Using AJAX to Serve Dynamic Web Content on the NetBurner Platform

©2006 Tod Gentille

Syncor Systems, Inc.

http://www.syncorsystems.com

Table of Contents

Table of Contents 2
Revision History: 2
Abstract 3
AJAX In a nutshell 3
The Server Request 3
The Client Request Object 3
HTML 6
Final Considerations and Further reading 7

Revision History:

First Publication: May 16, 2006

Revision 1: May 23, 2006

Copyright 2006 Syncor Systems, Inc.

Abstract

A NetBurner solution will often monitor dynamic conditions like temperature and voltage. It is often desirable to see an updated view of this dynamic state data without manually refreshing the web client. It is a simple matter to make a web page dynamically refresh. However, the standard technique of using a metatag or JavaScript timeout setting causes the entire page to flash, which is often very distracting. This white paper describes a technique using AJAX that can keep sections of a page up to date in a seamless manner. This paper is accompanied by a zipped DevC++ project containing all the code.

AJAX In a nutshell

For our purposes we want to have the HTML page issue a request to the server, have the page receive the answer from the server asynchronously and display the received data dynamically. The request is created and responded to using JavaScript and the browser's DOM. It has nothing to do with the NetBurner and is no different from doing AJAX on any platform. (I should note that I won't actually be using XML in this example.) The actual request to the server is NetBurner specific and is accomplished using the FUNCTIONCALL tag.

The Server Request

Let's start with the server request code because it is very simple and we can get it out of the way. Create a page called ajax.htm with the content shown in Example 1.This will call a C method named AjaxCallback in our NetBurner. This is the equivalent of a cgi call on a more traditional web server.

```
<!--FUNCTIONCALL AjaxCallback -->
```

Example 1. The Entire Server Request page

Now create a C method in the NetBurner named AjaxCallback as shown in Example 2. This code writes out a simple string with the always changing TimeTick value. Not too useful but useful enough for testing and demonstration purposes.

```
//===========================================================================
// Example function that can be called via AJAX. This one just writes out the
// the current timetick value but of course this could be much more elaborate.
//===========================================================================
void AjaxCallback(int sock, const char* url)
{
  char java_script[100];
  sprintf(java_script, " System Tick count:%u", TimeTick);

  writestring(sock, java_script );
}
```

Example2. Netburner method that responds to AJAX request

The Client Request Object

If all web browsers worked the same this would be easy. However, you get paid the big bucks because vendors can't agree on standards. The solution is to query the DOM first and then issue

 Copyright 2006 Syncor Systems, Inc.

the proper request based on what the DOM supports. I prefer to keep all my JavaScript external to my html pages. Example 3 contains a sample method from SupportScripts.js that should work with most of the major browsers.

```
function CreateRequestObject()
{
	var request_obj;
	if(window.XMLHttpRequest)
	{
		// pretty much all browswer except IE support this
		request_obj = new XMLHttpRequest();
	}
	else if(window.ActiveXObject)
	{
		// Internet Explorer 5+
		request_obj = new ActiveXObject("Microsoft.XMLHTTP");
	}
	else if (window.createRequest)
	{
		request_obj = window.createRequest();
	}
	else
	{
		// There is an error creating the object,
		// just as an old browser is being used.
		alert('Problem creating the XMLHttpRequest object. Dynamic udpating will not work with this browser.');
	}
	return request_obj;
}
```

Example 3. Multiple Browser Request Object

The CreateRequestObject() can be called inline right at the top of our SupportScripts.js file as shown in Example 4.

```
//===============================================================================
// Executable code that runs when the page that includes this file loads.
//===============================================================================
// create the AJAX http request object with global scope.
var requestObj = CreateRequestObject();
```

Example 4. Inline call to CreateRequestObject()

We now have a generic request object. We have to set some properties of this object to get it to do the work. CallbackNetburner shown in Example 5 will set the open and onreadystatechange properties. Notice that the open property specifies 'ajax.htm' (see Example 1) as our server request.

Copyright 2006 Syncor Systems, Inc.

```
//============================================================================
// CallbackNetburner uses http.open to callback to the netburner via the ajax.htm
// page which just contains a single line of html with the netburner function
// we want to call
//============================================================================
function CallbackNetburner()
{
	requestObj.open('get','ajax.htm'); //invoke the desired function on the netburner
	//by sending the htm page that has the <!--FUNCTIONCALL AjaxCallback -->
	//The netburner method AjaxCallback() is in webformcode.cpp
	//Tell the ProcessAjaxResponse method to handle the returned data
	requestObj.onreadystatechange = ProcessAjaxResponse;
	requestObj.send(null);
}
```

Example 5. Making the request object do useful work.

Notice that the onreadystatechange property is calling ProcessAjaxResponse. This is where we set up the code that returns data to our HTML page. The code is returned by setting the innerHTML property of a DIV on our HTML. This DIV has an id of "ajaxTest" as we will see in the next section when we examine the HTML.

```
//============================================================================
// When the netburner writes data out the socket this method will get invoked
// when it is finished so we can retrieve the data.
//============================================================================
function ProcessAjaxResponse()
{
	var REQUEST_DONE = 4;
	var STATUS_NORMAL = 200;
	if((requestObj.readyState == REQUEST_DONE)&& (requestObj.status == STATUS_NORMAL)){
		var answer = requestObj.responseText;
		//ajaxText is the id of a DIV tag in index.htm
		document.getElementById('ajaxTest').innerHTML = answer;
	}
	else
	{
		document.getElementById('ajaxTest').innerHTML = "Error in ProcessAjaxResponse";

	}
}
```

Example 6. Sending the data back to the HTML page.

That's all the AJAX specific code we need. We now need to tie all this to the form and have it execute via a timer. There are many ways to do but I want to show an example of how to do it using the Observer design pattern and keeping all the JavaScript in the external file. Anyone familiar with C# delegates will find this familiar.

Copyright 2006 Syncor Systems, Inc.

```
//==============================================================================
// Executable code that runs when the page that includes this file loads.
//==============================================================================
// create the AJAX http request object with global scope.
var requestObj = CreateRequestObject();
//Using the Observer design pattern, add the doLoad function as the
//window.load event keeping any other event that might already be there
addEvent(window, 'load', doLoad);


//==============================================================================
// General purpose way to add events to objects without overwriting any
// existing events that might be on that object.
//==============================================================================
function addEvent(obj, eventType, functionToRun){
 if (obj.addEventListener){
   obj.addEventListener(eventType, functionToRun, true);
   return true;
 } else if (obj.attachEvent){
   var r = obj.attachEvent("on"+eventType, functionToRun);
   return r;
 } else {
   return false;
 }
}


//==============================================================================
// When the form loads set up all the GUI elements to reflect the current
// state of the netburner
//==============================================================================
function doLoad()
{
   StartClock();
}
//==============================================================================
// Set up a timer so that the callback to the netburner happens every 5 seconds.
//==============================================================================
var RESET_TIMER_VALUE = 5;
var timerSecs_ = 5;
function StartClock()
{
        --timerSecs_;
        setTimeout("StartClock()",1000); // delay one second
        if(timerSecs_==0)
        {
                timerSecs_ = RESET_TIMER_VALUE;
                CallbackNetburner();
        }
}
```
Example 7. Tying it all together.

Here we tell the page to run doLoad() at startup, doLoad() kicks off a timer. Every 5 seconds this timer will call CallbackNetburner() which sets the properties for our request object. Once the request is satisfied by the NetBurner, ProcessAjaxResponse() is called and our page is updated with the new information.

The final step is to look at our HTML page.

HTML

The HTML is very straightforward. All the work is done in our SupportScripts.js file so the only JavaScript you will see is the include for that file.

Copyright 2006 Syncor Systems, Inc.

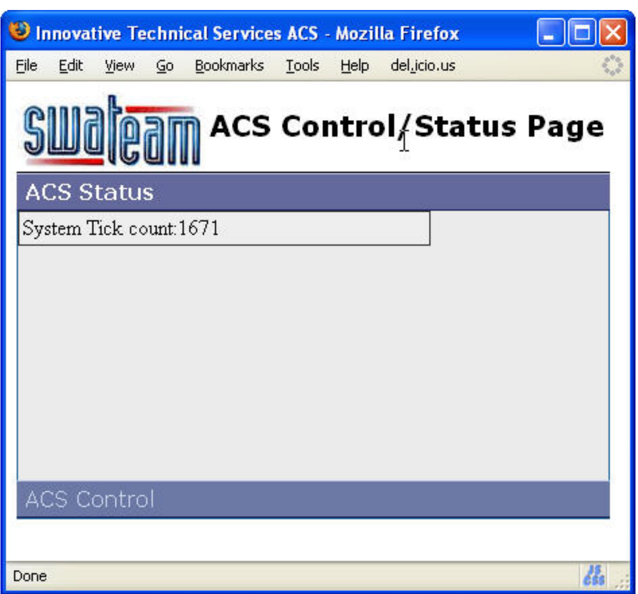
```
<html>

<head>
<title>Syncor Systems Inc.  Netbuner Main Page</title>
<script src="SupportScripts.js"></script>
<script language="javascript">
<LINK REL=stylesheet TYPE="text/css" HREF=styles.css>

</head>
<body>
<IMG SRC="swatteam_s.jpg" BORDER= "0">
<h1 class=TitleRight>Syncor Systems, Inc.</h1>

<!-- we want to let the user see an error message if they don't support scripting.
-->
<DIV ID="MainPage">

<noscript>
<H1>This netburner requires JavaScript to be enabled to run properly.</H1>
</noscript>

<DIV ID='ajaxTest'>
Waiting for Netburner callback...
</DIV>
</DIV>
</body>
</html>
```

Example 8. The HTML page .

Final Considerations and Further reading

You can save yourself some effort by using some of the dozens of libraries and AJAX frameworks that are available on the web. Two that I have used are the open source prototype at http://www.prototype.conio.net/ and the OpenRico framework (which requires prototype) at http://openrico.org.

If you really want to get into AJAX and the newer JSON syntax of JavaScript I found *Ajax In Action* by Crane and Pascarello very helpful (including Appendix B).



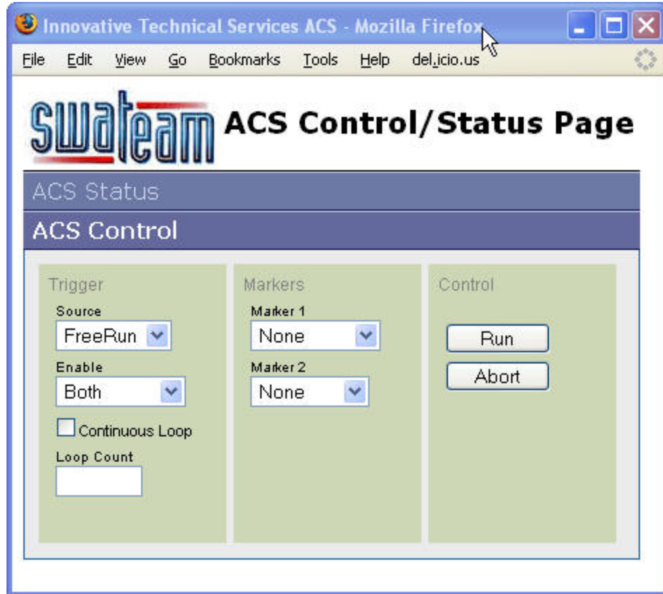


Figure 2. Same web app using Rico accordion.

Copyright 2006 Syncor Systems, Inc.