MAVERICK XML - A TUTORIAL

The Maverick IP Appliance has a default IP address of *192.168.1.10*. Assuming this default, the Maverick's XML output is obtained from http://192.168.1.10/pc10144.xml. The XML looks something like that shown on the next page in **Figure 1**.

AN INTRODUCTION TO XML

XML is a World Wide Web Consortium (W3C) recommendation. The W3C is made up of over 400 members from organizations all over the world that are working on specifications for web technologies. The W3C website (http://www.w3.org) states that XML is "*a simple, very flexible text format … playing an increasingly important role in the exchange of a wide variety of data on the Web and elsewhere.*"

Defining data in XML can be done in many different ways and, just like a database design, can be structured to hold data in many different ways. By defining data in XML, the data is separated from the presentation; but to present the data, its structure must be known.

Web pages can use a technology called AJAX, Asynchronous JavaScript and XML, to dynamically update web pages. The Maverick web pages use this technology so the page always displays the latest data from the Maverick. The JavaScript part of AJAX uses another W3C recommendation called the Document Object Model (DOM), which defines the functions or methods to access XML data easily. For example, the DOM defines a method called *getElementsByTagName()* which will return a list of all XML elements with that tag name. In **Figure 1**, to retrieve all Sensor elements, you would invoke the method *getElementsByTagName("Sensor")*.

The DOM has some standard language for referring to XML content. A node or element contains the starting and ending XML tags. It may contain child nodes. For example, in **Figure 1**, a <Sensor> node has child nodes: <SensorName>, <SensorValue>, <SensorUnits>, <SensorAlert>, <SensorRelay>, <Sensor LTValue>, <SensorGTValue>, and <SensorLog>.

An attribute has a value and is contained within the tag name of the node. It has the format *attribute="value"*. In **Figure 1**, the <Sensor> nodes have both a *Channel* and *ID* attribute. Both attributes have a value equal to the I/O number of the Maverick.

XML is used to define many other W3C technologies like XSL and SVG. To keep track of which XML is which, the W3C defines *namespaces* for each. A namespace can be given to any XML data. The Maverick uses namespace http://mamacsys.com/pc10144 as defined by the *xmlns* attribute of the Maverick document element.

This introduction is certainly not complete, but serves to define some of the concepts that will be used in this document. The W3C website is the ultimate source of information, and a very good, free online tutorial is available at http://www.w3schools.com.

Figure 1: Maverick XML Output

```
1.  <?xml version="1.0" encoding="ASCII"?>
2.  <Maverick xmlns="http://mamacsys.com/pc10144" version="1.13">
3.    <NodeID>MAMAC UNO</NodeID>
4.    <CurrentTime>13:34:59</CurrentTime>
5.    <CurrentDate>May 9, 2008</CurrentDate>
6.    <Sensor id="1" Channel="1">
7.      <SensorName>Lab Temp</SensorName>
8.      <SensorValue>70.3</SensorValue>
9.      <SensorUnits>F</SensorUnits>
10.     <SensorAlert>Alert</SensorAlert>
11.     <SensorRelay>None</SensorRelay>
12.     <SensorLTValue>0.0</SensorLTValue>
13.     <SensorGTValue>1.0</SensorGTValue>
14.     <SensorLog>1</SensorLog>
15.   </Sensor>
16.   <Sensor id="2" Channel="2">
17.     <SensorName>Lab Humidity:</SensorName>
18.     <SensorValue>19.0</SensorValue>
19.     <SensorUnits>% RH</SensorUnits>
20.     <SensorAlert>Alert</SensorAlert>
21.     <SensorRelay>None</SensorRelay>
22.     <SensorLTValue>0.0</SensorLTValue>
23.     <SensorGTValue>1.0</SensorGTValue>
24.     <SensorLog>1</SensorLog>
25.   </Sensor>
26.   <Sensor id="3" Channel="3">
27.     <SensorName>Duct Pressure</SensorName>
28.     <SensorValue>2.4</SensorValue>
29.     <SensorUnits>&amp;quot; wc </SensorUnits>
30.     <SensorAlert>Normal</SensorAlert>
31.     <SensorRelay>None</SensorRelay>
32.     <SensorLTValue>0.0</SensorLTValue>
33.     <SensorGTValue>1.0</SensorGTValue>
34.     <SensorLog>1</SensorLog>
35.   </Sensor>
36.   <Sensor id="4" Channel="4">
37.     <SensorName>Fan Current</SensorName>
38.     <SensorValue>25.0</SensorValue>
39.     <SensorUnits>Amps</SensorUnits>
40.     <SensorAlert>Normal</SensorAlert>
41.     <SensorRelay>None</SensorRelay>
42.     <SensorLTValue>0.0</SensorLTValue>
43.     <SensorGTValue>1.0</SensorGTValue>
44.     <SensorLog>1</SensorLog>
45.   </Sensor>
46.   <Output id="5" Channel="5">
47.     <OutputName>Supply Fan</OutputName>
48.     <OutputValue>On</OutputValue>
49.     <OutputControl>Automatic</OutputControl>
50.     <OutputSchedule enabled="0">
51.       <Sunday enabled="0" start="00:00" stop="00:00" />
52.       <Monday enabled="0" start="00:00" stop="00:00" />
53.       <Tuesday enabled="0" start="00:00" stop="00:00" />
```

(Continued on Page 3)

(Continued from Page 2)

```
      <Wednesday enabled="0" start="00:00" stop="00:00" />
      <Thursday enabled="0" start="00:00" stop="00:00" />
      <Friday enabled="0" start="00:00" stop="00:00" />
      <Saturday enabled="0" start="00:00" stop="00:00" />
    </OutputSchedule>
  </Output>
  <Output id="6" Channel="6">
    <OutputName>DX Cooling</OutputName>
    <OutputValue>On</OutputValue>
    <OutputControl>Automatic</OutputControl>
    <OutputSchedule enabled="0">
      <Sunday enabled="0" start="00:00" stop="00:00" />
      <Monday enabled="0" start="00:00" stop="00:00" />
      <Tuesday enabled="0" start="00:00" stop="00:00" />
      <Wednesday enabled="0" start="00:00" stop="00:00" />
      <Thursday enabled="0" start="00:00" stop="00:00" />
      <Friday enabled="0" start="00:00" stop="00:00" />
      <Saturday enabled="0" start="00:00" stop="00:00" />
    </OutputSchedule>
  </Output>
  <Output id="7" Channel="7">
    <OutputName>Humidifier</OutputName>
    <OutputValue>Off</OutputValue>
    <OutputControl>Automatic</OutputControl>
    <OutputSchedule enabled="0">
      <Sunday enabled="0" start="00:00" stop="00:00" />
      <Monday enabled="0" start="00:00" stop="00:00" />
      <Tuesday enabled="0" start="00:00" stop="00:00" />
      <Wednesday enabled="0" start="00:00" stop="00:00" />
      <Thursday enabled="0" start="00:00" stop="00:00" />
      <Friday enabled="0" start="00:00" stop="00:00" />
      <Saturday enabled="0" start="00:00" stop="00:00" />
    </OutputSchedule>
  </Output>
  <Output id="8" Channel="8">
    <OutputName>Exhaust Fan</OutputName>
    <OutputValue>On</OutputValue>
    <OutputControl>Manual</OutputControl>
    <OutputSchedule enabled="0">
      <Sunday enabled="0" start="00:00" stop="00:00" />
      <Monday enabled="0" start="00:00" stop="00:00" />
      <Tuesday enabled="0" start="00:00" stop="00:00" />
      <Wednesday enabled="0" start="00:00" stop="00:00" />
      <Thursday enabled="0" start="00:00" stop="00:00" />
      <Friday enabled="0" start="00:00" stop="00:00" />
      <Saturday enabled="0" start="00:00" stop="00:00" />
    </OutputSchedule>
  </Output>
</Maverick>
```

Figure 1: Maverick XML Output

The Maverick uses AJAX to dynamically update its main web page. Let's take a look at the code. **Figure 2** contains part of the main web page, *start.html*.

```
<table id="InputValues1" width="100%" border="0" align="center" cellpadding="0" cellspacing="0">
 <tr>
  <td width="100%" align="center">
   <table border="0" align="center" cellpadding="0" cellspacing="0">
    <tr>
     <td class="TitleSm" align="right">Sensor 1: </td>
     <td class="NormalBold" align="left"><span id="S1Name">Lab Temp</span> : </td>
     <td class="NormalBold" align="left"><span id="S1Val">70.3</span> <span id="S1U">F</span></td>
    </tr>
    <tr>
     <td class="TitleSm" align="right">Sensor 2: </td>
     <td class="NormalBold" align="left"><span id="S2Name">Lab Humidity</span> : </td>
     <td class="NormalBold" align="left"><span id="S2Val">19.0</span> <span id="S2U">% RH</span></td>
    </tr>
    <tr>
     <td class="TitleSm" align="right">Sensor 3: </td>
     <td class="NormalBold" align="left"><span id="S3Name">Duct Pressure</span> : </td>
     <td class="NormalBold" align="left"><span id="S3Val">2.4</span> <span id="S3U">&quot; wc</span></td>
    </tr>
    <tr>
     <td class="TitleSm" align="right">Sensor 4: </td>
     <td class="NormalBold" align="left"><span id="S4Name">Fan Current</span> : </td>
     <td class="NormalBold" align="left"><span id="S4Val">25.0</span> <span id="S4U">Amps</span></td>
    </tr>
   </table>
  </td>
 </tr>
</table>
```

***Figure 2: Maverick HTML Output from* start.html**

Figure 3 contains part of the JavaScript code that updates the HTML from the XML. The comments were added for clarification.

```
//Code to get the Sensor Value from the XML
SValue = xmlDocument.getElementsByTagName("SensorValue");
SensorVal = SValue[i].childNodes[0].nodeValue;

//Code used to update the HTML page with the XML data for the sensor
document.getElementsByTagName("span")[3*i+3].innerHTML = SensorVal;
```

***Figure 3: Maverick JavaScript code in* start.html**

The *id* attributes of the span node are highlighted in **Figure 2**. **Figure 4** shows another way to write the code to take advantage of the *getElementById()* method.

```
//Code to get the Sensor Value from the XML
SValue = xmlDocument.getElementsByTagName("SensorValue");
SensorVal = SValue[i].childNodes[0].nodeValue;

//Code used to update the HTML page with the XML data for the sensor
document.getElementById("S"+Number(i+1)+ "Val").innerHTML = SensorVal;
```

Figure 4: JavaScript* getElementById() *Example

TRANSFORMING MAVERICK XML INTO HTML

Another XML technology called XSL, eXtensible Stylesheet Language, is used to transform XML data into another format like HTML, SVG, or reformatted XML. **Figure 5** shows how XML combined with XSL can be transformed using XSL Transformations into HTML or SVG for presentation. XML used in this way provides a separation of data from presentation allowing the data to be presented in different ways without having to write script to get the data in each web page.

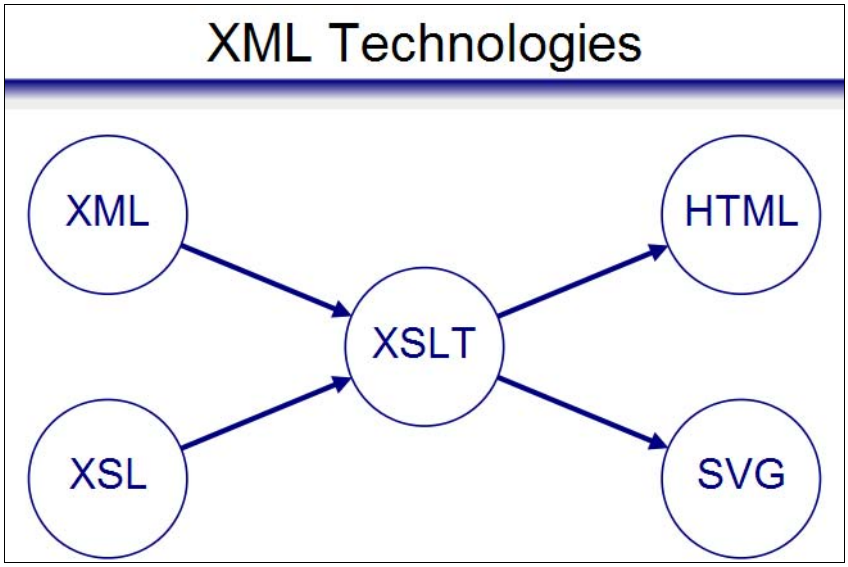


Figure 5: Transforming XML for Presentation

Figure 6 shows an XSL stylesheet that transforms the Maverick XML into HTML for presentation. The XSL stylesheet is XML and as such has an XML declaration as the first line of the file. The stylesheet has namespace definitions for the XSL itself and for the Maverick XML. These are defined in the *xmlns:xsl* and *xmlns:mav* attributes of the <xsl:stylesheet> node. Note the *xsl:* prefix to the '*stylesheet*' node. This informs the XSL transformation software that this node is XSL and not XML from some other technology or device like the Maverick. Similarly, the *mav:* prefix defines the XML for the Maverick and must be used when accessing the Maverick nodes.

The XSL stylesheet in **Figure 6** has a header where it displays the NodeID of the Maverick and the current date and time. It also creates two (2) tables -- one for the inputs and one for the outputs.

Transforming the XML from **Figure 1** using the XSL stylesheet from **Figure 6** is done with some php code shown in **Figure 7**. The php code reads in the XML file and the XSL file. An *xsltProcessor* object is created to do the transformation. The XSL stylesheet previously read in is imported into the *xsltProcessor* object, then the *transformToDoc()* method is called passing the XML file previously read in as a parameter. The *saveHTML()* method of the resulting document is called to get the result in HTML text for outputting via the print statement. The result of the transformation is shown in **Figure 8**.

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:mav="http://mamacsys.com/pc10144" exclude-result-prefixes="mav">
<xsl:output method="html" encoding="iso-8859-1" indent="yes"/>

<xsl:template match="mav:Maverick" xml:space="preserve">
  <html>
  <head>
    <link rel="stylesheet" type="text/css" href="MaverickStyle.css" />
    <title><xsl:value-of select="mav:NodeID" /></title>
  </head>
  <body>
    <center>
    <h2>Maverick IP Sensor Data</h2>
    <h4><xsl:value-of select="mav:NodeID" />:
          <xsl:value-of select="mav:CurrentDate" />  <xsl:value-of select="mav:CurrentTime" />
    </h4>

    <div id="inputs">
      <h3>Inputs</h3>
      <table border="1" cellspacing="0" class="inputs">
        <tr>
          <th>Channel</th> <th>Sensor Name</th> <th>Value</th> <th>Units</th> <th>Alert</th>
        </tr>
        <xsl:for-each select="mav:Sensor">
          <tr>
            <td align="center"><xsl:value-of select="@Channel" xmlns="http://mamacsys.com/pc10144" /></td>
            <td align="center"><xsl:value-of select="mav:SensorName" /></td>
            <td align="right"><xsl:value-of select="mav:SensorValue" /></td>
            <xsl:variable name="units"><xsl:value-of select="mav:SensorUnits" /></xsl:variable>
            <xsl:choose>
              <xsl:when test="string-length($units) &gt; 0">
                <td align="left"><xsl:value-of select="$units" /></td>
              </xsl:when>
              <xsl:otherwise><td align="center">-</td></xsl:otherwise>
            </xsl:choose>
            <td align="center"><xsl:value-of select="mav:SensorAlert" /></td>
          </tr>
        </xsl:for-each>
      </table>
    </div>

    <div id="outputs">
      <h3>Outputs</h3>
      <table border="1" cellspacing="0" class="outputs">
        <tr>
          <th>Channel</th> <th>Output Name</th> <th>Value</th> <th>Control</th>
        </tr>
        <xsl:for-each select="mav:Output">
          <tr>
            <td align="center"><xsl:value-of select="@Channel" xmlns="http://mamacsys.com/pc10144" /></td>
            <td align="center"><xsl:value-of select="mav:OutputName" /></td>
            <td align="center"><xsl:value-of select="mav:OutputValue" /></td>
            <td align="center"><xsl:value-of select="mav:OutputControl" /></td>
          </tr>
        </xsl:for-each>
      </table>
    </div>
    </center>
  </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

Figure 6: XSL Stylesheet for the Maverick XML

```
<?php
  header("Expires: -1");
  header("Content-Type: text/html");

  $xmlFile = "http://192.168.1.10/pc10144.xml";
  $xslFile = "mav.xsl";

  #-----------------------------------------
  #  Load the xml and xsl files
  #-----------------------------------------
  $xml = new DomDocument;
  $xml->load($xmlFile);

  # Load the XSL stylesheet
  $xsl = new DomDocument;
  $xsl->load($xslFile);

  #-----------------------------------------
  #  Return the output of the transformation
  #-----------------------------------------

  /* create the processor */
  $xslt = new xsltProcessor;

  /* import the stylesheet */
  $xslt->importStylesheet($xsl);

  /* transform and output the xml document */
  $result = $xslt->transformToDoc($xml)->saveHTML();
  print $result;

  /* free memory */
  unset($xslt);
  unset($xsl);
  unset($xml);
?>
```

Figure 7: PHP Code to Transform the Maverick XML into HTML

Maverick IP Sensor Data

MAMAC UNO: May 9, 2008 13:34:59

Inputs

Channel	Sensor Name	Value	Units	Alert
1	Lab Temp	70.3	F	Alert
2	Lab Humidity:	19.0	% RH	Alert
3	Duct Pressure	2.4	" wc	Normal
4	Fan Current	25.0	Amps	Normal

Outputs

Channel	Output Name	Value	Control
5	Supply Fan	On	Automatic
6	DX Cooling	On	Automatic
7	Humidifier	Off	Automatic
8	Exhaust Fan	On	Manual

Figure 8: HTML Presentation of Maverick XML Data

COMBINING XML FROM MULTIPLE MAVERICKS AND TRANSFORMING INTO HTML

This section will demonstrate how to combine the XML data from multiple Mavericks and transform that data into a single HTML page. First, the XML data needs to be combined into a single XML document. Since XML documents can only have one root element, that gets created first. Each Maverick XML document will then be appended as a child node of the root document node.

To see how to do that, look at the php code in **Figure 9** and compare that to the php code from **Figure 7**. The main document is created with a root tag <Mavericks> and has the same namespace attribute as the Maverick XML. Next the first XML file is loaded into a new XML document, then the root node and all child nodes are imported into the main document, and finally appended to the root node of the main document.

The second XML document is appended to the main document in the same fashion. The main document XML now has the structure shown in **Figure 10** (without showing all the child nodes for each Maverick). The remaining php code then reads an XSL stylesheet that supports the new XML structure and transforms the XML into HMTL just like the php code of **Figure 7** does. **Figure 11** shows the XSL stylesheet used for this transformation. **Figure 12** shows the HTML result.

```
<?php
  header("Expires: -1");
  header("Content-Type: text/html");

  $xmlFile = "mav.xml";
  $xmlFile2 = "mav2.xml";
  $xslFile = "Mavericks.xsl";

  #-----------------------------------------
  #  Create the main document and append child nodes
  #-----------------------------------------
  $doc = new DOMDocument();
  $doc->loadXML("<Mavericks xmlns='http://mamacsys.com/pc10144'></Mavericks>");

  #  Load the first xml file and import it and all child nodes into the main document
  $xml = new DomDocument;
  $xml->load($xmlFile);
  $node = $doc->importNode($xml->documentElement, true);
  $doc->documentElement->appendChild($node);

  #  Load the second xml file and import it and all child nodes into the main document
  $xml->load($xmlFile2);
  $node = $doc->importNode($xml->documentElement, true);
  $doc->documentElement->appendChild($node);
  #-----------------------------------------
  # Load the XSL stylesheet
  #-----------------------------------------
  $xsl = new DomDocument;
  $xsl->load($xslFile);
  #-----------------------------------------
  #  Return the output of the transformation
  #-----------------------------------------

  /* create the processor */
  $xslt = new xsltProcessor;

  /* import the stylesheet */
  $xslt->importStylesheet($xsl);

  /* transform and output the xml document */
  $result = $xslt->transformToDoc($doc)->saveHTML();
  print $result;

  /* free memory */
  unset($xslt);
  unset($xsl);
  unset($xml);
  unset($doc);
?>
```

Figure 9: PHP code to transform multiple Mavericks' XML into HTML

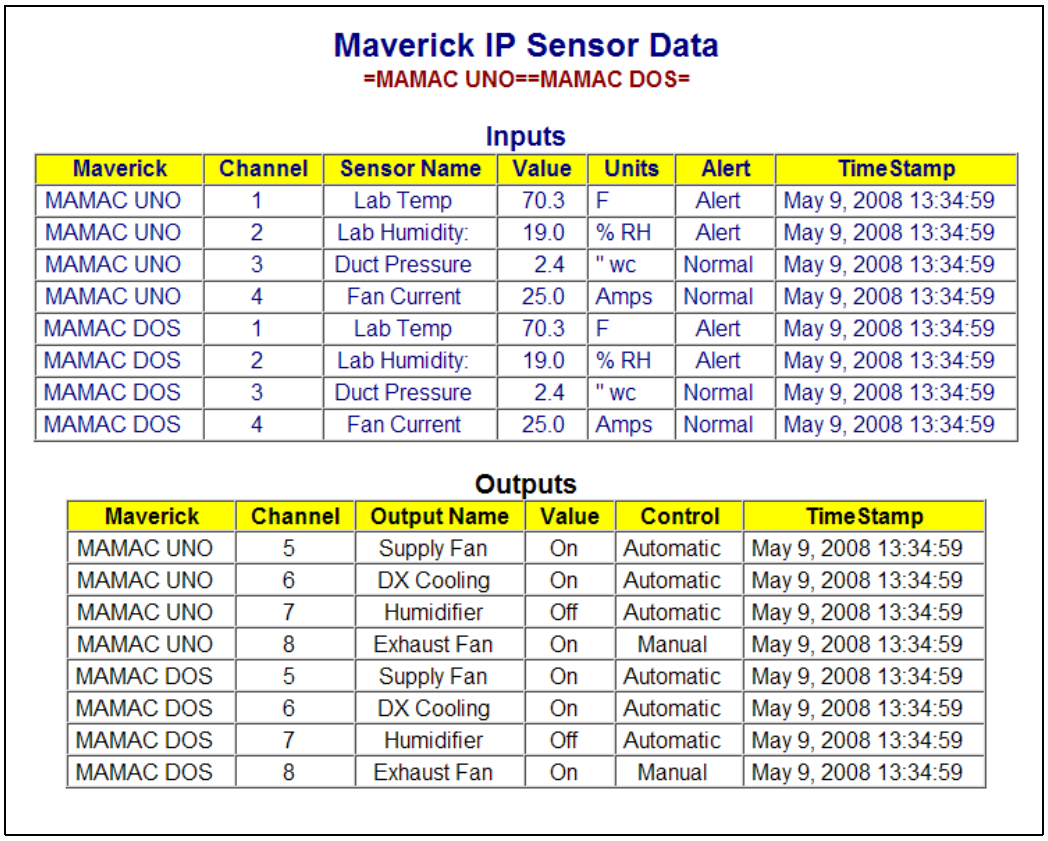
```
<?xml version="1.0"?>
<Mavericks xmlns="http://mamacsys.com/pc10144">
 <Maverick>
  ….
 </Maverick>
 <Maverick>
  ….
 </Maverick>
</Mavericks>
```

Figure 10: HTML XML Structure After Combining Multiple Mavericks' XML

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:mav="http://mamacsys.com/pc10144" exclude-result-prefixes="mav">
<xsl:output method="html" encoding="iso-8859-1" indent="yes"/>
<xsl:template match="mav:Mavericks" xml:space="preserve">
 <html>
 <head>
  <link rel="stylesheet" type="text/css" href="MaverickStyle.css" />
  <title>Maverick IP Sensor Data</title>
 </head>
 <body>
  <center>
  <h2>Maverick IP Sensor Data</h2>
  <h4><xsl:for-each select="mav:Maverick">=<xsl:value-of select="mav:NodeID" />=</xsl:for-each></h4>
  <div id="inputs">
   <h3>Inputs</h3>
   <table border="1" cellspacing="0" class="inputs">
    <tr>
     <th>Maverick</th><th>Channel</th><th>Sensor Name</th><th>Value</th><th>Units</th>
     <th>Alert</th><th>TimeStamp</th>
    </tr>
    <xsl:for-each select="mav:Maverick/mav:Sensor">
     <tr>
      <td align="center"><xsl:value-of select="../mav:NodeID" /></td>
      <td align="center"><xsl:value-of select="@Channel" xmlns="http://mamacsys.com/pc10144" /></td>
      <td align="center"><xsl:value-of select="mav:SensorName" /></td>
      <td align="right"><xsl:value-of select="mav:SensorValue" /></td>
      <xsl:variable name="units"><xsl:value-of select="mav:SensorUnits" /></xsl:variable>
      <xsl:choose>
       <xsl:when test="string-length($units) &gt; 0">
        <td align="left"><xsl:value-of select="$units" /></td>
       </xsl:when>
       <xsl:otherwise><td align="center">-</td></xsl:otherwise>
      </xsl:choose>
      <td align="center"><xsl:value-of select="mav:SensorAlert" /></td>
      <td align="center"><xsl:value-of select="../mav:CurrentDate" />  <xsl:value-of select="../mav:CurrentTime" /></td>
     </tr>
    </xsl:for-each>
   </table>
  </div>

  <div id="outputs">
   <h3>Outputs</h3>
   <table border="1" cellspacing="0" class="outputs">
    <tr>
     <th>Maverick</th><th>Channel</th><th>Output Name</th><th>Value</th>
     <th>Control</th><th>TimeStamp</th>
    </tr>
    <xsl:for-each select="mav:Maverick/mav:Output">
     <tr>
      <td align="center"><xsl:value-of select="../mav:NodeID" /></td>
      <td align="center"><xsl:value-of select="@Channel" xmlns="http://mamacsys.com/pc10144" /></td>
      <td align="center"><xsl:value-of select="mav:OutputName" /></td>
      <td align="center"><xsl:value-of select="mav:OutputValue" /></td>
      <td align="center"><xsl:value-of select="mav:OutputControl" /></td>
      <td align="center"><xsl:value-of select="../mav:CurrentDate" />  <xsl:value-of select="../mav:CurrentTime" /></td>
     </tr>
    </xsl:for-each>
   </table>
  </div>
  </center>
 </body>
 </html>
</xsl:template>
</xsl:stylesheet>
```

Figure 11: XSL Stylesheet to Transform Multiple Mavericks' XML to HTML



Maverick IP Sensor Data

=MAMAC UNO==MAMAC DOS=

Inputs

Maverick	Channel	Sensor Name	Value	Units	Alert	TimeStamp
MAMAC UNO	1	Lab Temp	70.3	F	Alert	May 9, 2008 13:34:59
MAMAC UNO	2	Lab Humidity:	19.0	% RH	Alert	May 9, 2008 13:34:59
MAMAC UNO	3	Duct Pressure	2.4	" wc	Normal	May 9, 2008 13:34:59
MAMAC UNO	4	Fan Current	25.0	Amps	Normal	May 9, 2008 13:34:59
MAMAC DOS	1	Lab Temp	70.3	F	Alert	May 9, 2008 13:34:59
MAMAC DOS	2	Lab Humidity:	19.0	% RH	Alert	May 9, 2008 13:34:59
MAMAC DOS	3	Duct Pressure	2.4	" wc	Normal	May 9, 2008 13:34:59
MAMAC DOS	4	Fan Current	25.0	Amps	Normal	May 9, 2008 13:34:59

Outputs

Maverick	Channel	Output Name	Value	Control	TimeStamp
MAMAC UNO	5	Supply Fan	On	Automatic	May 9, 2008 13:34:59
MAMAC UNO	6	DX Cooling	On	Automatic	May 9, 2008 13:34:59
MAMAC UNO	7	Humidifier	Off	Automatic	May 9, 2008 13:34:59
MAMAC UNO	8	Exhaust Fan	On	Manual	May 9, 2008 13:34:59
MAMAC DOS	5	Supply Fan	On	Automatic	May 9, 2008 13:34:59
MAMAC DOS	6	DX Cooling	On	Automatic	May 9, 2008 13:34:59
MAMAC DOS	7	Humidifier	Off	Automatic	May 9, 2008 13:34:59
MAMAC DOS	8	Exhaust Fan	On	Manual	May 9, 2008 13:34:59

Figure 12: HTML Presentation of Mavericks' XML Data

USING XSL TO TRANSFORM XML INTO REFORMATTED XML

While XSL can transform XML into HTML and SVG as shown in **Figure 5**, it is not limited to those formats. It can also be reformatted into XML. For example, the Maverick XML has eight (8) I/O points, each defined using a *Channel* attribute as shown in the XML of **Figure 1**. The Maverick XML is rather verbose with each of the I/O having several child nodes. It is possible to transform the XML, reducing its size and limiting the data to specific needs, using attributes instead of child nodes. Here is how to do that.

Figure 13 shows the php code, which is the same as the code in **Figure 7** with two main exceptions. The *Content-Type* of the output is "*text/xml*" instead of "*text/html*" because we are transforming the XML into XML, not HTML. The result of the XSL transformation uses the *saveXML()* method of the xsltProcessor instead of the *saveHTML()* method. Again, this is because the output is XML, not HTML.

```
<?php
  header("Expires: -1");
  header("Content-Type: text/xml");

  $xmlFile = "mav.xml";
  $xslFile = "reformat.xsl";

  #-----------------------------------------
  #  Load the xml and xsl files
  #-----------------------------------------
  $xml = new DomDocument;
  $xml->load($xmlFile);

  # Load the XSL stylesheet
  $xsl = new DomDocument;
  $xsl->load($xslFile);

  #-----------------------------------------
  #  Return the output of the transformation
  #-----------------------------------------

  /* create the processor */
  $xslt = new xsltProcessor;

  /* import the stylesheet */
  $xslt->importStylesheet($xsl);

  /* transform and output the xml document */
  $result = $xslt->transformToDoc($xml)->saveXML();
  print $result;

  /* free memory */
  unset($xslt);
  unset($xsl);
  unset($xml);
?>
```

Figure 13: PHP code to reformat the Maverick XML

Figure 14 shows the XML output that is desired. The *SensorValue* and *OutputValue* child nodes are now the text of the *Input* and *Output* nodes. Some of the remaining *Sensor* and *Output* child nodes have been replaced by attributes of the *Input* and *Output* nodes respectively. The resulting XML is considerably less verbose. The attributes can easily be obtained in code by using the Document Object Model (DOM) method *getAttribute()*.

```
<?xml version="1.0"?>
<Maverick xmlns="http://mamacsys.com/pc10144">
  <NodeID>MAMAC UNO</NodeID>
  <CurrentTime>13:34:59</CurrentTime>
  <CurrentDate>May 9, 2008</CurrentDate>

  <Input id="1" desc="Lab Temp" units="F" alert="Alert">70.3</Input>
  <Input id="2" desc="Lab Humidity:" units="% RH" alert="Alert">19.0</Input>
  <Input id="3" desc="Duct Pressure" units="&amp;quot; wc " alert="Normal">2.4</Input>
  <Input id="4" desc="Fan Current" units="Amps" alert="Normal">25.0</Input>

  <Output id="5" desc="Supply Fan" mode="Automatic">On</Output>
  <Output id="6" desc="DX Cooling" mode="Automatic">On</Output>
  <Output id="7" desc="Humidifier" mode="Automatic">Off</Output>
  <Output id="8" desc="Exhaust Fan" mode="Manual">On</Output>
</Maverick>
```

Figure 14: Reformatted Maverick XML

The XSL stylesheet in **Figure 15** was used to convert the Maverick XML into the reformatted XML. This stylesheet is considerably different from the others in this document. XSL is not a programming language, but more of a text replacement language. It reads the input XML matching nodes and replacing text with the text specified in the stylesheet. When XSL sees the *apply-templates* element it looks for matching templates for the text it is working on.

Besides the main template, there are three (3) additional templates in the stylesheet. If it matches a <Maverick> node with a child <Sensor> node, it outputs the text for that template. If it matches a <Maverick> node with a child <Output> node, it outputs the text for that template. If it matches a <Maverick> node, but is not a <Sensor> or <Output> node, the template will just copy the text to the output. This last template copies the <NodeID>, <CurrentTime>, and <CurrentDate> nodes to the output without any reformatting.

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:mav="http://mamacsys.com/pc10144" exclude-result-prefixes="mav">
<xsl:template match="/" xmlns="http://mamacsys.com/pc10144">
  <Maverick>
    <xsl:for-each select=".">
      <xsl:apply-templates />
    </xsl:for-each>
  </Maverick>
</xsl:template>

<xsl:template match="mav:Maverick/mav:Sensor" xmlns="http://mamacsys.com/pc10144">
  <xsl:if test="mav:SensorName!='xxxx'">
    <Input>
      <xsl:attribute name="id"><xsl:value-of select="@id" /></xsl:attribute>
      <xsl:attribute name="desc"><xsl:value-of select="mav:SensorName" /></xsl:attribute>
      <xsl:attribute name="units"><xsl:value-of select="mav:SensorUnits" /></xsl:attribute>
      <xsl:attribute name="alert"><xsl:value-of select="mav:SensorAlert" /></xsl:attribute>
      <xsl:value-of select="mav:SensorValue" />
    </Input>
  </xsl:if>
</xsl:template>

<xsl:template match="mav:Maverick/mav:Output" xmlns="http://mamacsys.com/pc10144">
  <xsl:if test="mav:OutputName!='xxxx'">
    <Output>
      <xsl:attribute name="id"><xsl:value-of select="@id" /></xsl:attribute>
      <xsl:attribute name="desc"><xsl:value-of select="mav:OutputName" /></xsl:attribute>
      <xsl:attribute name="mode"><xsl:value-of select="mav:OutputControl" /></xsl:attribute>
      <xsl:value-of select="mav:OutputValue" />
    </Output>
  </xsl:if>
</xsl:template>

<xsl:template match="mav:Maverick/*">
  <xsl:copy-of select="." />
</xsl:template>

</xsl:stylesheet>
```

Figure 15: XSL Stylesheet to Reformat the Maverick XML

MORE WAYS TO APPLY XML

Figure 5 shows that XML can be transformed into SVG. Examples of that are not shown in this document. But the XML could just as easily be transformed into an SVG graphic with the data displayed in a similar table format. But typically the real-time XML data will just update a static SVG graphic using JavaScript. If the logging data was XML instead of CSV, a stylesheet could transform the interval data into an SVG graph.

The Maverick XML data allows AJAX techniques to be used to update HTML pages and SVG graphics such as Dashboards, panel meters, real-time graphs, strip charts, etc. It is really up to one's imagination.

This document used server side XSL transforms using PHP, but XSL transforms can be done using JavaScript on the client browser. Unfortunately, it is a security risk to allow cross-domain access to XML data, so getting data from multiple Mavericks is not allowed. As a Firefox add-on or desktop application, this is not an issue and it works just fine.

MAVERICK WEB PAGES ARE XML

All the Maverick web pages are valid XML. If data not available from the Maverick XML is desired, the appropriate page can be read using *XMLHttpRequest* object and parsed using standard *DOM* methods. Web pages that are not valid XML must be searched for the desired data using string functions, which is very prone to error when any change is made to the web page. By using XML to parse the page for the desired data, the process is much less prone to error when changes are made. **Figure 16** shows an example of Maverick "screen scraping" using XML parsing techniques. Note that the *DOCTYPE* declaration needs to be removed from the XML text before loading the text as an XML document. The *debug* output from executing the code is shown in **Figure 17**.

```
  Dim xmlHttp As New MSXML2.ServerXMLHTTP
  Dim xmlDoc As New MSXML2.DOMDocument
  Dim nodes As MSXML2.IXMLDOMNodeList
  Dim element As MSXML2.IXMLDOMElement
  Dim html As String
  Const htmlDOCTYPE As String = "<!DOCTYPE html PUBLIC ""-//W3C//DTD XHTML 1.0
Transitional//EN"" ""http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"">"

  xmlHttp.open "GET", "http://192.168.1.10/channel1.htm", False, "admin", "password"
  xmlHttp.send

  html = Replace(xmlHttp.responseText, htmlDOCTYPE & vbLf, "")
  xmlDoc.loadXML html

  Set nodes = xmlDoc.getElementsByTagName("input")
  Set element = nodes(10)

  Debug.Print "input(10) id = " & element.getAttribute("id")
  Debug.Print "input(10) name = " & element.getAttribute("name")
  Debug.Print "input(10) value = " & element.getAttribute("value")
```

Figure 16: Visual Basic 6 Code to Parse a Maverick HTML Page as XML

```
input(10) id = RelayLTValue
input(10) name = MAV_01_07
input(10) value = 70.0
```

Figure 17: Debug Output From Running Visual Basic 6 Code in Figure 16

The document has shown some of what can be done with the Maverick XML. It demonstrates the powerful tools available for XML from the Document Object Model properties and methods to the XML languages like XSL, SVG, and XUL. To learn more, visit the World Wide Web Consortium (W3C) website http://www.w3.org and/or visit the free online tutorials on these standards at http://www.w3schools.com.

XML APPLICATIONS

XML and JavaScript are the tools of which widgets and gadgets are made. The two combine to provide a very powerful platform. For example, Firefox add-ons are all written in XML and JavaScript. Mozilla uses an XML language of their own to provide the menus, buttons, and other elements of the Firefox browser. They call it XUL. Firefox add-ons and desktop applications can be written using XUL. Like Firefox, XUL is free and open source.

A Firefox toolbar add-on for the Maverick is available and desktop applications are being developed as this document is being written. An initial release of a desktop application to manage the real-time data from multiple Mavericks is available. Another application is in development to manage Maverick logging data (.csv files) for long-term data storage using Firefox/XULRunner Storage API. The Storage API saves data in a SQLite database.

SQLite (http://www.sqlite.org) is a software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. An SQLite database is a single disk file, so backing up the database is as simple as copying a file. There are numerous implementations of SQLite with details available at http://www.sqlite.org/cvstrac/wiki?p=SqliteWrappers.

A free SQLite Manager is available at http://code.google.com/p/sqlite-manager.