

Squeaky Clean — Black-box Methods with SOAP

Presented by: **Lincoln Stoller. Braided Matrix. Inc.**



INTRODUCTION

This session is for developers interested in creating and publishing their own web services. It recommends designs for creating web services.

WEB SERVICE STANDARDS

Web services involve a web service Server and web service Client communicating over a LAN, or a WAN such as the internet. The client and server can be written in different languages, run under different operating systems, and utilize different hardware. The benefit of the Web Service standard is that the differences between the client and the server languages, OS's, and hardware are irrelevant.

All of this will be built into the web service standard when it is complete. Currently, only basic standards exist, so services generally do not provide all that is promised.

WEB SERVICE REQUIREMENTS

Web services involve 4 components. 3 are built in to 4D, and two of these are automated. I expect that the other two will be integrated and automated in 4D in the near future.

XML

The lingua franca of web services. All communication of, about, and between web services is conveyed in XML. 4D can “speak” XML. At present 4D provides developers with rudimentary XML tools.

SOAP

The core web service message protocol that is largely invisible to a developer. 4D creates SOAP “portals” that give you easy access to SOAP messaging. These are the proxy methods created by the 4D SOAP Wizard. Learn the SOAP Wizard and the proxy methods that it creates in order to gain greater control over the client/server process.

WSDL

4D automates the publishing of the WSDL for the web service that you create. Of the two available web service message styles 4D fully automates the RPC (Remote Procedure Call) style. The DOC (Document) style is not automated and requires the programmer to encode and decode XML-formatted information.

Support for the RPC style is a legacy feature of Web Services. It is widely used, but its potential and its future are limited. Building 4D services in DOC style will remain difficult until powerful XML parsing tools become available.

UDDI

4D web services are not currently published to UDDI directories (other directories exist as well). Most likely publishing indexes will be managed outside of the development environment. The extra information needed to support directories, such as UDDI, will be added to the WSDL once UDDI standards are in place. Most likely 4D will support this and automatically make your services visible to UDDI services.

NOT CLIENT/SERVER

A 4D Server or a copy of 4D stand-alone can function as a web server. The web service client may either be a stand-alone application, a client, a server, or any other piece of web-enabled software (even another web services server).

Web services are not constructed on the client/server database model, and there is no “client/server” relationship between the web client and the web server in the database sense. The way the Web Client and Web Server function should not be confused with the way the 4D Client and 4D Server function.

Web services are better described as a means of inter-application communication. Think of it as a messaging system with a language, a protocol, and a suite of supporting discovery and indexing services. At a programming level RPC-based web services have much in common with 4D Open.

MIDDLE-WARE

A 4D web server acts as “middle-ware”. This means that the web service is built using one application development environment (4D) to provide a “derived” application development environment (a web service). Or to put it another way, it is a program built in a programming environment to provide services to another programming environment (the web services client).

There are three defining features of middle-ware

- There is no graphical user interface.
- The middle-ware interface is an application programmer interface (API).
- The only services available to the client are those that you provide.

This is a departure from 4D’s use to create user-based applications using the form designer, or 4D’s use in a collaborative system that employs 4D’s Connectivity modules (4D Open, ODBC, 4D Oracle, etc.).

Comparison with 4D Open

SOAP and 4D Open can be used to create similar API’s, but there are large differences in what can be done.

	4D Open	SOAP
Procedure Based	Yes	Optional (RPC available now but likely to become obsolete)
Extensible	No	Yes
4D-to-4D Communication	Yes	Yes
Open Standard	No	Yes
Web Support	No	Yes
Compact Data	Yes	No, not in Unicode text form.
Interruptible	Yes, if Execute on Server No, if using 4D Open calls	No
Persistent Connection	Yes	No
Connection Protocol	Yes, proprietary	Yes, public
Embedded in a Language	Yes	No

Data Structure Syntax	No	Yes (XML)
Synchronous / Asynchronous	Asynchronous	Synchronous (4D does not support Asynchronous, but other environments do.)
Automatic Publishing	No	Yes, at both description and discovery levels.
Security	High, centrally managed	Variable, few standards, problematic
Cost to Deploy in 4D	5 connections take one server seat.	Requires Web Services license.

From a coding point of view 4D's SOAP acts similarly to 4D Open's Execute On Server command. It provides a means for parameter passing, error reporting, and will (hopefully) provide a means for managing complex XML data.

Designing middle-ware is designing a programming environment. You determine what features you are going to provide, rather than having the user (or client) decide, as is the case in most 4D development projects. The services you provide are designed to provide a complete "toolbox".

TYPES OF INTER-APPLICATION SERVICES

Simple

Simple services are generally look-up services where little data is passed and little returned. In addition, the structure of the data that is passed and returned is simple.

Simple services can be designed on a task-by-task basis, free from client input.

Examples of Simple Web Services:

Stock Quotes, Weather, Document Serving

Database

Custom database application service provides remote application support for selected database operations.

Database services may involve:

- Rules that are unique and are designed for a particular project.
- Protocols, parameters, and actions that are known only to internal developers.
- Private access.
- Optimization for a particular environment.

Examples of Database Services:

Custom 4D applications, connectivity for enterprise applications, synchronizing with a mobile database.

Complex

Complex web services can be thought of as a collaboration between two or more applications, each having their own rules and each possibly storing their own data. By "collaboration" I mean either *sharing data*, or having some *effect upon the data* stored by either the web client or the web server.

Complex services may involve:

- Database services (Insert, Modify, Delete)
- Submitting, receiving, or returning complex data.
- Close collaboration: multi-stage, conditional, or synchronized processing.
- High Security: identification, verification, authorization, and encryption.

A complex service usually requires that the server provide the client with a complete toolbox of services.

Examples of Complex Services:

Open connectivity for enterprise applications, B2B services order entry, delivery management.

SERVICE-ORIENTED ARCHITECTURE

The Web Services environment provides a basic level of service -oriented structure:

- 1) It exposes basic logic through the WSDL, supporting more complex rules through the DTD that can accompany XML data. However, it provides no guarantee that the services are logical, correct, or even authentic.
- 2) A registry where the service discloses itself to clients.

Service Oriented Architectures refer to, in general, a comprehensive design supporting a full suite of related services, what might be called a “complete toolbox”.

WHAT IS A COMPLETE TOOLBOX?

All web services are supported in the WSDL and conform to basic protocol. That does not mean the service is “complete”.

A complete service provides 3 additional levels of service necessary for its being utilized from a mission critical Web Service client.

Disclosure of Business Logic

The service must be able to communicate its business rules with the client in an automated fashion. The client must be able to verify that the server’s rules for data and processing conform to the client’s expectations.

Rules For Data

XML provides the syntax for describing data structure; the DTD (Data Type Definitions) provides data description (allowable values, rules that apply “locally”)

Rules For Processing

Reference to published standards (e.g. Order processing rules, which are not machine-readable) and requirements that are machine-readable (eg. Requiring the value of every customer ID >0).

Example: Provide a means for client to get a directory of services, service requirements, and business rules. (WSDL may need to be supplemented.)

Data Services

Basic Operations: these include the usual Insert, Modify, and Delete.

Logical Model: since structure should be hidden (it must be according to the standards), services are provided for the *Logical Structure*, not the *Physical Structure*.

Monitoring and Control

In addition to control of information, a toolbox provides for administration of the API itself. These are “meta tasks” for managing the data processing functions.

Response: Since the client may not be able to wait for tasks to complete on the server, the server should provide a means for rapid response.















- Examples:** immediately returning a code that does any of the following:
- verifies communication (handled automatically by SOAP),
 - verifies the data and input parameters (handled partially by SOAP),
 - returns a task ID, or provisional data (only the connection ID is handled by SOAP).

Update: If tasks cannot be carried out immediately, then the server must provide the client with information regarding the progress of the tasks.

- Examples:** provide a method to the client that:
- indicates the status of an on-going task (not handled by SOAP),
 - indicates the status of the server (not handled by SOAP).

Control: Since the logic on the client side may require a change in processing in the case of an extended task, the server must provide the client with a means for control of an ongoing (not yet complete) task.

- Examples:** provide a method to the client that:
- returns partially processed or provisional data (not handled by SOAP),
 - enables client to terminate and rollback on-going process (not handled by SOAP).

Monitoring & Control		in 4D Open	in SOAP
Response	Verifies Communication		
	Verifies Data		
	Returns Task ID		
Update	Returns Task Status		
	Returns Server Status		
Control	Preliminary Data		
	Task Roll-back		

 Handled automatically
  Handled partially
  Requires programming
  Not supported

DESIGN EXAMPLE: 4TH QUARTER’S API

In 2001, Braided Matrix, Inc. implemented a complex Service Oriented Architecture for 4th Quarter using 4D Open. The service was proprietary, local to the 4D network, written in 4D and supported entirely through 4D Open calls. This service has been rewritten in 4D v2003 using SOAP calls, the SOAP Wizard, and ITK.

API Disclosure and Open Standards**SOAP WSDL:**

Lists calls and parameters, complex data is embedded in BLOB's in proprietary format.

SOAP Call:

Data description: lists arrays and their elements required as input parameters.

Developer Documentation:

Explains required format, protocol and standards used in the calls.

Tasks**Invoices, Customers, Accounts, Transactions:**

Logical operations handled transparently. Relationships are input, data created, links updated.

Insert, Modify, and Delete: The following series of are executed and processing of the client request halts if processing fails at any step.

- 1) Check request syntax,
- 2) Check relational integrity (e.g. That the referenced data is in the data base.),
- 3) Operation is attempted.

Control Tasks**Retrieve Status:**

Locate current status of task referenced by Task ID.

Halt & Delete Task:

Halt processing of incomplete task. Remove from task queue. (This is accomplished by starting a web process that interacts with the previously started web service that is still processing the original request.)

Audit

Standard GUI in 4th Quarter for administrator to review submitted tasks. Each task is stored in a record that contains the date, time, user, task name, input data, and the processing outcome.

Review Submitted Tasks:

See tasks submitted by date, type, user, outcome, etc.

Resubmit Task:

Return a task to the queue for processing if its syntax and data are OK but it had not been processed. Handles the case where the condition that prevented original processing has been rectified (inventory now available, customer account paid off, records now available).

Delete Task:

Remove audit history of tasks and their processing outcomes.

Query, Sort & Print Task Reports:

Examine and create reports of past activity.

Batch Tasks: 2 Listed, 0 Selected

All Batch Tasks A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 0-9 Other Clear Bt20 ?

Search... Focus Sort Title Ascending

Attn	Task ID	Tabl	Rec.ID	Submit Date	Submit Time	Last Processed	Action	Err#	Title	Active
	2101	61	5520	5/12/03	16:35	5/12/03	Customer_New	0	New Customer ID=5520	
	2103	61	5531	5/12/03	16:51	5/12/03	Customer_New	0	New Customer ID=5531	

Run Batch Export BLOBs Import BLOBs

4th Quarter's Batch Task Administration screen. Records correspond to commands sent to the server.

CODE EXAMPLE: 4TH QUARTER'S API

Server Side

A single controlling method on the server directs all services through the use of calls to other 4Q modules. Information passed through SOAP parameters is assigned to new records. Record processing is handled by the same methods that handle records entered through 4Q's standard graphical user interface. Outcome messages are passed back to the SOAP client.

All web service tasks are handled automatically by 4D. We need only declare the SOAP variables, and start the web service. This methods used a BLOB of a proprietary and variable internal structure to carry information to the server and back to the client. The method operates under SOAP using RPC messaging style.

```

$CallerName:="SOAP_4QSubmitTask"      `method name
` Declare variables
C_BLOB (SOAP_oDataBLOB;vBT1Blob;$pDataBLOB)
C_TEXT (SOAP_tCallerIdentity;SOAP_t4QTRTask;SOAP_tResultMessage)
C_LONGINT (SOAP_LBatchRecordNum)

`Declare parameters
SOAP DECLARATION (SOAP_tCallerIdentity;Is Text ;SOAP Input ;
  "SOAP_tCallerIdentity")
SOAP DECLARATION (SOAP_t4QTRTask;Is Text ;SOAP Input ;
  "SOAP_t4QTRTask")
SOAP DECLARATION (SOAP_oDataBLOB;Is BLOB ;SOAP Input ;
  "SOAP_oDataBLOB")
SOAP DECLARATION (SOAP_LbatchRecordNum;Is LongInt ;SOAP Input ;
  "SOAP_LBatchRecordNum")
SOAP DECLARATION (SOAP_tResultMessage;Is Text ;SOAP Output ;
  "SOAP_tResultMessage")
SOAP DECLARATION (vBT1Blob;Is BLOB ;SOAP Output ;"vBT1Blob")
$BatchAction:=SOAP_t4QTRTask
$SOAPCaller:=SOAP_tCallerIdentity
$pDataBLOB:=->SOAP_oDataBLOB

$Err:=__SY ProceSInit ($CallerName;$BatchAction)
<>Tab:=Char (Tab_)

ARRAY TEXT (vylBTText;0)
ARRAY TEXT (vy2BTText;0)
$CUSTOMER:=(Position ("CUSTOMER";$BatchAction)>0)

```

```

$New_Customer:=(Position("_NEW";$BatchAction)>0)
  \Test for other actions.
  \...

Case of
: ($Err <0)
: ($CUSTOMER)
  Case of
  : ($New_Customer)
  $Err:=_CSAPISubmitCustomer ($CallerName;$BatchAction;
    ->v2Longint;->vylBTText;->SOAP_tResultMessage;
    ->vOutsideControl;$pDataBlob->;$SOAPCaller)
  If (($Err =0) & (v2Longint>0))
    SOAP_tResultMessage:=String(v2Longint)+<>Tab
    +SOAP_tResultMessage
  End if
  \Handle other customer actions.
  \...

End case
\Handle actions on other tables.
\...

End case

If (SOAP_tResultMessage="")
  Case of
  : ($ErrorText#"")
  SOAP_tResultMessage:=$ErrorText

  : (Records in selection([Batch_Task])>0)
  $TaskDate:=String([Batch_Task]DateProcessed;Short )
  $TaskTime:=String([Batch_Task]TimeProcessed;HH MM SS )
  $TaskID:=String([Batch_Task]Task_ID)
  SOAP_tResultMessage:=String($Err)+<>Tab+"TaskID="+$TaskID
    +<>Tab +$TaskDate+<>Tab +$TaskTime

  Else
  SOAP_tResultMessage:=String($Err)+<>Tab+
    String(Current date(*))+<>Tab +String(Current time(*))
  End case
End if

If ($Err<0)
  SEND SOAP_FAULT(SOAP Client Fault ;String($Err)+" "
    +SOAP_tResultMessage)
Else
  $Err:=_BTHdlBatchData ($CallerName;"SET_ERROR_CODE/"+$BatchAction;
    ->vErrorCode;->vylBTText;->SOAP_tResultMessage;$pDataBLOB->;
    $SOAPCaller)
End if

SET BLOB SIZE($pDataBLOB->;0)
_SYUNLoadAll ($CallerName;"")

\end of method

```

Client Side

In the following client methods an application prepares data used to create a new 4Q client record. 4D's SOAP Wizard created the proxy method that passes the information to the web server. Some error handling code was added to the proxy method to test the passed parameters and to install an On Event Call method, which should be done in all cases.

```

`method WebSvc_Create4QClient
`SOAP client method to create new 4Q Customer
`adapted from a method written by Kent Wilbur
C_POINTER($pointerBlob)
C_BLOB(FQTR_Descriptions)
ARRAY_TEXT(arrayCustomerInfo;0)
ARRAY_TEXT(arrayCustomerFormat;0)
$LOffset:=0
$ReturnMessage:=""
$Err:=0

SET BLOB SIZE(FQTR_Descriptions;0)
$tMessage:=SOAP_FQTR_HandleRecords ("GetCustomer_Description";
->FQTR_Descriptions)
If (BLOB size(FQTR_Descriptions)>0)
  BLOB TO VARIABLE(FQTR_Descriptions; arrayCustomerFormat)
  SET BLOB SIZE(FQTR_Descriptions; 0)

  $Err:=FQTR_FillCustomerBLOB(->arrayCustomerInfo ;
->arrayCustomerFormat)

  VARIABLE TO BLOB(arrayCustomerInfo; $pointerBlob->)

  If ($Err=0)
    $UserInfo:=Current User+";"+Current Machine
    $Task:="CUSTOMER_NEW"
    $pointerBlob:=->FQTR_Descriptions
    $Err:=proxy_4QsubmitTask($UserInfo; $Task;
      $pointerBlob; $ReturnMessage)
  End If
End If

```

The proxy method created by 4D has been modified by the insertion of a pair of On Err Call calls. These ensure that SOAP errors are handled correctly in the event that the soap server reports an error.

```

`proxy_4QsubmitTask: proxy method to call 4Q's SOAP service
C_TEXT($0)
C_TEXT($1)
C_TEXT($2)
C_POINTER($3)
C_LONGINT($4)
C_BLOB(proxy_Blob)
SET BLOB SIZE(proxy_Blob;0)

SET WEB SERVICE PARAMETER("SOAP_tCallerIdentity";$1)
SET WEB SERVICE PARAMETER("SOAP_t4QTRTask";$2)
If (Count parameters>2)
  proxy_oBlob:=$3->
  SET WEB SERVICE PARAMETER("SOAP_oDataBLOB";proxy_Blob)

```

```

If (Count parameters>3)
    SET WEB SERVICE PARAMETER("SOAP_LBatchRecordNum";$4)
End if
End if

SOAP_LErrorNumber:=0
`Stops the aborts on a connection error
ON ERR CALL("SOAP_HandleConnectionError")

CALL WEB SERVICE(<>SOAP_tFQTRSoapServer+"/4DSOAP/";
    "A_4DInc_WebService#SOAP_4QSubmitTask";"SOAP_4QsubmitTask";
    "http://SOAPServices.4d.com/SOAPServices";Web Service Dynamic)

ON ERR CALL("")

If (OK=1)
    GET WEB SERVICE RESULT($0;"SOAP_tResultMessage")
    If (Count parameters>2)
        `Memory clean-up on the final return value.
        GET WEB SERVICE RESULT(proxy_Blob;"vBT1Blob";*)
        $3->:=proxy_Blob

        SET BLOB SIZE(proxy_Blob;0)
    End if
End if

```

```

`SOAP_HandleConnectionError: method to handle SOAP error
`written by Kent Wilbur
C_LONGINT($LPosition)
C_TEXT($tErrorMessage)
C_LONGINT(SOAP_LErrorNumber)

$tErrorMessage:=Get Web Service error info(0)
Case of
: ($tErrorMessage="9910")
    $tErrorMessage:=Get Web Service error info(1)
    If ($tErrorMessage≤1≥="-")
        $LPosition:=Position("-";Substring($tErrorMessage;2))
        SOAP_LErrorNumber:=Num(Substring($tErrorMessage;1;$LPosition-1))
        SOAP_tErrorMessage:=Substring($tErrorMessage;$LPosition+3)
    Else
        SOAP_LErrorNumber:=Num($tErrorMessage)
        SOAP_tErrorMessage:=$tErrorMessage
    End if

: ($tErrorMessage="9912")
    $tErrorMessage:=Get Web Service error info(1)
    ALERT($tErrorMessage)

: ($tErrorMessage="9913") `Connection error
    $tErrorMessage:=Get Web Service error info(1)
    SOAP_LErrorNumber:=Num($tErrorMessage)
    SOAP_tErrorMessage:=$tErrorMessage

Else
    SOAP_LErrorNumber:=Num($tErrorMessage)
End case

```

BENEFITS**Opportunity**

The 4Q Web Services API's, easily transformed from the earlier Service Oriented Design implemented in 4D Open, have opened inter-application communication between 4th Quarter and the universe of web-enabled clients.

Utility

A complete API toolkit ensures 4th Quarter can support client applications with complex requirements.

Stability

Through the use of a single SOAP-manager method, the addition of new services only requires new branching statements in the server's controlling method.

SUMMARY

The SOAP architecture is a generally usable means of enabling applications to communicate. It includes automatic description and discovery of the service and the message structure. However this comes with less flexibility, and fewer programming tools than were available with 4D Open and other proprietary connectivity tools.

Web services provide a set of standards that requires replacing traditional end-user coding practices for Service Oriented designs more appropriate for middle-ware.

Web services promise a universally supported environment for linking data and coordinating operations between any number of systems located anywhere.