DYALOG

The tool of thought for expert programming

Dyalog™ for Windows

# SAWS* User Guide

*Stand-Alone Web Service Framework

## Version 1.4


Dyalog Limited

Minchens Court
Minchens Lane
Bramley
Hampshire
RG26 5BH
United Kingdom

tel: National (01256) 830030
International +44 1256 830030
fax: +44 (0)1256 830031
email: support@dyalog.com
http://www.dyalog.com

*Version 1.4, Developed for Conga v2.1*

***First Edition March 2011***

# Contents

CHAPTER 1

# What is SAWS?

## Background

SAWS is a workspace with tools to help you easily share the functionality you develop in Dyalog APL with others via Web Services. SAWS hides most of the underlying complexities of building Web Services, allowing you to focus on solving the problem at hand and make your results available to others. SAWS uses Conga[1] to communicate via TCP/IP. The "Web Service" protocol is one of the most widely used mechanisms for making functionality over the internet, and is supported by a very wide variety of programming languages and development tools.

## Introduction

APL programmers have long built highly functional applications and utilities to perform various types of analysis, query databases, and a myriad of other tasks. Sharing the results of these efforts with others, particularly those outside of APL realm, has often been cumbersome and sometimes problematic. Conversely, incorporating the results of functionality developed outside of APL has proven to be similarly challenging. Enter the Stand Alone Web Service (SAWS) framework.

SAWS:

- Enables an APL programmer to easily make results available via Web Services without having to become an expert in all of the standards and protocols necessary to develop and deploy Web Services.
- Makes it easy to retrieve the results of Web Services developed by others and use them in a natural "APL" manner.

SAWS will handle most Web Service needs, but is not intended to be a comprehensive offering that addresses all of the nuances of very complex Web Services. In particular, SAWS supports Web Services using the Simple Object Access Protocol (SOAP) standard and Web Service Description Language (WSDL). At present, SAWS does

[1] Please refer to the Conga User Guide for more information on Conga.

not support REpresentational State Transfer (RESTful) Web Services nor Web API Web Services.

# Testing SAWS

To quickly test SAWS on your system, load the SAWS workspace and enter:

```
      SAWS.Test 1
0  Conga ...
Web server 'HTTPSRV' started on port 8080
Handling requests using ##.SAWS.HandleRequest
 Running  100  tests...
100  calls in  611  msec =   163.7  calls/sec
 Response:  0   Regression  1  RegResult                    xmlns
http://localhost/MyWebService/
                            2  Coeff0     -0.05
                            2  Coeff1     2.03
                            2  Coeff2     0
                            2  Residual   0.00075
0
Object 'HTTPSRV' has been closed - Web Server shutting down
```

# Web Services Via .NET

In addition to SAWS, Dyalog APL provides a means to build and call Web Services using .NET. Please refer to the .NET Interface Guide for more information.

CHAPTER 2

# A Brief Web Service Primer

**Note:** This chapter presents some background information about Web Services. For those of you who are eager to roll up your sleeves and get to building and using Web Services, skip to Chapter 3.

## What is a Web Service?

A quick search on the Internet for the definition of a Web Service will turn up a rather large number of results. The W3C defines a web service as "a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically Web Services Description Language WSDL). Other systems interact with the web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other web-related standards."

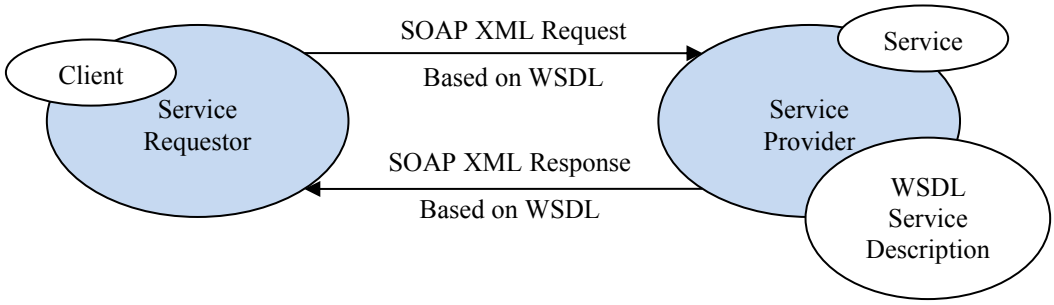
**Web Services are modular –** a Web Service is self-contained and self-describing. Everything necessary to invoke a Web Service and interpret its results is a part of the Web Service itself.

**Web Services are accessed via standard protocols –**Web Services can be accessed over the Internet or an intranet using a web browser or other client.

**Web Services are platform independent –** Web Service providers and requestors can communicate effectively without any knowledge of the platform that either is using.

**Web Services share data, business logic, or processes –** Web Services can deliver a wide range of function from very simple a query/response service to very complicated business processes.

# SOAP, WSDL, and many other Acronyms




The diagram above depicts a typical Web Service interaction. The provider has developed some sort of service that he wishes to make available. The service is developed in whatever language or technology the provider chooses. The service could implement several methods (functions) each with different parameters (arguments). To make the service available, the provider describes the service, its methods, their parameters and responses using the Web Service Description Language (WSDL). WSDL is formatted using XML.

The requestor uses the WSDL Service Description to build a request message using SOAP[2]. SOAP is formatted using XML as well. The SOAP message is transmitted from the requestor to the provider typically using HTTP. HTTPS can also be used when secure interaction is required. The request will indicate what method is being requested along with any parameters necessary for the request.

The provider receives the request message and parameters, and if valid, processes the request and formats the response according to the service description and transmits it back using HTTP (or HTTPS).

Finally, the requestor receives the response, parses it and uses the information contained therein.

# WSDL

WSDL uses XML to describe a Web Service. When you want to use a Web Service provided by someone else, you will need to examine its WSDL to understand how to invoke the service, what arguments will be passed, and what the results will look like. There are six primary elements in a WSDL document.

[2] SOAP once stood for “Simple Object Access Protocol”, but the acronym was dropped with Version 1.2 of the SOAP standard.

| Element | Description |
|---|---|
| <definition> | This is the root element of the document and contains all the other elements. |
| <types> | Describes the data elements and types in the input and output messages. |
| <message> | Describes the messages that will be transmitted. |
| <portType> | Describes the operations that are supported. |
| <binding> | Describes how the messages will be transmitted. |
| <service> | Describes the location of the service. |

We'll take a closer look at how to interpret a WSDL document in order to invoke a Web Service from APL in Chapter 4.

# SOAP

The request that is sent to a Web Service and the response it sends back use SOAP. Below is sample request using SOAP 1.1 over HTTP.

```
POST /stockquote.asmx HTTP/1.1
Host: www.webservicex.net
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://www.webserviceX.NET/GetQuote"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap=http://schemas.xmlsoap.org/soap/envelope/>
  <soap:Body>
    <GetQuote xmlns=http://www.webserviceX.NET/>
      <symbol>string</symbol>
    </GetQuote>
  </soap:Body>
</soap:Envelope>
```

The first part of the message is an HTTP header which describes the location of the Web Service, the fact that this is a SOAP request, the function that will be invoked, and how the content is encoded. The second part of the message is an XML message that contains the SOAP request with the function to be invoked and the parameters that are passed. There are several ways to invoke Web Services; SAWS uses SOAP 1.1 over HTTP.

CHAPTER 3

# Let's Build a Web Service

## Steps to Build and Run a Web Service

Building a Web Service using SAWS consists of the following steps:
In a namespace,

- Write one or more functions that do something useful, interesting, or amusing.
- Write a function called `BuildAPI` that describe your useful, interesting, or amusing functions, their arguments, and results.

Then, start the SAWS Server. That's it.... end of story.

## A Sample Web Service

The Web Service[3] we're going to build is a very simple database application. To simplify things, the database is just a matrix in the workspace, though it could easily be stored in a file or in a database system like Microsoft SQL Server or MySQL. The point is, you've got some data and you want to share it with others.

```
      ]display[4] DataBase
┌→──────────────────┐
↓ ┌→──┐ ┌→──┐ ┌→──┐ │
│ │Apple│ 15 6.99   │
│ └───┘ └───┘ └───┘ │
│ ┌→──┐ ┌→──┐ ┌→──┐ │
│ │Ball │  3  14.99 │
│ └───┘ └───┘ └───┘ │
│ ┌→────┐ ┌→──┐ ┌→──┐ │
│ │Cactus│ 0  9.99  │
│ └────┘ └───┘ └───┘ │
│ ┌→──┐ ┌→──┐ ┌→──┐ │
│ │Daisy│ 19 2.49   │
│ └───┘ └───┘ └───┘ │
└∊──────────────────┘
```

[3] This Web Service can be found in `#.PriceCheck` in the `SAWS` workspace.
[4] `]display` is a user command to display the structure of the result of an APL expression. For more information on user commands, please visit http://www.dyalog.com/documentation.

Our database consists of three columns:

`[;1]` The Item Name
`[;2]` Quantity on Hand
`[;3]` Item Price

We're going provide two functions. The first, **`ListItems`**, will return the names of items in the database. The second function, **`GetItemInfo`**, will take an item name as an argument and return the quantity on hand and price.

```
     )NS PriceCheck
#.PriceCheck

     )CS PriceCheck
#.PriceCheck

     ∇ r←ListItems a;result;noatt
[1]    ⍝ Implements the ListItems method for the PriceCheck web service
[2]    ⍝ arg - empty MLS (Markup Language Structure) there are no
         arguments to this method
[3]    ⍝ r[1] - 1 (indicates r[2] is an MLS)
[4]    ⍝ r[2] - MLS containing the  result
[5]    ⍝      [;1] - depth of nesting (origin 1)
[6]    ⍝      [;2] - element name
[7]    ⍝      [;3] - element value
[8]    ⍝      [;4] - 2 column attribute name/value pairs
[9]    ⍝ The result represents a 2 level nested structure of
[10]   ⍝  ItemList which contains 0 or more ItemNames
[11]   ⍝   equivalent to the XML:
[12]   ⍝ <ItemList>
[13]   ⍝   <ItemName>First Item Name</ItemName>
[14]   ⍝   <ItemName>Second Item Name</ItemName>
[15]   ⍝   ...
[16]   ⍝ </ItemList>
[17]   noatt←0 2⍴⊂'' ⍝ no attributes
[18]   result←1 4⍴1 'ItemList' ''noatt ⍝ build the ItemList Level
[19]   result⍪←2,(⊂'ItemName'),DataBase[;,1],⊂noatt ⍝ Add the ItemNames
         from the database
[20]   r←1 result
     ∇
```

```
     ∇ r←GetItemInfo arg;ind;name;qty;price;resp;noatt;result
[1]   ⍝ Implements the GetItemInfo method for the PriceCheck web service
[2]    ⍝ arg - 1 row Markup Language Structure (MLS)
[3]    ⍝      [;1] level (1)
[4]    ⍝      [;2] 'ItemName'
[5]    ⍝      [;3] character vector of the name of the item to retrieve
[6]    ⍝      [;4] ⊂0 2⍴''  indicating there are no attributes
[7]    ⍝ r[1] - 1 (indicates r[2] is an MLS)
[8]    ⍝ r[2] - MLS containing the result
[9]    ⍝      [;1] - depth of nesting (origin 1)
[10]   ⍝      [;2] - element name
[11]   ⍝      [;3] - element value
[12]   ⍝      [;4] - 2 column attribute name/value pairs
[13]   ⍝ The result represents a 2 level nested structure of
[14]   ⍝  ItemInfo which contains information for equivalent to the XML:
[15]   ⍝ <ItemInfo>{Not }Found
[16]   ⍝   <ItemName>name</ItemName>
[17]   ⍝   <ItemQty>quantity</ItemName>
[18]   ⍝   <ItemPrice>price</ItemPrice>
[19]   ⍝ </ItemInfo>
[20]   name←(arg[;2]⍳⊂'ItemName')⊃arg[;3],⊂'' ⍝ get the ItemName element
[21]   ind←DataBase[;1]⍳⊂name ⍝ look the name up
[22]   resp←'ItemName' 'ItemQty' 'ItemPrice',[1.5](DataBase⍪name ⍬
         ⍬)[ind;] ⍝ look up item information
[23]   noatt←0 2⍴⊂'' ⍝ no attributes
[24]   result←1 4⍴1 'ItemInfo'('Not Found'↓⍨4×ind≤⍬⍴⍴DataBase)noatt ⍝
         ItemInfo level
[25]   result⍪←2,resp,⊂noatt ⍝ item details
[26]   r←1 result
     ∇
```

Finally, we'll define our API (Application Programming Interface) for the Web Service.

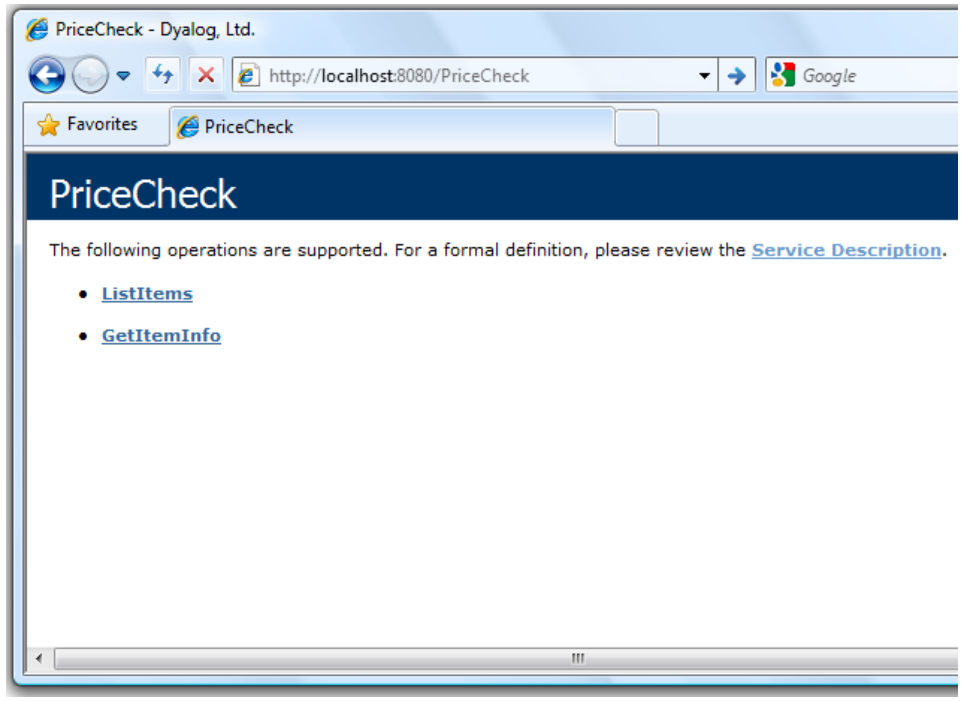
```
     ∇ api←BuildAPI;method;arg;result;⎕ML
[1]    ⍝ Construct the API for the PriceCheck Web Service
[2]    ⍝ api - vector of method definitions, one element per method in
         the Web Service
[3]    ⍝       [1] method description
[4]    ⍝       [2] argument(s) description
[5]    ⍝       [3] result(s) description
[6]    ⍝
[7]    ⍝ This Web Service has 2 methods, ListItems and GetItemInfo
[8]    ⍝ ListItems has no arguments and returns a list of the items in
         the database
[9]    ⍝ GetItemInfo takes the name of an item and returns information
         about the item
[10]   ⎕ML←1
[11]   api←0⍴⊂''
[12]  ⍝ ListItems method definition
[13]   method←1 4⍴1 'ListItems' ''(1 2⍴'documentation' 'List the items
         available via this service.')
[14]   result←arg←0 4⍴0 ⍝ initialize the ListItems method's argument and
         result descriptions
[15]  ⍝ as there is no argument to this method, arg remains a 0 row
         matrix
[16]   result⍪←1 'ItemList' ''(↑('minimum' 1)('maximum' 1)) ⍝ there is
         exactly 1 ItemList result...
[17]   result⍪←2 'ItemName' ''(↑('datatype' 'string')('minimum' 0)) ⍝
         ...which contains 0 or more ItemNames
[18]   api,←⊂method arg result ⍝ append the ListItems method definition
[19]  ⍝ GetItemInfo method definition
[20]   method←1 4⍴1 'GetItemInfo' ''(1 2⍴'documentation' 'Get information
         about an item')
[21]   result←arg←0 4⍴0 ⍝ initialize the GetItemInfo method's argument
         and result descriptions
[22]   arg⍪←1 'ItemName' ''(↑('datatype' 'string')('minimum' 1)
         ('maximum' 1)) ⍝ the argument is an ItemName
[23]   result⍪←1 'ItemInfo' ''(↑('minimum' 1)('maximum' 1)) ⍝ the result
         an ItemInfo which contains...
[24]   result⍪←2 'ItemName' ''(↑('datatype' 'string')('minimum'
         1)('maximum' 1)) ⍝ 1 ItemName
[25]   result⍪←2 'ItemQty' ''(↑('datatype' 'integer')('minimum'
         1)('maximum' 1)) ⍝ 1 ItemQty
[26]   result⍪←2 'ItemPrice' ''(↑('datatype' 'double')('minimum'
         1)('maximum' 1)) ⍝ and 1 ItemPrice
[27]   api,←⊂method arg result ⍝ append the GetItemInfo method definition
     ∇
```

The programming for our Web Service is done. Now all that's left to do is start up the SAWS server...

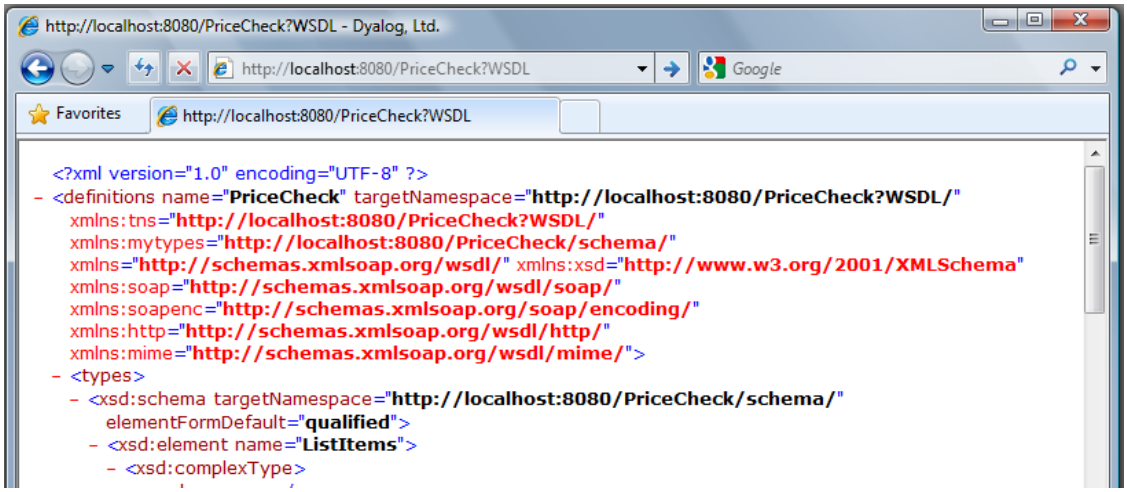
```
      SAWS.Init ⍬ Initialize SAWS
0  Conga loaded from: C:\Program Files\Dyalog\Dyalog...

      SAWS.Run 8080 1 ⍬ Start service on port 8080 in a
       new thread
Web server 'HTTPSRV' started on port 8080
Handling requests using ##.SAWS.HandleRequest
```

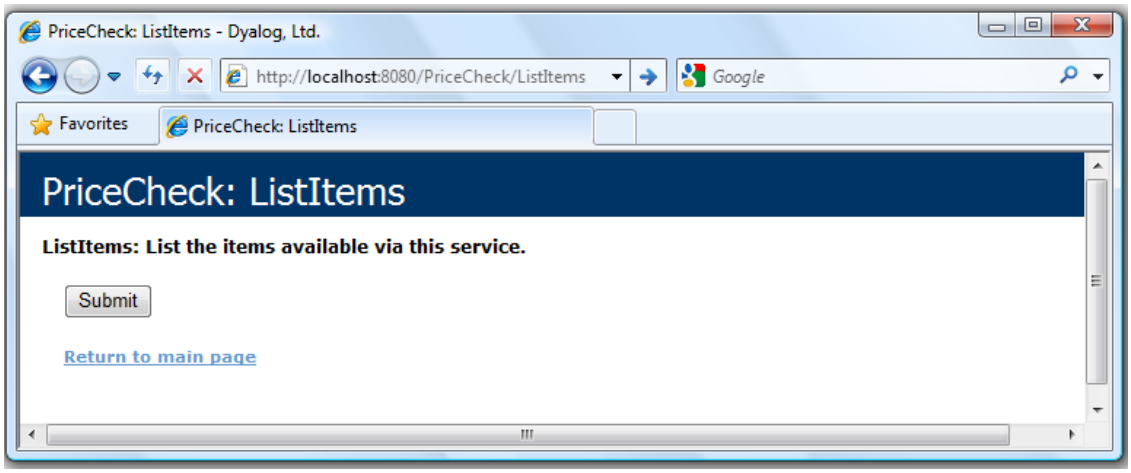
We're ready to try the service out. Since we started the service locally on port 8080, the URL for the service is http://localhost:8080/PriceCheck. The SAWS server includes HTML code to display information about your Web Service.



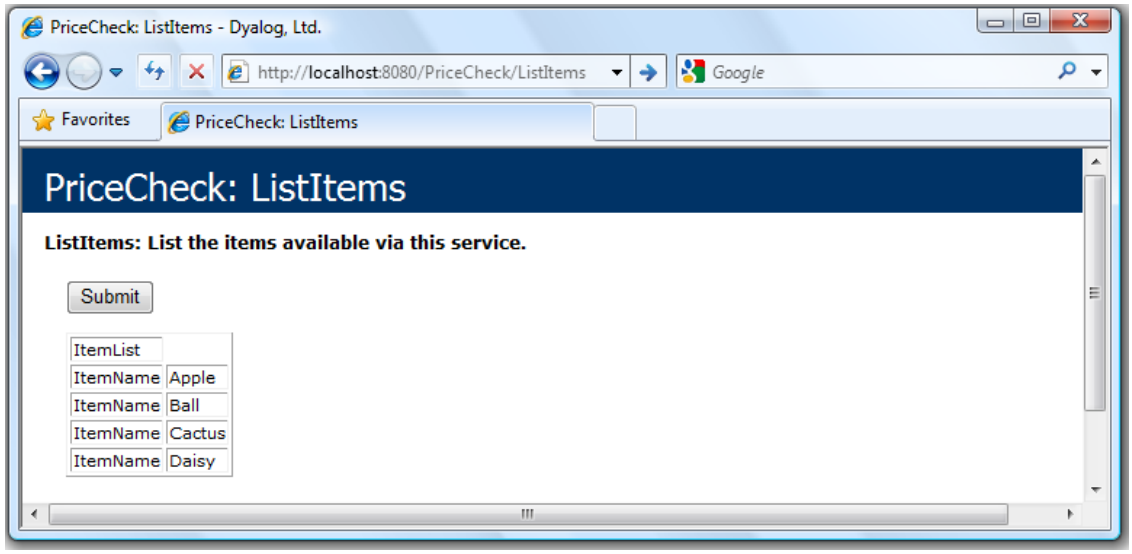
Clicking on the Service Description link will display the WSDL that was built by SAWS for your Web Service.




Clicking on the ListItems link will display a dialog to invoke the ListItems method.



Clicking on the Submit button will in invoke the ListItems method.




In less than 30 lines of code, plus comments, we have created a fully functional, albeit modest, Web Service.

CHAPTER 4

# Invoking Web Services Using SAWS

## Invoking Web Services

In Chapter 3 we saw how you can invoke a Web Service using a web browser. Many Web Services may be invoked in this manner. Using a browser is not an optimal way to integrate the results from a Web Service into your application or business process. For this, you use a Web Service client.

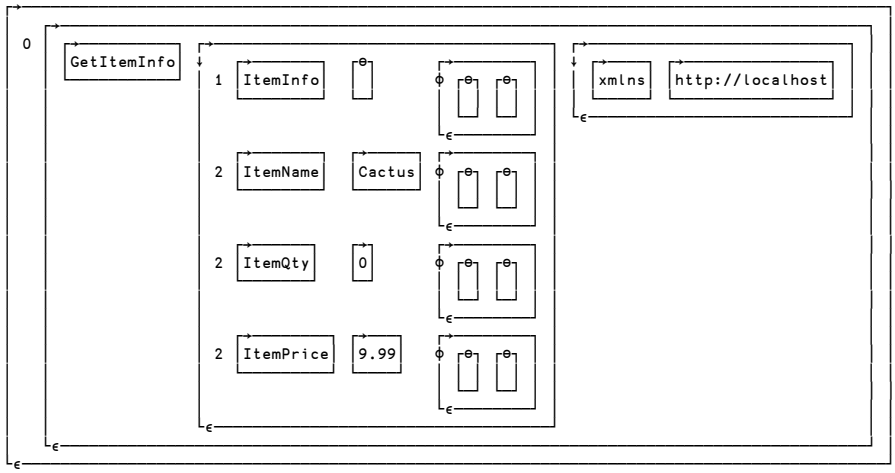
## Introducing... `SAWS.Call`

`SAWS.Call` is the client function used to invoke Web Services. Before you use `SAWS.Call` or the SAWS server, `SAWS.Run`, you need to initialize SAWS with:

```
SAWS.Init
```

This needs to be done only once in a session. Then, to invoke the Web Service we built in the previous chapter, you would use:

```
data←'localhost' 8080 'PriceCheck' SAWS.Call '' 'GetItemInfo'
   ('ItemName' 'Cactus')
]display data
```



The syntax for `SAWS.Call`[5] is:

```
r←host {port} {page} #.SAWS.Call service method pvm
```

| | |
|---|---|
| `host` | The host name for the service, in this case, 'localhost'. For other Web Services located on the Internet the host would be of the form 'www.domainname.com'. |
| `port (Optional)` | The port for the service. Defaults to port 80, the HTTP port. |
| `page (Optional)` | The webpage for the service, or, in the case of SAWS-hosted service, the namespace name containing the service. |
| `service` | The name of the service. In many cases, either the page name or the service name will be `''`. |
| `method` | The method name to be invoked. |
| `pvm` | The parameter/value matrix that contains the arguments to the method. |
| `r` | `r[1]` is a return code of 0 for success, non-zero for failure.<br>The remaining part of the result can vary based on the particular Web Service, or where the failure occurred. See Appendix II for details on SAWS.Call. |

# Reading a WSDL File

Building a successful `SAWS.Call` request can involve a bit of research, investigation, and experimentation. The primary source of information used to divine how to invoke a Web Service is its WSDL. Some Web Services also publish sample SOAP over HTTP messages. We'll start by looking at the WSDL for a public Web Service[6] that retrieves stock quotes.

1. At the top of the file, find **targetNamespace** attribute in the **definitions** tag.

```
<wsdl:definitions
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
 xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
 xmlns:tns="http://www.webserviceX.NET/"
 xmlns:s="http://www.w3.org/2001/XMLSchema"
```

[5] `SAWS.Call` is fully documented in Appendix II.

[6] See Appendix IV for the complete WSDL file listing.

```
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
targetNamespace="http://www.webserviceX.NET/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
```

The **targetNamespace** will be the server that is used as the first argument to `SAWS.Call`. The capitalization and punctuation are significant. Some **targetNamespace** tags may have a trailing `'/'` as this one does.

2. Find the **service** tag near the bottom of the file.

```
<wsdl:service name="StockQuote">
  <wsdl:port name="StockQuoteSoap" binding="tns:StockQuoteSoap">
 <soap:address location="http://www.webservicex.net/stockquote.asmx" />
 </wsdl:port>
</wsdl:service>
```

There may be several **port** tags within the service tag. Locate the **port** tag corresponding to SOAP 1.1, in this case its named StockQuoteSOAP. There are two pieces of information of interest here. The first is the **address** tag which will indicate the location and page of the web service. The second is the **binding** attribute which will lead us to the definition of the service.

3. Working up the file, find the matching "StockQuoteSOAP" **binding** tag.

```
<wsdl:binding name="StockQuoteSoap" type="tns:StockQuoteSoap">
<soap:operation soapAction="http://www.webserviceX.NET/GetQuote"
      style="document" />
```

The type attribute of the **binding** tag (StockQuoteSoap) will point us to the appropriate **portType** tag. The **soapAction** attribute of the **operation** tag is useful for debugging Web Service calls; more on this later.

4. Again, working up the file, find the matching **portType** tag.

```
<wsdl:portType name="StockQuoteSoap">
  <wsdl:operation name="GetQuote">
  <wsdl:input message="tns:GetQuoteSoapIn" />
  <wsdl:output message="tns:GetQuoteSoapOut" />
  </wsdl:operation>
</wsdl:portType>
```

There may be more than one **operation** defined. Find the **operation** you're interested in, in this case GetQuote. Within the **operation** tag, you may find **input** and **output** tags.

5. If an input tag exists, find the matching message tag

```
<wsdl:message name="GetQuoteSoapIn">
  <wsdl:part name="parameters" element="tns:GetQuote" />
</wsdl:message>
```

Each **part** tag names the element that describes the input parameters, in this case "GetQuote".

6. Find the matching **element** tag.

```
<s:element name="GetQuote">
  <s:complexType>
  <s:sequence>
  <s:element minOccurs="0" maxOccurs="1" name="symbol" type="s:string" />
  </s:sequence>
  </s:complexType>
 </s:element>
```

Here we find that there is one parameter, "symbol" and that it is a string. The **minOccurs** and **maxOccurs** attributes indicate the minimum and maximum number of occurrences of this parameter.

7. Similar to the input tag, if there is an **output** tag from Step 4 above, find the matching **message** tag.

```
<wsdl:message name="GetQuoteSoapOut">
  <wsdl:part name="parameters" element="tns:GetQuoteResponse" />
</wsdl:message>
```

8. Find the matching **element** tag,

```
<s:element name="GetQuoteResponse">
<s:complexType>
<s:sequence>
<s:element minOccurs="0" maxOccurs="1" name="GetQuoteResult"
      type="s:string" />
</s:sequence>
</s:complexType>
</s:element>
```

We see that the result is named GetQuoteResult and is a string.

Based on the WSDL file, here is what we know about this Web Service.

| | |
|---|---|
| Server address | www.webserviceX.NET (notice the capitalization matches the **targetnamespace)** |
| Port | Unless otherwise specified, port 80 (HTTP) or 445 (HTTPS) |
| Page Name | stockquote.asmx |
| Service Name | None in this case. |
| Method Name | GetQuote |

| | |
|---|---|
| Input Parameters | Name = 'symbol'<br>Value = string containing stock symbol of interest |
| Output Parameters | Name = "GetQuoteResult"<br>Value = string |

# Sample SOAP Request

A Web Service provider may publish a sample SOAP request and response. The request is useful to compare to the request that **SAWS.Call** builds.

```
POST /stockquote.asmx HTTP/1.1
Host: www.webservicex.net
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: http://www.webserviceX.NET/GetQuote

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap=http://schemas.xmlsoap.org/soap/envelope/>
  <soap:Body>
    <GetQuote xmlns=http://www.webserviceX.NET/>
      <symbol>string</symbol>
    </GetQuote>
  </soap:Body>
</soap:Envelope>
```

You can display intermediate results for both the SAWS client and server by setting **SAWS.TRACE** to 1. The HTTP header (**hdr**) and SOAP request (**req**) are displayed. You can examine SOAPAction in **hdr** to ensure that it matches the soapAction attribute in the WSDL (see step 3 above), or in the sample SOAP request.

```
      SAWS.TRACE←1
      z←'www.webserviceX.NET/' 80 'stockquote.asmx'
        SAWS.Call '' 'GetQuote' ('symbol' 'k')
hdr:
POST /stockquote.asmx HTTP/1.1
Host: www.webserviceX.NET
SOAPAction: http://www.webserviceX.NET/GetQuote
Content-Type: text/xml; charset=utf-8
Content-Length: 450
req:
<?xml version="1.0"?><SOAP-ENV:Envelope SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><SOAP-
ENV:Body><GetQuote
```

```
xmlns="http://www.webserviceX.NET/"><symbol>k</symbol></GetQuote
></SOAP-ENV:Body></SOAP-ENV:Envelope>
```

Chapter.NET Interface Guide

# Let the User Beware

There are free Web Services and there are Web Services that can be subscribed to for a fee. Be mindful that when you use a free service, the provider is under no obligation to maintain the service, inform you of changes, or even keep the service running.

CHAPTER 5

# Providing Web Services With SAWS

## Using SAWS.Run

SAWS.Run is used to run Web Service(s). It starts a Conga-based web server and processes SOAP requests.

## Dedicated Web Services

When you supply namespace name as the left argument to SAWS.Run, only methods defined in that namespace are available via the Web Service. In effect, the port you specify (8080 in this case) is dedicated to running that particular Web Service.

```
      '#.MyWebService' SAWS.Run 8080 1
Web server 'HTTPSRV' started on port 8080
Handling requests using ##.SAWS.HandleRequest
```

If you want to run multiple dedicated Web Services, you will need to run each in a separate APL session.

## Running Multiple Web Services

If you don't specify a namespace, SAWS will search for a namespace that matches the name in the request. In this way you can offer a number of Web Services from a single server. Namespaces can be nested.

Consider the following example, we have a namespace named WebServices and it contains three namespaces, PriceCheck, Regression, and Weather, each of which implements a Web Service.

To run all of these services, use:

```
    SAWS.Run 8080 1
```

This will start a web server on port 8080 in a new thread. The page name for each Web Service coincides with the namespace name. Use '/' instead of '.' to delimit the namespace names.

Using a web browser, you would specify:
http://localhost:8080/WebServices/PriceCheck/GetItemInfo?ItemName=Apple

Using `SAWS.Call`, you would specify:

```
'localhost' 8080 'WebServices/PriceCheck' SAWS.Call ''
        'GetItemInfo' ('ItemName' 'Apple')
```


# Secure Web Services Using `SAWS.RunSecure`

`SAWS.RunSecure` is an analogous function to `SAWS.Run` that provides secure Web Services over HTTPS. Please see the reference for `SAWS.RunSecure` function `SAWS.TestSecure` in the workspace for an example of how to use `SAWS.RunSecure` and `SAWS.Call` to provide and consume secure Web Services.

CHAPTER 6

# Integrating SAWS within Your Application

## Steps to Integrate SAWS

1. Have an application. It could be an existing application that you desire to make available as a Web Service. It could be an application in which you desire to use functionality available through third-party Web Services. It could be a brand new application. Or any combination of the above.

2. Copy the `SAWS` namespace into your application workspace. `SAWS` can be installed in the root or in any namespace. For this discussion we'll copy `SAWS` into the root namespace.

```
      )CS
#
      '#.SAWS' ⎕CY 'SAWS'
```

3. If your application already uses Conga, you may want to assign `SAWS.DRC` to reference the existing `DRC` namespace as follows (assuming that Conga is in `#.DRC`):

```
      #.SAWS.DRC←#.DRC
```

4. In your application initialization code, include the line.

```
      {}#.SAWS.Init
```

   This will initialize SAWS when your application runs.

5. If you are a Web Service consumer, you'll use `SAWS.Call` to invoke and retrieve results from Web Services.

6. If you are a Web Service provider, you'll need to write functions for each method in your Web Service as well as `BuildAPI` to describe your Web Service and then use `SAWS.Run` or `SAWS.RunSecure` to provide access to your WebService.

# BuildAPI

`BuildAPI` is the function which returns the description of your Web Service. SAWS uses `BuildAPI` to build the WSDL for your Web Service.

The result of `BuildAPI` is a vector with one element per method in your Web Service. Each element of the vector contains three Markup Language Structure (`MLS`)[7] elements. These are: a description of the method itself, a description of the argument(s) to the method, and a description of the result(s) of the method.

## Describing the Method

The method description can be as simple as the character vector name of the method.

```
'ListItems'
```

Or the method could be an `MLS`as follows:

`[;1]` 1
`[;2]` method name
`[;3]` `''`
`[;4]` parameter/value matrix (`pvm`)[8]

Valid parameters are:

`'documentation'` character vector description of the service
`'pattern'` integer 1,2,or 4 describing the type of interaction with the method.
1 – one way, input only
2 – two way, request/response
4 – one way, output only (notification)

```
method←1 4⍴1 'GetItemInfo' ''(2 2⍴'pattern' 2
          'documentation' 'Get information about an item')
```

## Describing the Method's Arguments and Results

The argument and result descriptions are `MLS` structures and describe the allowable structure of the data for each element.

`[;1]` level - generally 1 for argument elements, but can be used to represent more complex structures
`[;2]` element name
`[;3]` `''` – not currently used
`[;4]` parameter/value matrix (`pvm`)

---

[7] See Common Data Structures for more information on the structure of an `MLS`.
[8] See Common Data Structures for more information on the structure of a `pvm`.

Valid parameters are:

| | |
|---|---|
| `'documentation'` | character vector description of the argument or result |
| `'datatype'` | generally, one of `'string' 'boolean' 'integer'` or `'double'`. It can also be a data type described by http://www.w3.org/TR/xmlschema-2/ . If you use one of these data types, use the prefix `'xsd:'` as in `'xsd:dateTime'`. |
| `'minimum'` | integer indicating the minimum number of times this element can occur within its parent element. |
| `'maximum'` | integer indicating the maximum number of times this element can occur within its parent element. |

```
      3 2⍴'datatype' 'integer' 'minimum' 1 'maximum' 1
```

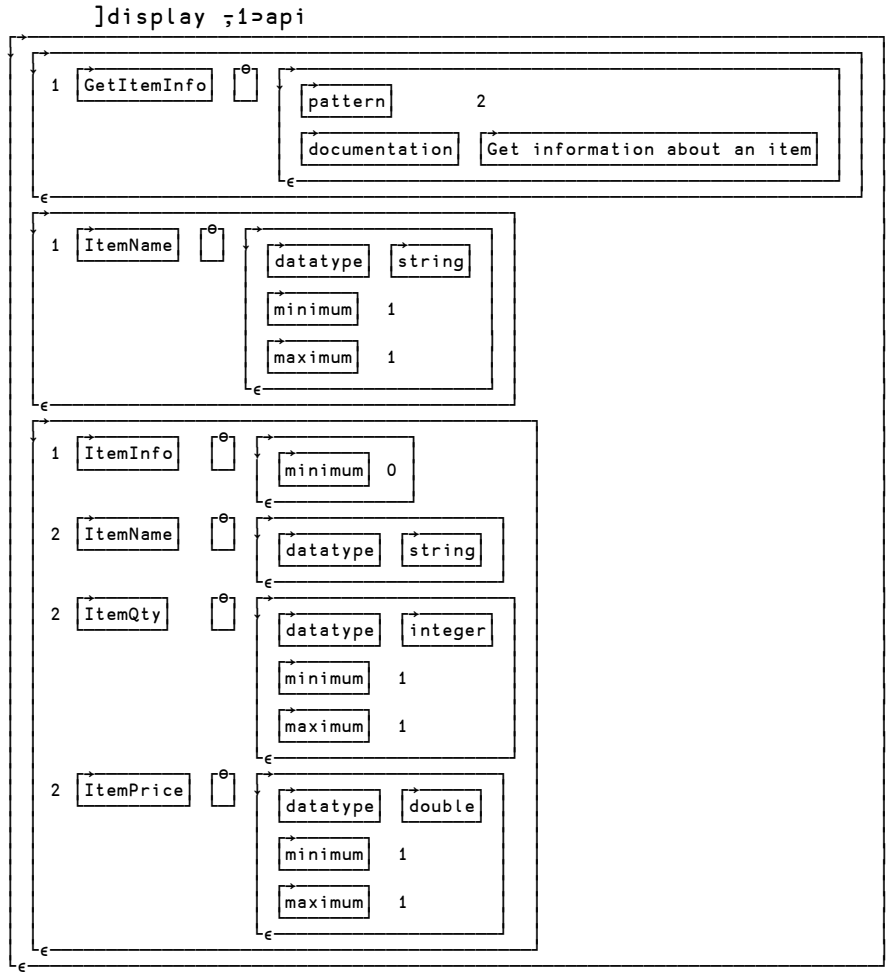
Describes a single required integer element.

```
      1 2⍴'minimum' 0
```

Describes an optional array of any length.

The following code fully describes the GetItemInfo method of the PriceCheck Web Service:

```
method←1 4⍴1 'GetItemInfo' ''(2 2⍴'pattern' 2
      'documentation' 'Get information about an item')
result←arg←0 4⍴0
arg⍪←1 'ItemName' ''(3 2⍴'datatype' 'string' 'minimum' 1
      'maximum' 1)
result⍪←1 'ItemInfo' ''(1 2⍴'minimum' 0)
result⍪←2 'ItemName' ''(1 2⍴'datatype' 'string')
result⍪←2 'ItemQty' ''(3 2⍴'datatype' 'integer' 'minimum'
      1 'maximum' 1)
result⍪←2 'ItemPrice' ''(3 2⍴'datatype' 'double'
      'minimum' 1 'maximum' 1)
api←,⊂method arg result
```




# Implementing Your Methods

SAWS Web Service methods are monadic functions which return a result. The right argument represents the input message and the result represents the output message. The input and output messages are each contained in an MLS. These MLS's correspond to their descriptions contained in the API.

## Input Message

The input message contained in the right argument is an `MLS`. It contains elements representing the arguments for the method where columns `[;2]` and `[;3]` are of particular interest.

`[;1]` contains the level number, this can be ignored
`[;2]` contains the element name
`[;3]` contains the element value
`[;4]` is not currently used and contains the value `0 2⍴⊂''`

The function g`etelement` in `#.WebServices` is a useful utility to retrieve elements. It will return the element value if found, or an empty vector if not found.

```
getelement←{(⍺[;2]⍳⊂⍵)⊃⍺[;3],⊂''}
```

## Output Message

The result of your function used to implement the Web Service method is a 2 element array.

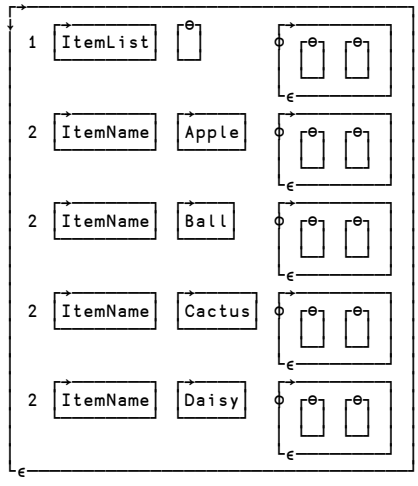
`result[1]` 1 indicates that `result[2]` contains an `MLS`
`result[2]` `MLS` if `result[1]=1`, otherwise it is any APL array

The `MLS` contains:

`[;1]` contains the level number. This can be used to indicate nesting of data in the output message/result.
`[;2]` contains the output message element name
`[;3]` contains the output message element value
`[;4]` is not currently used and contains the value `0 2⍴⊂''`

The structure below depicts an element called ItemList which contains two ItemName elements.

```
      ]dis 2⊃#.PriceCheck.ListItems 0 4⍴0
```



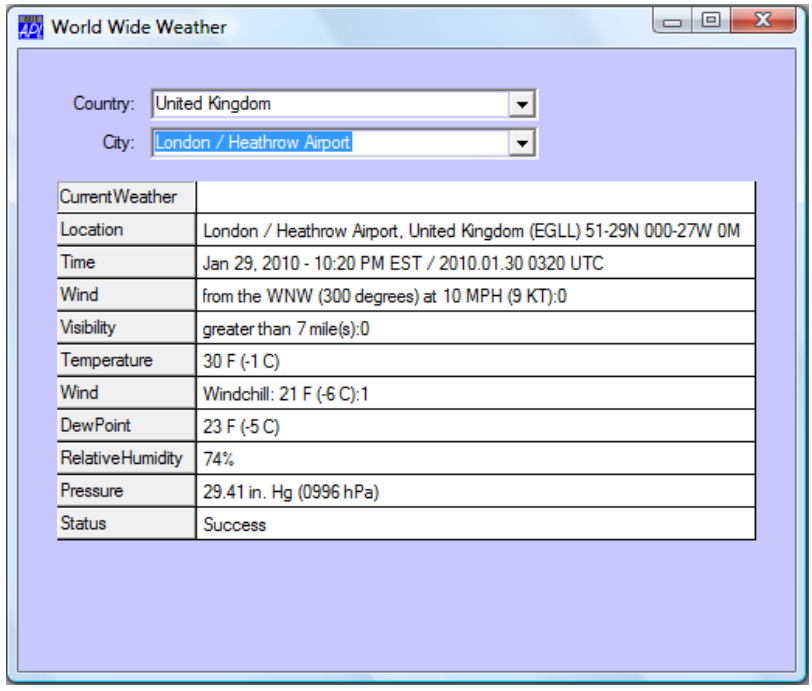
APPENDIX I

# Namespaces Provided with SAWS

`ClientSample`

Contains a class, `WWWeather`, which implements a sample world-wide weather dialog using the SAWS Client, `SAWS.Call`.

```
z←⍬new ClientSample.WWWeather
```




`MyWebService`

Contains a sample Web Service which implements a simple regression analysis service.

`PriceCheck`

Contains a sample Web Service which implements a simple database driven application.

`SAWS`

Contains the functions to invoke and serve Web Services.

`WebServices`

Contains nested namespaces, each of which implements a Web Service. This is a sample to demonstrate one way to organize the delivery of multiple WebServices using a single SAWS server.

APPENDIX II

# SAWS REFERENCE

## Common Data Structures

There are two data structures that are used frequently in SAWS. These are:

`MLS` – Markup Language Structure
`[;1]` Level
`[;2]` Element/Tag name
`[;3]` Element/Tag value
`[;4]` pvm

`MLS`'s are used in the result of the SAWS Web Service `BuildAPI` function and as the result of any functions which return the results of methods within the Web Service.

`pvm` – parameter/value matrix
`[;1]` Parameter name
`[;2]` Parameter value

If no parameters exist, the `pvm` is `0 2⍴⊂''`. `pvm`'s are used within an `MLS` and elsewhere in SAWS.

# Functions

## SAWS.Call Invoke a Web Service

```
r←host {port} {page} {soapaction} {SSLparms}
      #.SAWS.Call soaprequest|service method arg
```

| Left Argument | |
|---|---|
| `host` | A character vector containing the host (IP address or URI) for the service, optionally prefixed by http:// or https://.<br>Examples:<br>`'localhost'`<br>`'www.webservice.net'`<br>`'https://www.securews.com'` |
| `port` `(optional)` | The port for the service.<br>If `port` is not specified, the default HTTP port 80 will be used, or 443 if HTTPS is specified in the host parameter.<br>Ports 80 and 443 are the typical ports for HTTP and HTTPS services respectively. |
| `page` `(optional)` | The webpage for the service, or, in the case of SAWS-hosted service, the namespace name containing the service. |
| `soapaction` `(optional)` | If a non-standard SOAPAction is required in the HTTP header, it may be specified here as a character vector. |
| `SSLparms` `(optional)` | If calling a secure Web Service, this element is used to contain the appropriate Conga parameters in the form:<br>`('X509' client-certificate)`<br>`{('SSLValidation' n)}`<br>where `client-certificate` is a type `DRC.X509Cert` certificate. `SSLValidation` specifies any optional flags for the processing of certificates. Please refer to the Conga v2.1 or later documentation for more information on `X509Cert` and `SSLValidation`. |

### Right Argument

The right argument to `SAWS.Call` can be either:

- A character vector containing the entire SOAP over HTTP request; OR
- A 3 element nested vector

| | |
|---|---|
| `service` | The name of the service. In many cases, this will be either the page name or the service name will be `''`. |
| `method` | The method name to be invoked. |
| `arg` | Can be in one of three formats:<br>• A simple non-empty character vector containing the XML SOAP message. No HTTP header information should be included here<br>• An empty vector if the web service method takes no parameters.<br>• A nested vector containing the names and values for the parameters to the web service method in any of the following formats:<br>  • Depth 2 vector of vectors, for example<br>`('param1' 'value1' 'param2' 'value2')`<br>  • Depth 3 vector where each element contains a parameter/value pair. This format is similar to that used for Dyalog GUI objects.<br>`(('param' 'value1')('param2' 'value2'))`<br>  • 2 column matrix of `[;1]` parameter names `[;2]` parameter values<br>`((2,2)⍴'param1' 'value1' 'param2' 'value2')` |
| Result | |
| `r` | `r[1]` is a return code of 0 meaning no exception was detected, or non-zero to indicate that some exception was detected. |
| | If `r[1]=0` no exception was detected<br>`r[2]` will be a 3 element vector containing<br>`[1]` the method name<br>`[2]` the method result either in the form of an `MLS` or arbitrary APL data<br>`[3]` pvm for the attributes applied to the method<br>Depending on the SOAP response is built by the Web Service, a return code of 0 may not guarantee success. Additional examination of the contents of `r[2]` may be necessary to validate the response. |
| | If `r[1]= 1` the Web Service sent a SOAP Fault response<br>`r[2]` a vector of vectors containing information about the fault |

If `r[1]=¯1` the SOAP message could not be decoded
`r[2]` is a 3 element vector containing
`[1] 'Client.Invalid'`
`[2]` error message
`[3] ''`

---

If `r[1]>1` Conga signalled an exception, then `r` contains
`r[1]` exception code
`r[2]` exception name
`r[3]` additional exception information, if any

## SAWS.Init — Initialize SAWS

```
r←SAWS.Init
```

| | |
|---|---|
| `r` | `r[1]` is 0 for success, otherwise a Conga exception code<br>`r[2]` is a character vector status message<br>If `r[1]=0` then `r[2]` is either:<br>`'Conga loaded...'` if Conga has not been previously loaded<br>`'Conga reset'` if Conga has been previously loaded |

**Discussion:**
`SAWS.Init` is used to initialize SAWS. You must call `SAWS.Init` prior to running the SAWS client or server, but you need only call it once during your session.

**Note:**
If Conga is in use elsewhere in your application, you should use that instance of Conga instead of running `SAWS.INIT`. This is done by assigning `SAWS.DRC` to the path of the `DRC` namespace. For example, if Conga (the `DRC` namespace) is in the root you would assign `SAWS.DRC` as follows:

```
SAWS.DRC←#.DRC
```

## SAWS.Run — Start a SAWS Server

```
{r}←{svc} SAWS.Run {port} {threaded} {srvname} {address}
```

| | |
|---|---|
| `address` | The address (domain name or IP address) for the service. This is used is the WSDL that SAWS generates.<br>Defaults to “localhost” if empty or not supplied. |

| | |
|---|---|
| port | The port number for the server (defaults to port 8080) |
| threaded | 1 indicates run the server in a separate thread<br>0 (the default) indicates run the server in the current thread |
| srvname | The name to assign to the web server.<br>If not supplied, Conga will assign the name 'HTTPSRV' |
| svc | A reference to the namespace containing the Web Service.<br>If svc is supplied, the server runs in "dedicated" mode and will service only requests for the Web Service defined in the namespace.<br>If svc is not supplied, the server will search for a namespace matching the Web Service name. |
| r | If arg[2] is 1 (run threaded), r is the thread number for the server.<br>If arg[2] is 0 (run in current thread), r isn't particularly interesting. |

**Description:**
SAWS must have been initialized prior to using SAWS.Run. See SAWS.INIT.

The expression:

```
SAWS.Run 0
```

will start a SAWS server named 'HTTPSRV' on port 8080 in the current thread. To stop a SAWS server running in the current thread, you need to generate a strong interrupt and the server should interrupt within 10 seconds.

## SAWS.RunSecure Start a Secure SAWS Server

```
{r}←{svc} SAWS.RunSecure {port} {threaded} {srvname}
{address} {rootcertpath} {cert}
```

| | |
|---|---|
| address | The address (domain name or IP address) for the service. This is used is the WSDL that SAWS generates.<br>Defaults to "localhost" if empty or not supplied. |
| cert | The server certificate as an instance of DRC.X509Cert. Please refer to Conga v2.1 or later documentation for information on X509Cert. |
| port | The port number for the server (defaults to port 8080) |

| | |
|---|---|
| r | If arg[2] is 1 (run threaded), r is the thread number for the server.<br>If arg[2] is 0 (run in current thread), r isn't particularly interesting. |
| rootcertpath | The path containing the certification authority (CA) certificates to be used.<br>(defaults to the result of SAWS.Samples.CertPath) |
| srvname | The name to assign to the web server.<br>If not supplied, Conga will assign the name 'HTTPSRV' |
| svc | The name of the namespace containing the Web Service.<br>If svc is supplied, the server runs in "dedicated" mode and will service only requests for the Web Service defined in the namespace.<br>If svc is not supplied, the server will search for a namespace matching the Web Service name. |
| threaded | Is 1 to run the service in a separate thread, 0 (the default) to run in the current thread |

**Discussion:**
SAWS.RunSecure is almost identical to SAWS.Run except that it uses SSL/TLS to provide secure communications.

SAWS must have been initialized prior to using SAWS.RunSecure. See SAWS.INIT.

```
      cert←DRC.X509Cert.ReadCertFromFile 'c:\mycert.cer'
      rootcertpath←'c:\mycertpath'
      SAWS.RunSecure 445 1 'HTTPSRV' '' rootcertpath cert
```

Will start a secure SAWS server named 'HTTPSRV' on port 445 in a separate thread and look in the directory c:\mycertpath for root certificates to use.

## SAWS.Stop — Stop a SAWS Server

```
{r}←SAWS.Stop srvname
```

| | |
|---|---|
| Svrname | The SAWS server name to stop. If empty (⍬ or ''), all SAWS servers will be stopped. |

## SAWS.Test, SAWS.TestSecure — Test SAWS

```
{address} SAWS.Test close
{address} SAWS.TestSecure close
```

| | |
|---|---|
| Address | The address of the web server to use. Defaults to 'localhost'. |
| Close | Flag to indicate how to run the test.<br>close=0 just start the SAWS server, do not run tests<br>close=1 start the SAWS server and run tests<br>close=¯1 SAWS server already running, just run tests |

**Discussion:**
SAWS.Test is an easy way to verify that SAWS will function in your environment. SAWS.TestSecure will perform the same tests securely using HTTPS.

# Variables

## SAWS.DEBUG — Debug Mode Settings

SAWS.DEBUG uses additive powers of 2 to turn on different debugging features.

| Mode | Description |
|---|---|
| 1 | Controls the server error trapping when executing your Web Service methods. Normally SAWS will trap any unhandled error in your Web Service method and return an error code. Setting SAWS.DEBUG to 1 will disable SAWS' built in error handling. This is useful for debugging your Web Service methods, but should not be left on in general. |
| 2 | Captures the last request and response pair.<br><br>When using SAWS.Call, the last request and response pair are stored in SAWS.LastCallRequest and SAWS.LastCallResponse.<br><br>When using SAWS.Run, the last request and response pair are stored in SAWS.LastRunRequest and SAWS.LastRunResponse.<br><br>These are useful for testing and debugging. |
| 4 | Use alternate messages.<br><br>When using SAWS.Call, you may construct an entire SOAP over HTTP request and assign it to the variable SAWS.AltRequest. SAWS will send |

| | |
|---|---|
| | this request to the Web Service rather than the one built from the arguments to `SAWS.Call`. This is useful when testing for basic connectivity to a Web Service.<br><br>When using `SAWS.Run`, you may construct a SOAP response and assign it to the variable `SAWS.AltResponse`. SAWS will send this response back to the client instead of the response built by your application code. This is useful for testing the interface to new clients to your Web Service. |

```
SAWS.DEBUG←1 ⍝ turn debug mode 1 on
SAWS.DEBUG←3 ⍝ turn debug modes 1 and 2 on
SAWS.DEBUG←¯1 ⍝ turn all debugging on
SAWS.DEBUG←0 ⍝ turn all debugging off
```

## SAWS.SILENT Suppress Session Output

SAWS.SILENT allows you to suppress output that would normally be sent to the APL session. This is useful when running SAWS in a runtime environment. Tracing output displayed when setting SAWS.TRACE∈1 3 is not suppressed.

```
SAWS.Silent←1 ⍝ suppress session output
```

Note: All output is funneled through the function SAWS.Output which could be modified to log SAWS output to a file or some similar use.

## SAWS.TIMEOUT Client Timeout Setting

SAWS.TIMEOUT allows you to set the number milliseconds that SAWS.Call should wait for a response from a Web Service before timing out. The default is 10000 (10 seconds).

```
SAWS.TIMEOUT←15000 ⍝ set 15 second timeout
```

## SAWS.TRACE Trace Client and Server

SAWS.TRACE uses additive powers of 2 to turn on different tracing features.

| Mode | Description |
|---|---|
| 1 | Controls the display of trace information. Setting this value will cause SAWS to display more detailed information about requests and responses that it processes. |
| 2 | Set terse or verbose mode. As some messages in the trace output can be rather lengthy, setting this value will truncate the output to the first 65 characters of a message. This is useful for monitoring the flow of requests and responses within SAWS rather than their detailed content. |

```
SAWS.TRACE←1 ⍝ turn verbose tracing on
SAWS.TRACE←3 ⍝ turn terse tracing on
SAWS.TRACE←0 ⍝ turn all debugging off
```

## SAWS.STYLE Cascading Style Sheet for Web Pages

The HTML web pages that SAWS.Run produces are built in the function SAWS.ServiceHTML. While you could modify this function to change the content, look and feel of the web pages, most changes can be accomplished by modifying the variable SAWS.STYLE, which is a Cascading Style Sheet (CSS).

APPENDIX III

# A Sampling of Public Web Services

This appendix lists some free public Web Services the author happened upon while doing research for this manual.

**www.webserviceX.NET**

There are a large number of free web services offered from www.webserviceX.NET ranging from stock quotations, to weather services, to Bible quotations, to unit conversions, including... if you ever wondered how many miles are in a parsec... well... here it is:

```
      3⊃,2 2⊃'www.webserviceX.NET/' 80 'Astronomical.asmx' SAWS.Call ''
         'ChangeAstronomicalUnit' ('AstronomicalValue' '1'
         'fromAstronomicalUnit' 'parsec' 'toAstronomicalUnit' 'miles')
191735141772O5.121
```

**www.wsdll.com**

This is a website that catalogs both free and fee Web Services including:

http://staging.mappoint.net/standard-30/
Microsoft MapPoint Web Services are XML Web services with a SOAP API that allows you to add location-based functionality to your application that calls on the high-quality maps, as well as the location finding and routing capabilities of MapPoint Web Services.

http://river.sdsc.edu/wateroneflow/NWIS/DailyValues.asmx?WSDL
The USGS National Water Information System (NWIS) provides access to millions of sites measuring streamflow, groundwater levels, and water quality. This Web Service provides methods for retrieving daily values data, such as discharge and water levels, from NWIS.

http://ws.cdyne.com/WeatherWS/Weather.asmx?wsdl
Get Weather information by zipcode.

```
   'ws.cdyne.com' 80 'WeatherWS/Weather.asmx' SAWS.Call 'WeatherWS'
                                'GetCityWeatherByZIP' ('ZIP' '20001')
```

**http://www.nws.noaa.gov/xml/**

This is the National Oceanic and Atmospheric Administration's National Weather Service National Digital Forecast Database Web Service.

**http://www.usgovxml.com/**

USGovXML is an index to publically available web services and XML data sources that are provided by the US government. USGovXML indexes data sources from all 3 branches of government as well as its boards, commissions, corporations and independent agencies.

APPENDIX IV

# Sample WSDL

This Appendix lists the entire WSDL for the StockQuote Web Service. At the time of this writing, this file was located at:
http://www.webservicex.net/stockquote.asmx?WSDL

```
<?xml version="1.0" encoding="utf-8" ?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
    xmlns:tns="http://www.webserviceX.NET/"
    xmlns:s="http://www.w3.org/2001/XMLSchema"
    xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
    xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
    targetNamespace="http://www.webserviceX.NET/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
    <wsdl:types>
        <s:schema elementFormDefault="qualified"
            targetNamespace="http://www.webserviceX.NET/">
            <s:element name="GetQuote">
                <s:complexType>
                    <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="symbol" type="s:string" />
                    </s:sequence>
                </s:complexType>
            </s:element>
            <s:element name="GetQuoteResponse">
                <s:complexType>
                    <s:sequence>
                        <s:element minOccurs="0" maxOccurs="1"
                        name="GetQuoteResult" type="s:string" />
                    </s:sequence>
                </s:complexType>
            </s:element>
            <s:element name="string" nillable="true" type="s:string" />
        </s:schema>
    </wsdl:types>
    <wsdl:message name="GetQuoteSoapIn">
        <wsdl:part name="parameters" element="tns:GetQuote" />
    </wsdl:message>
    <wsdl:message name="GetQuoteSoapOut">
        <wsdl:part name="parameters" element="tns:GetQuoteResponse" />
    </wsdl:message>
    <wsdl:message name="GetQuoteHttpGetIn">
        <wsdl:part name="symbol" type="s:string" />
```

```
</wsdl:message>
<wsdl:message name="GetQuoteHttpGetOut">
    <wsdl:part name="Body" element="tns:string" />
</wsdl:message>
<wsdl:message name="GetQuoteHttpPostIn">
    <wsdl:part name="symbol" type="s:string" />
</wsdl:message>
<wsdl:message name="GetQuoteHttpPostOut">
    <wsdl:part name="Body" element="tns:string" />
</wsdl:message>
<wsdl:portType name="StockQuoteSoap">
    <wsdl:operation name="GetQuote">
        <wsdl:documentation
            xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Get Stock quote for a
company Symbol
        </wsdl:documentation>
        <wsdl:input message="tns:GetQuoteSoapIn" />
        <wsdl:output message="tns:GetQuoteSoapOut" />
    </wsdl:operation>
</wsdl:portType>
<wsdl:portType name="StockQuoteHttpGet">
    <wsdl:operation name="GetQuote">
        <wsdl:documentation
            xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Get Stock quote for a
company Symbol
        </wsdl:documentation>
        <wsdl:input message="tns:GetQuoteHttpGetIn" />
        <wsdl:output message="tns:GetQuoteHttpGetOut" />
    </wsdl:operation>
</wsdl:portType>
<wsdl:portType name="StockQuoteHttpPost">
    <wsdl:operation name="GetQuote">
        <wsdl:documentation
            xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Get Stock
quote for a company Symbol
        </wsdl:documentation>
        <wsdl:input message="tns:GetQuoteHttpPostIn" />
        <wsdl:output message="tns:GetQuoteHttpPostOut" />
    </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="StockQuoteSoap" type="tns:StockQuoteSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="GetQuote">
        <soap:operation soapAction="http://www.webserviceX.NET/GetQuote"
            style="document" />
        <wsdl:input>
            <soap:body use="literal" />
        </wsdl:input>
        <wsdl:output>
```

```
                <soap:body use="literal" />
            </wsdl:output>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:binding name="StockQuoteSoap12" type="tns:StockQuoteSoap">
        <soap12:binding transport="http://schemas.xmlsoap.org/soap/http" />
        <wsdl:operation name="GetQuote">
            <soap12:operation soapAction="http://www.webserviceX.NET/GetQuote"
                    style="document" />
            <wsdl:input>
                <soap12:body use="literal" />
            </wsdl:input>
            <wsdl:output>
                <soap12:body use="literal" />
            </wsdl:output>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:binding name="StockQuoteHttpGet" type="tns:StockQuoteHttpGet">
        <http:binding verb="GET" />
        <wsdl:operation name="GetQuote">
            <http:operation location="/GetQuote" />
            <wsdl:input>
                <http:urlEncoded />
            </wsdl:input>
            <wsdl:output>
                <mime:mimeXml part="Body" />
            </wsdl:output>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:binding name="StockQuoteHttpPost"
        type="tns:StockQuoteHttpPost">
        <http:binding verb="POST" />
        <wsdl:operation name="GetQuote">
            <http:operation location="/GetQuote" />
            <wsdl:input>
                <mime:content type="application/x-www-form-urlencoded" />
            </wsdl:input>
            <wsdl:output>
                <mime:mimeXml part="Body" />
            </wsdl:output>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="StockQuote">
        <wsdl:port name="StockQuoteSoap" binding="tns:StockQuoteSoap">
            <soap:address location="http://www.webservicex.net/stockquote.asmx" />
        </wsdl:port>
        <wsdl:port name="StockQuoteSoap12"
            binding="tns:StockQuoteSoap12">
            <soap12:address location="http://www.webservicex.net/stockquote.asmx" />
```

```
            </wsdl:port>
            <wsdl:port name="StockQuoteHttpGet"
                binding="tns:StockQuoteHttpGet">
                <http:address location="http://www.webservicex.net/stockquote.asmx" />
            </wsdl:port>
            <wsdl:port name="StockQuoteHttpPost"
                binding="tns:StockQuoteHttpPost">
                <http:address location="http://www.webservicex.net/stockquote.asmx" />
            </wsdl:port>
        </wsdl:service>
    </wsdl:definitions>
```

# Document Change Log

This section describes the major changes made with each version of this document.

| Version | Date | Description |
|---|---|---|
| 1.0 | Dec 2009 | Original version |
| 1.1 | Jul 2010 | Added address parameter to `SAWS.Run/SAWS.RunSecure`<br>Added `SAWS.DEBUG` |
| 1.2 | Aug 2010 | Added `SAWS.CallSOAP`<br>Added additional modes to `SAWS.DEBUG` |
| 1.3 | Sep 2010 | Removed `SAWS.CallSOAP` and modified `SAWS.Call`<br>Added additional mode to `SAWS.TRACE`<br>Added certificate parameter to `SAWS.Call` |
| 1.4 | Mar 2011 | Updated areas addressing secure web services to be compatible with Conga v2.1 |