Software Tool & Die Inc.

Presents

Web Services Primer

Getting Started
A Web Services Tour!

By Chris Pollach – President: Software Tool & Die Inc.

- Email: cpollach@travel-net.com
- Blog: http://chrispollach.blogspot.ca
- PBDJ: http://chrispollach.sys-con.com
- LinkedIn: http://ca.linkedin.com/in/chrispollach
- SourceForge: http://sourceforge.net/projects/stdfndclass
- STD: http://www.softdie.com
Web Services Goals

- Facilitate communication between systems
  - Different platforms
  - Different programming languages
  - Through firewalls easily
  - Self descriptive API
  - Self descriptive data
What are Web Services?

• A collection of operations that can be described, published, located, and accessed over a network using standardized XML messaging
• Proposed to World Wide Web Consortium (W3C) in Mar 2001 – [http://www.w3c.org](http://www.w3c.org)
• Web Services utilize XML making them both platform and language independent
• XML gives us a mechanism for making cross-platform and/or cross-language communications
The primary components that make up Web Services are:

- **WSDL** – Web Services Description Language
  - Used to describe Web services
- **SOAP** – Simple Object Access Protocol
  - Used for sending and receiving messages from Web services
Describing Web Services

• Why does a Web service need to be described?
  – Web services could be used by anyone, anywhere, using any language on any platform
  – A description allows a developer to know how to interact with a Web service
    • PowerBuilder provides tools to read and integrate WSDL

• Web services are described using Web Services Description Language (WSDL)

• WSDL is written in XML

• Usually a developer of a Web Service does not have to manually write WSDL
  – PowerBuilder 11 (or higher) creates the ASMX, DISCO and WSDL
CREATING .NET Web Services in PowerBuilder Classic
PowerBuilder/.Net Web Services

- PowerBuilder gives you the choice of outputting PowerScript code as an
  - Assembly
  - Web Service
- PowerBuilder Web Services are deployed to your Microsoft IIS Web Server or EAServer (EOL 2016)
.Net Web Service Target
The wizard is virtually the same as for .NET assemblies, etc.
You must specify a **virtual directory** name for your Web Service as it will live on the IIS server.
• PBL, Application Object, Project, NVO
NVUOs – Code as you normally would

```sql
SELECT BLOB Image_Data
  INTO :lblb_data
  FROM PCI_Human_Image_Worker_Queue
  WHERE image_id = :iIdec_image_id

SELECT BLOB Image_Data
  INTO :lblb_data
  FROM PCI_Human_Image_Worker_Queue
  WHERE image_id = :iIdec_image_id
```

```javascript
if (this_of_is_debug_mode) {
  this_of_write_log("Image service Controller started on invocation of method 'of_deskew_image'. ", true);
  this_of_write_log("Session ID: "+ as_session_id + ", Image ID: "+ as_image_id);
}
```

```java
Integer li_file_no
Long li_width
Long li_height

this_of_set_cpu_start(); // Record Start
this_of_set_session_id(as_session_id); // Set Session ID

if (this_of_is_debug_mode) {
  this_of_write_log("Image service Controller started on invocation of method 'of_deskew_image'. ", true);
  this_of_write_log("Session ID: "+ as_session_id + ", Image ID: "+ as_image_id);
}
```

// Connect to DB
ii_rc = io_tr_dbs.of_connect();

// Check connection!
ii_rc = fn_check_db_status(this, io_tr_dbs);

// Convert data type
iIdec_image_id = DEC(as_image_id)

// YES => Get FORM image
ii_rc = fn_check_db_status(this, io_tr_dbs);
```

// YES => Check DB Status!
Wizard selections may always be changed in the Project.
Deployment Options

- Directly to IIS
- or
- create an MSI install File
• You **must** select which methods you want to expose

• You can view WSDL and test your Web Service
Viewing WSDL

- Must deploy your .NET Web Service target first
- Project View WSDL button OR
- In a browser
  http://hostname/virtdirname/service.asmx?WSDL
IIS Directory – What is here?
Web Service Virtual Root Directory
Global.asax file

• A source file where developers can add application level logic into their Web applications. Located at the root of a particular Web application's virtual directory tree
• Application events such as Application_Start, Application_End, Session_Start, Session_End reside here.
• Automatically parsed and compiled into a dynamic .NET Framework class
• The first time any resource or URL within the application namespace is activated or requested Configured to automatically reject any direct URL request so that external users cannot download or view the code within

<%@ Application Codebehind="Global.asax.cs" Inherits="PBWebApp.Global" %>
DISCO Files

- DISCO is a Microsoft technology for publishing and discovering Web Services
- DISCO files make it possible to discover the Web Services exposed on a given server
- DISCO files make it possible to discover the capabilities of each Web Service (via documentation) and how to interact with it
- DISCO files live in the Web Application’s virtual root

```xml
<?xml version="1.0" encoding="utf-8"?>
<discovery xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    <contractRef ref="http://localhost/pci_image_services/nc_image_service_interface.asmx?wsdl"
        docRef="http://localhost/pci_image_services/nc_image_service_interface.asmx"
        xmlns="http://schemas.xmlsoap.org/disco/scl/" />
</discovery>
```
ASP.NET provides support for Web Services with the .asmx file (a wrapper to your Web Service)

- Similar to an .aspx file
- From a browser, enter the following:
  - http://hostname/virtdirname/service.asmx
- The ASMX file lists your Web Service methods
- Clicking a link takes you to a test "harness" for that method
Testing your Web Service

GetEmployees

Test

To test the operation using the HTTP POST protocol, click the 'Invoke' button.

Invoke

SOAP 1.1

The following is a sample SOAP 1.1 request and response. The placeholders shown need to be replaced with actual values.

```xml

  <employee>
    <emp_id>123</emp_id>
    <manager_id>456</manager_id>
    <emp_name>John Doe</emp_name>
    <emp_email>johndoe@example.com</emp_email>
    <emp_address>123 Main St</emp_address>
    <emp_phone>555-1234</emp_phone>
  </employee>
</ns1:GetEmployeesResponse>
```
This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<employees>
  <employee>
    <emp_id>102</emp_id>
    <manager_id>501</manager_id>
    <emp_fname>Fran</emp_fname>
    <emp_lname>Whitney</emp_lname>
    <dept_id>100</dept_id>
    <street>49 East Washington Street</street>
    <city>Needham</city>
    <state>MA</state>
    <zip_code>02192</zip_code>
    <phone>6175553985</phone>
    <status>A</status>
    <ss_number>017349033</ss_number>
    <salary>45700.00</salary>
    <start_date>1994-02-26T00:00:00</start_date>
    <termination_date>0001-01-01T00:00:00</termination_date>
    <birth_date>1966-06-05T00:00:00</birth_date>
    <bene_health_ins>Y</bene_health_ins>
    <bene_life_ins>Y</bene_life_ins>
    <bene_day_care>N</bene_day_care>
    <sex>F</sex>
  </employee>
  <employee>
    <emp_id>105</emp_id>
    <manager_id>501</manager_id>
    <emp_fname>Matthew</emp_fname>
    <emp_lname>Cobb</emp_lname>
    <dept_id>100</dept_id>
    <street>77 Pleasant Street</street>
    <city>Waltham</city>
  </employee>
</employees>
```
Why Did We Do This?

- Interoperability
- You now have a Web Service ready to be accessed from:
  - PowerBuilder
  - Appeon
  - Java
  - C#
  - VB
  - Delphi
  - ...

namespace DotNetWSClient
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            Cursor.Current = Cursors.WaitCursor;
            pbWebService.s_employees.employees = new DotNetWSClient.pbwebservice.s_employees();
            pbWebService.s_webService.proxy = new DotNetWSClient.pbwebservice.s_webService();
            employees = proxy.of_getemployees();
            dataGridView1.DataSource = employees.employees;
            dataGridView1.AutoSizeColumns();
        }
    }
}
CONSUMING Web Services
Once you have the details and have built your web service consumer application, how do you call that web service’s methods?

- Create a Simple Object Access Protocol (SOAP) message
  - PowerBuilder provides two options capable of reading and writing SOAP messages
    - “Legacy” EasySoap PBNI extension
    - “New” .NET Engine
• An XML-based communications protocol
  – “Everything is XML”
• Industry standard for cross-platform distributed messaging
• Defined by World Wide Web Consortium (W3C)
Consuming a Web Service from a PowerBuilder client requires a Web Service proxy.

A network connection is needed, but Web Services require a special Soap Connection.

The Web Service is similar to an NVUO as a container of methods which could be called via SOAP (Simple Object Access Protocol).

Invoking Web services through SOAP requires:

- Serialization and deserialization of data types
- The building and parsing of XML-based SOAP messages

A PowerBuilder Web Service client proxy performs these tasks for you eliminating the need to have extensive knowledge of:

- The SOAP specification and schema
- The XML Schema specification
- The WSDL specification and schema
**Prerequisites**

**PB 11.x/11.5.x:** .NET 2.0 Framework SDK on development machine + .NET 2.0 Framework (Runtime) on both development and deployment machine.

**PB 12.x:** .NET 3.5 Framework SDK on development machine + .NET 3.5 Framework (Runtime) on both development and deployment machine.

**PB 12.5.x:** .NET 4.0 Framework SDK on development machine + .NET 4.0 Framework (Runtime) on both development and deployment machine.

**PB 12.6.x:** .NET 4.0 Framework SDK on development machine + .NET 4.5 Framework (Runtime) on both development and deployment machine.
Web Service Proxy Wizard
Choose the Web Service Engine

About Web Service Proxy Wizard

This wizard will help you create a project object that can build proxies for a web service. You will need a proxy for each service that you want to access from your client application.

Once you have built and deployed the proxy, you can use the web service in PowerScript.

If this is what you want to do, click Next to proceed; otherwise, click Cancel.

Use .Net Engine

You must first select the type of SOAP engine you want to use to access a Web service. The .NET engine supports more features and a slightly larger range of datatypes than the EasySoap engine, but it also requires the .NET Framework SDK on your development machine and the .NET framework on all deployment machines. If you keep this check box unchecked, your application will use the EasySoap engine to access the Web services that you select in this wizard.
Specify WSDL

In order to create a proxy, enter the WSDL file name with its full path, or URL.
For a local WSDL file, enter the file name with its full path.
For a remote WSDL file, enter its URL.

Which WSDL file do you want to access?

WSDL File Name:
http://localhost/webservice/n_webservice.asmx?WSDL

Firewall Option
You can set or modify the firewall settings in PowerBuilder System Options dialog box.

Assembly Name: webservicedemo
Select a Service From WSDL

There is only 1 service in the WSDL file.
Please select it, then proceed.

Services:
-WebService.n_webservice
Define Prefix for Proxy (Optional)

By default, the proxy name for each port is: portname. But you can add a prefix to them. Note: The proxy name MUST be less than 40 characters.

Prefix For Proxy Name
Specify Project Name and Library

Specify a project name and select a library where it will be stored.

Project Name:
```
p_pbwsclient_wsproxy
```

Project Library:
```
C:\Documents and Settings\bruce\My Documents\TechWave\web
```
It is a *good* practice to store your proxies in a separate PBL in your library list.
Upon completion of the WSP Wizard, the new project is visible in the System Tree, and the project will be open in the painter.

Next, deploy the project to have the PB IDE build the appropriate proxy components!
• If your company uses a Proxy Server to bridge between you and the Firewall, visit the Tools → System Options dialog
• Input the name of your Proxy Server, port, your user id and password to that proxy server
• This is for design-time Internet connections only
The Web Service Proxy

- System Tree (expanded), following the deploy of the proxy project
- The function(s) available from the Web Service will be visible under the proxy
- Be sure you understand that the proxy project is separate from the actual proxy object
Use of Aliases in Proxy

- PowerBuilder is **not** case sensitive
- XML (SOAP) and .NET **are** case sensitive
- To get around that difference, each method in the proxy uses an alias
- The string that follows “alias for” contains the case-sensitive name and the signature of the corresponding XML or SOAP method
Exported Web Service Proxy

• Note the “alias for” clauses in the function or subroutine declarations
.Net Web Service Engine – Files Created from Proxy
Web Service Runtime Engines

- **EasySoap Engine** – `pbsoapclientnnn.pbd/pbx`
  - This engine is backward compatible with the PB9=>PB12.6 Web Service engine
  - It can work on machines that don’t have the .NET framework
- **.NET Engine** – `pbwsclientnnn.pbd/pbx`
  - This is new .NET SOAP engine
- **Both of the above define two classes:**
  - `SoapConnection`
  - `SoapException`
What Was that PBX Reference?

- An extension to PowerBuilder functionality created using the PowerBuilder Native Interface (PBNI)
- *Before 10.5*, a PBNI extension (*.pbx or *.dll) developer had to:
  - Use the `pbx2pbd` utility to create a PBD file from an extension
  - Be sure to put the extension file (PBX) in the application's search path *and* add the PBD file to the target's library list
- *Now there are fewer steps:*
  - Import the *.pbx directly into your *.pbl’s using the System Tree
  - Must still deploy the extension in the application’s path
Importing PowerBuilder Extensions

- *Prior* to PB 10.5, to gain a SoapConnection, you needed to add `pbsoapnnn.pbd` to your library list.
- `Pbsoapnnn.pbd` was a PBNI extension for EasySoap.
- Now you can import the `*.pbx` directly to a PBL.
- To do so, right-click over a PBL.
Choosing the SOAP Flavour

- `PbwsclientNNN.pbx` is the extension for the .NET Web Service engine
- `PbsoapclientNNN.pbx` is the extension for EasySoap
Important Points About These Imports

• Using `pbwsclientnnn.pbx` requires the .NET 2.0, 3.5, 4.0 or 4.5 Framework on design-time and runtime machines. **Note**: .Net 4.5 can not be used with PB 12.5.x or lower!

• Both extension files contain the same objects, and you use these objects and their methods in similar ways

• The `Sybase\Shared\PowerBuilder` directory contains PBD versions of the extension files that may still be used instead of importing the extensions (add PBDs to library list instead)

• When you create a Web service client application, you must deploy the extension file that you use along with the client executable to a directory in the application's search path
  – The Runtime Packager tool automatically includes the extension files required by your Web service applications
Will help to ensure PBNI extensions are deployed to your end users:
• Following the import of the .NET extension, you will see two new objects in the System Tree:
  – SoapConnection
  – SoapException
• Notice the CreateInstance method in soapconnection
• After importing the SoapConnection object, you are ready to write code to communicate with the Web Service

• Begin by instantiating the soapconnection object:
SoapConnection Methods

• New methods that were added to SoapConnection in PowerBuilder v10.5 & higher
• Prior to PB v10.5, most connection options were passed in as arguments to the SetOptions( ) method of SoapConnection
• Now, there are individual methods you may call
• For EasySoap use:
  – SetSoapLogFile( )
  – SetTimeout( )
  – UseConnectionCache( )
Securing Web Services

- Securing Web Services has been secondary from the beginning of the specification
- However, you have seen some security measures are in place
  - The ability to secure a Web Service:
    - Basic authentication (user id and password)
    - Use of digital certificates
- You may also secure a Web Service through the use of SOAP Headers
- This section will show you how to use SOAP Header authentication
• Declare a reference variable of type Web Service proxy
• Create an instance of the Web Service proxy
Sample SOAP Message

- Use of SOAP Headers is optional
- Below is an example of calling a Web Service method named *GetEmployees*

```xml
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
        <GetEmployees xmlns="http://teamsybase.com" />
    </soap:Body>
</soap:Envelope>
```
Note about SOAP Headers

• Be aware that authenticating callers by encoding plaintext user names and passwords in SOAP Headers is not secure.

• To secure SOAP Header information you could:
  – Encrypt SOAP messages by writing a SOAP extension that unencrypts requests and encrypts responses.
  – Use SSL / HTTPS to publish the Web Service.
Web Service
DATAWINDOWS
In PowerBuilder v11.0 and higher, you can use a Web Service as the data source for a DataWindow object:

- Supports a disconnected client model
- Eliminates requirement that database vendor’s client software reside on end-user machine
- Web Service ‘result set’ support
• Are an extension of the Web Services support that has been in PowerBuilder since Version 9.0
  – Uses the .NET Web Service engine
  – Creates a .NET assembly to do the work behind the scenes
• Web Service DataWindows are modeled on the way the Stored Procedure DataWindow works
• Two components:
  – Design-time component that allows you to browse, select a Web Service, then a specific method
  – Run-time component that
    • Retrieves data and maps to DataWindow columns
    • Updates data mapping columns to Web Service method inputs
Restrictions on Web Service Methods

- The return of the Web Service method must be:
  - **Simple** data types such as Integer, String, Date, Time, Double, Blob (base64Binary), Boolean, Decimal, Float, Long, DateTime, Char (byte), etc
    - DWO will have a single column/row
  - Array of simple types
    - DWO will have $n$ rows of a single column depending on the size of the array
  - Structure of simple types
    - DWO will have 1 row with $n$ columns depending on the number of variables in the structure
  - Array of structure
    - DWO will have $n$ rows, $n$ columns
- Some Web Service methods *will not* work with the DataWindow
• Web Service DataWindows will allow *Retrieval Arguments* (If the Web Service method has input parameters)

• Query Mode is **not** supported

• The Web Service method metadata is used to create the actual DataWindow object

• You will use the Retrieve( ) & Update ( ) methods just as you do today!
Most Presentation Styles are supported:
RichText and OLE are not supported
First, select a WSDL file describing the Web Service

Enter the URL to a WSDL, ASMX, or XML file, or browse a mapped drive for a WSDL file

- The file selected should be in a publicly accessible location for all members of the development team
The Assembly File serves as an interface between the DataWindow and the Web Service.

Name the Assembly File:
- If you do not name the Assembly file, the wizard will select a name based on the name of the WSDL file entry.
Next, you must select a service described in the WSDL and then one of its **public** methods.
Select which of the methods arguments or its return value to use as the result set.

Continued ...
After completing the wizard the DataWindow is displayed.
• PowerBuilder automatically generates a .NET assembly (dll) used to interact with the Web Service at runtime

• The generated .NET dll must be copied along with the application executable and required PowerBuilder runtime DLLs for Web Service applications
Some Web services support or require a user ID and password, and other session-related properties. The `wsconnection` can provide this information:

- [Image of a browser window showing properties and events of the `Wsconnection` object]
Sample WSConnection Code

```csharp
w_main (dwwsclient) (C:\Documents and Settings\bruce\My Documents\T
Script - open for w_main returns long

w_main

wsconnection  wsconn
wsconn = CREATE wsconnection
wsconn.endpoint = "http://localhost/webservice/n_websevice.asmx"
dw_1.setwsobject( wsconn)
dw_1.Post Retrieve()
```
Updates on WS DataWindows

• There are **no** transaction standards provided with Web Services
• Web Services are inherently **stateless**
  – Call a method, get a response, finished
• Given the above limitations, if updating data via a Web Service DataWindow, you will use the “Trust” methodology
  – Basically, you are throwing the data “over the fence” to the Web Service and trusting he will do the right thing
  – For example, if you have a DataWindow doing an insert, update and delete, and the call to the Web Service method for the delete fails, the Web Service DataWindow doesn’t retain knowledge of the other two operations
Defining Update Properties

- As mentioned before, the Web Service DataWindow was modeled from the Stored Procedure DataWindow
- The DataWindows Rows menu item now has a new item for Web Services Updates...
- Instead of mapping the DataWindow to a particular Stored Procedure, you will map the DataWindow (columns) to a particular Web Service method input parameter(s)
Web Service DataWindow Updates

- Similar to Stored Procedure update options
• New **WSError** event is analogous to the existing DataWindow DbError event when using a Web Service data source

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation</td>
<td>Type of operation (Retrieve, Update, Insert, Delete, …)</td>
</tr>
<tr>
<td>Rownum</td>
<td>Row number (or 0 if not applicable such during)</td>
</tr>
<tr>
<td>BufferName</td>
<td>Name of the buffer being accessed while the error occurred</td>
</tr>
<tr>
<td>WSInfo</td>
<td>The WSDL file, the URL that defines the Web service, or the assembly that is used access the Web service</td>
</tr>
<tr>
<td>Method</td>
<td>Name of the Web service method invoked</td>
</tr>
<tr>
<td>ErrorMessage</td>
<td>Exception message returned from the method</td>
</tr>
</tbody>
</table>
You can also perform *limited* tracing of the Web Service DataWindow.

Do so by adding a key-value pair to PB.INI:

```
[DataWindow]
dataWindow = 1
```
Questions?
Have you hugged your DataWindow today?

Obrigado!  Gracias  Grazie

THANK YOU

Merci  Vielen Dank  Köszönettel