JavaScript and pbuilder compression

1 Document History

Date	Version	Change Description	
1/20/2014	V1.0	Initial Entry	
1/22/2014	V1.1	Add additional content	
1/23/2014	V1.2	Add more content	
1/29/2014	V1.3	Cleanup	
1/29/2014	V1.4	Fill in glossary section	
1/29/2014	V1.5	Add Bubba's comments	

2 Table of Contents

3 Introduction

This paper describes a method for suppressing pbuilder utility tag compression in JavaScript files.

3.1 Problem Solved

The pbuilder utility compresses certain HTML tag sequences into 1 byte tokens. This is used to save RAM space when dealing with large web pages. The problem is that in some cases when the tokens are uncompressed and turned back into HTML tags, line feeds are added. In certain instances that we will explore, these line feeds can break JavaScript files. We will look at identifying the problem and we will discuss addressing the issue.

3.2 Audience

This paper is geared for software developers with knowledge of HTML and JavaScript. Additionally the user of this paper should have had experience developing web applications using the NET+OS advanced web server (AWS) component including the pbuilder utility.

3.3 Scope

This paper has a very narrow focus. This paper looks specifically at the pbuilder utility's compression of HTML tags, how this can negatively affect JavaScript files and how to overcome this issue.

This paper does not address any of the following topics:

- Developing web applications within the NET+OS development environment
- Using the pbuilder utility
- HTML coding
- C coding
- JavaScript coding
- Anything else not described in the first paragraph of this section.

3.4 Theory of Operation

This paper will first describe what to look for in JavaScript files that are adversely affected by the compression actions of the pbuilder utility. Following that this paper will demonstrate a method we recommend for shutting off compression for JavaScript files.

4 Handling pbuilder utility compression of JavaScript files

The primary purpose of the pbuilder utility is to convert files used within a web application into C code for compilation into your application. The converted files could be HTML, JavaScript, jpeg or many other formats. A second purpose of the pbuilder

utility is to parse through comment tags and generate stub functions. The stub functions allow the web pages (browser) to have access to device data. A third purpose of the pbuilder utility is to compress HTML tags. The HTML tags are converted into one byte tags. The tags are uncompressed by the advanced web server engine when the final HTML is generated and sent back to the browser. But while stored in FLASH or RAM, they are in compressed form.

4.1 The application we will use for the example.

The application we will use for our example is made up of a simple HTML page and a JavaScript file. The HTML file includes an anchor with an href attribute that points to an onClick attribute. The value of the onClick attribute is a call to a function contained within the JavaScript file. Additionally the HTML file contains two fields (text) and a submit button.

The JavaScript file creates a new web page and fills in information into the form entry of one of the fields of the HTML file. The real purpose of the JavaScript file is to demonstrate how pbuilder utility can adversely affect a JavaScript file and how to avoid the issue. By having the JavaScript file modify the field of one of the HTML entry points we can show that the JavaScript function is being called. That is, it is for demonstration purposes only.

Get example code here.

4.2 The JavaScript file

```
var myPopupWindow;
var myWindowsDoc;
var anInteger = 0

function makeMeAWindow()
{
    myPopupWindow = window.open("new_window.html", "thePopupWindow",
"height=500, width=500, left=100, top=100, resizable=yes,
scrollbars=yes,toolbar=yes,menubar=no,location=no, status=yes");
    myWindowsDoc = myPopupWindow.document;
    myWindowsDoc.open();
    myWindowsDoc.writeln("<html><head><title>");
    myWindowsDoc.writeln("My Document Title");
    myWindowsDoc.writeln("</title>");
    myWindowsDoc.writeln("</head>");
    myWindowsDoc.writeln("<body>");
    myWindowsDoc.writeln("Hello from javascript");
    myWindowsDoc.writeln("<p>");
    myWindowsDoc.writeln("This page was created on the fly");
    myWindowsDoc.writeln("by having a javascript function");
    myWindowsDoc.writeln("write HTML to a newly created window");
    myWindowsDoc.writeln("<p>");
    myWindowsDoc.writeln("An onClick associated with an href and an
");
    myWindowsDoc.writeln("anchor tag initiated the function call");
    myWindowsDoc.writeln("</body>");
    myWindowsDoc.writeln("</html>");
        theNameElement=document.getElementsByName("theNameField");
        anInteger = anInteger + 1;
        if(theNameElement != null)
        {
           theNameElement[0].value="javascript filled in this field  "
+ anInteger +" times";
        }
    myWindowsDoc.close();
    //myPopupWindow.close();
}
```

As you can see, the JavaScript file is kept simple so as not to obscure the message we are trying to pass along. One thing I need to mention is that the pbuilder utility will not pass the id attribute/value pair (used in HTML files to add identity to a particular object) into the final C code. Thus in the JavaScript file I use, getElementsByName as opposed to getElementById.

4.3 The HTML File

```
<HTML>
<HEAD>
<TITLE>A Test Page</TITLE>
<!-- a javascript file to put up a small window -->
<script language="javascript" type="text/javascript" src="simple_javascript_newwin.js">
</script>
</HEAD>
<BODY>
Hello from HTML
<P>
<!-- a live link that calls a javascript function when clicked -->
<A HREF="javascript:void(0);" onclick="makeMeAWindow(); return false;">Make me a window</a>
<form method="post">
Enter a name:
<input type="text" size="64" maxlength="64" name="theNameField" id="theNameField"/>
<p>
Enter Street Address:
<input type="text" size="64" maxlength="64" name="theStreetAddressField" id="theStreetAddressField"/>
<p>
<input type="submit" value="submit"/>
</form>
</BODY>
</HTML>
```

As in the case of the JavaScript file, this HTML file is kept simple. Also notice that this is shown before any pbuilder utility comment tags are added for accessing device data. pbuilder utility comment tags have no bearing on the pbuilder utility's processing of JavaScript files.

Also I am showing the id attribute in the HTML file. They are allowed by HTML but will not be passed through the pbuilder utility to the resultant .c file.

4.4 Running the pbuilder utility against raw JavaScript

In the example below, I have run the HTML file and the JavaScript file through the pbuilder utility. As a first pass I will compare interesting parts of the JavaScript file against the pbuilder utility output. The exhibit directly below is a portion of the raw JavaScript file.

The following is some of the "raw" javascript file:

```
    myPopupWindow = window.open("new_window.html", "thePopupWindow",
"height=500, width=500, left=100, top=100, resizable=yes,
scrollbars=yes,toolbar=yes,menubar=no,location=no, status=yes");
    myWindowsDoc = myPopupWindow.document;
    myWindowsDoc.open();
    myWindowsDoc.writeln("<html><head><title>");
    myWindowsDoc.writeln("My Document Title");
    myWindowsDoc.writeln("</title>");
    myWindowsDoc.writeln("</head>");
    myWindowsDoc.writeln("<body>");
    myWindowsDoc.writeln("Hello from javascript");
    myWindowsDoc.writeln("<p>");
    myWindowsDoc.writeln("This page was created on the fly");
    myWindowsDoc.writeln("by having a javascript function");
    myWindowsDoc.writeln("write HTML to a newly created window");
    myWindowsDoc.writeln("<p>");
    myWindowsDoc.writeln("An onClick associated with an href and an ");
    myWindowsDoc.writeln("anchor tag initiated the function call");
    myWindowsDoc.writeln("</body>");
    myWindowsDoc.writeln("</html>");
```

Pre-compression HTML tags

I have encircled some HTML tags that are contained in the JavaScript code. The HTML tags are used for building a web page on the fly.

This is the same JavaScript code but shown after being run through the pbuilder utility.

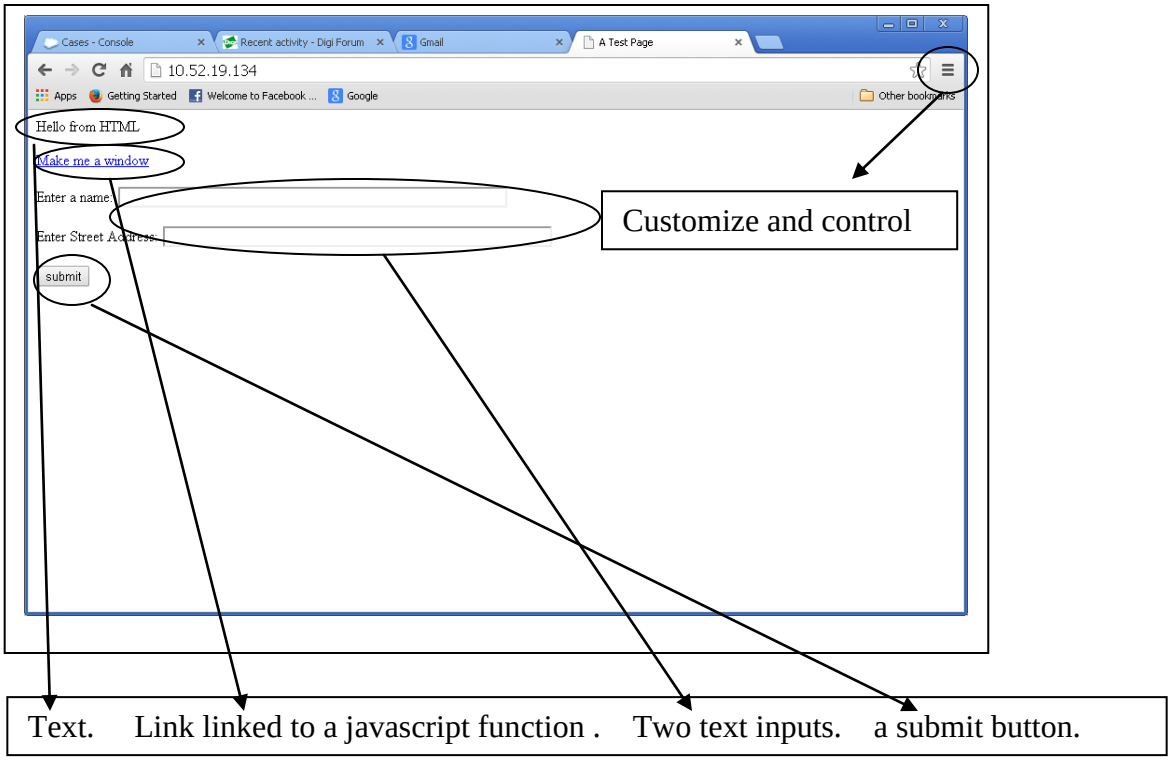
```
      "myPopupWindow = window.open(\"new_window.html\", \"thePopupWindow\",
"
      "\"height=500," C_WIDTH "500, left=100, top=100, resizable=yes, "
      "scrollbars=yes,toolbar=yes,menubar=no,location=no, status=yes\");\n"
      "myWindowsDoc = myPopupWindow.document;\n"
      "myWindowsDoc.open();\n"
      "myWindowsDoc.writeln(\"" C_oHTML_oHEAD_oTITLE "\");\n"
      "myWindowsDoc.writeln(\"My Document Title\");\n"
      "myWindowsDoc.writeln(\"" C_xTITLE "\");\n"
      "myWindowsDoc.writeln(\"" C_xHEAD "\");\n";

static char Pgsimple_javascript_newwin_Item_2[] =
      "myWindowsDoc.writeln(\"" C_oBODY ">\");\n"
      "myWindowsDoc.writeln(\"Hello from javascript\");\n"
      "myWindowsDoc.writeln(\"" C_oP "\");\n"
      "myWindowsDoc.writeln(\"This page was created on the fly\");\n"
      "myWindowsDoc.writeln(\"by having a javascript function\");\n"
      "myWindowsDoc.writeln(\"write HTML to a newly created window\");\n"
      "myWindowsDoc.writeln(\"" C_oP "\");\n"
      "myWindowsDoc.writeln(\"An onClick associated with an href and an
\");\n"
      "myWindowsDoc.writeln(\"anchor tag initiated the function call\");\n"
      "myWindowsDoc.writeln(\"" C_xBODY "\");\n";

static char Pgsimple_javascript_newwin_Item_3[] =
      "myWindowsDoc.writeln(\"" C_xHTML "\");\n"
```

HTML tags that have been "compressed"

Please notice that the HTML tags have been replaced with pbuilder utility compressed tags. I have encircled a few for emphasis.

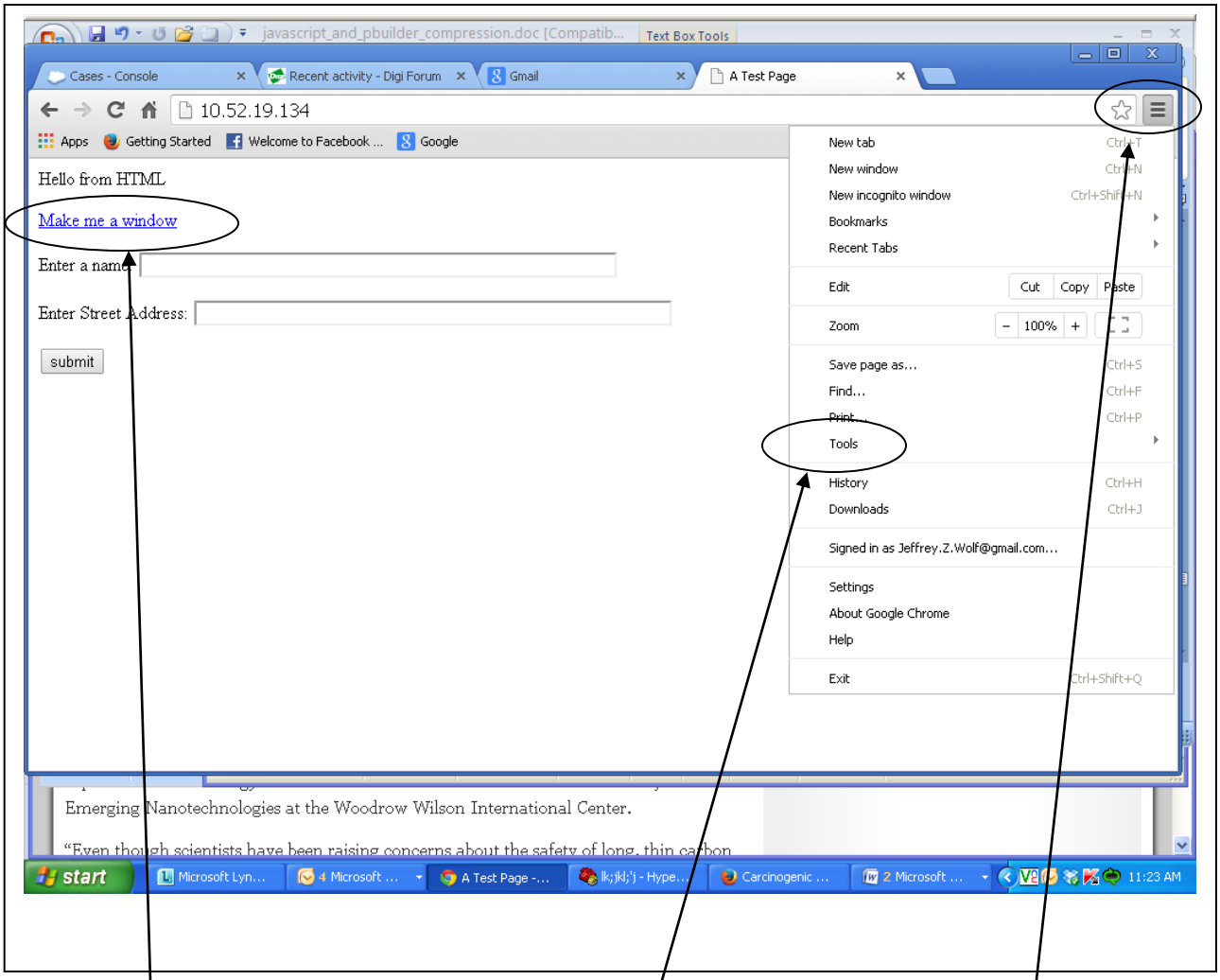
4.4.1 Running the application against this compressed JavaScript

We have created a simple web page and a simple JavaScript file to go with it. After compiling and building the application, we surf to our device and get the following page:



I have highlighted the main objects on the web page. “Hello from HTML” is straight text. “Make me a window” is text associated with an anchor tag and an href attribute. The customize and control button is a Google Chrome button that we will encounter shortly. I wanted to ensure that I pointed it out early.

We want to click on the “Make me a window” live link. The problem is that when I click on the link, nothing happens. So we need to diagnose why nothing happens. I am using Google’s Chrome browser. If you click in the customize and control icon (shown earlier) you get the following:

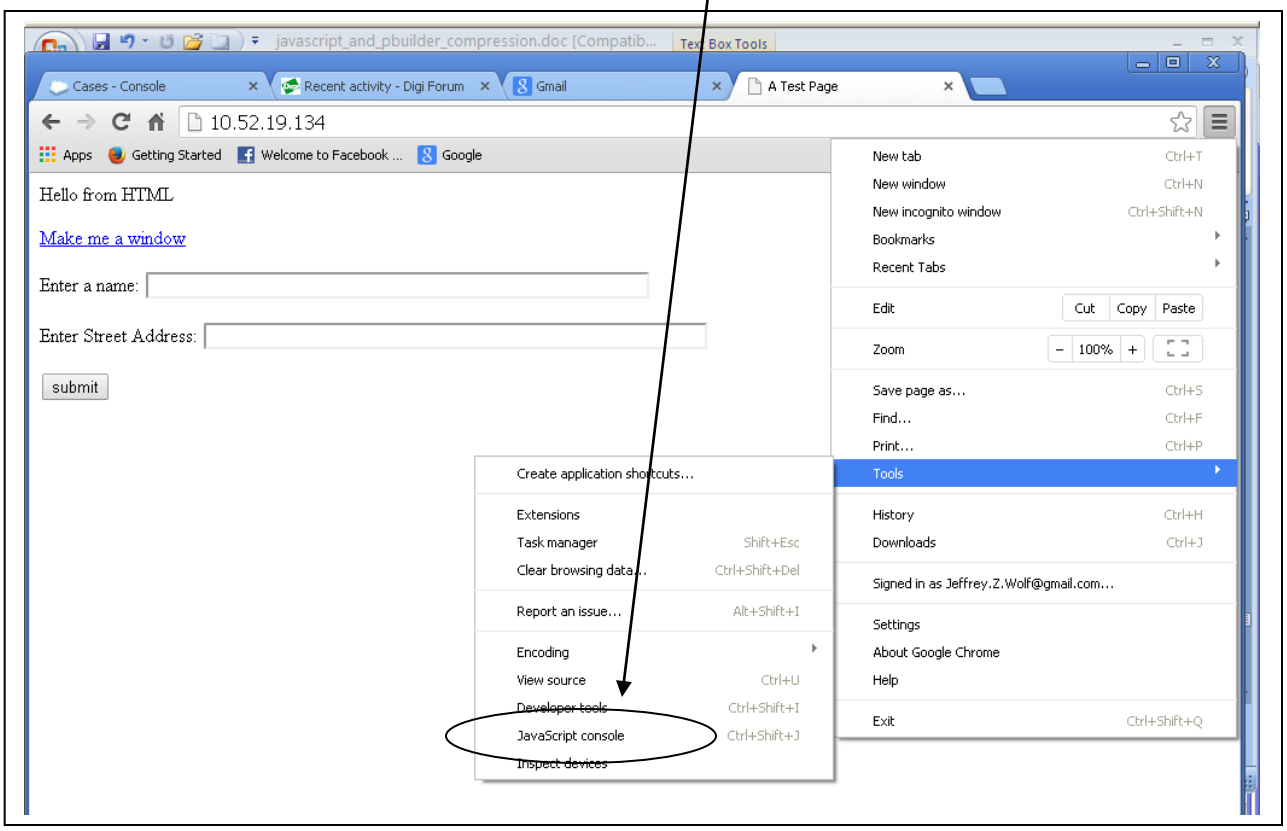


I clicked here but I got nothing.

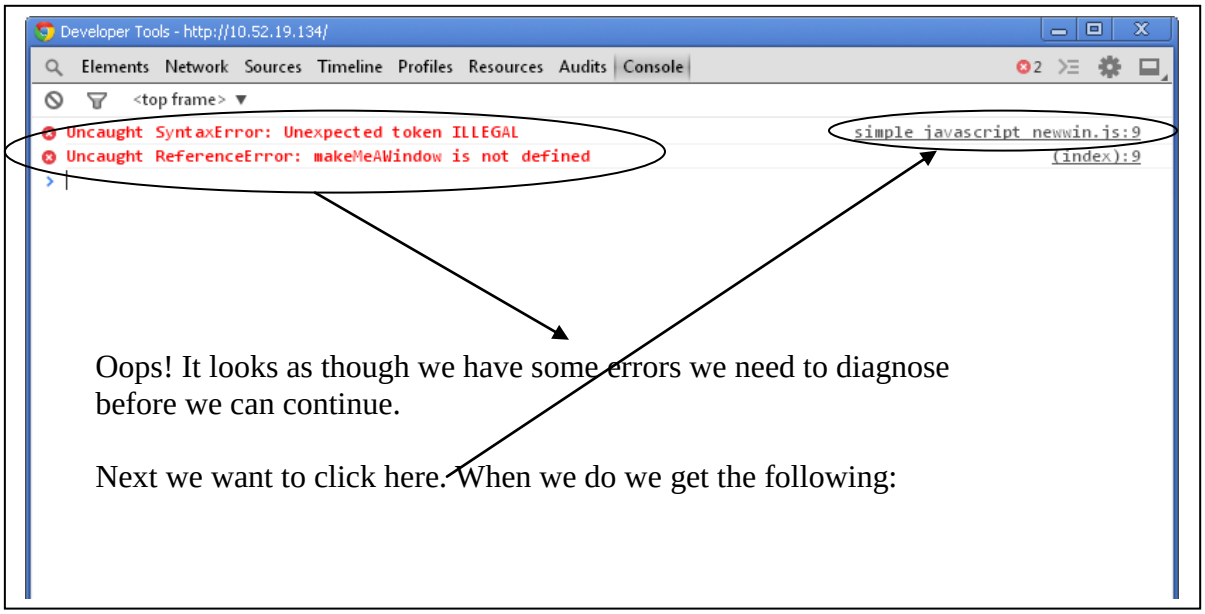
Click the customize and control icon

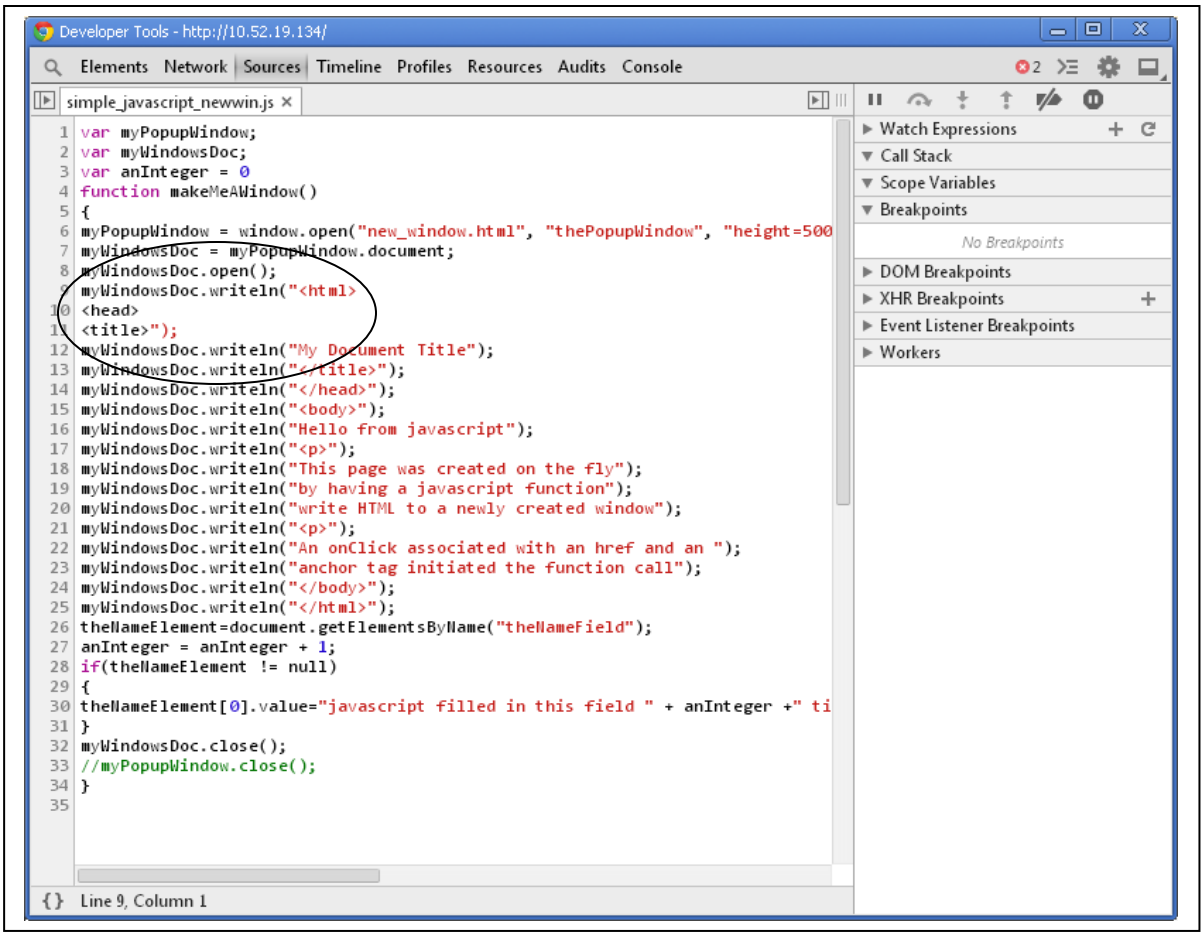
From the customize and control menu click the tools item

From the tools menu item click on the JavaScript console menu item as show below:



The JavaScript console gives the following view:





The view above from the Javascript console shows the javascript code as the browser received it form our device. Notice the myWindowsDoc.writeln that I have encircled above. Notice that the API has been broken up into three lines. Now if we compare this to the original javascript as follows:

```
myWindowsDoc.writeln("<html><head><title>");
```

and the compressed version as follows:

```
"myWindowsDoc.writeln(\"" C_oHTML_oHEAD_oTITLE "\");\n"
```

we can guess that something bad happened in the decompression performed by the AWS engine. What happened was the oHTML was converted to "<HTML>\n". oHEAD was converted to "<head>\n". oTITLE was converted to "<title>\n". The new lines broke the myWindowsDoc.writeln API into three lines, which the javascript interpreter is clearly not happy about. So, we must somehow tell the pbuilder utility, "Keep your hands off my javascript".

4.5 Changing pbuilder utility's actions

The first thing we could do is surround the entire JavaScript file with the comment tag pair "<!—RpDZT-->" and "<!—RpEnd-->". DZT is shorthand for data zero terminated. That is text. It turns out that this is not enough because even though we told pbuilder utility that the code between the tags is text, it will still attempt to compress the HTML as shown above. What you must do is add the attribute RpParse with a value of 0. The following demonstrates the RpDZT comment tag with the RpParse:

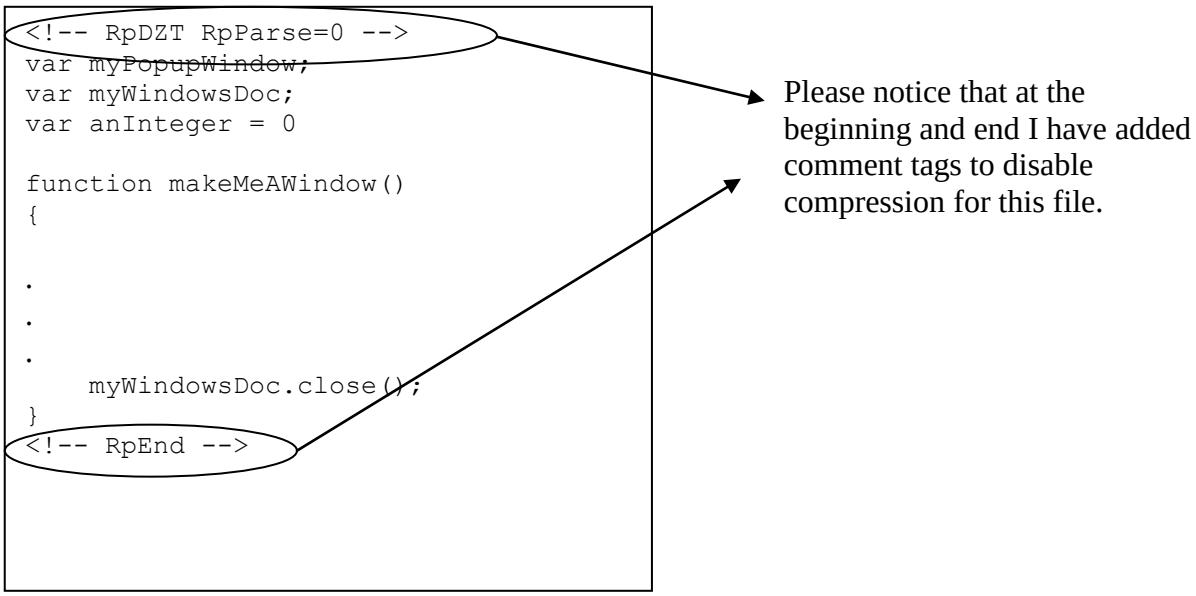
"<!— RpDZT RpParse=0 →" and "<!—RpEnd →". (Please note that the beginning of a pbuilder utility comment tag is a less than sign following by an exclamation mark followed by two dashes. Next a space, the tag contents and a space. After the space are two dashes and a greater than sign).

So let's make these changes and see what happens.

4.5.1 Updated files

4.5.1.1 JavaScript file

The following is from the JavaScript file we are using, updated to disable compression:



4.5.1.3 .c file output from the pbuilder utility

The following piece is from the .c file generated by running the .html file and the JavaScript file through the pbuilder utility:

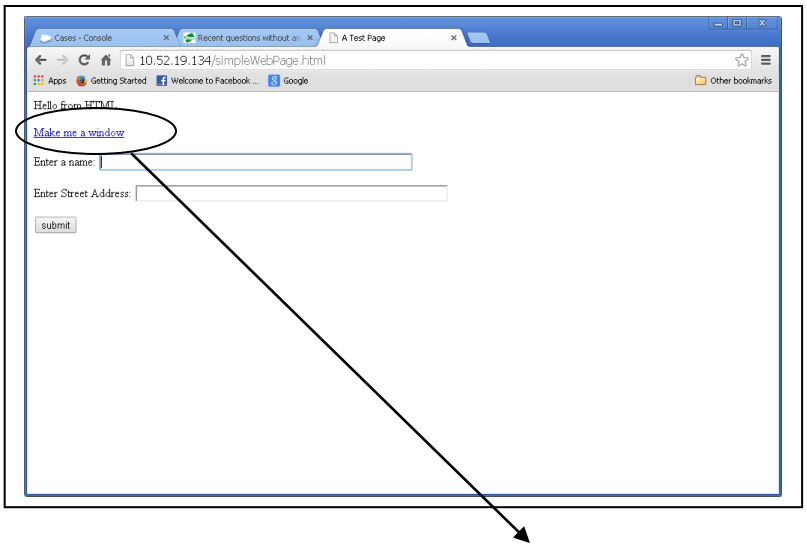
```
static char Pgsimple_javascript_newwin_Item_1[] =
      "\n"
      "var myPopupWindow;\n"
      "var myWindowsDoc;\n"
      "var anInteger = 0\n"
      "\n"
      "function makeMeAWindow()\n"
      "{\n"
      "    myPopupWindow = window.open(\"new_window.html\", "
      "\"thePopupWindow\", \"height=500, width=500, left=100, top=100, "
      "resizable=yes, scrollbars=yes,toolbar=yes,menubar=no,location=no,
"
      "status=yes\");\n"
      "    myWindowsDoc = myPopupWindow.document;\n"
      "    myWindowsDoc.open();\n"
      "    myWindowsDoc.writeln(\"<html><head><title>\");\n"
      "    myWindowsDoc.writeln(\"My Document Title\");\n"
      "    myWindowsDoc.writeln(\"</title>\");\n";
```

Compare the updated .c file with the original one that we saw previously. Notice in the first writeln call in the updated file that HTML tags are in the clear. While below in the previous version, the HTML tags in the first writeln call are “compressed”.

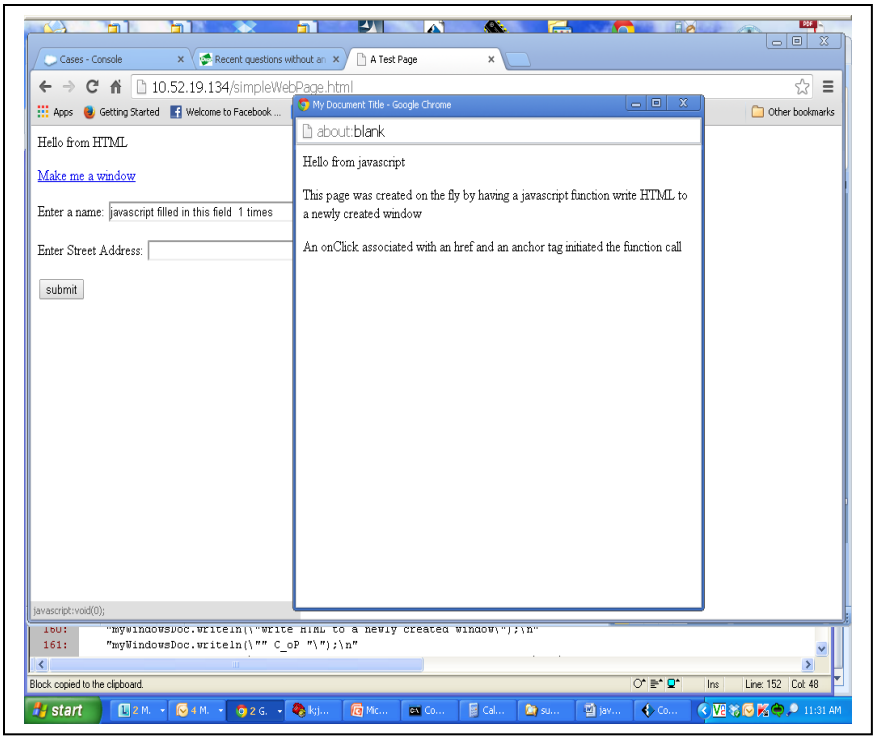
```
static char Pgsimple_javascript_newwin_Item_1[] =
      "var myPopupWindow;\n"
      "var myWindowsDoc;\n"
      "var anInteger = 0\n"
      "function makeMeAWindow()\n"
      "{\n"
      "myPopupWindow = window.open(\"new_window.html\",
\"thePopupWindow\", "
      "\"height=500," C_WIDTH "500, left=100, top=100, resizable=yes, "
      "scrollbars=yes,toolbar=yes,menubar=no,location=no,
status=yes\");\n"
      "myWindowsDoc = myPopupWindow.document;\n"
      "myWindowsDoc.open();\n"
      "myWindowsDoc.writeln(\"" C_oHTML_oHEAD_oTITLE "\");\n"
      "myWindowsDoc.writeln(\"My Document Title\");\n"
      "myWindowsDoc.writeln(\"" C_xTITLE "\");\n"
      "myWindowsDoc.writeln(\"" C_xHEAD "\");\n";
```

4.5.1.4 Running the application

The image below should be familiar. The exhibit shows the main (and only) web page of the application:

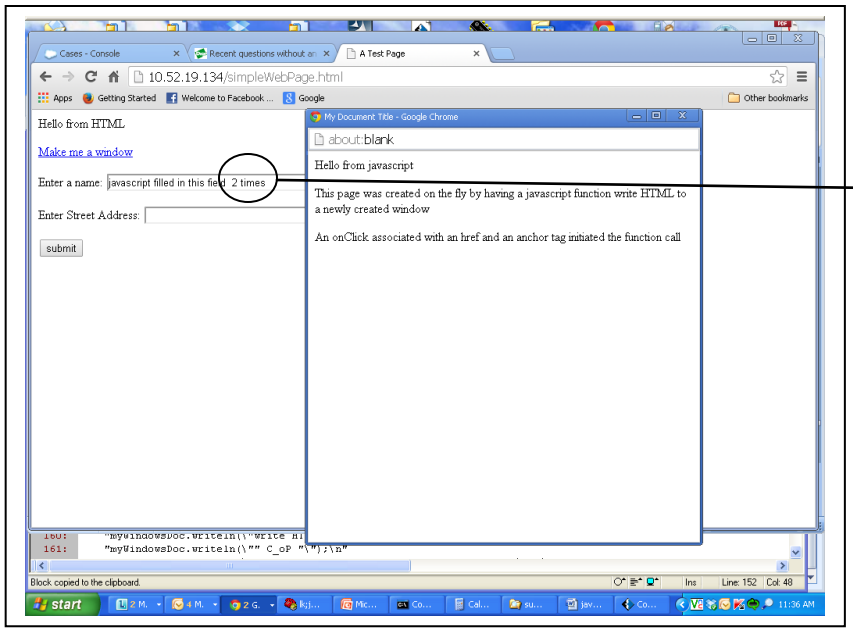


But this time when I click the "Make me a window" link, something different and good happens, as follows:



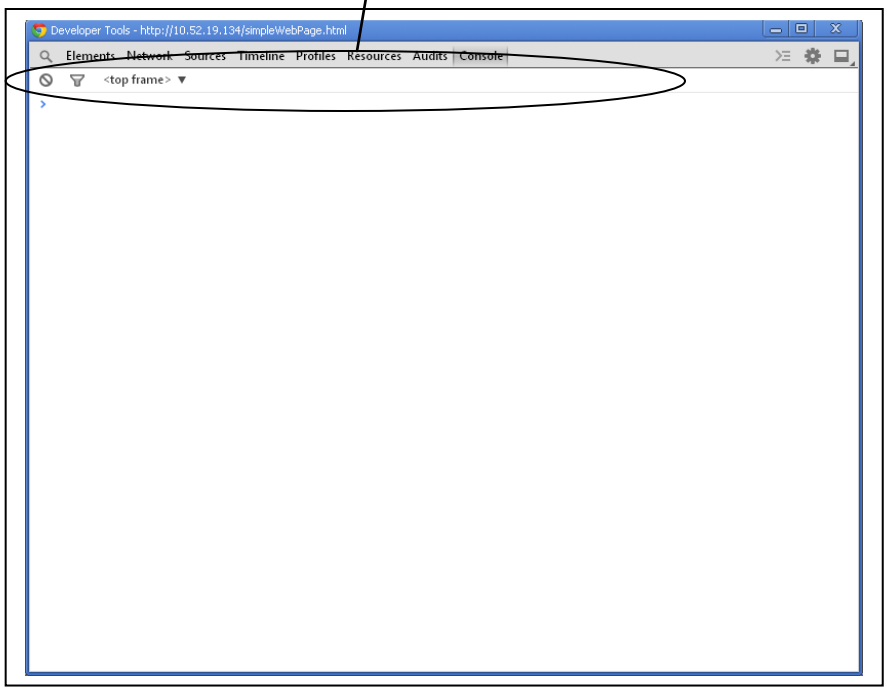
First, notice that this time we successfully created a new web page. Secondly the JavaScript code updated the "Enter a name" input box. Before, that field was blank. Now it contains "javascript filled in this field 1 times."

The "1 times" text is also interesting. Since the integer variable is global in the javascript file. Everytime I hit the "Make me a window" link, that number will increment.



Notice the second time I clicked the link, I got a new web page and the text was updated to "2 times".

Also notice that if we run the Customize and control->tools->javascript console again, this time no errors are displayed.



5 Conclusion

What have we demonstrated? We have shown that certain JavaScript sequences, especially those containing HTML tags, can be broken by the pbuilder utility's attempt at compression. Further we have demonstrated that by surrounding the JavaScript file with a <!—RpDZT RpParse=0 → and <!—RpEnd → comment tag pair, we can disable compression for the file and cause the JavaScript code to work within the NET+OS AWS-based web application.

6 Appendix

6.1 Glossary of terms

- Advanced web server - An embedded web server included in the NET+OS development environment.
- AWS – acronym for advanced web server
- Comment tags – modified HTML comments used to control the output of the pbuilder utility
- FLASH – a type of memory that does not lose its contents when power is removed.
- Google Chrome – a browser produced by the Google Company. According to Wikipedia, "a freeware web browser developed by Google".
- HTML – according to Wikipedia, "Hypertext Markup Language, is the main markup language for creating web pages and other information that can be displayed in a web browser".
- JavaScript – according to Wikipedia, "is a dynamic computer programming language. It is most commonly used as part of web browsers, whose implementations allow client-side scripts to interact with the user, control the browser, communicate asynchronously, and alter the document content that is displayed".
- JPEG – according to Wikipedia, "The term JPEG is an acronym for the Joint Photographic Experts Group. The JPEG standard specifies the codec, which defines how an image is compressed into a stream of bytes and decompressed back into an image, but not the file format use dot contain the stream.
- NET+OS – an embedded operating system and an embedded development environment developed by Digi International.
- pbuilder utility – a utility provided as part of the advanced web server component of the NET+OS development environment. Its purpose is to convert HTML, JavaScript, JPEG and other components of a web project into C programming code so that those components can be compiled into a project for execution.
- RAM – Random access memory. A type of computer memory whose contents are lost when power is removed.