



Serving Dynamic NetBurner Pages (without a full page refresh)

Tod Gentile

Syncor Systems, Inc.



Table of Contents

Abstract.....	3
Frames.....	3
JavaScript.....	3
HTML.....	4
NetBurner Code	5
JavaScript Revisited.....	6
HTML Revisited.....	8
Conclusion	9
Remaining NetBurner Code that interacts with HTML.....	9

Revision History:

First Publication: February 8, 2006



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 frames that minimizes that distraction. Also this paper will show one technique for keeping the web page up to date with user submitted data changes.

Frames

The general solution involves creating a two frame web page where one frame is “hidden” and the other frame displays all the data. The starting index.html page for your NetBurner solution should look like the code shown in Example 1.

Note that in the example, to make the border as hidden as possible, we set the border of the frameset to 0. Also, since Internet Explorer allows hidden borders to be resized the NORESIZE option was used on the frame row that holds the hidden frame.

```
<HTML>
<HEAD>
<TITLE>Syncor Systems Netburner Frameset</TITLE>
</HEAD>
<FRAMESET BORDER = "0", ROWS = "0,100%">
<FRAME NAME="hidden" NORESIZE SRC ="hidden.html">
<FRAME NAME="netburner" SRC = "mainpage.html">
</FRAMESET>
</NOFRAMES>
<BODY>
    This NetBurner interface requires the use of frames. You need to use a browser
that supports and has frames enabled.
</BODY>
</NOFRAMES>
```

Example 1. Index.html Frameset Page

JavaScript

All of the scripting is going to take place in frame[0] which is named “hidden”. All of the displayed information will be shown in frame[1] which is named “netburner”. JavaScript can refer to frames by array index or name, in our code we use the names. The basic approach is to have a NetBurner callback write all the state information into a structured object with properties on the hidden frame. Then scripts will use that object to dynamically set the values of form widgets on the page. Before we get into the JavaScript lets look at the HTML.

HTML

The HTML for the “hidden” frame is shown in Example 2. Notice the NetBurner callback to WriteSystemVarsToJavaScript. This creates our JavaScript object with properties that we use in the “doLoad()” function which is called from the <body> tag.

An example of the web page is shown in Figure 1. The HTML for the “netburner” frame is shown in Example 3. It’s worth noting that we are dynamically updating the user entry widgets. This serves our purpose for demonstrating the technique but normally you would only want to update user interface elements on a Submit, not on a timeout. Otherwise the user can be in the middle of changing a value and the timeout will cause it to revert to its old value. In our case we actually want that to happen as one way of seeing that our technique is working. In your production application separate the state data from the user entry data.

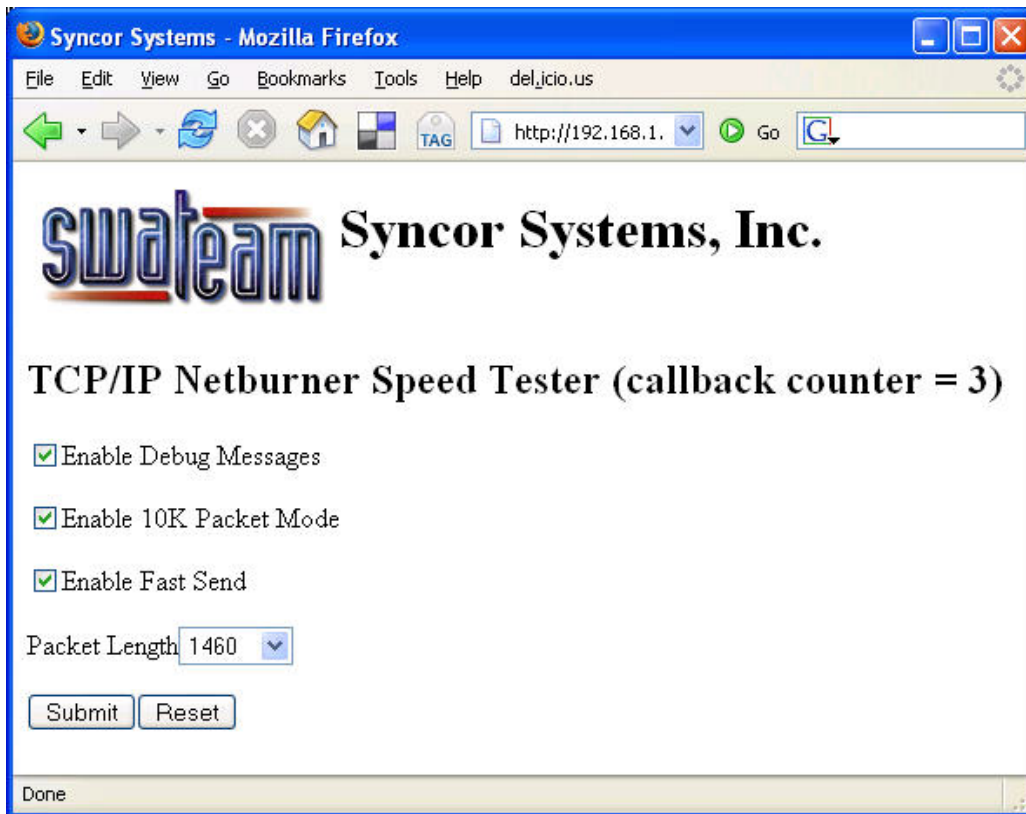


Figure 1. Sample Netburner Web page.

Other form widgets can easily be added and manipulated in a similar manner to what is shown here. One interesting bit of HTML to notice is that the Form uses the target attribute and sets it to “_top”. If it didn’t do this then our NetBurner code that redirects us back to this page after the Submit would end up loading a frame inside a frame. It wouldn’t have a visible effect because borders are set to 0 but it is much cleaner to just load the redirect into the top.

```

<HTML>
<HEAD>
  <TITLE> Syncor Systems, Inc. Netburner Interface
</TITLE>
<script language="javascript">
  <!--FUNCTIONCALL WriteSystemVarsToJavaScript -->
</script>
<script src="SupportScripts.js"></script>
</HEAD>
<body onload="doLoad()">
<BODY>
</HTML>

```

Example 2. HTML Code for “hidden” frame.

```

<html>
<head>
<title>Syncor Systems Inc. Netburner Main Page</title>
</head>
<body>
<table >
  <tr>
    <td ><IMG SRC="swatteam.bmp" BORDER= "0"></td>
    <td ><h1>Syncor Systems, Inc.</h1></td>
  </tr>
</table>
<noscript>
<H1>This netburner requires JavaScript to be enabled to run properly.</H1>
</noscript>
<!-- Just for fun to see the difference between dynamic updating of the page and refreshing -->
<!--FUNCTIONCALL DoSampleCallback -->
<form target="_top" method="POST" >
  <p><input type="checkbox" name="ckb_debug" >Enable Debug Messages</p>
  <p><input type="checkbox" name="ckb_10K" >Enable 10K Packet Mode</p>
  <p><input type="checkbox" name="ckb_fastsend" >Enable Fast Send</p>
  <p>Packet Length<select size="1" name="cmb_packet">
    <option>1460</option>
    <option>4096</option>
    <option>4380</option>
    <option>7300</option>
    <option>8192</option>
    <option>16060</option>
  </select> </p>
  <p><input type="submit" value="Submit" name="B1"><input type="reset" value="Reset"
name="B2"></p>
</form>
</body>
</html>

```

Example 3. HTML for “netburner” frame.

NetBurner Code

For the sake of completeness I have included all the NetBurner code that interacts with the Web page at the end of this document. You probably have your own post handler and there is nothing special in most of this code. The key method for the technique we are discussing is WriteSystemVarsToJavaScript and that is shown here.

```

//=====
// Send hidden info from the server to the browser. Used to allow javascripts
// to setup gui elements to values that reflect the state of the machine.
// Data is stored in an object with properties using object literal syntax like
// var machineState_ =
// {
//     packetLength: 4096,
//     debugging: true,
//     fastSendState: true,
//     send10Kpackets: false
// };
//=====
void WriteSystemVarsToJavaScript(int sock, PCSTR url)
{
    char java_script[100];
    //declare the property strings - these must match exactly in the javascript
    //that uses them.
    const char MACHINE_STATE_VAR[] = "machineState_";
    const char PACKET_LENGTH_PROP[] = "packetLength";
    const char DEBUGGING_PROP[] = "debugging";
    const char FAST_SEND_PROP[] = "fastSendState";
    const char SEND_10K_PROP[] = "send10Kpackets";

    const char TRUE_STR[] = "true";
    const char FALSE_STR[] = "false";

    siprintf (java_script,"var %s =\n{\n",MACHINE_STATE_VAR);
    writestring (sock, java_script);
    siprintf (java_script,"\t %s: %d,\n",PACKET_LENGTH_PROP,packetLength_);
    writestring (sock, java_script);
    siprintf (java_script,"\t %s: %s,\n",DEBUGGING_PROP, DEBUGGING_? TRUE_STR:FALSE_STR);
    writestring (sock, java_script);
    siprintf (java_script,"\t %s: %s,\n",FAST_SEND_PROP, fastSendState_? TRUE_STR:FALSE_STR);
    writestring (sock, java_script);
    //note no trailing comma and a trailing close curly brace
    sprintf (java_script,"\t %s: %s\n;",SEND_10K_PROP,packets10k_? TRUE_STR:FALSE_STR);
    writestring (sock, java_script)}

```

JavaScript Revisited

Now that we have our machineState_ object and properties available on our web page we can look at how to use it in JavaScript. First we are going to call doLoad() when the page loads. This script has two functions. The first is to setup a timeout to automatically call our refresh() method every 2 seconds. The second is to call selectPacketLength() and setupCheckboxes() to make the form reflect the current state of the NetBurner. Note that refresh() doesn't update the entire page instead it just reloads the hidden frame using

```
parent.hidden.location.reload();
```

When this happens if nothing needs to be updated the user will see very little activity. If something does happen the user will only see the appropriate widget change value, there will not be a "flash" of the entire page.

```

//=====
// Javascript for Netburner Support
// DESCRIPTION: This file contains the basic code showing how to write state
// data from the netburner to a hidden web page frame and then use that data
// to update the widgets on the frame that the user sees. We can set a timeout
// and refresh the hidden frame which will then update any changes to the
// non-hidden frame.
// The machineState_ object is written by Netburner method WriteSystemVarsToJavaScript()
// You will see a call to that method in hidden.html
// Copyright 2006 Tod Gentile Syncor Systems, Inc.
//=====

//=====
// When the form loads set up all the GUI elements to reflect the current
// state of the NetBurner
//=====
function doLoad()
{
    // the timeout value should be the same as in the "refresh" meta-tag
    // update once every 2 seconds.
    setTimeout ("refresh()", 2*1000);
    selectPacketLength();
    setupCheckboxes();
}

//=====
// Loop through the select options and when the text matches the value written
// out to the machineState_ object set the selected property to true. This will
// make that option show in the popup.
//=====
function selectPacketLength()
{
    //the user visible frame is called "netburner" and we use it in the var
    // below, we also could have used frames[1] instead
    var packet_list = parent.netburner.document.forms[0].cmb_packet;
    //pkt_len = document.forms[0].packetLength.value;

    for (var idx = 0; idx < packet_list.options.length; ++idx)
    {
        if (packet_list.options[idx].text == machineState_.packetLength)
        {
            packet_list.options[idx].selected = true;
        }
        else packet_list.options[idx].selected = false;
    }
}

//=====
// Check the machineState_ settings and enable the corresponding checkbox
// if the corresponding machineState_ setting is true.
// Checkboxes are not checked by default
//=====
function setupCheckboxes()
{
    //instead of .netburner. could have used .frames[1]
    var the_form = parent.netburner.document.forms[0];
    var ms = machineState_;
    if (ms.debugging) the_form.ckb_debug.checked = true;
    else the_form.ckb_debug.checked = false;

    if (ms.fastSendState) the_form.ckb_fastsend.checked = true;
    else the_form.ckb_fastsend.checked = false;

    if (ms.send10Kpackets) the_form.ckb_10K.checked = true;
    else the_form.ckb_10K.checked = false;
}

//=====
// Refresh just the hidden frame when the timeout counter expires

```

```
//=====
function refresh()
{
    parent.hidden.location.reload();
}
```

HTML Revisited

There is still some activity happening on the web browser that is less than desirable. First the status bar displays the progress bar every time the page refreshes. Also, on FireFox, the Cancel button flashes and a slight animation occurs in the upper right hand corner. Internet Explorer doesn't have these latter two issues, nevertheless, it is probably a good idea to limit these problems. In fact most embedded apps don't need most of the window decorations that come along with a browser window so let's get rid of them. Replace the index.htm page with an intermediate page that opens a new window and loads our frameset into that new window. Rename the current index.htm page frameset.htm and create the following new index.htm page.

```
<HTML>
<HEAD>
    <TITLE> Syncor Systems, Inc. Netburner Interface</TITLE>
<script language="javascript">
function doLoad()
{
var w = window.open("frameset.htm", "statuswin", "width= 400, height=400, status = no, resizable
=no");
}
</script>
</HEAD>
<body onload="doLoad()">
<body>
</HTML>
```

New Index.htm page

The only remaining problem seems to be that the cursor can flash between the standard cursor and the wait cursor with the hourglass. The behavior varies by browser. If you find a solution to this let me know. The final browser window is shown below.



Conclusion

You may have noticed that in the example code there isn't really anything dynamic going on that requires a constant update. In my code, the NetBurner also has a process that handles incoming Ethernet messages and changes the state of the variables shown here. I wrote a simple PC GUI to send those change commands. I then opened up two browser windows in FireFox and IE and watched the screens change in real time. I can also just make changes to the user interface and not hit submit. The refresh will cause the old values to be displayed. During testing, I also turned off the refresh to make sure the Submit works correctly without a refresh. I hope you found something helpful here. If you have better ideas that you would like to share please write to me at tod_02@syncorsystems.com.

Remaining NetBurner Code that interacts with HTML

For the sake of completeness here is the rest of the NetBurner code that is called from callbacks or from form submission actions that hasn't been shown previously. I tend to gather all my form related code into a module named after the page where it is used. Since this application has only one form all this code is in a single module.

```
//=====
// This page holds the C++ code that is exposed to the HTML pages for use
// as callbacks. We minimize the callbacks by using javascript when possible.
// One of the primary purposes of this code is to write the machine state out
// to the web page using a javascript object with properties. Then the javascript
```

```

// can analyze that info to customize the HTML to reflect the machine state.
//=====
//All of the functions exposed to the HTML subsystem must be extern C
//so that the names don't get mangled.
extern "C"
{
    void DoSampleCallback( int sock, PCSTR url );
    void WriteSystemVarsToJavaScript(int sock, PCSTR url);
};

//=====
// Just a counter so that we can easily see in the web page every time
// a callback refresh is made.
//=====
void DoSampleCallback( int sock, PCSTR url )
{
    static int counter = 0;
    char display_string[80];
    sprintf(display_string, "<h2>TCP/IP Netburner Speed Tester (callback counter =
%d)</h2>", ++counter);
    writestring( sock, display_string );
    //if we wanted to write safe escaped HTML we would use writesafestring()
}

//=====
// Post Handler - When the user submits a form this method is called
// Set up using SetNewPostHandler() via the RegisterPost() method
//=====
int MyDoPost( int sock, char *url, char *pData, char *rxBuffer )
{
    //In a Real application one would likely have multiple forms.
    //one would then select the proper processing function by dispatching based
    //on the url data member.
    //In this case we will ignore that....
    const int MAX_CB_RET_LEN = 3; //Checkboxes don't even exist in returned post if the box is not
    enabled.
    const int MAX_PACKET_SIZE_LEN = 6; //65536 wold be our biggest value so make it one extra
    //Declare the strings for all the HTML interface widget names
    const char PACKET_10K_CB [] = "ckb_10K";
    const char DEBUG_CB[] = "ckb_debug";
    const char FASTSEND_CB[] = "ckb_fastsend";
    const char PACKET_SIZE_CMB[] = "cmb_packet";
    const char REDIRECT_PAGE[] = "index.htm";

    //ExtractPostData returns -1 if field not found, 0 if found but empty
    char checkbox_data[MAX_CB_RET_LEN];
    char packet_size[MAX_PACKET_SIZE_LEN];

    if (ExtractPostData( DEBUG_CB, pData, checkbox_data, sizeof(checkbox_data) ) != -1 )
    {
        //in the case of a checkbox, the actual value doesn't matter if it shows up then
        //it is checked otherwise it doesn't show up. The actual value would be "on" if
        //we wanted to check it.debug_checkbox
        DEBUGGING_ = true;
    }
    else DEBUGGING_ = false;

    if (ExtractPostData( PACKET_10K_CB, pData, checkbox_data, sizeof(checkbox_data) ) != -1 )
    {
        packets10k_ = true;
    }
    else packets10k_ = false;

    if (ExtractPostData( FASTSEND_CB, pData, checkbox_data, sizeof(checkbox_data) ) != -1 )
    {
        fastSendState_ = true;
    }
}

```

```
else fastSendState_ = false;

if (ExtractPostData( PACKET_SIZE_CMB, pData, packet_size, sizeof(packet_size) ) != -1 )
{
    int packet_length = atoi(packet_size);
    packetLength_ = packet_length;
    fprintf("Packet Length from web: %d\n",packetLength_);
}
else
{
    fprintf( "Error in extracting text from packet size popup.\r\n" );
}
//We have to respond to the post with a new HTML page...
//In this case we will redirect so the browser will go to that URL for the response...
RedirectResponse( sock, REDIRECT_PAGE );

return 0;
}

//=====
// Install the method we want to have handle form submissions
//=====
void RegisterPost()
{
    SetNewPostHandler( MyDoPost );
}
```