A REVELATION WHITE PAPER

AJAX & Opensight

By David Goddard
Revelation Software Pty Ltd
and featured in International Spectrum Magazine
Issue dated May/June 2006

Contents

CONTENTS 3

INTRODUCTION 4

- The "buzzword" part 4
- The "understanding" part 4
- How AJAX works 5
- Using AJAX with MultiValue Databases (e.g. OpenInsight) 6

HOW TO AJAX YOUR OPENINSIGHT 7

- Step 1. - Create an HTML page 7
- Step 2 – Add JavaScript to request the XML data from OpenInsight 8
- JavaScript Function number 1 – getXMLData 8
- JavaScript Function No 2 – loadXMLDoc 8
- Step 3 – Capture the XML Document in the state_Change function 9
- A quick note on the XML Document Object Model (DOM) 11

PUTTING IT ALL TOGETHER 12

COPYRIGHT NOTICE
© 2004 Revelation Software Limited. All rights reserved.

No part of this publication may be reproduced by any means, be it transmitted, transcribed, photocopied, stored in a retrieval system, or translated into any language in any form, without the written permission of Revelation Software Limited.

TRADEMARK NOTICE
OpenInsight is a registered trademark of Revelation Technologies, Inc. Advanced Revelation is a registered trademark of Revelation Technologies, Inc. OpenInsight for Workgroups is a registered trademark of Revelation Technologies, Inc. Report Builder+ is a trademark of Revelation Technologies, Inc.

Microsoft, MS, MS-DOS, Windows, are registered trademarks of Microsoft Corporation in the USA and other countries. IBM is a registered trademark of International Business Machines Corporation. Intel is a registered trademark of Intel Corporation. Lotus, Lotus Notes and Notes are registered trademarks of Lotus Development Corporation. All other product names are trademarks or registered trademarks of their respective owners.

Introduction

The "buzzword" part

When I was kid growing up in the suburbs, my mom used AJAX® branded products to clean the clothes, the bath, the floor, well just about everything! I was more an ACME brand guy, something to do with Wile E. Coyote I guess.

Anyway, fast-forward 30 years and now I'm using AJAX, not the cleaning products, but the latest buzzword to hit cyberspace. AJAX, the buzzword, stands for **A**synchronous **J**avaScript **A**nd **X**ML.

So what's so new about JavaScript and XML? Not much really. They're the same as they were last week when AJAX was still a cleaning product. The important letter in this an acronym is the first "A", which stands for Asynchronous. This clever little letter is the reason Internet companies in the know, like Google, are using AJAX to build their latest and greatest web based applications.

But you've heard all this before. Web Portals were it, a bit and a packet of chips for a while. What about the Push- Pull and dot Net? So why is AJAX any different? Why should you spend your valuable time on this latest development craze? Because this stuff is easy, it actually works, and you can start doing it now with the tools and knowledge you already have!

The "understanding" part

Since the early days of the Internet, developers have searched for a standard and browser independent way to asynchronously load content on an existing web page without requiring a full reload of the page. Microsoft made early attempts in 1996 when they introduced the iframe element into Internet Explorer 3.0. Netscape responded when they introduced the now defunct layer element into Netscape 4.

Over the last 10 years techniques like Remote Scripting (1998) and custom Java applets have come and gone. The web development community, collaborating through newsgroups and blogs, refined these techniques until, in 2002, a new standard was created called XMLHttpRequest. This standard is a set of API's that can be used by many web browser scripting languages to transfer and manipulate XML data to and from a server using HTTP, and thus establishing an independent connection channel between the client and the server.

XMLHttpRequest is the glue that makes AJAX work. It's that important first "A".

AJAX is not a product or a language; it is a methodology for building interactive web applications. It makes web pages feel more responsive by using JavaScript and XML to send and receive small amounts of data between a browser and the server behind the scenes, therefore removing the requirement for the web page to be fully reloaded to view any changes. This helps to make web applications smaller, faster and more user friendly.

AJAX is based on the following open web standards:

- XHTML (eXtensible HyperText Markup Language)
- JavaScript
- XML (eXtensible Markup Language)
- CSS (Cascading Style Sheets)

XHTML is compatible with the W3C standard HTML 4.01 and will eventually replace HTML. It is supported by all new browsers including Internet Explorer, Firefox, and Opera on most operating systems.

XHTML, in combination with CSS, controls the styling of the web page.

JavaScript is used as a client-side scripting language to interact with the HTML DOM (Document Object Model) to dynamically display and interact with the user and the information presented. Other client-side scripting languages can be used (like PHP) as well as server-side scripting like ASP, although a client-side language is required to interact with the user via their browser.

XML is a commonly used format for transferring data from and to the server, although any format will work including preformatted HTML or plain text.

AJAX is web server, browser, and operating system independent. Its applications can reach a larger audience and are easier to install and support than desktop applications.

As you can see, the technologies behind AJAX have been around for several years, but now, under the AJAX banner, have become more widely known and adopted. "Google Earth" and "Google GMAIL" are examples of web applications based on the AJAX methodology.

How AJAX works

A traditional web application will submit input (using an HTML form) to a web server. After the web server has processed the data, it will return a completely new web page to the user. Because the server returns a new web page each time the user submits input, traditional web applications often run slowly and tend to be less user-friendly.

With AJAX, web applications can send and retrieve data without reloading the whole web page. This is done by sending an XMLHttpRequest to the server and by modifying only parts of the web page using JavaScript.

The preferred way to communicate with the server is by sending and receiving data as XML, but other methods can be used.

Using AJAX with MultiValue Databases (e.g. OpenInsight)

MultiValue databases like OpenInsight work easily with XML. This is because XML is delimited data, but instead of using field (attribute) marks and value marks, XML uses tags.

OpenInsight, for example, has built in functions to send and receive XML data. This makes it very easy to use OpenInsight as the database in an AJAX application.

In this white paper we have a simple example of using AJAX techniques to receive data from OpenInsight and display it in an HTML page.

In the example we will be using OECGI.EXE as the means of connecting to OpenInsight from the web server. For more information on setting up and using OECGI.EXE please refer to the OpenInsight help files or the knowledge base section on www.revelation.com.

The example will request the lname, fname, city, state, and zip columns from the customer table, found in the OpenInsight EXAMPLES application, sorting the data by lname, and then display this on the HTML page.

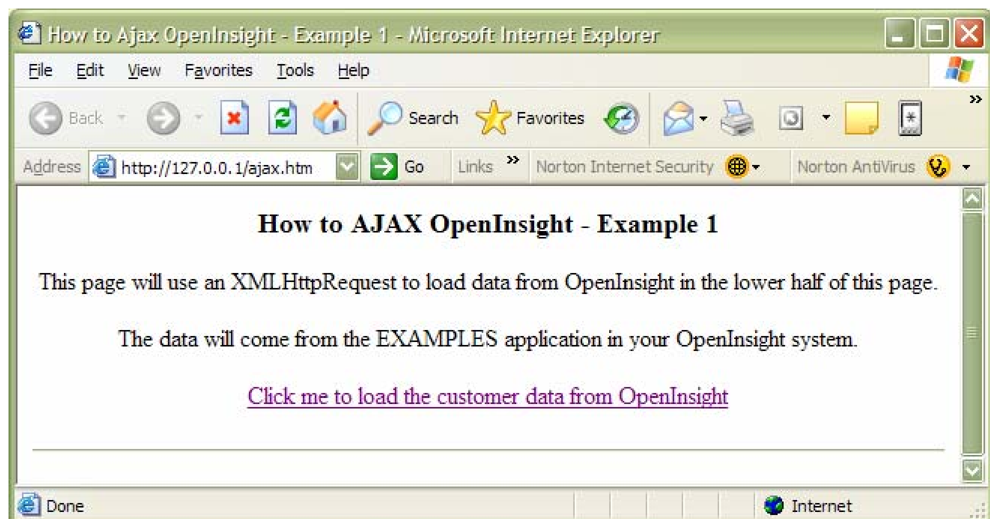
How to AJAX your OpenInsight

Step 1. - Create an HTML page

Create an HTML page that contains the code shown below:

```
<html>
<head>
    <title>How to AJAX OpenInsight</title>
</head>
<body>
    <h3 align="center">How to AJAX OpenInsight</h3>
    <p align="center">
    This page will use an XMLHttpRequest to load data from
    OpenInsight in the lower half of this page.<br /><br />
    The data will come from the EXAMPLES application in your
    OpenInsight system.
    </p>
    <p align="center"><a href="#" onclick="getXMLData()">Click me to
    load the customer data from
    OpenInsight</a></p>
    <hr/>
    <span id="datafromoi"></span>
</body>
</html>
```

This will produce a page that appears in the browser as seen below:



Step 2 – Add JavaScript to request the XML data from OpenInsight

We need a few simple JavaScript functions to perform the connection to OpenInsight and retrieve the XML data. This code needs to be added inside the head element of your html page.

JavaScript Function number 1 – getXMLData

In the first function, getXMLData (shown below), we set up the URL we need to pass to the web server to retrieve the XML data. In this case, we are using OECGI.EXE to make the connection to OpenInsight. We call this function from the onload event in the body tag of the HTML page above.

```
function getXMLData() {
    var url = "http://localhost/cgi-bin/oecgi.exe/inet_xml?
    cmd=list customers by lname lname fname company phone fax email"
loadXMLDoc(url)
}
```

OECGI.EXE makes a call to the BASIC+ function INET_XML. The source code for INET_XML can be found in the SYSPROG account of your OpenInsight system. We pass a standard RLIST statement to INET_XML via OECGI .EXE, which it uses to select the table, rows, and columns to return as an XML document.

For Example:

The following call to OECGI.EXE will start the inet_xml BASIC+ routine. It will take the Rlist command specified by the cmd= parameter and select the customers table by the lname column, returning the lname, fname city state city columns as an XML document.

Eg: http://localhost/cgi-bin/oecgi.exe/ inet_xml?cmd=list customers by lname fname lname city state zip

The getXMLData function then calls a JavaScript function called loadXMLDoc which will make the connection to OpenInsight to get the data.

JavaScript Function No 2 – loadXMLDoc

The loadXMLDoc function uses the XMLHttpRequest object to make a connection to the web server to request the data.

Please note that the syntax used to setup the XMLHttpRequest varies between browsers. The example below handles IE and Mozilla browser types.

XMLHttpRequest makes an asynchronous connection to the server. This means that it sends the request, but does not wait for a response. Instead, we have to tell XMLHttpRequest the name of a function to call when the state of our request changes.

This is done by setting the onreadystatechanged property of the XMLHttpRequest object.

In the example shown below, we set this to call a function called status_Change. As the status changes for our XML request, this function will be called. (See step 3 for more information).

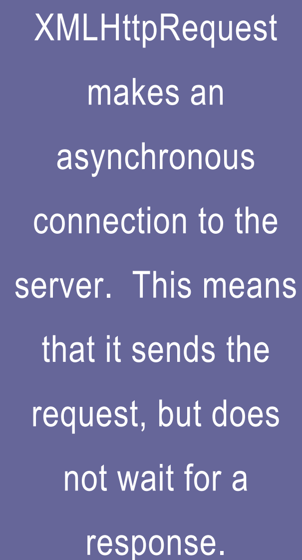
```
function loadXMLDoc(url) {
// code for Mozilla, etc.
if (window.XMLHttpRequest)
    {
    xmlhttp=new XMLHttpRequest()
    xmlhttp.onreadystatechange=state_Change
    xmlhttp.open("GET",url,true)
    xmlhttp.send(null)
    }
// code for IE
else if (window.ActiveXObject)
    {
    xmlhttp=new ActiveXObject("Microsoft.XMLHTTP")
    if (xmlhttp)
    {
    xmlhttp.onreadystatechange=state_ChangeForm
    xmlhttp.open("GET",url,true)
    xmlhttp.send()
    }
    }
}
```

Step 3 – Capture the XML Document in the state_Change function

This is where all the magic happens. The XMLHttprequest state_Change event (shown below) will call the specified function whenever the state of the XMLHttpRequest changes. The different states are as follows:

- 0 = uninitialized
- 1 = loading
- 2 = loaded
- 3 = interactive
- 4 = complete

Once the state value is 4, the XML data has arrived back at the browser. Any errors are returned in the status and statusText properties.



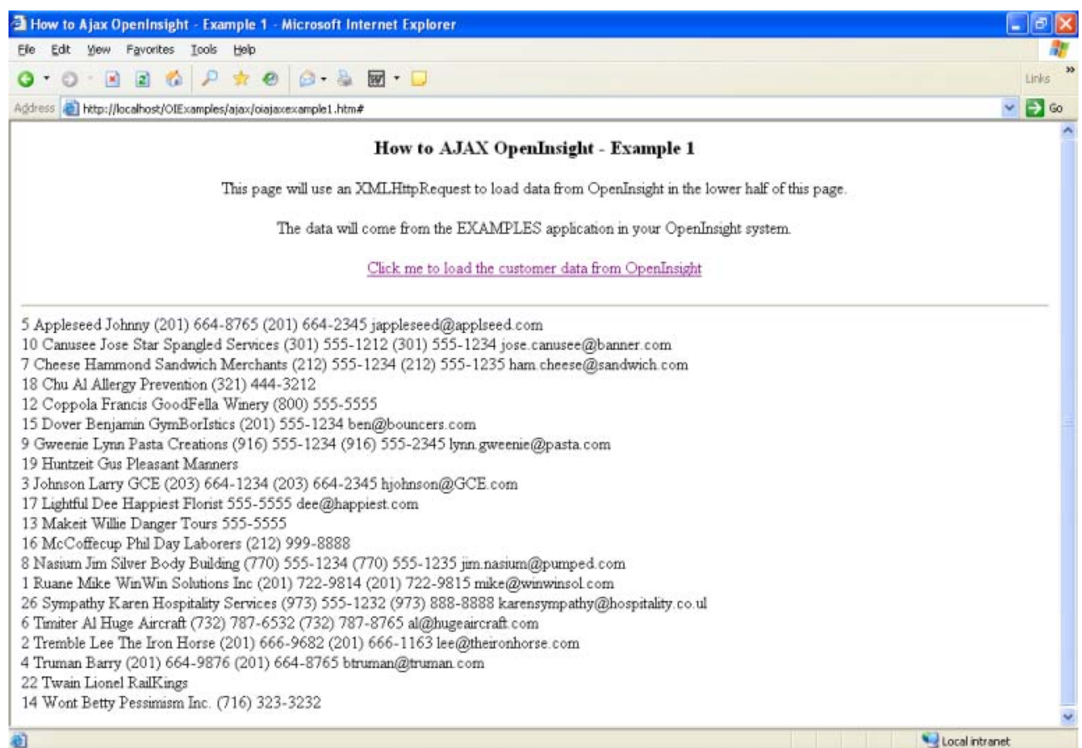
The stateChanged function then initialises an XMLDocument object to hold the XML data returned from the web server. Once we have an XML document object initialized, we can start displaying the XML data in our HTML page. In the example we loop through each row returned and display the fname and lname columns down the page inside the span tag.

```
function state_Change() {
// if xmlhttp shows "loaded"
if (xmlhttp.readyState==4)
    {
    // if "OK"
    if (xmlhttp.status==200)
    {
    //alert("XML data OK")
    //Create the xmldoc object and load it
    // with the data returned from the server
    xmldoc=new ActiveXObject("Microsoft.XMLDOM")
    xmldoc.async="false"
    xmldoc.loadXML(xmlhttp.responseText)

    //Loop through the list of rows returned
    var myhtml = ""
    var rows = xmldoc.selectNodes("/rows/row")
    for (i=0;i<rows.length;i++)
    {
        var thisrow = rows.item(i)

        // Count the number of columns
        var cnt=thisrow.childNodes.length
        for (j=0;j<cnt;j++) {
        // Write the current row to the form
        myhtml = thisrow.childNodes(j).text + " "
        }
        myhtml = myhtml + "<br/>"
        }
    }
}
else
{
    alert("Problem retrieving XML data:" + xmlhttp.statusText)
}
    }
}
```

A quick note on the XML Document Object Model (DOM)

The XML document object model provides a text-based means to describe and apply a tree-based structure to information. There are many ways to read this tree structure. XPATH is one way to navigate and extract information in an XML document.

XPATH is a major element in the W3C XML standards as so is fundamental to a lot of advanced XML usage. Please visit www.w3schools.com for more information on XPATH.



This is the expected result from working through this white paper using the code which is provided in full over the page.

Putting it all together

The code below shows the complete html page containing all the elements required to get you up and going with your first AJAX enabled web page. The source code for this page and another two examples are available to download from the knowledge base section on www.revelation.com. Search the site for the word AJAX.

So, again we can strike up another victory for our little MultiValue product. The market has come up with yet another, "all new, improved" buzzword, and we are buzzword compliant with just a little bit of work. Will it always be this easy? Maybe not. But OpenInsight can give you the AJAX functionality required by customers in a quick and easy fashion.

```
<!doctype html public "-//w3c//dtd html 4.0 transitional//en">
<html>
<head>
<title> How to Ajax OpenInsight - Example 1</title>
<meta name="generator" content="openinsight">
<meta name="author" content="David Goddard">
<meta name="keywords" content="way cool">
<meta name="description" content="OpenInsight, AJAX">
<script type="text/javascript">

function getXMLData() {
    var url = "http://localhost/cgi-bin/oecgi.exe/inet_xml?cmd=list
    customers by lname lname fname company phone fax email"
loadXMLDoc(url)
}

function loadXMLDoc(url) {
// code for Mozilla, etc.
if (window.XMLHttpRequest)
    {
    xmlhttp=new XMLHttpRequest()
    xmlhttp.onreadystatechange=state_Change
    xmlhttp.open("GET",url,true)
    xmlhttp.send(null)
    }
// code for IE
else if (window.ActiveXObject)
    {
    xmlhttp=new ActiveXObject("Microsoft.XMLHTTP")
    if (xmlhttp)
    {
    xmlhttp.onreadystatechange=state_Change
    xmlhttp.open("GET",url,true)
    xmlhttp.send()
    }
}
}
```

```
function state_Change() {
// if xmlhttp shows "loaded"
if (xmlhttp.readyState==4)
{
    // if "OK"
    if (xmlhttp.status==200)
    {
        //alert("XML data OK")
        //Create the xmldoc object and load it
        // with the data returned from the server
        xmldoc=new ActiveXObject("Microsoft.XMLDOM")
        xmldoc.async="false"
        xmldoc.loadXML(xmlhttp.responseText)

        //Loop through the list of rows returned
        // We are using an XPATH statement to select all the rows
        // returned in the XML document.
        var rows = xmldoc.selectNodes("/rows/row")
        var myhtml = ""
        for (i=0;i<rows.length;i++)
        {
             //var thisrow=xmldoc.documentElement.childNodes(i)
             var thisrow = rows.item(i)
             // Count the number of columns in the row
             var cnt=thisrow.childNodes.length
             for (j=0;j<cnt;j++)
             {
             // Write the current row to the form
             myhtml = myhtml + thisrow.childNodes(j).text + " "
             }
             myhtml = myhtml + "<br />"
        }
        document.getElementById("datafromoi").innerHTML=myhtml
        }
    else
    {
    alert("Problem retrieving XML data:" + xmlhttp.statusText)
    }
}
}
</script>
</head>
<body>
```