search engine request (or from a visitor without javascript enabled). That can be done with either a cookie, or with a request parameter. I personally prefer the request parameter option, since cookies is something that a visitor can have disabled. Then, on server side, based on that we must choose either to send only the desired part or the entire page. Here's a small example:

```
<html> <head> <script src="sack.js"></script> <script src="seo_ajax.js"></script> <title>The title</title> </head>
<body> <table> <tr> <td colspan="2"> <div id="header_content">The header content</div>
<tr> <tr> <td> <div id="menu"> <a class="ajax_link" href="link1.html">Menu Link 1</a>
class="ajax_link" href="link2.html">Menu Link 2</a> <a href="www.externalpages.com">External Page</a>
</div> </td> <td> <div id="center_content">Heres the index content</div> </td>
```

<div id="ajax_buffer" style="display: none"></div> </body></html>This page has three parts that change for each url. The page title, the header_content and the center_content. Then we have the menu that is the same for each page, and we identify the links that are meant to be converted with the class name ajax_link. We also have the script call, and the entire site structure that never changes. Finally we have a div called ajax_buffer, that is meant to temporarily store the response from our request, I'll come back to it later. Now we need to identify the ajax links and convert them, here's our seo_ajax.js content:

```
function AjaxLink(anchor) { var url = anchor.href; //store the anchor url var self =
this; addEvent(anchor, 'click', function() { //set the onclick event to make the request self.run(); //we use
addEvent function defined on // http://www.hrcerqueira.com/object-oriented-javascript-widgets/
anchor.href = 'javascript:void(0)'; //set the href to javascript: void(0), //so the anchor remains with
look and feel, but it won't do nothing addEvent(anchor, 'mouseover', function() { //a small cosmetic change, we don't
want that "javascript: void(0) return true; //shows up in the status bar }; this.run = function() { sa
Sack(); //instantiate a sack object, see description bellow sackObj.requestFile = url; //set it's url sackObj.method =
"POST"; //POST method sackObj.encVar('is_ajax', 'true'); //add a POST parameter, //telling th
request is meant to be read as an ajax request sackObj.onCompletion = function()
{self.requestComplete(sackObj.response);}; //on complete call the proper function sackObj.onFailed = function()
{self.requestError(sackObj.response);}; //same with fail and error sackObj.onError = function()
{self.requestError(sackObj.response);}; //put here any code that will warn the user of the request being started
sackObj.runAJAX(); //run the request } this.requestComplete = function(response) {
document.getElementById('ajax_buffer').innerHTML = response; //put the response on the "buffer"
distributeBufferContent(); //this function will leave for latter } this.requestError = function(response) { //put here any code
that will warn users of an error //use the response var to be a bit more specific with the error //example:
window.alert("There was an error with your request"); }} function convertLinks(el) { var ajax_anchors =
el.getElementsByTagName('A'); //get all anchors in a certain element var pattern = new RegExp('(^\s)ajax_link(\s|$)');
//a regexp to test the class name for (var i = 0; i < ajax_anchors.length, i++) { //for each anchor in the array if
(pattern.test(ajax_anchors[i].className)) { //if is a ajax_link new AjaxLink(ajax_anchors[i]); //convert it }
}} addEvent(window, 'load', function() { //convert the links after the page is loaded convertLinks(document.body);}); As you
see I use sack for my ajax requests. You can use whatever you feel like, but if you stick with sack you won't
regret, it's just wonderful. With this code, we can already make the requests, but we need to define what happens
on the server side. I'll use php for demonstration purposes, because it's simpler to understand.
Here's what happens when we click on Menu Link 1: $is_ajax = isset($_POST['is_ajax']) && $_POST['is_ajax'] ==
'true'; //is it an ajax request? $page_title = 'The new page title'; $header_content = 'Computed header content';
$center_content = 'Computed center content'; if ($is_ajax) { echo <<<RESULT<div id="ajax_title">$page_title</div><div
id="ajax_header_content">$header_content</div><div id="ajax_center_content">$center_content</div>RESULT; } else
{echo <<<RESULT<html> <head> <script src="sack.js"></script> <script src="seo_ajax.js"></script>
<title>$page_title</title> </head> <body> <table> <tr> <td colspan="2"> <div
id="header_content">$header_content</div> </td> <tr> <tr> <td> <div id="me
class="ajax_link" href="link1.html">Menu Link 1</a> <a class="ajax_link" href="link2.html">Menu Link 2</a>
<a href="www.externalpages.com">External Page</a> </div> </td> <td> <div id="me
id="center_content">$center_content</div> </td> <tr> </table> <div id="ajax_buffer" style="display:
none"></div> </body></html>RESULT; }I guess this code is pretty self explanatory. It's not written thinking about
performance and usability and other stuff, but with demonstration purposes. The important thing to retain here is that if
the request is not an ajax request we retrieve the entire page, otherwise we retrieve only the new parts wrapped in
```

div's or any other element properly identified for each part we want to change on the client side. Now we need to write the distributeBufferContent that will pick up the new content and distribute it by the page:

```
function distributeBufferContent() {
    var title = document.getElementById('ajax_title').innerHTML;
    var headerContent = document.getElementById('ajax_header_content').innerHTML;
    var centerContent = document.getElementById('ajax_center_content').innerHTML;
    if (title) //just to be sure if we defined the new title
        document.title = title;
    if (headerContent) //same here
        document.getElementById('header_content').innerHTML = headerContent;
    if (centerContent) //and again
        document.getElementById('center_content').innerHTML = centerContent;
    convertLinks(document.getElementById('header_content'));
    convertLinks(document.getElementById('center_content'));
}
```

There is some code repeating on this function, but I think is good to demonstrate what it's meant to do. There's just one thing to be careful about. Remember that the newcomer links must have absolute paths, or they can be relative paths, but they need to be relative to the page that is already in browser and not to the script we just call. Everything should work fine, but we still have a problem. What about the bookmarks and the back button? Let's use hash signs (#) for that, and make a small retouch to our AjaxLink object:

```
function AjaxLink(anchor) {
    var url = anchor.href;
    var self = this;
    addEvent(anchor, 'click', function() {
        self.run();
    });
    //nothing new until here
    anchor.href = '#' + url; //instead of javascript: void(0), we set it as if it was a page link
    /**addEvent(anchor, 'mouseover', function() {
        return true;
    });*/ //we can remove this, or keep it, your choice
    //the rest remains the same
    Now the bookmarks should work, and the back button should do nothing. But if we visit the bookmark, we are still visiting the original page. So we need to find a way of retrieving the content users want to see. Let's change our page "onload" method:
    addEvent(window, 'load', function() {
        //convert the links after the page is loaded
        convertLinks(document.body);
        var loc = new String(window.location);
        var url = document.URL;
        var bookmarkedUrl = loc.substr(url.length + 1, loc.length - url.length); //we just want the url after the hash sign
        //I should have designed the AjaxLink object differently, but I was not thinking about this
        //so, I'll just create one with a fake anchor tag and run it
        var fakeAnchor = {};
        fakeAnchor.href = bookmarkedUrl;
        new AjaxLink(fakeAnchor).run();
    });
    That's good now, just missing the back button feature, right now it doesn't do nothing. I don't have an answer to that yet, the best solution I can think of is to provide some kind of self made browser history with the ajax links and a back button on the page which calls a javascript function that reads the self made history and does it's "getting back" job. If anyone has a good solution please post a comment here and let me know. I hope that my poor english was enough to explain my theory. I hadn't tested the code I've written here, but I think it should work fine. Maybe some misplaced character or something, but if you want to test it you should find those "bugs" quickly. Hope this helps someone. Cheers,

```