User HTML for Control Solutions Web Servers

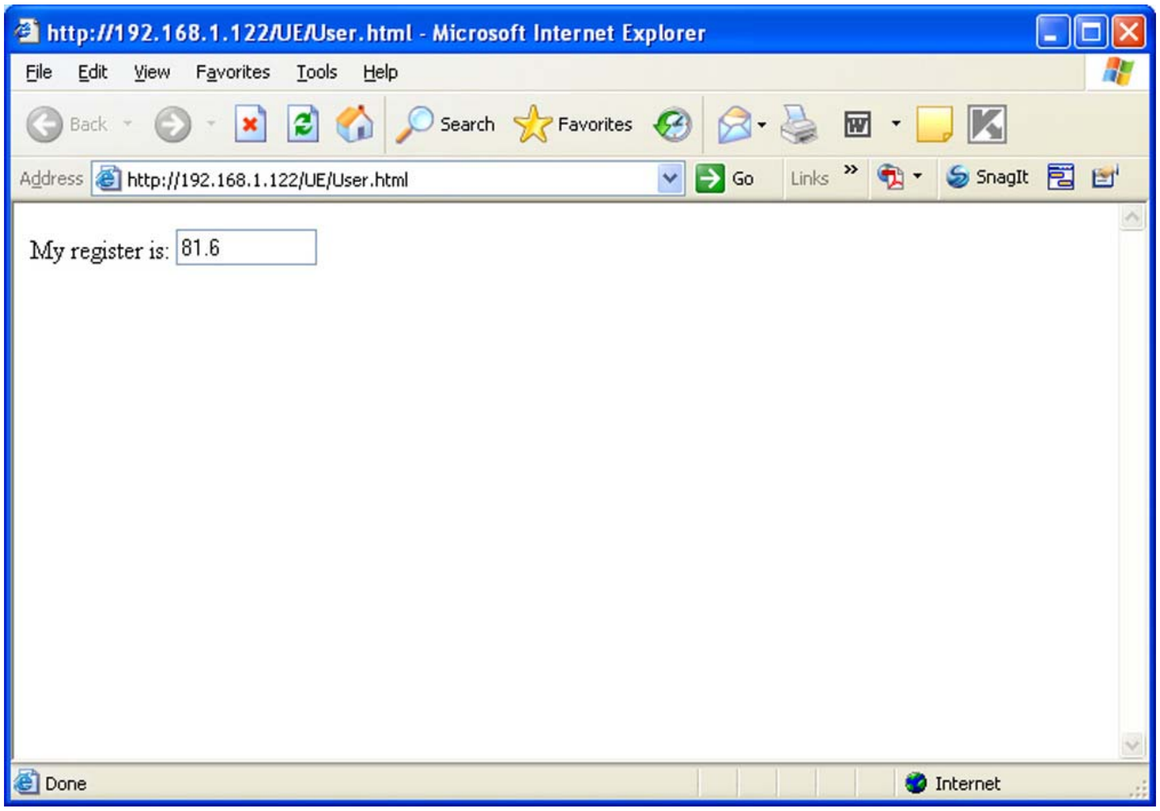
The following is a brief introduction to the sample web sites that can be downloaded to an AddMe III or AddMe Jr. web server. Each sample directory includes a screen shot of the page constructed, and a directory named “upload”. The entire contents of the upload directory should be loaded into the /FLASH0 directory of the server.

Simple Text – Example #1

The most basic form of a web page showing a single register value looks like this in raw HTML code form:

```
<html>
<head></head>
<body>
      <p>My register is: <input type="text" name="reg1001" value="%.1f"
      readonly size="10" /></p>
</body>
</html>
```

The resulting page looks pretty simple, like this:


http://192.168.1.122/UE/User.html - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
Address http://192.168.1.122/UE/User.html
Go
Links
SnagIt
Done
Internet

My register is: 81.6

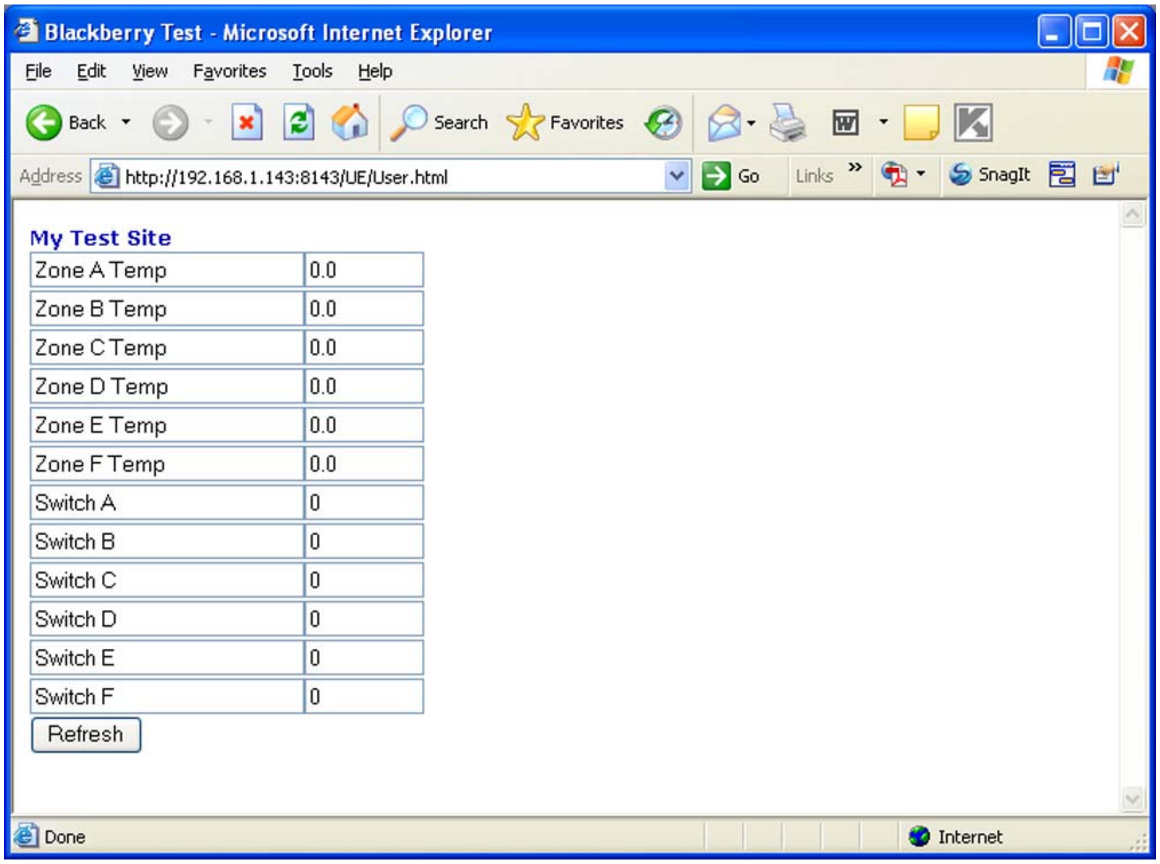
Note that this example includes no buttons, and is a “read only” page meaning we are not able to input anything back to the server.

Blackberry – Example #2

This sample page is designed to fit on a Blackberry screen. It looks a little odd on a PC's browser, but fits cleanly on the Blackberry screen.

This page still consists of only a single page, and is "read only", not accepting input back to the browser. This page does illustrate the technique for building a tabular representation of data. You start by placing an HTML table on the page in your web authoring tool, then add input boxes with the "readonly" tag selected.

The resulting page should come out looking something like this on the PC browser:


Blackberry Test - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://192.168.1.143:8143/UE/User.html Go Links SnagIt
Refresh
Done Internet

My Test Site

Zone A Temp	0.0
Zone B Temp	0.0
Zone C Temp	0.0
Zone D Temp	0.0
Zone E Temp	0.0
Zone F Temp	0.0
Switch A	0
Switch B	0
Switch C	0
Switch D	0
Switch E	0
Switch F	0

You could easily convert this page to accept input. Remove the "readonly" tag from each input box to be enabled, and add a second submit type button named "submitChange" with a value of "Change".

Building Monitor – Example #3

This is a subset of what you might have for a building monitoring site, but it illustrates a few points worth knowing about.

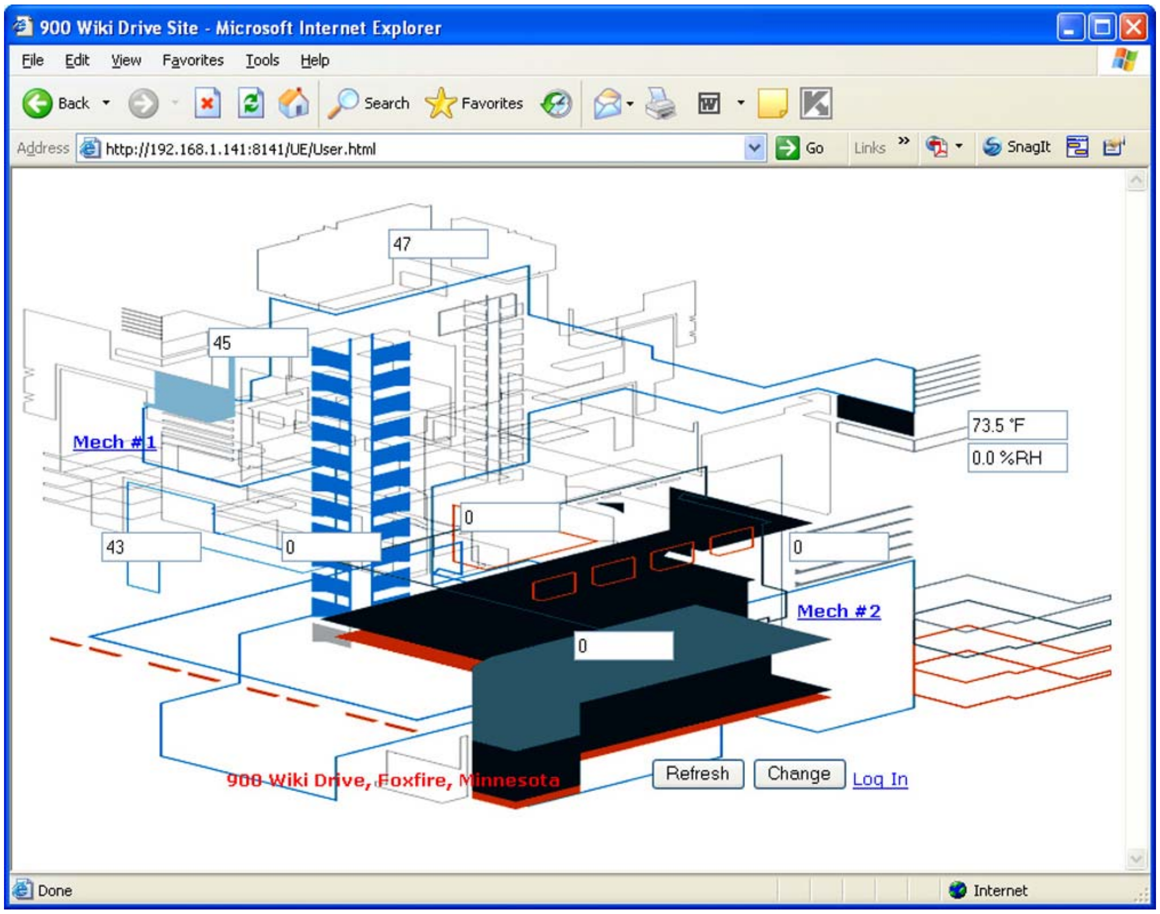
This page includes a submit button that does accept changes. Values are sent back to the same local registers that they came from when the page was served.

This page illustrates linking both to additional user pages, and to accessing the preprogrammed pages in the server.

The additional user pages in this sample site illustrate the alternate form of input, the drop box, referred to in HTML as a <select>.

When the pages in this example are served, you will see a picture with input boxes and drop boxes appearing to be superimposed on the picture. The trick for doing this is to create a table with many rows and columns, and place the input box in the desired cell of the table to position it over the picture. Set the table's border to zero to get rid of any visible hint of a table.

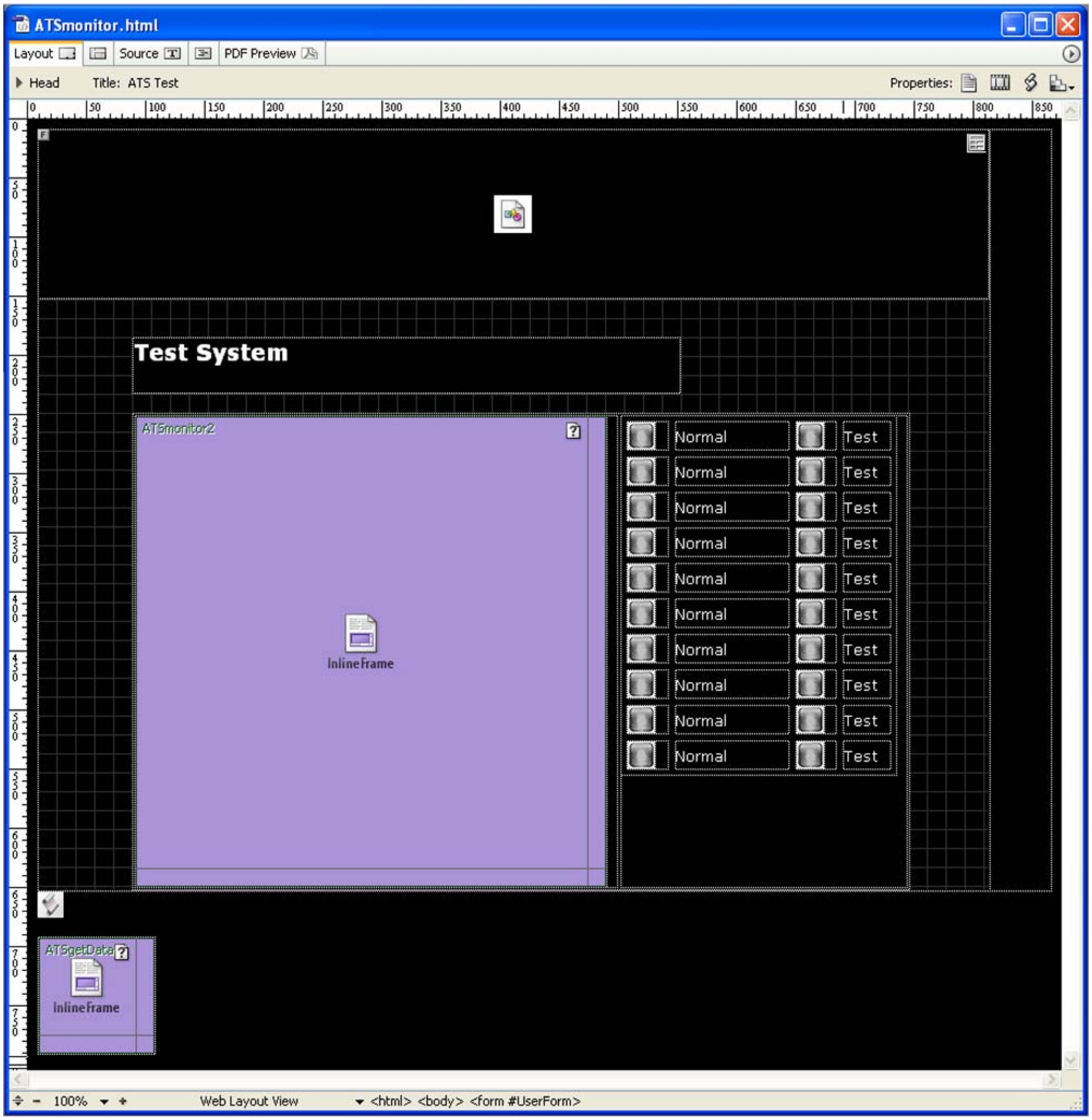
Please note that learning basic HTML is beyond the scope of this write-up or any tutorial that Control Solutions plans to offer. Your local book store has many books on the subject.


900 Wiki Drive Site - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://192.168.1.141:8141/UE/User.html Go Links SnagIt
47
45
Mech #1
73.5 °F
0.0 %RH
0
43
0
0
Mech #2
0
900 Wiki Drive, Foxfire, Minnesota
Refresh
Change
Log In
Done
Internet

Test System – Example #4

This sample is where we really start to have some fun. The data is indicated by red and green lights. The submit button is a graphic button. The page is self-refreshing.

The web authoring tool's layout view of the page looks something like this:



The large violet block represents the area we have allocated to an inline frame that will be our input frame. The purpose of separating this frame is to avoid repainting the entire page each time we click the submit button.

The small violet block is the inline frame for the page that contains nothing but hidden variables. This page is set to be self refreshing using the reload timer. Since it is invisible, it will have minimal impact on the view of the page.

The graphic “lights” in the normal and test columns will turn on red or green as provided by register data mapped to the hidden variables in the invisible frame. A JavaScript function set to be called periodically will reassign the image source based on register data. By selecting red, green, or gray images, the lights will appear to turn on and off.

The submit button in the input frame is a graphic image of a yellow button. Clicking it does not normally do anything; however, by adding a little more JavaScript, we can make it act as a button. Better yet, we can make it mimic the submitChange=Change button.

Inside the form, we include this HTML:

```
<input type="hidden" name="submitChange" value="Change">
<img src="specialbutton.gif" onclick="sendMeAway();" alt="" height="25" width="133"
border="0">
```

Outside the form but still within the body, we include this JavaScript:

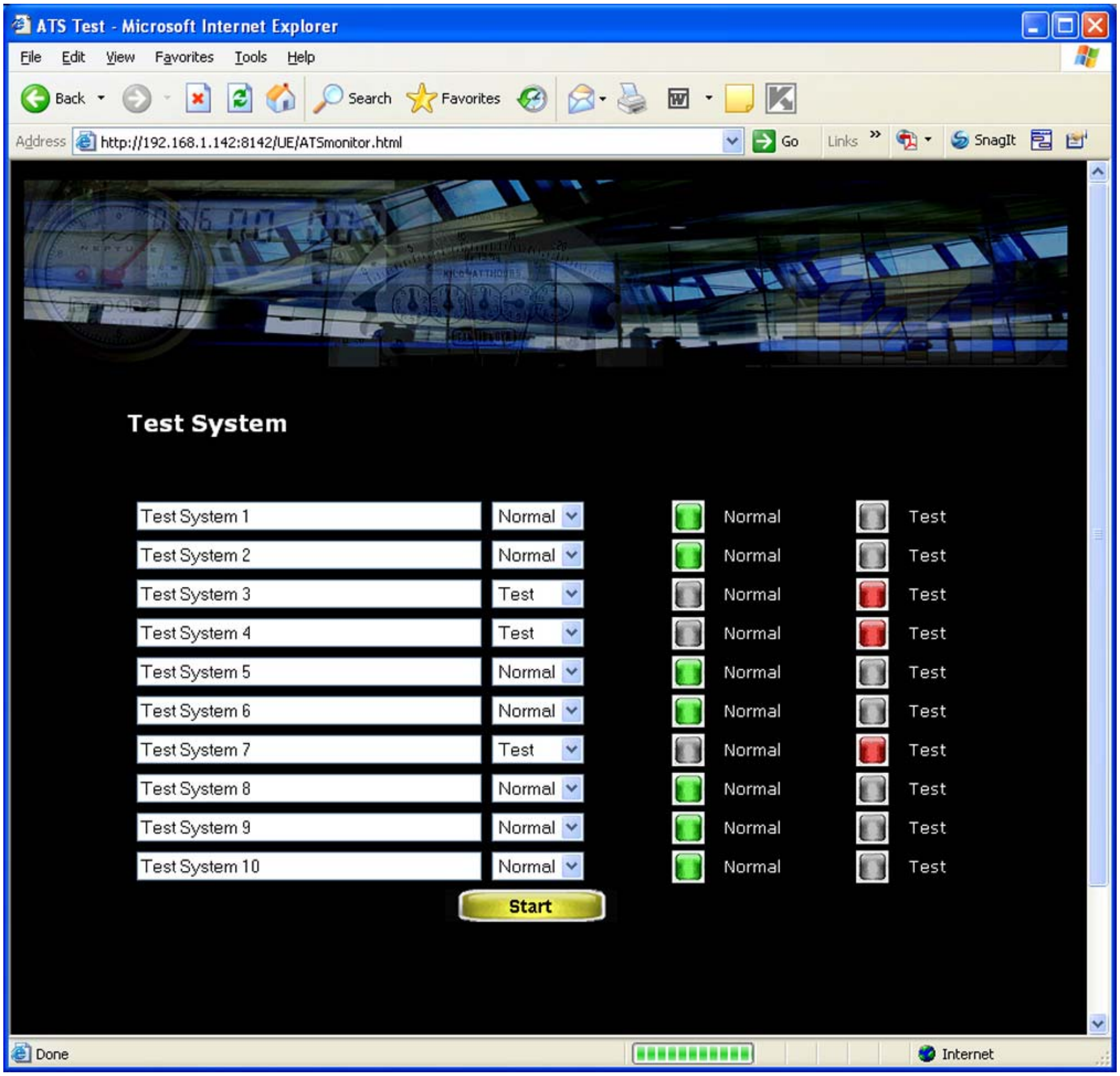
```
<script type="text/javascript"><!--
function sendMeAway() {
      document.UserForm.submitChange.value="Change";
      document.UserForm.submit();
}
//--></script>
```

The `onclick` included in the image reference causes the JavaScript function `sendMeAway()` to be executed when the image is clicked. The function sets up a hidden variable that will produce the return value `submitChange=change` in the post, and then initiates a submit (or post).

Important: Part of the functionality in this page is actually provided by threshold rules found in the BootConfig.xml file. Here is where the content of register 101 determines the state of registers 201 and 301. Registers 201 and 301 in turn decide what color the Normal and Test “lights” are. This pattern continues for 1 through 10. (The sample provided is actually for an AddMe Jr.)

One more note – you need to refer to the CGI overview document to see how to obtain data from local registers in real time. You have access to the register contents and the register name, and it is up to you to decide whether to allow the users to change or only view the data. You can also access the system description to use as a site name, allowing the same HTML template to be used for multiple sites.

The “automated” page resulting from the multiple page set viewed simultaneously via the inline frames should appear as follows.



ATS Test - Microsoft Internet Explorer

Address http://192.168.1.142:8142/UE/ATSmonitor.html

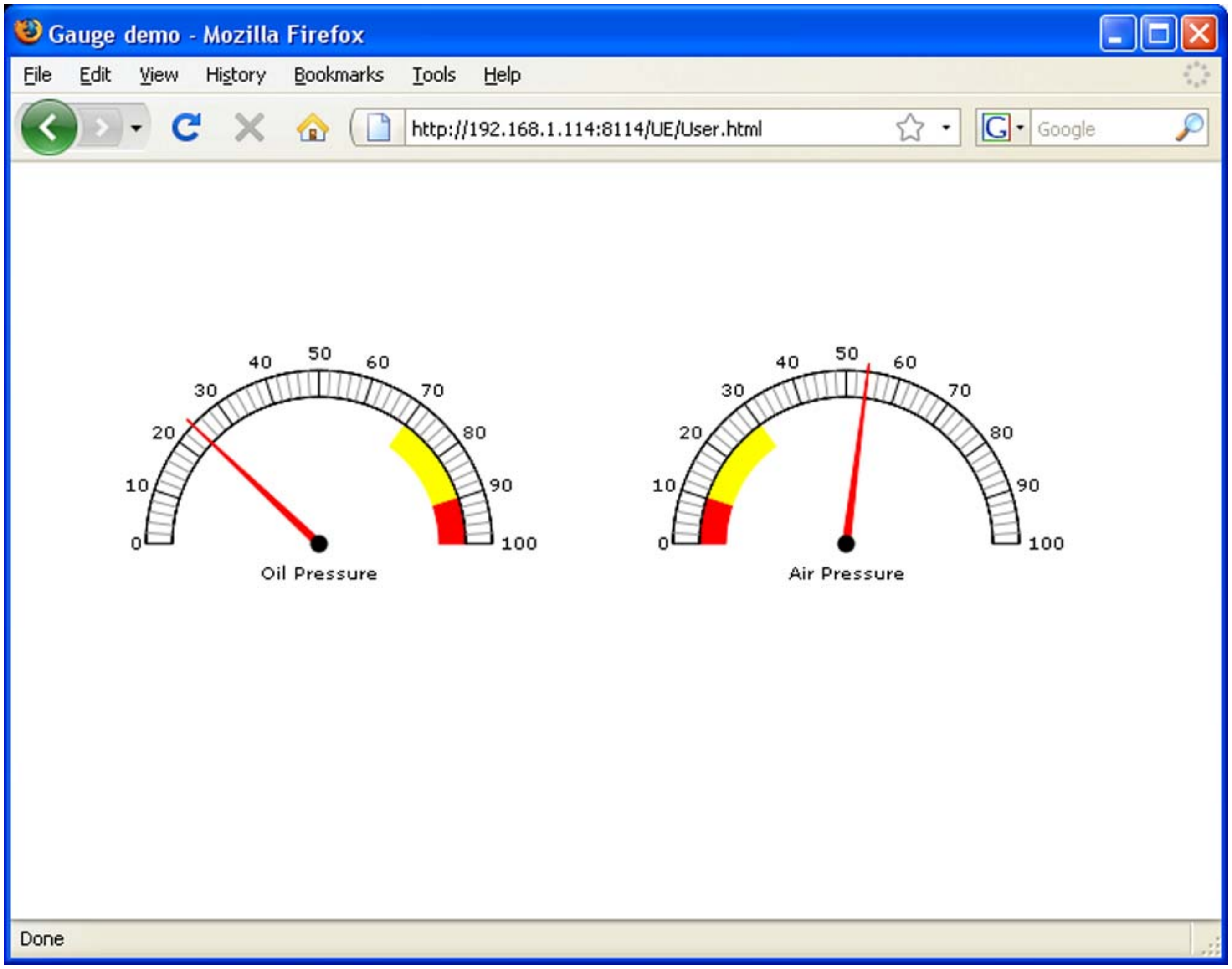
Test System

Test System 1	Normal	🟩	Normal	⬜	Test
Test System 2	Normal	🟩	Normal	⬜	Test
Test System 3	Test	⬜	Normal	🟥	Test
Test System 4	Test	⬜	Normal	🟥	Test
Test System 5	Normal	🟩	Normal	⬜	Test
Test System 6	Normal	🟩	Normal	⬜	Test
Test System 7	Test	⬜	Normal	🟥	Test
Test System 8	Normal	🟩	Normal	⬜	Test
Test System 9	Normal	🟩	Normal	⬜	Test
Test System 10	Normal	🟩	Normal	⬜	Test

Live Gauges - Example #5

All of the examples so far required no additional licensed software libraries. Live gauges are a popular item to display on web pages, so we are providing an example and information about obtaining the necessary JavaScript library.

Our live gauge demo looks like this:



The needles on our gauges move in real time as data changes. In this case, data is being obtained from inputs on a AddMe Jr. The scale, labels, etc., may be changed and you may have any number of gauges on the page.

This demo is based on the Emprise JavaScript Chart package from Emprise Corporation. You may buy this package online at www.ejschart.com. You can also find numerous live demos at their web site.

The only code written for this demo is shown on the following 2 pages. The rest is contained in the library packages purchased from Emprise.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<!-- User.html -->

<html>

      <head>
            <meta http-equiv="content-type" content="text/html;charset=iso-8859-1">
            <meta name="generator" content="Adobe GoLive">
            <title>Gauge demo</title>
      <script type="text/javascript" src="/FS/FLASH0/EJSChart.js"></script>
      <script type="text/javascript" src="/FS/FLASH0/EJSChart_Gauges.js"></script>
      <script type="text/javascript" src="/FS/FLASH0/code.js"></script>
      <style type="text/css">
            #myChart1 { height: 250px; width: 250px; position: absolute; top: 50px;
                  left: 50px; }
            #myChart2 { height: 250px; width: 250px; position: absolute; top: 50px;
                  left: 350px; }
      </style>

      </head>

      <body bgcolor="#ffffff">
            <div id="myChart1" class="chart"></div>
            <div id="myChart2" class="chart"></div>
            <script type="text/javascript" id="script1">
      var chart1 = new EJSC.Chart(
            "myChart1" ,
            {
                  show_titlebar: false ,
                  show_x_axis: false ,
                  show_y_axis: false ,
                  show_legend: false,
                  show_messages: false
            }
      );
      var chart2 = new EJSC.Chart(
            "myChart2" ,
            {
                  show_titlebar: false ,
                  show_x_axis: false ,
                  show_y_axis: false ,
                  show_legend: false,
                  show_messages: false
            }
      );

      var myGauge1 = new EJSC.AnalogGaugeSeries(
            new EJSC.ArrayDataHandler(
                  [ [40,'Oil Pressure'] ]
            ) ,
            {
            ranges: [
                              [ 70 , 90 , 'rgb(255,255,0)' ] ,
                              [ 90 , 100 , 'rgb(255,0,0)' ]
                        ]
            }
      );
      var myGauge2 = new EJSC.AnalogGaugeSeries(
            new EJSC.ArrayDataHandler(
                  [ [40,'Air Pressure'] ]
            ) ,
            {
            ranges: [
                                [ 10 , 30 , 'rgb(255,255,0)' ] ,
                                [ 0 , 10 , 'rgb(255,0,0)' ]
                        ]
            }
      );

      chart1.addSeries(myGauge1);
      myGauge1.__points[0].x = 55;
      chart1.__doDraw();
      chart2.addSeries(myGauge2);
      myGauge2.__points[0].x = 55;
      chart2.__doDraw();

      function updateChart() {
            var x = parseInt(phantom.document.UserForm.reg101.value);
            var y = parseInt(phantom.document.UserForm.reg102.value);
            myGauge1.__points[0].x = x;
            chart1.__doDraw();
            myGauge2.__points[0].x = y;
            chart2.__doDraw();
      }

      window.setInterval(updateChart, 1000);
      </script>
            <iframe name="phantom" src="UserGetData.html" frameborder="0"
                  height="50" width="50"></iframe>
      </body>

</html>



<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<!-- UserGetData.html -->

<html>

      <head>
            <meta http-equiv="content-type" content="text/html;charset=iso-8859-1">
            <meta name="generator" content="Adobe GoLive">
            <title>Untitled Page</title>
            <meta http-equiv="refresh" content="2">
      </head>

      <body bgcolor="#ffffff">
            <form id="UserForm" action="/UE/icanform" method="post"
                  name="UserForm">
                  <div align="left">
                        <input type="hidden" name="reg101" value="%d">
                        <input type="hidden" name="reg102" value="%d">
                  </div>
            </form>
            <p></p>
      </body>

</html>
```