Testing Web Services (.NET and Otherwise)

Software Test Automation Conference -- Spring 2003
Presented by Tom Arnold, Xtend Development, Inc.

Introduction

The past 2 years have shown many companies embracing web services and striving to have their systems interoperate both with different operating systems within their own walls and outside of their walls making it easier to work with strategic partners. How do we as engineers test those services before they are ever rolled out into production?

This paper will help you understand what a web service is, give you an example of how web services are being put to use, and determine how to approach testing a web service manually through a web page and with automation using different programming languages.

In the Beginning

A *Web Service* is not a new idea, but because it's the latest buzzword -- especially as IBM, HP and Microsoft advertise their new server products that form what have become known as *Web Technologies* -- it seems to many people like it is the latest and greatest thing.

A Brief History

Web services have actually been around since Tim Berners-Lee[1] started his research on serving up hypertext files in 1990, and certainly existed before Marc Andreessen[2] first released the Mosaic browser in 1993 that helped introduce the world to the Internet and what was termed the World Wide Web.

By 1994, the Web was gaining serious momentum, and mainstream software companies jumped in to create their own web servers and services. Microsoft Corporation, for example -- although not as quick to jump into the fray as others -- released the *Microsoft Internet Information Server* (MS IIS) in 1995 as its answer to web server software for the Microsoft Windows operating system. Since the creation of the Web, software companies have continued to add additional layers to their web server products to handle the distribution of server load, session tracking, security (authentication & authorization), database integration, data exchange, and more, with the goal of making it easier to develop Web applications.

XML

An example of new technologies -- created specifically for Web Services as a solution to sharing data -- is XML, born in February 1998. XML did not become a mainstream solution, however, until 2000-2001 as companies incorporated it into their Web development tools. XML is an important component in today's version of Web Services and provides a standardized way to represent complex data structures in a text format that is both readable to humans and, because it is standardized, readable by any program adhering to the XML specification. Transferring data between different operating systems, databases, and other programs that previously relied upon their own proprietary binary formats for data storage, suddenly became easier as everyone embraced (and continues to embrace) XML. XML has emerged as the "standard platform for convergence of information"[3] and isn't limited to any one operating system, programming language, or object-model specific protocols. The same cannot be said for COM

[1] Tim Berners-Lee is credited by the World Wide Web Consortium (W3C) (www.w3c.org) as the "inventor of the Web."
[2] Marc Andreessen is known for being a founder of Mosaic Communications Corporation (later Netscape) and creating a web browser in 1993.
[3] "Happy Birthday, XML!" (5 years old) page on the W3C.org site: http://www.w3.org/2003/02/xml-at-5.html

(Component Object Model), CORBA (Common Object Request Broker Architecture), IIOP (Internet Inter-ORB Protocol), RMI (Remote Method Invocation), or other web service-like solutions.

Back to the Basics

Bringing it back to the basics: Any computer program that monitors and responds to port 80 [4] network traffic is a web server and any programs working with a web server that responds to a specific type of request is a web service. It sounds simple -- and it is simple -- but additional layers have been added to help break programmatic responsibilities into pieces that form an overall picture of Web architecture. Understanding those pieces and how they all fit together is where there can be challenge.

Let's begin with an example of how some companies have evolved into using web services.

Example of Using a Web Service

This example is of a fictitious e-commerce company -- *XYZ Company* -- that sells videos, compact discs and books on the Internet. This company handles the shopping cart that keeps track of a customer's selections as well as the final checkout where billing information is gathered and shipping charges added to the final total. Because this is a small business (2 people), the approach taken by XYZ Company is very simple: charge a single rate for shipping & handling and ship only within its home country. This is a limiting approach to a potentially larger market, to be sure, but as a small start-up the company does not feel the need to take on too much while it works out the kinks in its business strategy. An added benefit is that XYZ Company can then shop around with the different shipping companies to find the best rates and pocket any savings.

As time goes by, business booms and XYZ Company grows beyond its two-person staff. The company needs room for stocked items, a shipping department, a marketing staff that includes a Webmaster, as well as other employees crucial to handling the day-to-day operations of the company. But as the company grows and attracts new customers, competition heats up on the Internet. Having a single rate for shipping products is no longer something customers will tolerate. Customers want options, such as overnight service, 2nd day delivery, and 5-day ground service.

> **Problem**: A single shipping rate no longer provides the flexibility demanded by customers.

> **Solution**: Webmaster writes a utility that goes to the preferred shipping company's website once a day, parses the "rates.htm" page looking for the keywords *overnight*, *2nd day*, and *ground*, and uses those current rates in its shipping calculations. (This type of utility is commonly known as a *screen scraper*.)

Using a screen scraper utility allowed XYZ Company to provide reasonable and flexible shipping rates to its customers. Unfortunately, this utility wasn't a long-term solution. The biggest problem was that XYZ Company relied on their shipping company -- WWW Shipping[5] -- not changing the format of their pages that had the rates, keywords associated with each level of delivery service, or worse, the location (URL) of the page holding the required information. A screen scraper utility will break frequently requiring constant maintenance. A better solution had to be found.

> **Problem**: Using a screen scraper relies too heavily on a third party company not changing the format of its web page, keywords, and keeping key pages in the same location on the web server.

[4] A *port* is a logical channel used on a network to distinguish between the different types of requests/responses sent through a TCP/IP-based network. Port 80 is used by HTTP (Hyper-text Transport Protocol). When adding a firewall to a network, port 80 is one of the few ports commonly left open. Another port typically left open is 25 (for incoming email traffic).

[5] WWW Shipping.com is a fictitious company created as part of this example of how a third-party company may provide web services to other companies considering the use of a freight company's services.

Solution: Use the new web service provided by WWW Shipping to determine shipping rates in real-time with the guarantee that changes to how the information is requested and data is returned will not change.

WWW Shipping rolls out a new service that e-Commerce companies can use when calculating shipping costs. The format is straightforward and customers of WWW Shipping are assured the parameters will not change. Companies like XYZ Company that would like to use the service need only provide the following information in programmatic requests to the shipping company's web site:

- Pick-up location (zip code)
- Destination (zip code)
- Package weight (pounds)
- Level of service (overnight, 2nd day, 5-day)
- Company ID (optional, but lower rates are available to frequent shippers)

Requests to the shipping company for rate information are in the form of an HTTP-GET that looks similar to the following:

```
http://www.WWW-shipping.com/rate-quote.cgi?pickup=98136&dest=32073&wt=5.5&service=1
```

In our example, the value returned to the requestor -- known as the *consumer* in Web Services terminology -- is a single number: the cost (in U.S. dollars) to ship the package. This type of query is done in real-time from the XYZ Company website by having the user click on a link to submit the shipping details to the WWW Shipping site to request a quote from its web service -- also known as the *provider*. The customer confirms the rate they've selected and it is entered into XYZ Company's final checkout page.

This is a very simple form of a web service that returns a single value from a third party web service. When the data being returned is more complex, such as a multi-field record or a list of records, the structure of the information and its relationship to the data around it must be kept intact. Keeping that data and its relationships intact is the solution XML provides.

Let's take the example one final step: XYZ Company has grown its presence on the Internet and now ships enough packages with WWW Shipping to qualify for their Preferred Shipper Rate. XYZ Company must provide its unique identity number in its requests for rate quotes so that WWW Shipping provides the correct preferred price. Instead of using an HTTP-GET approach that shows the data being requested in the browser's Address control, XYZ Company wants to hide these details from the user. In addition, they want to make the request for shipping information to be a little more transparent so that their site looks even more polished.

Problem: XYZ Company wants to take advantage of WWW Shipping's the lower rates provided to WWW Shipping's clients that ship a high volume of products. To do this, XYZ Company must provide its unique ID as part of the request for the shipping rate quote. Furthermore, XYZ Company would like to hide their ID from its customers as well as hide the web request -- previously displayed in a separate browser window -- that made the XYZ Company website somewhat clunky.

Solution: XYZ Company moves from an HTTP-GET request to an HTTP-POST request resulting in data being embedded in the header of the HTTP data request. This means the request data is no longer seen on the Address line of the browser.

By changing the method used in submitting the web form from *GET* to *POST*, the data is now sent without showing it on the *Address* / *URL* line of the browser. This helps to hide the unique ID used by

XYZ Company and also hides the fact that XYZ Company queries another site for shipping information.[6] Furthermore, XYZ Company uses the WWW Shipping's new XML schema when sending the data with the POST method. Part of this approach includes the use of SOAP (Simple Object Access Protocol) that allows the XML data to be embedded into the HTTP request. The result is a block of information sent to WWW Shipping that has the following format and still adheres to the HTTP standard[7]:

```
POST /rate-quote.cgi
Content-Type: text/xml; charset=utf-8
Content-Length: 312
SOAPAction: "http://www.WWWshipping.com"
<?xml version="1.0" encoding="utf-8" ?>
<soap:envelope>
    <soap:body>
        <shipment>
            <company name="XYZ Company" id="12345">
            <parcel id="1" service="overnight">
                <pickup zip="98136"/>
                <delivery zip="32073"/>
                <weight pounds="5" ounces="8"/>
            </parcel>
        </shipment>
    </soap:body>
</soap:envelope>
```

Where this becomes most interesting is to the XYZ Company's Webmaster. IBM, HP, Microsoft and others have developed tools that make all of this data exchange transparent to the web developer. All the developer needs to know is that the data is submitted and results received back. If the results come back in XML, the data it is placed into a data object and queried as easily as any other data object available to the programmer.

Testing a Web Service

Now that you know what a web service is and can look like, along with an example of how a company's website could evolve to take advantage of such a service, let's flip over to the WWW Shipping Company's side of the equation and how their testing staff might go about testing the web service.

A Black Box Affair

Testing a web service is very much like testing a compiled programming library. In cases where the test engineer is unable to see the inner workings or source code, the only information the engineer has to go on is available entry points or interfaces into that black box. Web services have interfaces that are considered *exposed* when they are remotely accessible. Each exposed object has a function or *method* that acts as an interface allowing the web service to share its capabilities. Send a request to the interface with the information it requires and you will receive a response.

There are two common methods used to test a web service: through a web page or through a programming language. We will look at both. But before we do, let's look at the required values and their ranges when invoking the WWW Shipping web service method for obtaining a rate quote.

---

[6] The approach of using the POST method hides the submitted data from an everyday user. However, it is not secure communication. There are many programs on the market that make it easy to grab an HTTP-request allowing the data to be revealed. Data that needs to be secured, such as credit card or social security numbers, should be sent through a secured connection. XYZ Company would be well advised to first link to a secured server using HTTPS, for example, before sending such data.

[7] By continuing to follow the HTTP standard, it is not necessary for a network administrator to open additional ports other than port 80 for sending and receiving data. This is one of the strengths of XML and SOAP: because it is textual data, it cannot contain a binary executable file that could act as a virus.

Name	Required	Legal values	Comments
pickup	yes	5-digit integer	U.S. zip code for the shipper
dest	yes	5-digit integer	U.S. zip code for recipient
wt	yes	0< float <= 100 pounds	Decimal value for weight of package
service	yes	1, 2 or 5	1-day, 2-days or 5-days delivery
cid	no	5-digit integer	Unique customer ID for preferred shipping rates

Test data to be used when exercising this web service include:

1) Checking boundary conditions for the different values
2) Submitting values known to be illegal or out of bounds
3) Omitting some of the required values and verifying error codes
4) Testing legal values and comparing them to data proven to be correct
5) Verifying the order in which the values are passed has no adverse effect

These are just a few of the obvious tests that come to mind and a good starting point. Let's try some of these test scenarios out through the two methods of testing.

Testing with a Web Page

Testing via a web page is by far the easiest approach and is an option with many web services. However, for complex web services that require more complex data to be sent, and many times relying on XML, this is not the best option.

The GET Method

For the approach of testing with the GET method the user need only create a page full of links. A form may also be created allowing the test engineer to type the data in interactively instead of having a set of links on a page. Below is an example[8] of links that could be created to test the WWW Shipping web service using the GET method:

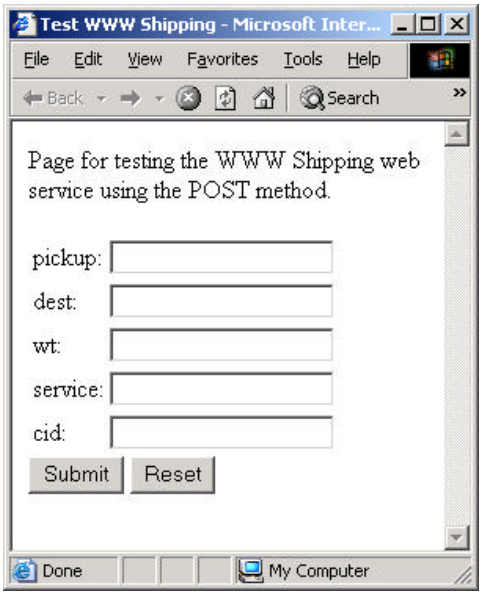
```
<html>
<head>
<title>Test WWW Shipping</title>
</head>
<body>
<p>Page for testing the WWW Shipping web service using the GET request
method.</p>
<p>
Test 1: pickup=98136, dest=98116, wt=0, service=1<br>
Test 2: pickup=98136, dest=98116, wt=8.5, service=1<br>
Test 3: pickup=98136, dest=98116, wt=100, service=1<br>
Test 4: pickup=98136, dest=98116, wt=100.1, service=1<br>
etc.
<p>
<a href="http://www.WWWshipping.com/rate-
quote.cgi?pickup=98136&dest=98116&wt=0&service=1"> Execute Test 1</a><br>
<a href="http://www.WWWshipping.com/rate-quote.cgi?pickup="
98136&dest="98116&wt=8.5&service=1&quot;">Execute Test 2</a><br>
<a href="http://www.WWWshipping.com/rate-
quote.cgi?pickup=98136&dest=98116&wt=100&service=1">Execute Test 3</a><br>
<a href="http://www.WWWshipping.com/rate-quote.cgi?pickup="
98136&dest="98116&wt=100.1&service=1&quot;">Execute Test 4</a><br>
etc.
</body>
</html>
```

[8] Note: the text editor word-wrapped the HTML line that links to rate-quote.cgi. There is not a carriage return in the original HTML. A carriage return would be interpreted as a space and result in a broken link.

The POST Method

Testing from a web page using the POST method is a little more involved, but still pretty straight forward. The requirement here is that a web form must be used to submit the values to the web service. Also, the "method" parameter is set to the value of "POST" so that the data is sent as part of the HTTP header and not as a query string in the browser's Address line. Also, keeping the flexibility of a form but generating an HTTP-GET is as simple as changing the method attribute from POST to GET in the <form> tag.

```
<html>
<head>
<title>Test WWW Shipping</title>
</head>
<body>
<p>Page for testing the WWW Shipping web service using the POST method.</p>
<form method="POST" action="http://www.WWWshipping.com/rate-quote.cgi">
<table>
  <tr><td>pickup:</td><td><input type="text" name="pickup"></td></tr>
  <tr><td>dest:</td><td><input type="text" name="dest"></td></tr>
  <tr><td>wt:</td><td><input type="text" name="wt"></td></tr>
  <tr><td>service:</td><td><input type="text" name="service"></td></tr>
  <tr><td>cid:</td><td><input type="text" name="cid"></td></tr>
</table>
<input type="submit" value="Submit"> <input type="reset" value="Reset">
</form>
<p>
</body>
</html>
```



For those using Microsoft Visual Studio .NET to create web services, a page similar to the one above (and pictured to the right) is built for the developer allowing her to test the source code. If your group is using .NET for development, you can get a page like this from the programmers instead of building your own. For those testing web services not using .NET, that instead rely on Perl, C, PHP, or other programming languages, you may need to create your own web forms for testing purposes. As you can see from the HTML above, this isn't necessarily a difficult task.

Testing with a Programming Language

While it is important to have an easy way to manually test a web service so that bugs can be verified as fixed and new testing ideas can be verified at the spur of the moment, web services are the perfect candidate for programmatic or automated testing.

The approach that I would take in this case would be to have a pool of data that could be read in and used by a simple program to send requests to the web service. This is commonly known as *data-driven automated testing* and provides great flexibility to a testing team. For example, once the script has been written, anyone following the format for entering additional data into the data file can easily add more tests without having to touch any source code. It is also possible to generate the data programmatically if the goal is to test every conceivable combination of values.

Let us look at two examples of how one might go about automating the testing of a web service. The first example is using the tool that accompanies Microsoft Visual Studio .NET. The second one is an example of how to post a request to a web site using another common programming language, Perl[9].

[9] Perl -- created in 1987 by Larry Wall -- runs on Linux, Unix, Macintosh and Windows. Perl is freely available, well supported by www.perl.org, and one of the most popular web programming languages.

VB Script (using MS ACT)

Microsoft includes Application Center Test[10] (ACT) with the enterprise version of Visual Studio .NET. This is an extremely handy tool when generating load testing on a web site -- its main focus as a test automation tool. However, because it has been designed to work with programs that process forms (typically ASP.NET applications), the Test object model that defines ACT comes in handy for working with web services as well. In addition, the object model does not only work with .NET executable files, it can be used with any compiled program that processes HTTP requests. Whether the program on the web server is written in Python, Perl, C, PHP, C#, or anything else, ACT can send it a request and receive the results back.

The scenario recorder that is a part of ACT not only allows you to generate load tests on a web server, it helps you learn more about the Test object model and quickly figure out how to write your own VB Script that assembles an HTTP header and submit it as a request to the service on the web server you are testing.

We will continue with the WWW Shipping web service example. The following source code reads in a data file that holds the values required by WWW Shipping when providing a quote on its shipping rates. This script will read in a single line, split it out into an array of values, create a query string, and submit it to the web service:

```
Dim sDatafile    'Name of the data file
Dim fso          'Object for working with file system
Dim oFile        'Object directly referencing data file
Dim hTextStream  'Handle to an instance of the open file
Dim sQueryLine   'Buffer holding single text line from file
Dim sQueryArray  'Array holding split up values from text line
Dim oConnection  'Object referencing connection to web site
Dim oRequest     'Object used when GETting or POSTing HTTP request
Dim oResponse    'Object holding response from web server

sDatafile = "c:\data.txt"
'data.txt contains the test data
'format: pickup,dest,weight,service,company,result
'example: 98136,32073,5.5,1,12345,23.09

set fso = CreateObject("Scripting.FileSystemObject")

if fso.FileExists(sDatafile) Then
    set oFile = fso.GetFile(sDatafile)

    'Open the file for reading (1=read, 0=opens as ASCII)
    set hTextStream = oFile.OpenAsTextStream(1, 0)

    'Open connection to the web server
    set oConnection = Test.CreateConnection("www.WWWshipping.com")

    if (oConnection Is Nothing) Then
        Test.Trace "Error: Unable to create connection." & VBCRLF
    else
        set oRequest = Test.CreateRequest

        'Loop through the text file
        do while hTextStream.AtEndOfStream <> TRUE

            'Read in a line from the text file
```

[10] Additional information about learning and using Microsoft Application Center Test can be found at: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/act/htm/actml_main.asp

```
            sQueryLine = hTextStream.ReadLine

            'Break the line into 6 array values
            'The 1 means use string comparison to find the ','
            sQueryArray = split(sQueryLine,",", 6, 1)

            'If the data file is correct, the values are arranged as follows:
            'sQueryArray(0)=pickup, (1)=destination, (2)=weight (in pounds),
            '(3)=service type, (4)=company ID, (5)=expected result

            'Build the query string
            oRequest.Path = _
            "/rate-quote.cgi?&pickup=" & sQueryArray(0) & _
            "&dest=" & sQueryArray(1) & "&wt=" & sQueryArray(2) & _
            "&service=" & sQueryArray(3) & "&cid=" & sQueryArray(4)

            'Method of sending request
            oRequest.Verb = "POST"

            'Log what it is that is being submitted
            Test.Trace("Submitting: " & oRequest.Path & VBCRLF)

            'Send the request and get a response
            set oResponse = oConnection.Send(oRequest)

            if (oResponse.ResultCode = 200) then
                'Write out PASS/FAIL results (include details if FAIL)
                if (oResponse.Body = sQueryArray(5)) then
                    Test.Trace("PASS" & VBCRLF)
                else
                    Test.Trace("FAIL: " & oResponse.Body & VBCRLF)
                end if
            else
                Test.Trace("Request Error: " & oResponse.ResultCode & VBCRLF)
            end if
        loop
    end if
    hTextStream.Close    'Close the datafile
    oConnection.Close    'Close the web connection
else
    Test.Trace "Error: Cannot find data file '" & sDatafile &"'." & VBCRLF
end if
```

Walk-Through

Let's walk through this VB Script using the following data: `98136,32073,5.5,1,12345,23.09`

1. The data file is opened by creating a *Scripting.FileSystemObject*[11] object, assigning it to the *fso* variable, and using the *FileExists()* and *GetFile()* methods.
2. Next, the file is opened for *reading* in ASCII text format. Data is not yet read in, but it is open and ready to go.
3. The ACT Test object's *CreateConnection()* method is called to assign details of a connection to the www.WWWshipping.com website to the *oConnection* variable. *oConnection* can then be checked to verify it is able to see the web server.
4. If a connection can be made, it is time to build the request that will be sent to the web server. To create a request object, a call is made to the *Test.CreateRequest()* method and the new request object assigned to the variable *oRequest*.

[11] Information about the FileSystemObject object can be found on Microsoft's MSDN site at: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/script56/html/jsfsotutor.asp

5. With the heavy lifting done, it is time to loop through the data file verifying with each loop that the end of the file has not yet been reached.
    a. A line is read into the variable *sQueryLine*.
    b. The split function relies on a comma (",") separating each value and separates the line into 6 separate values that are placed into the variable *sQueryArray*. Those values can then be accessed individually.
    c. The *Path* property of the *oRequest* object is set to have the path to the web service program along with the values to be passed.
    d. The method of how those values are passed ("POST" vs. "GET") is set by assigning the *oRequest.Verb* property equal to "POST".
    e. *Test.Trace()* method allows us to write results out to a data file whose location is specified in the ACT user interface.
    f. The *oConnection.Send()* method dispatches the initialized *oRequest* object to the domain pointed to by *oConnection*.
    g. Calling the *Send()* method causes a *Response* object to be created which is assigned to the *oResponse* variable.
    h. The *ResultCode* property of the *oResponse* object can be checked to see if the request succeeded or failed (200 means "ok").
    i. The *Body* property in the *oResponse* object contains what was returned by the web service (minus the HTTP header information). If it equals the expected result (included which was included in the data file), the test passes. Otherwise, it fails and the value of the *oResponse.Body* property is written to the log file.
6. Loop continues until the end of the data file is reached.
7. The data file and web connection are both closed and the program comes to an end.

Perl

Perl -- formerly PERL, which stood for "Practical Extraction and Report Language" -- was started in 1987 by Larry Wall and developed as an open source project. Its syntax was derived from a number of languages, including C, sed, awk, Lisp, and others. Perl's strength is in its string manipulation capabilities through the use of regular expressions and is particularly popular for writing CGI (Common Gateway Interface) programs (similar to WWW Shipping's fictitious *rate-quote.cgi* web service). Perl is so pervasive that it has earned a reputation as being "the duct-tape of the Internet"[12] and is available for Macintosh, Linux, Unix and Microsoft Windows operating systems.

For those of you that have chosen[13] Perl as your language, the World Wide Web Library for Perl (LWP[14]) provides you with what you need to simulate user requests for web pages as well as HTTP-GET and POST calls to Web Services.

Here is a simple script using the POST method to send the necessary data to the rate-quote web service:

```
# Create a user agent object
use LWP::UserAgent;
$ua = LWP::UserAgent->new;
$ua->agent("MyTestApp/0.1 ");

# Create a request
my $req = HTTP::Request->new(POST => 'http://www.WWWshipping.com/rate-
quote.cgi');
$req->content_type('text/x-www-form-urlencoded');
$req->content('pickup=98136&dest=32073&wt=5.5&service=1&cid=12345');
```

[12] Taken from the Perl Mongers website: www.perl.org/press/fast_facts.html
[13] In 1999, my team was hired to simulate high server loads on the eToys.com web site (now owned by Toys R Us) to help the company prepare for the November/December holiday shopping rush. We chose Perl to generate 10-million user sessions and orders in a 24-hour period to confirm the site's readiness. It was one of the few e-commerce sites that did not turn away customers because of high server loads.
[14] Documentation on LWP can be found online at www.perldoc.com. Code snippets found on that site have been used in creating the sample Perl script.

```
# Pass request to the user agent and get a response back
my $res = $ua->request($req);

# Check the outcome of the response
if ($res->is_success) {
    print $res->content;
} else {
    print "No content returned.\n";
}
```

In addition to sending simple URL request strings, one may also send XML text as the content to a web service. What is returned depends solely on that web service. That is, in the case of WWW Shipping, a single value is returned without HTML or XML tags surrounding it.

Summary

The concept of a web service isn't something that has just arrived on the scene; it has been around since HTTP came into being. What is new is how data is becoming easier to share between different operating systems and applications using a common approach to data representation, as well as the new layers being added by companies like Microsoft, HP and IBM, to make data transfer more transparent to the web developer.

In cases where the source code isn't available to the test engineer, working with a web service is very much a black box approach to testing, even though using a programming language to test a web service's exposed interfaces into its functionality seems to indicate otherwise. It all boils down to sending and receiving HTTP traffic, and the data that is sent depends on what format the web service expects. The required format could be as simple as a query string sent from a browser's Address line, or, an XML document wrapped in a SOAP envelope, all embedded within an HTTP header. The return values also depend on how the web service was written, anything could be returned: XML, HTML, or a single byte of information indicating the response.

Building a web page to provide a user interface for testing a web service is a good way to verify the service is behaving as expected. However, getting into the guts of testing a web service through the use of a programming language like Perl, or an automation tool like Application Center Test, that provides specific support for sending and receiving HTTP data, these programming languages allow a test engineer to take more of a data-driven approach to testing.

The programming language isn't the main thing to focus on. Perl, Java, Python, C, and even VB Script can all be used for testing. What the test engineer does need to focus on is what the web service expects as input and what it defines as its output. To anyone experienced with testing functions and subroutines for compiled libraries, such as Windows DLLs, this is already familiar ground. For those that find this to be new territory, know that this isn't "yet another" set of skills you need to acquire. That is, getting the skills for testing exposed routines will carry over to the area of testing web services. Consider web services a new face on an old adversary that no longer exists solely on one's individual computer; this testing challenge uses the same principals from years ago, but now the exposed functions can be distributed globally. Methods and approaches to testing this class of application remain the same.

In this document we focused on a single web service where we knew its required inputs and expected outputs. For those interested in knowing more about how to track down other web services, what those services can do, and how to put them to use, please refer to my STAR West 2002 presentation slides found at http://www.xtenddev.com/starwest2002/.

#####

This presentation and paper can be found on the **StickyMinds.com** and **AutomationJunkies.com** web sites. I can be contacted at *tom@automationjunkies.com* if you'd like to share any comments on this paper.