Key Eleven

WebIO

WebIO Internet Automation Gateway

Software API

Datasheet, draft V1.2

www.keyeleven.com

KeyEleven
1305 58th Ave North
Moorhead, MN 56560

Introduction

WebIO is an Ethernet to Control Network and Sensor Network Gateway with a built-in embedded web server as the main user interface for local and remote internet access and control. The WebIO web server following the HTTP protocol provides the HTTP 1.1 GET Response Method URI interface to WebIO functions.

In addition to the HTTP interface, WebIO v3 and v4 will send out UDP Network messages containing WebIO events, wireless temperature readings (v3) and wireless security sensor readings (v4) and WebIO Expansion port hardwired input change events (v3 and v4).

This document describes the two interfaces to WebIO:

1. HTTP-GET URL Interface
2. UDP Network Messages

KeyEleven
1305 58th Ave North
Moorhead, MN 56560

WebIO HTTP Interface

WebIO Software Development

WebIO contains a built in Web Server on a chip for simple out-of-the-box operation, no PC or server required. But with the WebIO HTTP interface, WebIO can also be easily integrated with your own web page, web application or PC software.

Web applications can be created, both simple and complex including:

- A simple web page with WebIO URL links to send commands
- A full Web 2.0 based Internet Home Automation application portal
- Use WebIO URL as HTML SRC to send control command on web page load
- Include the WebIO web pages into your own web app, using iFrames
- Use HTML Forms/Javascript to create your own WebIO user interface

For example, copying the following HTML code into your own web page will cause a WebIO command to be sent on your web page load. This could be useful for ringing a chime on page visits, etc.

```
<iframe name ="FRAME1" src="http://webio.us:8087/x10.spi?house=C&unit=3&cmd=2&x10event=1"
style="width:0px; height:0px" frameborder="0" ></iframe>
```

This example hides the WebIO page response in an HTML iFrame.

The domain (webio.us) directs to the WebIO web site, where Port :8087 directs this request through a router to the WebIO Demo Unit. "house=C unit=3 cmd=2, is the X10 command C3-On, causing a chime to ring, a motor to turn, or light to come on, which ever way you configure WebIO.

KeyEleven
1305 58th Ave North
Moorhead, MN 56560

WebIO HTTP GET URI Interface

These examples use URI query strings to send commands to WebIO via HTTP-GET Method. To try out or test the WebIO HTTP query strings you can copy them into a Web Broswer's URL Address box.

Controlling AC power line X10 devices using WebIO HTTP interface

WebIO query string has the following parameters:

house= {X10 House Code A-P, R (for Expansion port Relay)}
unit= {X10 Unit Code 1-16}
cmd= {WebIO specific X10 Commands where values are:}

- 2 = On
- 3 = Off
- 21 = On-Wait-Off
- 22 = Off-Wait-On
- 4 = Bright Up
- 5 = Dim Down
- 1 = All Units Off
- 6 = All Lights On
- 7 =All Lights Off

x10event= 1 {required to execute command}
userpass= {passcode} - required if a WebIO passcode is configured
ups= {passcode for WebIO v3 and v4} – required if passcode is configured

Several different WebIO CGI programs can receive the X10 command query strings, each providing a different return value/web page.

WebIO CGI programs:

- x10.spi - This CGI returns a web page form for human user interface of X10 commands
- webio.spi - Returns a web page only containing HTML text: "WebIO: Message Received". This is useful for web applications that provide their own user interface
- webiox10.spi - Returns a web page only containing HTML text of the last received command. Example: "A 1 21"
  This is useful for software applications to programmatically validating the last command received by WebIO.

Note: CGI webio.spi and webiox10.spi are new to WebIO Version 1.5 (February 2008)

WebIO response pages can be hidden into an HTML iFrame when WebIO page return is not desired for use in your own web application.
See section: Embedding WebIO URL into web pages

WebIO HTTP Examples

The following examples work with the WebIO Demo at:
http://www.webio.us/demo/
Copy the following URL examples into a web browser URL Address bar:

Send X10 A1-On with no passcode:

http://webio.us:8087/x10.spi?house=A&unit=1&cmd=2&x10event=1
Note the use of port 8087 for network router forwarding to WebIO, Your setup maybe different.
The query string in this example is sent to x10.spi which returns to the WebIO X10 input form web page.

Send X10 A1-Off with no passcode:

http://webio.us:8087/webio.spi?house=A&unit=1&cmd=3&x10event=1
This example sends command to WebIO page webio.spi, which returns only HTML text: "WebIO: Message Received".
Note: this is useful for visually validating that WebIO received a message.

Send X10 A1-On-Wait-Off:

http://webio.us:8087/webiox10.spi?house=A&unit=1&cmd=21&x10event=1
This example sends command to WebIO page webiox10.spu, which returns HTML text of the last received command. Example: "A 1 21"
Note: this is useful for visually validating that WebIO received the command.

Send X10 E1-On-Wait-Off with passcode:

http://www.webio.us:8087/x10.spi? house=E&unit=1&cmd=21&x10event=1&userpass=q2w3

More WebIO HTTP-GET API Examples

When sending HTTP query strings to WebIO that are used to configure WebIO, such as setting its IP address, defining an X10 label, the WebIO "Lock-Prg" switch needs to be in the "Program" position.

Set WebIO X10 Labels

WebIO v2, v3, and v4 have labels for naming up to 8 X10 devices. Naming these X10 devices can be done using the WebIO labels webpage or from the WebIO v2 Setup software, WebIO-TM software for v3, or using the KeyOn software for WebIO v4.

The query string for setting WebIO X10 Labels contains the X10 device name and the associated house code and unit code for the labeled device.
WebIO v2 and v3 X10 labels have a length limit of 16 characters where as WebIO v4 has a 12 character limit.
WebIO v2 Label Example:

```
http://192.168.1.75/label.spi?labelevent=1&label01=Rope Light&h01=A&u01=1&label02=Night Light&h02=A&u02=2&label03=Color Lights&h03=A&u03=3&label04=Motor&h04=C&u04=3&label05=Room Lamp&h05=E&u05=1&label06=Radio&h06=B&u06=3&label07=&h07=A&u07=8&label08=&h08=B&u08=1
```

Setting WebIO v3 X10 labels is the same as V2.
Setting WebIO v4 X10 labels uses a different query string.
Example:

```
URL=HTTP://192.168.1.77/label.spi?l1=Radio/Chime&h01=M&u01=1&l2=Light&h02=M&u02=2&l3=Fan&h03=M&u03=5&l4=Label 4&h04=A&u04=1&l5=Alarm&h05=A&u05=5&l6=Color Lights&h06=A&u06=6&l7=Air Duct Fan&h07=A&u07=7&l8=Last XLabel&h08=A&u08=8&lbevent=1
```

WebIO v4 Automation Setup

Automation setup defines what X10 or Relay event occurs on a wireless sensor Alert/Normal, Open/Close or Lock/Unlock events. An HTTP-GET query string is sent for each automation channel plus alarm channel and for each state of Alert and Normal

WebIO v4 automation setup example:
Set: RF channel 6, A-Alert, R-relay, state B, no alarm, rf3event=9

```
http://192.168.1.71/label.spi?u06=6&h06=A&h07=R&u07=2&u08=66&ualt=0&rf3event=9
```

Set: RF channel 6, N-Normal, R-relay, state A, no alarm, rf3event=9

```
http://192.168.1.71/label.spi?u06=6&h06=N&h07=R&u07=2&u08=65&ualt=0&rf3event=9
```

WebIO Expansion Port Setup

Both WebIO v3 and v4 have an expansion port that can be use with the 4-Relay/4-Input board or other future device. The query string is used to set the label/names of the 4 relays and 4 inputs. The 4 inputs also can have automation parameters for house code and unit code, where when the input goes to a "high" state it sends out and X10 "On" Command for the corresponding input channels defined house/unit code.

Set WebIO version 3, EXP labels and automation to defaults (blank):

http://192.168.1.72/label.spi?label01= &label02= &label03= &label04= &label05= &label06= &label07= &label08= &labelevent=3&h01=0&u01=0&h02=0&u02=0&h03=0&u03=0&h04=0&u04=0

WebIO v3, just labels:

http://192.168.1.72/label.spi?label01=Test0001&label02=Test2002&label03=Test3003&label04=Test4004&label05=Test5005&label06=Test6006&label07=Test7007&label08=Test8008&labelevent=3

WebIO v3, labels with House and Unit codes for each of 4 inputs

http://192.168.1.72/label.spi?label01=Test0001&label02=Test2002&label03=Test3003&label04=Test4004&label05=Test5005&label06=Test6006&label07=Test7007&label08=Test8008&labelevent=3&h01=A&u01=1&h02=A&u02=2&h03=A&u03=3&h04=A&u04=4

WebIO v4 uses “lbevent” not “labelevent” and “l1”-“l8” not “label01”-“label08”:

http://192.168.1.72/label.spi?l1=Test0001&l2=Test2002&l3=Test3003&l4=Test4004&l5=Test5005&l6=Test6006&l7=Test7007&l8=Test8008&lbevent=3&h01=A&u01=1&h02=A&u02=2&h03=A&u03=3&h04=A&u04=4

WebIO v4 Register a Sensor

WebIO v4 requires wireless sensors to be registered to 1 of 8 channels. Each sensor has either a 2 byte ID (wireless security sensors) or a House/Unit code as 2 bytes for wireless X10 sensors and remotes. Note that WebIO will display the 2 byte ID as a 2 character Hex string but the HTTP query string sends the ID as a decimal integer.

Clear a sensor channel (un-register a sensor by defining an ID=0). Note that “u04=101”, where 101 represents sensor channel 1, and 102 would represent sensor channel 2, etc. “u05” is assigned to the sensor ID (0 in this example).

HTTP://192.168.1.78/rfs.spi?u04=101&u05=0&rf3event=2

Clear sensor channel 7:

http://192.168.1.78/rfs.spi?u04=107&u05=0&rf3event=2

Example, register sensor with ID “3F” Hex (send as 63 decimal)

HTTP://192.168.1.78/rfs.spi?u04=102&u05=63&rf3event=2

Register sensor ID=96h =150 decimal to channel 3 (103)

http://192.168.1.72/rfs.spi?u04=103&u05=150&rf3event=2

WebIO Temperature

All WebIO units have an internal temperature sensor. WebIO v3 can read from wireless temperature and humidity sensors. Wireless sensors are assigned to 3 channels using a channel switch on the sensor. Any number of wireless sensors (which contain a unique ID) are send over the TCP Network as a UDP message.

WebIO v4 internal temperature sensor
To request temperature update:

```
http://192.168.1.71/x10.spi?xmevent=3
```

To retrieve the temperature results, page must be refreshed from the browser

```
http://192.168.1.71/rsptemp.html
```

WebIO v3 has a minimum HTML page for displaying the internal temperature sensor and 3 channels of wireless temperature sensors

You can see this minimum HTML page at:
http://www.webio.us:8087/rsptemp.html

Contents of the minimal temperature value page may look like this:

```
T0 C: 17.1
T0 F: 62.9
T1 N:Solar Thermal F: 113.9 H:0 ID:9 C S:255
T2 N:Garage F: 102.6 H:0 ID:A 6 S:255
T3 N:Fireplace F: 72.5 H:23 ID:0 1 S:253
```

Where:

```
T0 - is the internal temperature sensor
T1 - is wireless sensor set to channel 1
T2 - is wireless channel 2
T3 - is wireless channel 3
N: - is followed by the user label/name of the sensor at the given channel
C: - is followed by Celsius temperature
F: - is followed by Fahrenheit temperature
H: - is followed by Humidity value
ID:- is followed by 2 byte hex ID of sensor (ignore spacebar between bytes)
S: - is followed by sensor time state, a value of 255 indicates a very recent sensor reading
        a value of 0 indicated a sensor reading has not been received for several minutes
        this number is decremented about every 2 minutes.
```

KeyEleven
1305 58th Ave North
Moorhead, MN 56560

WebIO v3, Temperature message UDP format
Every time WebIO receives a temperature update from a wireless sensor, WebIO sends the temperature data vi UDP to the WebIO-TM software or to your own application.
UDP data is sent to port UDP port 4040 by default.

UDP packet contains this data:

Byte - Label - Description
0 - "T" - the character "T" for temperature, 1 byte
1 - MAC4 - Byte number 4 of the WebIO's MAC address, bytes 1-3 are not sent, 1 byte
2 - MAC5 - MAC address byte 5, 1 byte
3 - MAC6 - MAC address byte 6, 1 byte
4 - CHANNEL - Channel of the Sensor (1-3)
5 - rfcs - Sensor temperature Celcius sign, 1 char ="-" if negative value
6 - rfci - Sensor temperature Celcius integer portion of temperature, 1 byte
7 - rfcd - Sensor temperature Celcius decimal portion of temperature, 1 byte
8 - rfh - Sensor humidity as percent, 1 byte, sensors that do not support humidity this value=0
9 - rfid1 - Sensor ID high order byte
10 - rfid2 - Sensor ID low order byte
11 - state - Sensor state byte, indicated low battery, unknown

For code samples see files:

KeyWebIO3Server.c

Sample code for Linux using GCC C compiler. This code receives UDP network messages from WebIO, forms the message data into an HTTP-GET querystring and sends it to a web server.

WebIO_V3_UDP-Rx_Code.pdf

Contains sample C# code and the C code for KeyWebIO3Server.c

KeyEleven
1305 58th Ave North
Moorhead, MN 56560

WebIO, Using HTML Forms, iFrames and Javascript

Web Application Development

Simple web applications can be created using client side HTML and Javascript code, using HTML Forms, iframes and Javascript AJAX/XMLHttpRequest.

In using WebIO across the Internet, WebIO must be setup to be "Internet Routable". See the WebIO Manual and Network Topologies sections in Documentation.

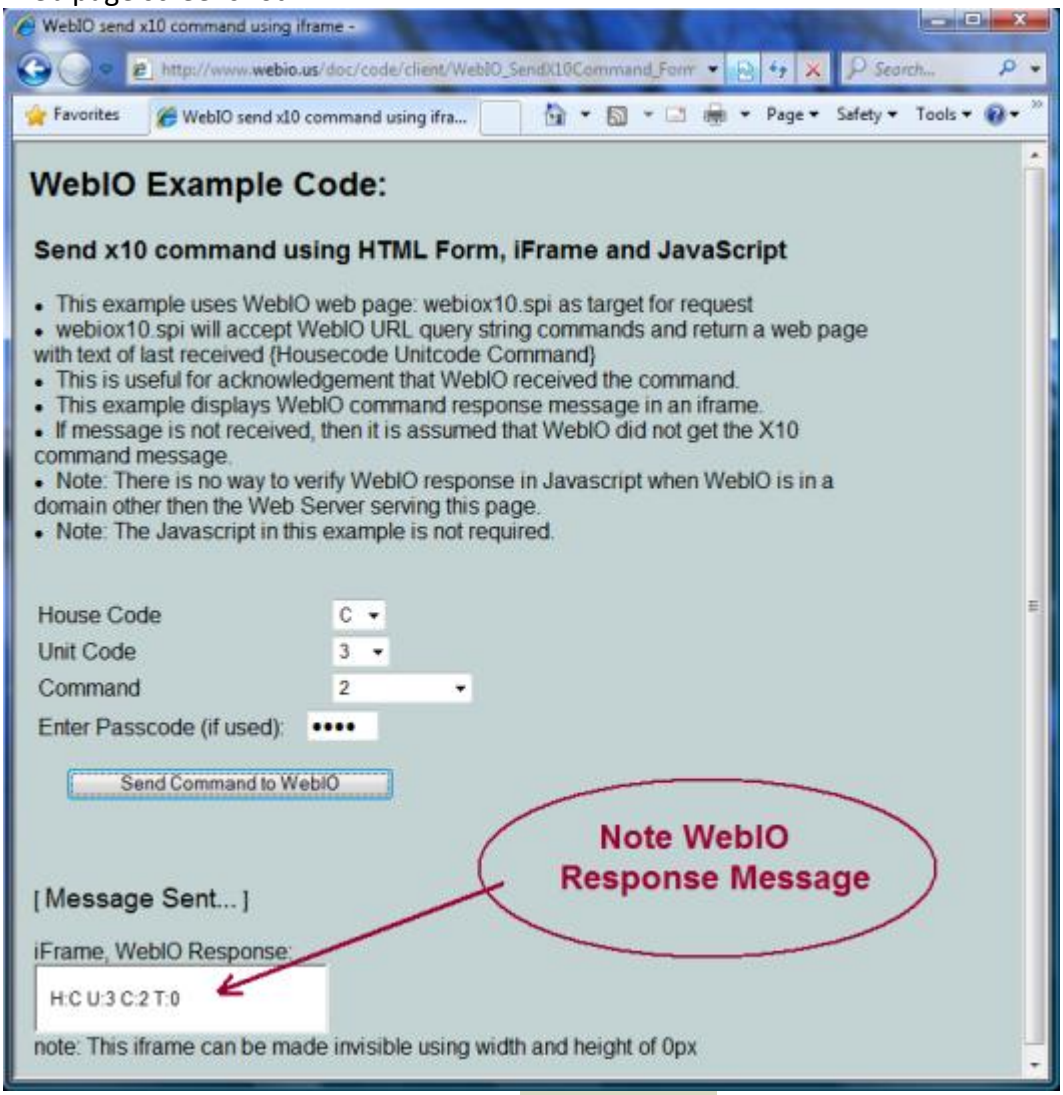
The diagram to the right shows an example of a Internet Web Server hosting a web app/page that has routable access to a WebIO installed in the home.
This web app embeds WebIO HTTP requests, hiding the details and HTML interfaces of the WebIO, creating a more seamless and integrated application.

Using HTML Forms, iFrame and JavaScript

From the URL examples above, you will notice that WebIO returns a webpage result. These web page responses can be hidden or contained using iframes and in some cases processed using Javascript.

This example uses an HTML Form for an interface to selecting and submitting an X10 request to WebIO.
The following examples work with the WebIO Demo at:
http://www.webio.us/doc/code/client/WebIO_SendX10Command_Form-iFrame_7.html

Web page screenshot:



Note the WebIO response message of: **H:C U:3 C:2 T:0**
This identifies that WebIO last send X10 message House Code "C", Unit Code "3" with Command/Function 2 (On), with a wait time of 0 seconds.

File: WebIO_SendX10Command_Form-iFrame_7.html

```
<html>
<head>
```

```
<title>WebIO send x10 command using iframe</title>
<!-- Key Eleven - WEbIO - tav 02/17/2008 -->

<style type="text/css">
    body
    {
        font: 14px arial;
        background-color: "#C3D3D3";
    }
</style>

<script type="text/javascript">
<!--

// Flag when form has been submit vs page refreshed
var iFormSubmitted=0;

function frameLoaded()
{
    var iFind=0;
    var sFrameData="";

    // only diplay if form has been submit, not on page refresh
    if (iFormSubmitted==1)
    {
        document.getElementById("MSG").innerHTML =
            "<span style='font-size:14pt'>Message Sent...</span>";
    }
}

function WebIOSubmitForm()
{
    // This is Javascript for form response sent to iFrame
    // but instead were using {target="iframename"} in Form tag to sumit to iframe by name
    //var form = document.getElementById("FORM1");
    //  this doesnt work (opens in new page): form.target = "IFRAME1";
    //form.target = "iframename";
    //form.submit();

    // flag that the form has been submitted
    iFormSubmitted=1;

    document.getElementById("MSG").innerHTML =
            "<span style='font-size:14pt'>Sending Request...</span>";

    //alert ("wait");
}
-->
</script>
</head>

<body>
<h2>WebIO Example Code:</h2>
<table><tr><td width="600">
<h3>Send x10 command using HTML Form, iFrame and JavaScript</h3>

<li>This example uses WebIO web page: webiox10.spi as target for request </li>
```

```
<li>webiox10.spi will accept WebIO URL query string commands and return a web page with
text of last received {Housecode Unitcode Command}</li>
<li>This is useful for acknowledgement that WebIO received the command. </li>
<li>This example displays WebIO command response message in an iframe. </li>
<li>If message is not received, then it is assumed that WebIO did not get the X10 command
message. </li>
<li>Note: There is no way to verify WebIO response in Javascript when WebIO is in a domain
other then the Web Server serving this page. </li>
<li>Note: The Javascript in this example is not required.</li>
<br/><br/><br/>


<!-- Form -->
<!-- note that form target="iframename", the name of the iframe not the frame ID, not
webio.us at port 8087 -->
   <form method="get" action="http://webio.us:8087/webiox10.spi" id="FORM1"
target="iframename">
      <input type="hidden" name="x10event" value="1"/>
          <table border="0" width="100%">
            <tr>
              <td
width="35%">House Code       </td>
              <td>
                <select name="house">
                    <option value="C" selected>C</option>
                    <option value="A">A</option>
                    <option value="B">B</option>
                    <option value="C">C</option>
                    <option value="D">D</option>
                    <option value="E">E</option>
                    <option value="F">F</option>
                    <option value="G">G</option>
                    <option value="H">H</option>
                    <option value="I">I</option>
                    <option value="J">J</option>
                    <option value="K">K</option>
                    <option value="L">L</option>
                    <option value="M">M</option>
                    <option value="N">N</option>
                    <option value="O">O</option>
                    <option value="P">P</option>
                </select>
              </td>
            </tr>
            <tr>
              <td width="35%">Unit Code</td>
              <td>
                <select name="unit">
                    <option value="3" selected>3</option>
                    <option value="1">1</option>
                    <option value="2">2</option>
                    <option value="3">3</option>
                    <option value="4">4</option>
                    <option value="5">5</option>
                    <option value="6">6</option>
                    <option value="7">7</option>
                    <option value="8">8</option>
                    <option value="9">9</option>
```

```
                    <option value="10">10</option>
                    <option value="11">11</option>
                    <option value="12">12</option>
                    <option value="13">13</option>
                    <option value="14">14</option>
                    <option value="15">15</option>
                    <option value="16">16</option>
                  </select>
                </td>
              </tr>
              <tr>
                <td width="35%">Command</td>
                <td>
                  <select name="cmd">
                      <option value="2" selected>2</option>
                      <option value="2">On</option>
                      <option value="3">Off</option>
                      <option value="21">On-Wait-Off</option>
                      <option value="22">Off-Wait-On</option>
                      <option value="4">Bright Up</option>
                      <option value="5">Dim Down</option>
                      <option value="1">All Units Off</option>
                      <option value="6">All Lights On</option>
                      <option value="7">All Lights Off</option>
                  </select>
                </td>
              </tr>

              <tr>
                <td colspan="2" nowrap>Enter Passcode (if used):   
                  <input type="password" name="userpass" value="    " maxlength="4"
size="4"/>
                  <br /><br />    
                  <!-- <input type="submit" name="submit2" value="Execute" /> -->
                  <input type="submit" name="submit2" value="Send Command to WebIO"
onclick="WebIOSubmitForm()"/>
                </td>
              </tr>
            </table>
        </form>



<!-- message placeholder,  sending... sent... -->
<br/><br/>
[ <span id="MSG">...</span> ]
<br/><br/>



<!-- the iFrame loaded with WebIO response -->
iFrame, WebIO Response:<br />
<!-- note:  iframe can be made invisible using width and height tof 0px -->
<iframe name="iframename" id="IFRAME1" src="about:blank" style="width:210px; height:50px"
onload="frameLoaded()"
    frameborder="1" scrolling="no">
</iframe>
<br/>
```

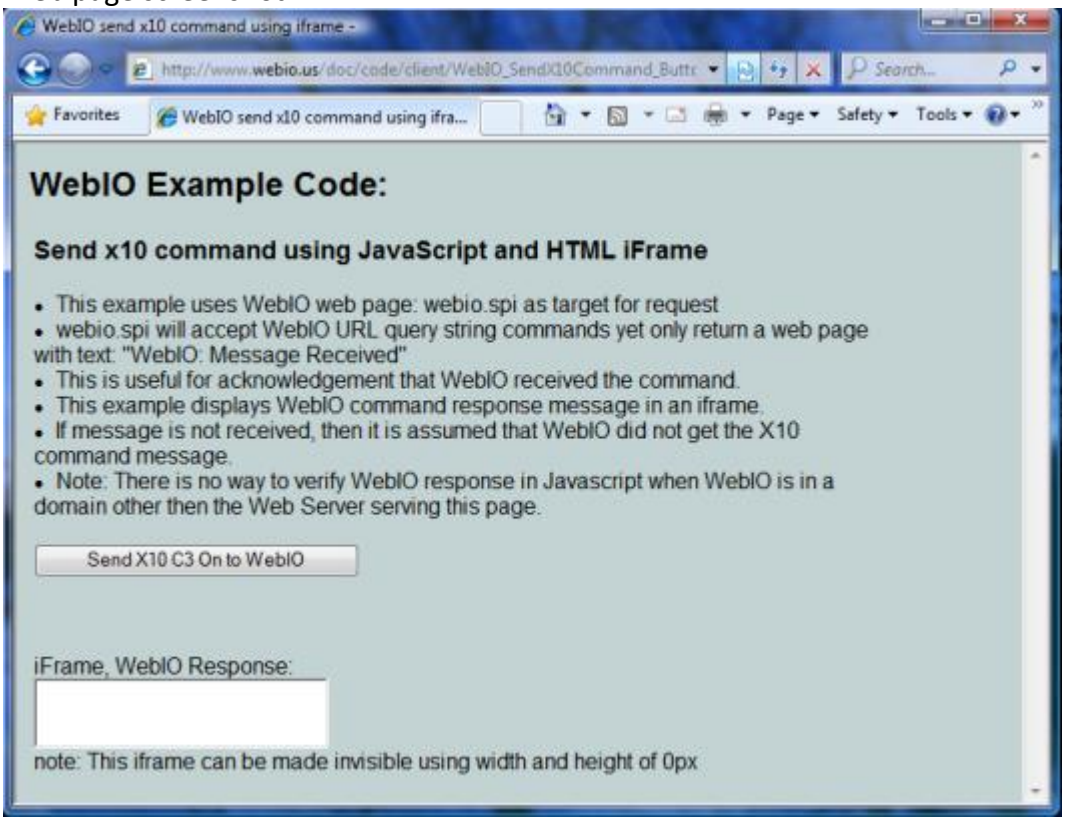
```
note: This iframe can be made invisible using width and height of 0px
<br />


</td></tr></table>
</body>
</html>
```

Using Form Buttons, iFrames and JavaScript

This example uses an HTML Form Button and Javascript to submit a WebIO command.
See demo at:
http://www.webio.us/doc/code/client/WebIO_SendX10Command_Button-iFrame_6.html

Web page screenshot:



File: WebIO_SendX10Command_Button-iFrame_6.html

```
<html>
<head>
<title>WebIO send x10 command using iframe</title>
<!-- Key Eleven - WEbIO - tav 02/17/2008 -->

<style type="text/css">
    body
    {
        font: 14px arial;
```

```
        background-color: "#C3D3D3";
    }
</style>

<script type="text/javascript">
<!--
// WebIO URL:
// this url returns "WebIO: message received", note port 8087
var url="http://webio.us:8087/webio.spi?house=C&unit=3&cmd=2&x10event=1";
// this url returns most recent X10 command: "{HouseCode}{UnitCode}{Command}"
//var url="http://webio.us:77/webiox10.spi?house=B&unit=3&cmd=2&x10event=1";
// Flag when button pressed vs page refreshed
var iButtonPressed=0;

function SendWebIOCommand()
{
    // first load blank into iFrame
    loadBlank();

    // Send WebIO Command and return page to iframe1
    document.getElementById("IFRAME1").src=url;

    // display message that message was sent
    document.getElementById("MSG").innerHTML =
            "<span style='font-size:14pt'>Sending Message...</span>";
    // Flag that button was pressed
    iButtonPressed=1;
}
function loadBlank()
{
    // clear iframe.. while waiting for a reply from WebIO
    // this doest work due to not reshreshing of iframe
    document.getElementById("IFRAME1").src='about:blank';
    // could do this but, then need to set back to visible
    //document.getElementById("IFRAME1").style.display='none';
}
function frameLoaded()
{
    var iFind=0;
    var sFrameData="";

    if (iButtonPressed==1)
    {
        document.getElementById("MSG").innerHTML =
            "<span style='font-size:14pt'>Message Sent...</span>";
        getStatus();
    }
}
function getStatus()
{
    // Search for "WebIO" in page to verify we actually sent request to WebIO
    // Note:  this does not work across domains, i.e. WebIO and the Web Server are at
diffent domains.

    //alert("done loading iframe");
    //alert(document.frames['IFRAME1'].document.documentElement.outerHTML)
    //sFrameData = document.frames['IFRAME1'].document.documentElement.outerHTML;
```

```
    /*
    iFind=document.frames['IFRAME1'].document.documentElement.outerHTML.indexOf("WebIO");
    alert("found retval: " + iFind);
    if (iFind!=-1)
    {
        alert("Ok, WebIO Found");
        document.getElementById("MSG").innerHTML =
            "<span style='font-size:14pt'>WebIO Command Sent Ok</span>"
        return true;
    }
    else
    {
        alert("Failed, WebIO not found");
        document.getElementById("MSG").innerHTML =
            "<span style='font-size:14pt'>Sending WebIO Command Failed</span>"
        return false;
    }
    */
}
-->
</script>
</head>

<body>
<h2>WebIO Example Code:</h2>
<table><tr><td width="600">
<h3>Send x10 command using JavaScript and HTML iFrame</h3>

<li>This example uses WebIO web page: webio.spi as target for request </li>
<li>webio.spi will accept WebIO URL query string commands yet only return a web page with
text: "WebIO: Message Received" </li>
<li>This is useful for acknowledgement that WebIO received the command. </li>
<li>This example displays WebIO command response message in an iframe. </li>
<li>If message is not received, then it is assumed that WebIO did not get the X10 command
message. </li>
<li>Note: There is no way to verify WebIO response in Javascript when WebIO is in a domain
other then the Web Server serving this page. </li>
<br/><br/>

<!-- button for sending command to WebIO -->
<input  type="button"  value="Send X10 C3 On to WebIO"  id="Button1"  name="Button1"
    onclick="return SendWebIOCommand()"/>

<!-- message placeholder,  sending... sent... -->
<br/><br/>
<span id="MSG"></span>
<br/><br/>

<!-- the iFrame loaded with WebIO response -->
iFrame, WebIO Response:<br />
<!-- note:  iframe can be made invisible using width and height tof 0px -->
<iframe id="IFRAME1" src="about:blank" style="width:210px; height:50px"
onload="frameLoaded()"
    frameborder="1" scrolling="no">
</iframe>
<br/>
note: This iframe can be made invisible using width and height of 0px
<br />
```

```
</td></tr></table>
</body>
</html>
```

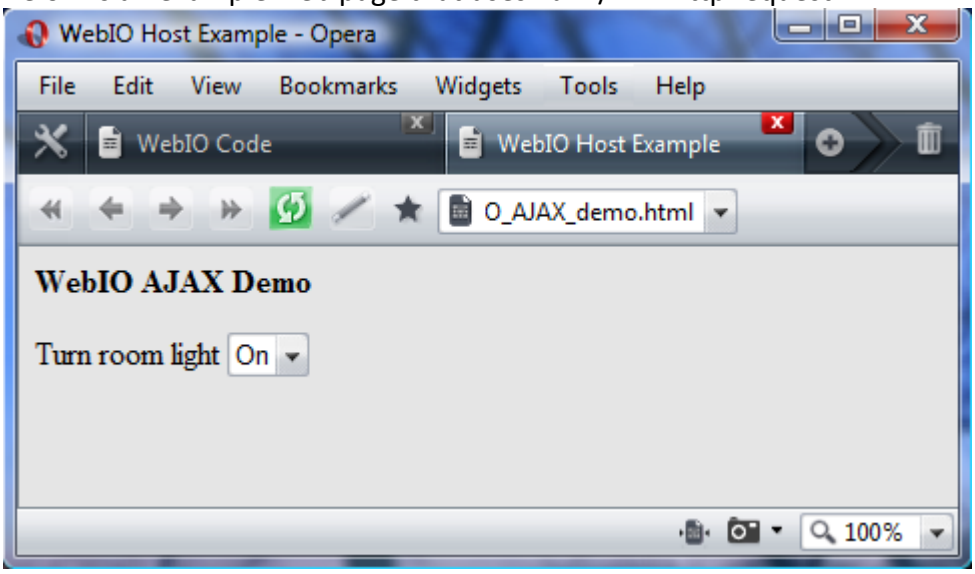
Using iFrames and JavaScript AJAX (XMLHttpRequest)

Javascript including iFrames and AJAX can be used for developing WebIO Web Apps. In somecases this can be difficult as Javascript cannot evaluate the contencts of iframes that span domains, (e.g. mydomain.com, someotherdomain.com). This restriction includes the use of AJAX/XMLHttpRequest.

The examples above use iframes and Javascript. These examples should work, even if the WebIO and Web Server have different domains because they do not need to process the response from WebIO. It just happens that in these demos/examples that the WebIO and Web Server are in the same domain, this would allow for evaluating and validating the WebIO responses, even though WebIO and this webserver use different port numbers 80 vs 8087 on the same domain.

For more information see Microsoft MSDN note:
About Cross-Frame Scripting and Security
http://msdn.microsoft.com/en-us/library/ms533028.aspx

Below is an example web page that uses AJAX/XMLHttpRequest:



Demo located at:
http://www.webio.us/doc/code/client/WebIO_AJAX_demo.html

File: WebIO_AJAX_demo.html

```
<html>
<head>
<title>WebIO Host Example</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
```

```
</head>

<script language="javascript" type="text/javascript">
// Create Http Object
function makeHttpObject() {
    var xmlHttpObj;
    // branch for Activex version (Microsoft IE)
    /*@cc_on
    @if (@_jscript_version >= 5)
        try {
            xmlHttpObj = new ActiveXObject("Msxml2.XMLHTTP");
        } catch (e) {
            try {
                xmlHttpObj = new ActiveXObject("Microsoft.XMLHTTP");
            } catch (E) {
                xmlHttpObj = false;
            }
        }
    @else
        xmlHttpObj = false;
    @end @*/

    // branch for native XMLHttpRequest object (Mozilla & Safari)
    if (!xmlHttpObj && typeof XMLHttpRequest != 'undefined') {
        try {
            xmlHttpObj = new XMLHttpRequest();
        } catch (e) {
            xmlHttpObj = false;
        }
    }
    return xmlHttpObj;
}
var httpObj = makeHttpObject(); // create the HTTP Object

// Http Response Event Handler
function getHttpResponse() {
    if (httpObj.readyState == 4) {
        if (httpObj.status == 200) {
            content = httpObj.responseText;
            div = document.getElementById("Status");
            div.innerHTML = "";
            // insert WebIO response HTML content into "Status" <div>
            //div.innerHTML = content;
                  // or just say "Message sent"
                  div.innerHTML ="Message sent";
        } else {
            alert("There was a problem with the response" + httpObj.statusText);
                  div = document.getElementById("Status");
            div.innerHTML = "";
                  div.innerHTML ="Message failed";
        }
    }
}


// URL for WebIO X10 commands
// http://webio.us:77/x10.spi?house=A&unit=2&cmd=2&x10event=1
// NOTE:  this URL must be the Domain as hosted site of this page
```

```
var url = "http://www.webio.us:8087/x10.spi?house=E&unit=1&x10event=1&cmd=";

// Send HTTP Form "Get" Event
function getWebIOResponse(ev) {
    ev = (ev) ? ev : ((window.event) ? window.event : null);
    if (ev) {
        var el = (ev.target) ? ev.target : ((ev.srcElement) ? ev.srcElement : null);
        if (el) {
            if (el.selectedIndex > 0) {
                httpObj.open("GET", url + el.options[el.selectedIndex].value, true);
                httpObj.onreadystatechange = getHttpResponse;
                httpObj.send(null);
                        div = document.getElementById("Status");
                        div.innerHTML = "Sending Request...";
            }
        }
    }
}

</script>


<body bgcolor="#E5E5E5">
<h4>WebIO AJAX Demo</h4>
<form id="demo">
  <label for="E1">Turn room light </label>
  <select id="E1" name="E1" size="1" onChange="getWebIOResponse(event)">
     <option value="1">-</option>
     <option value="2">On</option>
     <option value="3">Off</option>
  </select>
</form>
<div id="Status"><span></span></div>
</body>
</html>
```

Furthermore, client side scripting could make use of AJAX libraries such as Dojo and YUI.

Using Server Side Scripts - PHP, JSP, ASP.NET

More complex web applications be created using server side scripting tools/languages such as PHP, ASP.NET, JSP, Perl, Mono.net, etc. where the server side scripts perform the HTTP request/response with WebIO, relieving the client side (Javascript, etc) from doing the work.

WebIO v4 wireless sensor data sent via UDP

When WebIO v4 received a wireless sensor message or Expansion port input state change, WebIO sends the data via UPD message over the TCP/IP network to the WebIO KeyOn software. This data is encrypted for security purposes. If you want to receive this data by your own application you will need to either use the KeyOn UDP2XML feature or use the UDP2XML driver app. See documentation on UDP2XML for protocol definition.

For more information and software updates, visit the WebIO web site:

http://www.webio.us

http://www.keyeleven.com

or contact:

Key Eleven
1305 58th Ave North
Moorhead, MN 56560 USA