

# The AJAX Architecture and Scripting Services with E4X

Paul Fremantle  
*VP of Technology*  
WSO2

[paul@wso2.com](mailto:paul@wso2.com)

[www.wso2.com](http://www.wso2.com)



Thanks to Anthony Elder, IBM, who co-wrote the examples,  
and IBM who supported much of the effort in putting this together



# About the Presenter

---

- Ex-IBM Senior Technical Staff Member
  - Led the creation and development of
    - IBM Web Services Gateway
    - Web Services Invocation Framework
    - JWSDL / WSDL4J
- VP of Technology at WSO2
  - [www.wso2.com](http://www.wso2.com)
  - A startup aiming to develop and support leading edge Open Source Web services software
    - Provides support, training and consultancy for Apache Axis 1.x
- Co-Chair of the OASIS WS-Reliable eXchange Technical Committee
  - [http://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=ws-rx](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ws-rx)
- Co-author of “Building Web Services in Java 2<sup>nd</sup> Edition”
- Blog <http://www.bloglines.com/blog/paulfremantle>



# Contents

---

- Introducing AJAX
- Advantages
- Building AJAX clients for REST services
- Rest and SOAP
- Introducing E4X
- Sample browser code
- Writing server logic in E4X
- Conclusions



# Introducing AJAX

---

- Asynchronous JavaScript and XML
  - The practice of using “under the covers” XML/HTTP conversations to enrich a web page
  - The requests do not constitute *page refresh*
  - Results in more intuitive applications
  - Remember the “network is the computer”?
    - The network is the application?
  - Examples:
    - Google – Maps, Mail, autocomplete
    - Google maps can scroll round indefinitely, pulling in information from the network as needed

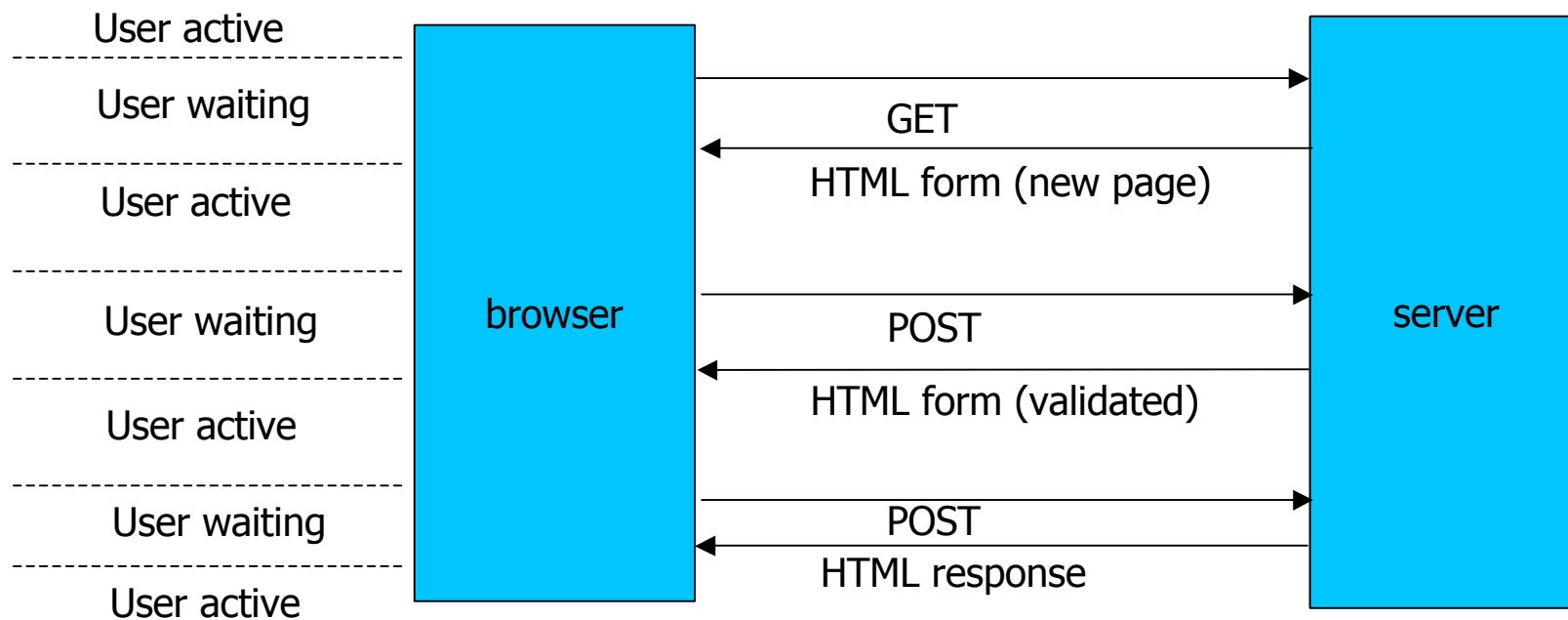


# Why Use AJAX?

---

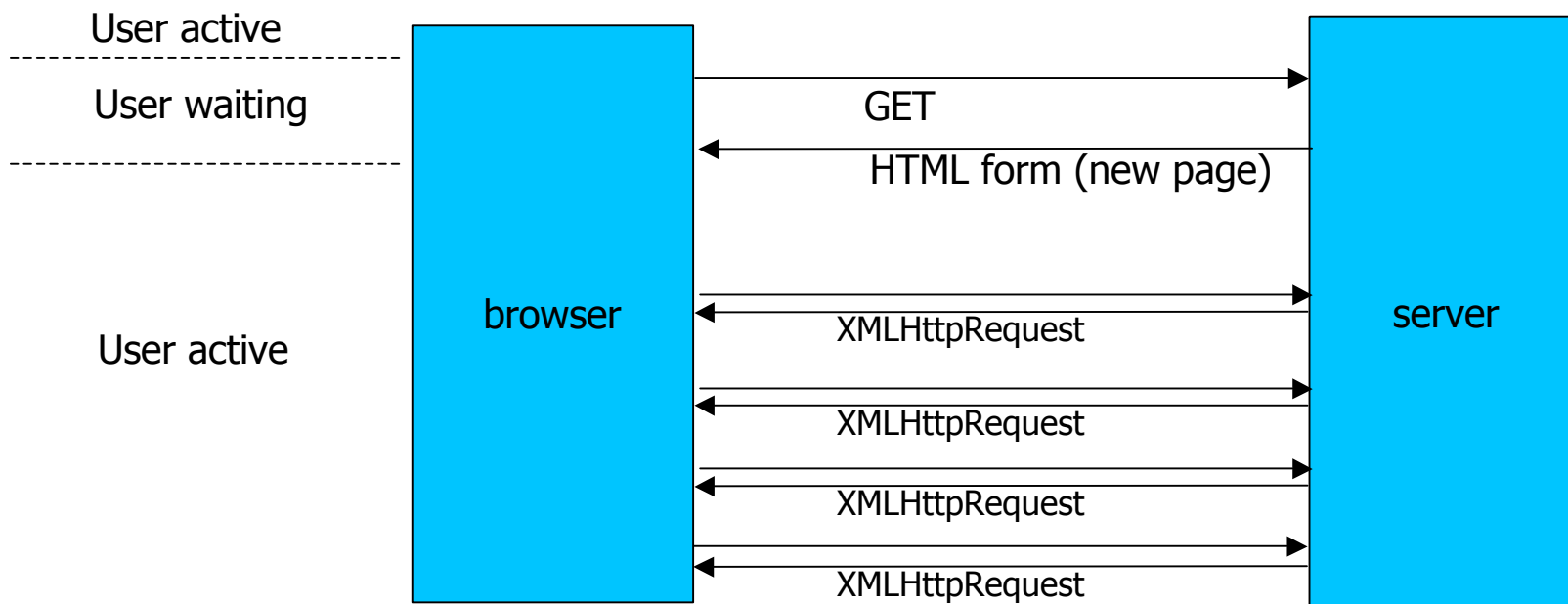
- Er... it's the latest thing?
- Subliminal validation
  - Checking entries *while* the user is typing
  - Prompting users with possible answers
  - Minimising network delays by interleaving network activity with user activity

# Normal Web Application



User gets form, submits, returned with marked errors, resubmits, reads response

# AJAX Web Application



- No page refreshes – user is always in control
- Validation of one field can happen while user is typing another.
- Validation can be less intrusive (like changing the graphical attributes)



# How Does It Work?

---

- XMLHttpRequest
  - Introduced in IE
  - Supports programmatic sending of HTTP requests and parsing the results
  - Not tied to XML – but much easier to correctly parse
  - Usually uses DOM for parsing the XML



# Getting an XMLHttpRequest

---

- Er... there is no standard, cross-browser, single syntax ☹
- Microsoft syntax:
  - `xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");`
    - or
  - `xmlhttp = new ActiveXObject("Msxml2.XMLHTTP");`
- Everyone else
  - `xmlhttp = new XMLHttpRequest();`



# One Size Fits All Code

---

```
var xmlhttp=false;
try {
  xmlhttp = new ActiveXObject("Msxml2.XMLHTTP");
} catch (e) { try {
  xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
} catch (E) { xmlhttp = false; }
}
if (!xmlhttp && typeof XMLHttpRequest!='undefined') {
  xmlhttp = new XMLHttpRequest();
}
```

➤ Thanks to:

- <http://jibbering.com/2002/4/httprequest.html>



# XMLHttpRequest

---

- `var xmlhttp = new XMLHttpRequest();`
- `xmlhttp.open ( method , URL , async );`
  - METHOD = { "GET", "POST", "PUT", *etc.* }
  - URL – as you expect
  - async – block for response or callback
- The async ability is one of the key attributes
- Careful – if you use "false" (*i.e.* sync), then the browser window will appear "hung"
- Browsers are naturally asynchronous



# XMLHttpRequest *(Continued)*

---

```
// set up any HTTP headers
xmlhttp.setRequestHeader("Content-type", "text/xml");
// create the callback function for async
var callback = function() {
    if (xmlhttp.readyState==4) {
        // do something with xmlhttp.responseText
    }
}
// set the event handler
xmlhttp.onreadystatechange = callback;
// send the request
xmlhttp.send("");
```



# DOM in JavaScript

---

- The Document Object Model is a standardised, general purpose Tree structured interface for manipulating XML
- Good for small-ish XML fragments
- Not the simplest or easiest of APIs!
  - `result = document.getElementById("result");`
  - `child = document.createElement("child");`
  - `child.createTextNode("text");`
  - `result.appendChild(child);`



# Using REST with AJAX

---

- REST – Representational State Transfer is the term Roy Fielding coined for the underlying semantic of HTTP {GET, POST, PUT, DELETE}
- It has come to be associated with XML/HTTP as opposed to SOAP
  - REST+XML uses the correct meanings of the HTTP verbs
  - Supports redirects, *etc.*
  - The standard SOAP binding into HTTP uses a POST obscuring the real meaning at the HTTP level



# REST Example

---

- HTTP GET of:

```
http://webservices.amazon.com/onca/xml?Service=AWSECommerceService
&SubscriptionId=XXXXX*
&Operation=ItemSearch
&SearchIndex=Books
&Keywords=Paul%20Fremantle
```

*\* Amazon requires you to subscribe and use your own ID*





# Aside on Security

---

- You should all know that browser security models do not allow:
    - access to data
    - without the user being aware
    - from other sites than the main page has come from
  - So you must download the webpage from the same URL host as the XMLHttpRequest is going to
    - or explicitly allow the browser
- ```
try {  
    netscape.security.PrivilegeManager.enablePrivilege("UniversalXPConnect  
    UniversalBrowserAccess");  
}  
catch (E) {}
```



# Response (Edited to Fit!)

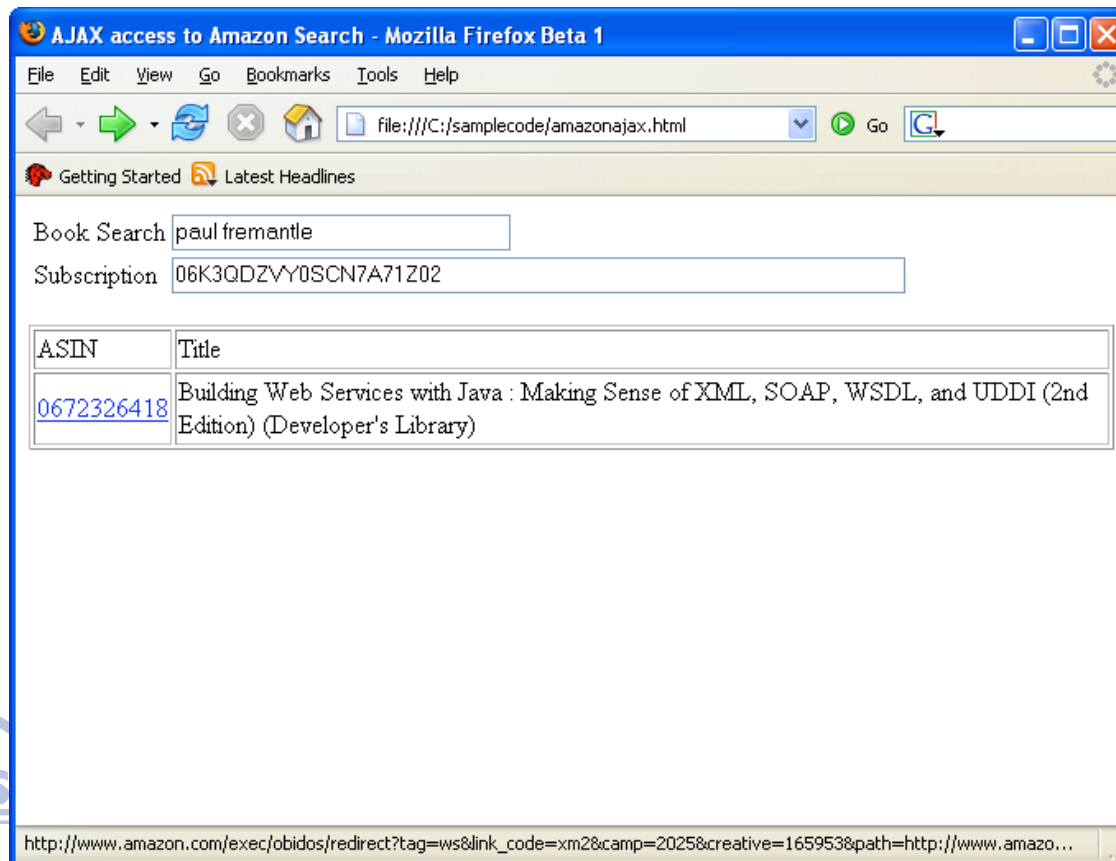
---

```
<ItemSearchResponse xmlns="...">
  <OperationRequest>...</OperationRequest>
  <Items>
    <Request>
      <IsValid>True</IsValid>
      <ItemSearchRequest>...</ItemSearchRequest>
    </Request>
    ...
    <Item>
      <ASIN>0672326418</ASIN>
      <ItemAttributes>
        ....
        <Title>Building Web Services with Java : Making Sense of XML, SOAP, WSDL,
and UDDI (2nd Edition) (Developer's Library)</Title>
      </ItemAttributes>
    </Item>
  </Items>
</ItemSearchResponse>
```

# Using AJAX with the Amazon Search

- Full code is on CD, and also available:

➤ <ftp://wso2.com/css2005/>





# Main Points

---

- Waits 2 seconds after you finish typing before sending the request
  - Gives enough time to pause, without undue wait
- XMLHttpRequest is a simple GET
- DOM is used to extract the data



# Example of the DOM Code

---

```
var aItems = resp.getElementsByTagName("Item");
  for (var i=0; i < aItems.length; i++)
    thisItem = aItems[i];
    ASIN = thisItem.
      getElementsByTagName("ASIN")[0].firstChild.nodeValue;
    linkurl = thisItem.

      getElementsByTagName("DetailPageURL")[0].firstChild.nodeValue;
    title =
      thisItem.getElementsByTagName("Title")[0].firstChild.nodeValue;
  }
```



# The Browser as a SOAP Client

---

- The same model can be used to access SOAP/HTTP services
  - Simply use DOM or string manipulation to create a SOAP request
  - Pull out tags from the response
    - getElementByTagName
  - Use the data to update the page

# REST vs SOAP

---

- There are MANY arguments over REST+XML versus SOAP
  - *My personal opinions*
  - There isn't much benefit in SOAP if the only intended client is a browser
    - because browsers are unlikely to support WS-Security, WS-ReliableMessaging
  - but... since things hardly ever turn out as planned
    - It's worth having a SOAP Envelope to allow for headers to be added later
  - The challenge is there for WS-supporters to make SOAP as easy as REST/XML – including on Linux (and come to my Axis2 talk ☺ to hear about Axis2 support for REST)

# JSON – JavaScript Object Notation

---

- Not specific to JavaScript – libraries exist for
  - C, C++, C#, Java, JavaScript, Perl, Python, *etc.*
- An alternative to XML that is simpler and easier to parse
  - Directly parsed by JavaScript
  - <http://www.json.org>



# Example

---

```
function Circle() {  
    this.x = 0; this.y = 0; this.radius = 0;  
}  
var circ = new Circle();  
circ.x = 10; circ.y = 20; circ.radius = 100;
```

## JSON:

```
{"x":10,"y":20,"radius":100}
```

Note reading JSON is inherent in JavaScript but creating it requires 3k of open source JavaScript  
Likewise, to create requires libraries



# A More Complex Example

---

```
{ "menu": {  
  "id": "file",  
  "value": "File:",  
  "popup": {  
    "menuitem": [  
      { "value": "New", "onclick":  
"CreateNewDoc()" },  
      { "value": "Open", "onclick": "OpenDoc()" },  
      { "value": "Close", "onclick": "CloseDoc()" }  
    ]  
  }  
}}
```



# And the Equivalent XML

---

```
<menu id="file" value="File" >  
  <popup>  
    <menuitem value="New" onclick="CreateNewDoc()" />  
    <menuitem value="Open" onclick="OpenDoc()" />  
    <menuitem value="Close" onclick="CloseDoc()" />  
  </popup>  
</menu>
```



# Using JSON in a Client

---

- Use normal xmlhttp to get the response

```
var json_data = xmlhttp.responseText;
```

And then use the JavaScript evaluate

```
eval("var the_object = (" + json_data + ")");
```



---

```
{"x":10,"y":20,"radius":100} ;  
do_something_nasty_here;
```

Safer to use the 3k OS code:

```
var myObject = JSON.parse(jsontext);
```



# My Take on JSON

---

- It may have some performance benefits
  - But no better than binary XML
- Much easier than DOM
- But no easier than E4X – wait and see
  
- I'm a firm believer in XML, but I can see situations where JSON has value



# SOA / AJAX Intersection

---

- Open model, allows interactions, extensions, composition, integration
  - <http://www.google.com/apis/maps/documentation/>
  - <http://www.thomasscott.net/tripods/>
- Allows thin, web-only clients direct to Web Services and SOA
  - An extension of the XML everywhere architecture



# E4X

---



# E4X – ECMAScript for XML

---

- ECMAScript is the official name for JavaScript
- E4X is a recent standard that adds built in XML support into the language – ECMA357
  - <http://www.ecma-international.org/publications/standards/Ecma-357.htm>
- Originally from BEA and AgileDelta
- Doesn't JS already have XML support with DOM?
  - Yes, but....
- E4X is simply, easier, more powerful
- Similar in concept to research prototypes from IBM and Microsoft
  - XJ, Comega

[ECMA – European Computer Manufacturer's Association  
The standards body where JavaScript was standardised  
Not to be confused with ECMA.ORG – the European Carton Manufacturer's Association]



# Examples in E4X

---

```
var x1 = new XML("<some>xml</some>");
```

```
var x2 = <people>
```

```
  <person gender="male">
```

```
    <name>Paul</name>
```

```
    <hair>Spiky</hair>
```

```
    <eyes>Grey</eyes>
```

```
    <height measure="cms">
```

```
      178
```

```
    </height>
```

```
  </person>
```

```
</people>;
```



# E4X Examples

---

```
js> print(x.person[0].name);
```

Paul

```
js> for each (var h in x..height) {  
    print(h+" " +h.@measure );  
}
```

178 cms

```
js> print(x.person.(name=="Paul").hair);
```

Spiky



# Trying It Out

---

- E4X is supported in two runtimes currently
- Mozilla's Rhino
  - (Java implementation of JavaScript)
- Mozilla's latest builds of SpiderMonkey
  - (C implementation of JavaScript available in Mozilla Browsers)
    - Mozilla 1.8+
    - Firefox 1.5 beta 1

# Rhino



- Download Rhino 1.6R1 / 1.6R2
  - <http://www.mozilla.org/rhino/>
- Download XMLBeans 1.0.4 or XMLBeans 2.0
  - 1.0.4
    - set classpath=<rhino>\js.jar;<xbeans>\xbean.jar
  - 2.0
    - set classpath=<rhino>\js.jar;<xbeans>\xbean.jar;<xbeans>\xmlpublic.jar
- `java org.mozilla.javascript.tools.shell.Main`



# E4X

---

- Variables can natively be XML
- The tree structure maps to fields
- Useful stuff:
  - `'..'` matches anywhere in the tree
    - *e.g.* `x..height` will match `x.person.height`
  - Deals with arrays transparently
    - *e.g.* if `x.length=1` then `x == x[0]`
  - Ordering
    - E4X supports FULL ORDERING of XML



# An Example

---

```
function metricToImperial(cms) {  
  var fudge=cms+3; // feeling a little short  
  var totalinches = Math.round(fudge/2.54)  
  var inch = totalinches%12;  
  var ft = (totalinches-inch)/12;  
  var response = <height/>  
  response.feet = ft;  
  response.inches = inch;  
  response.@measure = "imperial";  
  return response;  
}
```

```
js> metricToImperial(178)  
<height measure="imperial">  
  <feet>5</feet>  
  <inches>11</inches>  
</height>
```



# Namespaces

---

- E4X also has full support for XML namespaces
  - Inline
    - `var soapMsg = <s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/" />`
  - Set a default:
    - `default xml namespace = new Namespace("http://wso2.com/ns/2005/test");`
  - Using `::`
    - `var f = new Namespace("http://fremantle.org/testns");`
    - `soapMsg.Body.f::GetStockQuote="IBM";`

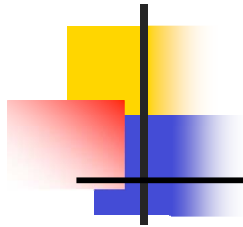


# Namespace Examples

---

```
js> var soapMsg = <s:Envelope
  xmlns:s="http://schemas.xmlsoap.org/soap/envelope/" />
js> var f = new Namespace("frem", "http://fremantle.org/testns");
js> soapMsg.Body.f::GetStockQuote="IBM";
js> print(soapMsg);
```

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <Body>
    <tes:GetStockQuote xmlns:tes="http://fremantle.org/testns">
      IBM
    </tes:GetStockQuote>
  </Body>
</s:Envelope>
```



# { } – “ex-lining”

---

```
var num = 36;
```

```
var p = <paul><age>{num}</age></paul>
```

```
print(p);
```

```
<paul>
```

```
  <age>36</age>
```

```
</paul>
```



# Using E4X in a Browser

---

- Firstly get a browser where it works!!!
- I recommend Firefox 1.5 Beta
  - <http://www.mozilla.org/projects/firefox/>
- Now you need to tell Firefox you are using E4X:
  - `<script type="text/javascript;e4x=1">`



# Issues

---

- There is one nagging issue using E4X together with XMLHttpRequest
  - *You can't convert directly from a DOM to an E4X XML*
- So you can't do:
  - `resp = new XML(xmlhttp.responseXML)`
- You *used* to be able to do
  - `resp = new XML(xmlhttp.responseText);`
  - but then an overzealous reading of the E4X spec put paid to that



# Fix

---

```
function getAsXML(s) {  
    var sx = s;  
    if ("<?xml" == s.substr(0,5)) {  
        var l = s.length;  
        var i = s.indexOf("?>");  
        sx = s.substring(i+2,l);  
    }  
    var xml = new XML(sx);  
    return xml;  
}
```

```
resp = getAsXML(xmlhttp.responseText); // works!!!
```



# E4X and SOAP

---

- Using E4X it's positively easy to create SOAP clients!
- You can either use E4X fields to create the request:

```
var s = new Namespace( "s",  
    "http://schemas.xmlsoap.org/soap/envelope/");  
var envelope = <s:Envelope xmlns:s={s}/>;  
envelope.s::Body="";  
var body = envelope.s::Body;  
var x = new Namespace("x","urn:xmltoday-delayed-quotes");  
body.x::getQuote = <x:getQuote xmlns:x={x}/>;  
var getQuote = body.x::getQuote;  
getQuote.symbol=symbol;
```



# Or Simply Inline the XML and "Ex-line" the Parameters

---

(I grabbed this from a SOAP sniffer)

```
var env =  
<s:Envelope xmlns:c="urn:xmethods-CurrencyExchange"  
  xmlns:s="http://schemas.xmlsoap.org/soap/envelope/" >  
  <s:Body>  
    <c:getRate>  
      <country1>US</country1>  
      <country2>{country}</country2>  
    </c:getRate>  
  </s:Body>  
</s:Envelope>
```

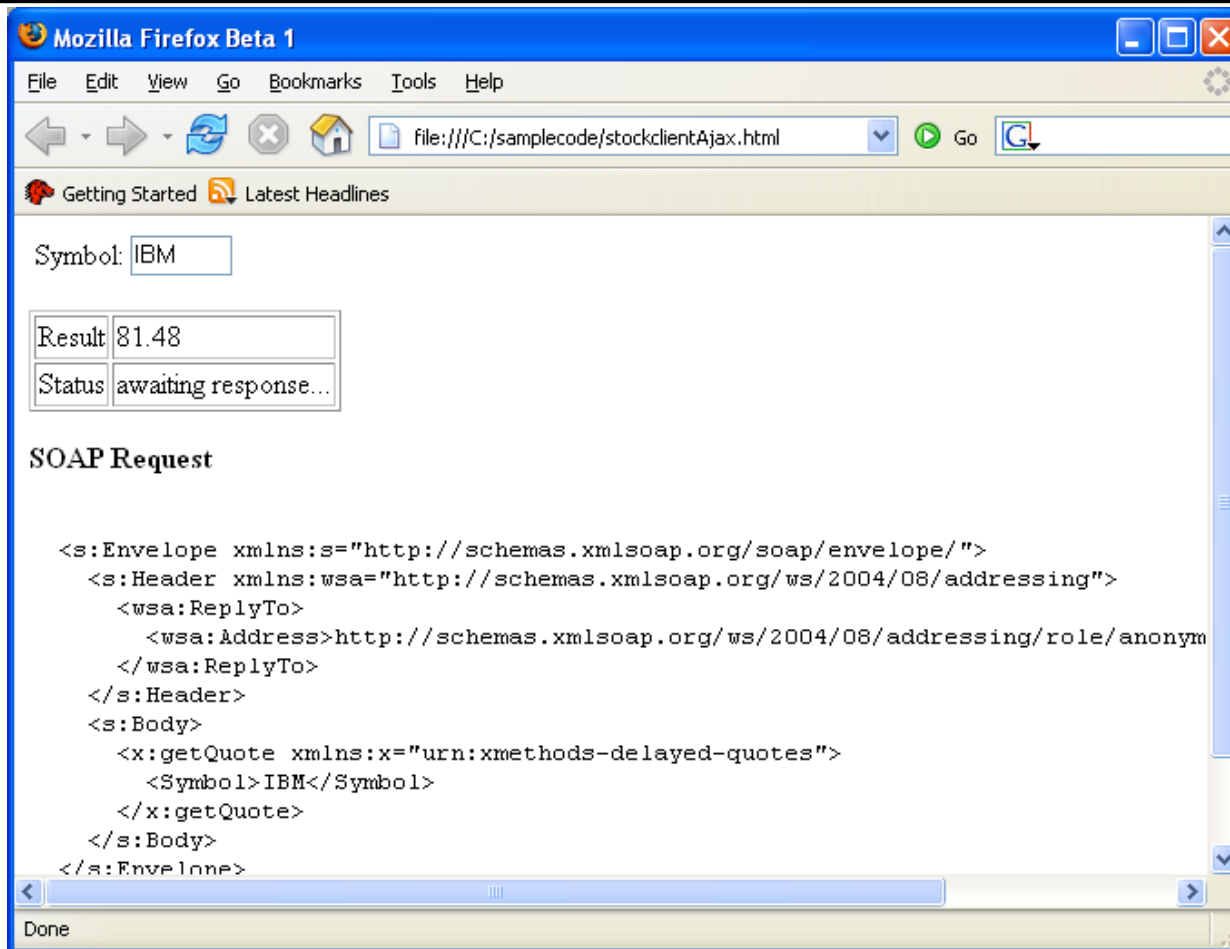


# execService

---

```
function execService(url, xml, callback) {
  var xmlhttp = new XMLHttpRequest();
  var async=false; if (arguments.length==3) async=true;
  xmlhttp.open("POST", url, async);
  xmlhttp.setRequestHeader("SOAPAction", "\"\"");
  xmlhttp.setRequestHeader("Content-Type", "text/xml")
  if (async) {
    var f = function() {
      if (xmlhttp.readyState==4) {
        callback(getAsXML(xmlhttp.responseText));
      }
    }
    xmlhttp.onreadystatechange = f;
  }
  xmlhttp.send(xml.toString());
  if (!async) return getAsXML(xmlhttp.responseText);
}
```

# StockClientAjax



Mozilla Firefox Beta 1

File Edit View Go Bookmarks Tools Help

file:///C:/samplecode/stockclientAjax.html

Getting Started Latest Headlines

Symbol:

Result	81.48
Status	awaiting response...

**SOAP Request**

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Header xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
    <wsa:ReplyTo>
      <wsa:Address>http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
    </wsa:ReplyTo>
  </s:Header>
  <s:Body>
    <x:getQuote xmlns:x="urn:xmethods-delayed-quotes">
      <Symbol>IBM</Symbol>
    </x:getQuote>
  </s:Body>
</s:Envelope>
```

Done

# Parsing the Result in One Line

```
resultBox.innerHTML = resp..*::Result[0];
```

- \_ The result as an E4X XML ●
- \_ Search the tree for tags ●
- \_ in any ●
- \_ namespace ●
- \_ with tag name Result ●
- \_ and take the first one ●

# Using XMLHttpRequest in Rhino

---

- Unfortunately there is no native support for XMLHttpRequest in Rhino
- Why would you want it?
  - So you can run E4X Web service clients outside a browser
  - So you can script Web services in a server environment
- Anthony Elder (IBM) wrote one
  - Available in the sample code

# E4X + XMLHttpRequest + Server

---

- Now we can script web services simply and easily
- How about we start to write the services in E4X?
- E4X provider for Axis allows very simple E4X scripts to be exposed as Web services



# WSDL

---

- It is very simple to extract the URL of a service from a WSDL in E4X:

```
function getAndParseWSDL(wsdlURL) {  
    var xh = new XMLHttpRequest();  
    xh.open("GET", wsdlURL ,false);  
    xh.send(null);  
    var resp = getAsXML(xh.responseText)  
    return resp..*::address.@location[0];  
}
```





# Axis WSDD

---

- The Web Service Deployment Descriptor
- Contains the meta-data around a service (usually)
- In our case it also contains the service – since it's just a script we can embed it



# A "normal" WSDD

---

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/" ...>
  <service name="Bid" provider="java:RPC">
    <namespace>http://www.soapinterop.org/Bid</namespace>
    <parameter name="className" value="samples.bidbuy.BidService"/>
    <parameter name="allowedMethods"
      value="RequestForQuote SimpleBuy Buy Ping"/>
  </service>
  <beanMapping qname="bid:PurchaseOrder"
    languageSpecificType="java:samples.bidbuy.PurchaseOrder"/>
  <beanMapping qname="bid:Address"
    languageSpecificType="java:samples.bidbuy.Address"/>
  <beanMapping qname="bid:LineItem"
    languageSpecificType="java:samples.bidbuy.LineItem"/>
</deployment>
```



# Axis E4X WSDD Example

---

```

<deployment name="test" xmlns="http://xml.apache.org/axis/wsdd/"
  xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">

  <service name="E4XStock" provider="java:E4X">
    <parameter name="type" value="body"/>
    <documentation>
      <![CDATA[
        function service(x) {
          var value = (x..*::Symbol=="IBM"?983:52.5;
          var n = new Namespace("urn:quoteservice");
          default xml namespace = n;
          response = <getQuoteReturn>
            <Result>{value}</Result>
          </getQuoteReturn>;
          return response;
        }
      ]]>
    </documentation>
  </service>
</deployment>

```

(Yes I have some IBM shares still, so I'm maybe a little overoptimistic)

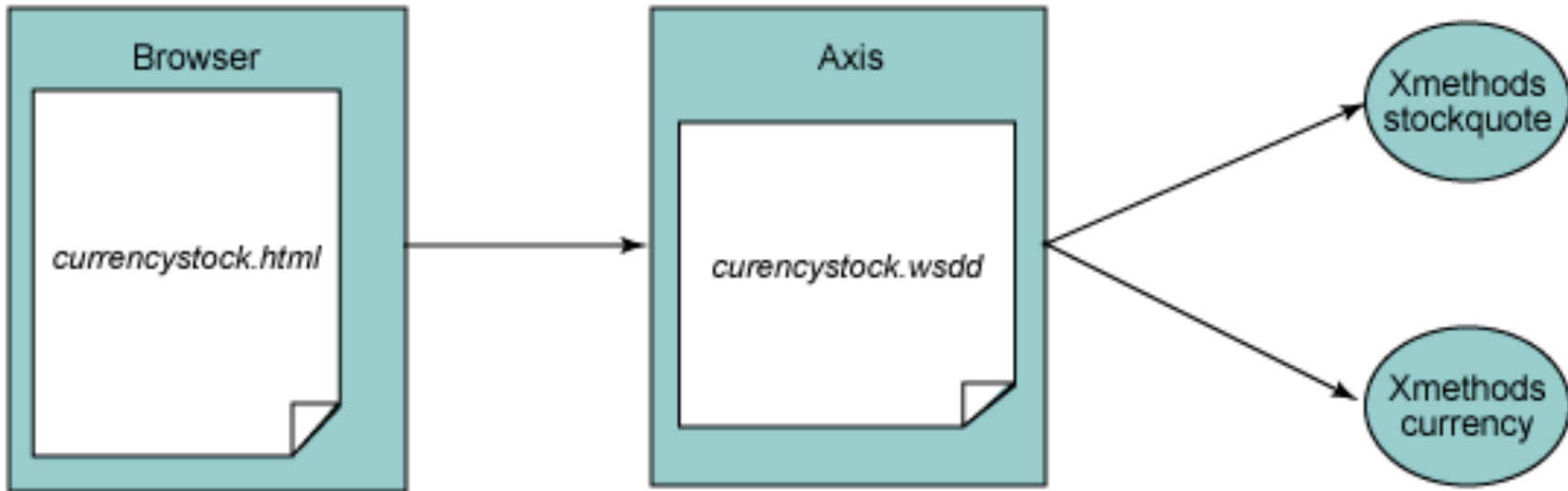


# Using E4XProvider

---

- Axis has a clever “auto-deploy” for providers
- Simply add E4XProvider.jar, js.jar, xbean.jar and maybe xmlpublic.jar to your Axis installation classpath
- Deploy the WSDD
- You have a new service!

# Choreography





# Overall Structure

---

```
function service(x) {  
    0. extract stock symbol from request body  
    1. extract currency symbol from request body  
    2. get WSDL for stock service and extract location  
    3. get WSDL for currency service and extract location  
    4. call stock service  
    5. call currency service  
    6. multiply  
    7. create response body  
    8. return  
}
```



# Code

---

```
function service(x) {
    var symbol = x.*::Symbol[0].toString();
    var country = x.*::Country[0].toString();

    java.lang.System.out.println(country.toString());
    var currURL = getAndParseWSDL
    ("http://www.xmethods.net/sd/2001/CurrencyExchangeService.wsdl");
    var stockURL = getAndParseWSDL
    ("http://services.xmethods.net/soap/urn:xmethods-delayed-quotes.wsdl");

    var rate = getCurrencyValue(country, currURL);
    var ticker = getStockQuote(symbol, stockURL);

    var response =
        <n:getQuoteResponse xmlns:n="urn:xmethods-delayed-quotes">
            <Result>{rate*ticker}</Result>
        </n:getQuoteResponse>
    return response;
}
```



# Summary

---

- We have:
  - built simple AJAX clients calling REST services
  - used DOM to parse the response
  - learnt about E4X
  - built E4X AJAX clients accessing SOAP services
  - built services using E4X
  - created simple choreography services using E4X



# Resources

---

- Read the E4X specification ECMA357
  - <http://www.ecma-international.org/publications/standards/Ecma-357.htm>
- Read the JavaScript specification ECMA262
  - <http://www.ecma-international.org/publications/standards/Ecma-262.htm>
- Rhino: <http://www.mozilla.org/rhino/>
- XMLBeans <http://xmlbeans.apache.org/>
- Two great articles on E4X
  - <http://www.ibm.com/developerworks/webservices/library/ws-ajax1>
  - <http://www.ibm.com/developerworks/webservices/library/ws-ajax2/>
- A third great article on E4X and IBM's SIBus
  - <http://www.ibm.com/developerworks/webservices/library/ws-mediation/>
- Apache Axis: <http://ws.apache.org/axis>
- Apache Axis2: <http://ws.apache.org/axis2>
- <http://www.ajaxpatterns.org/>