26. Web Service Publishing

LabWare 8 enables LIMS Functions and LIMS Subroutines to be exposed as web services. The web services published by LIMS are dynamic, and are defined by LIMS configuration. Once deployed, these web services may be consumed by any remote application, written in a language of the user's choice.

Built in Web Services are provided to log-into and log-out of LIMS (services *Authenticate* and *Close*), and to list and optionally re-create the enabled web functions and web subroutines (services *List Functions* and *List Routines*). Web Functions and Web Subroutines are wrappers of LIMS Basic Functions and LIMS Subroutines for use in the web. Each web function and web subroutine can be deployed as a web service.

LabWare supports the SOAP 1.1 and SOAP 1.2 specifications. In addition, WS-Security is supported (using the UsernameToken profile), allowing the user to invoke a web service with a single call.

Web functions are pre-defined for the LIMS Basic Functions in base LIMS, except those that are not relevant for the web environment. For security reasons, only those web functions that do not access the database are exposed as web services by default.

This web service mechanism allows total flexibility of the functionality that can be made available, as the capabilities are determined by LIMS Basic functions in user-written LIMS subroutines.

26.1 Using Web Services

Web Services are consumed by a remote application either with or without WS-Security and single sign-on . WS-Security is enabled and disabled as an application server environment variable (refer to the Web Reference chapter or the sytem install manual for more on configuring these variables. Single-sign on is configured on the CM Service that the remote application will be sending the request.

NOTE: The WS-Security setting is global for a LIMS installation, effective for all LIMS Services and Datasources that use the same JEE deployment In extreme cases, you could envisage two deployments, one with, and one without. It is off by default.

26.1.1 Without WS-Security:

The communication is a three step process:

1. The calling application authenticates with LIMS by providing a user name and password and LIMS returns an authentication token. This is then used to authenticate subsequent calls. This is also known as "opening a session."
2. The desired web service(s) is/are then invoked with the token
3. The web session is closed, known as "closing the session".

From a LIMS point of view, the authentication step logs the named user into LIMS, who remains logged in until the web session is closed or until it times out due to session inactivity. The session timeout period is set in the Web Configuration table.

TIP: For frequent or continuous web service use, it is more efficient to obtain an authentication token once, and then consume web services using this token for as long as desired. The token will remain valid (and the user will remain logged in) until the *close* service is invoked or the session times out due to session inactivity.

26.1.2 With WS-Security:

There is only one step to this process:-

- The application invokes a web service using a single message that includes the authentication details and LIMS responds with the answer.

In this case, with a single web service call, the user is logged into LIMS, the desired web service is invoked, and the user is automatically logged out.

LabWare Web Services support the UserNameToken Profile of WS-Security. For details on how to invoke LabWare LIMS Web Services using WS-Security, refer to the section on Using WS-Security.

This method is typically more efficient when a single web service call is made per session, or for infrequent web service calls. For frequent or continuous web service use, there is an overhead involved in authenticating and closing the web service for each call.

26.2 Deployed Web Services

The built in web services are Authentication, List Functions, List Routines, and Close. The WSDL documents for these services are available at the following URL's:

http://<server_address:port>/LabWare-8/services/labware_weblims_authenticate?wsdl
http://<server_address:port>/LabWare-8/services/labware_weblims_listFunctions?wsdl
http://<server_address:port>/LabWare-8/services/labware_weblims_listRoutines?wsdl
http://<server_address:port>/LabWare-8/services/labware_weblims_close?wsdl

For any deployed Web Function or Web Subroutine, the WSDL is available at

http://<server_address:port>/LabWare-8/services/function_name?wsdl

For example, for the ABS Web Function (which invokes the ABS LIMS Basic function),

http://<server_address:port>/LabWare-8/services/ABS?wsdl

where <server_address:port> is the address and port number (the port number is 80 if port is omitted) of the Java EE application server.

NOTE: Throughout this manual, the URL addresses assume the LabWare-8 war has been deployed with its default name LabWare-8. If the war is deployed with a different name, replace LabWare-8 with the deployed name.

NOTE: Throughout this manual, a SOAP 1.1 format is used for the examples. The SOAP 1.2 specification is also supported.

TIP: The examples in the following sections may be easier to understand if the xml portion is copied and pasted into notepad, then saved as a file with extension .xml, which may then be opened in a browser or suitable editor.

NOTE: Tags must be specified in the order shown here, which matches the order in the WSDLs.

26.2.1 Authentication

Authentication of a session is required before invoking any other service. The Authentication service is called with mandatory and optional parameters via one of two possible operations, *authenticate* or *authenticateWithRole*. For the authenticate operation, the parameters are a valid LIMS Username, Password, LIMS Datasource and LIMS Service. The LIMS User must have the *WebServices* function privilege. For the *authenticateWithRole* operation, the parameters are a valid LIMS Username, Password, Role, LIMS Datasource and LIMS Service. In this case, the Role must have the *WebServices* function privilege.

The LIMS Service parameter is optional and may be null or omitted altogether. In this case, the Default Service, as defined in the Web Configuration record, is used.

If authentication is successful, an authentication token (session token) is returned. This token is then used for all other services in this session. If unsuccessful, an error is returned.

NOTE: If WS-Security or Single Sign-On is enabled, the Authentication service is not used.

A SOAP Authentication request for a user without roles is as follows. The ***placeholders*** need to be replaced with actual values.

```
POST /LabWare-8/services/labware_weblims_authenticate HTTP/1.1
SOAPAction:
Host: <server_address>
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="UTF-8"?><soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<soapenv:Body>
            <authenticate soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        xmlns="labware_weblims_authenticate">
              <username>username</username>
              <password>password</password>
              <limsDSName>DataSource </limsDSName>
              <limsServiceName>Service</limsServiceName>
            </authenticate>
        </soapenv:Body>
</soapenv:Envelope>
```

The SOAP response for a successful authentication request is as follows. The ***placeholders*** are replaced by actual values. There may be additional headers, depending on the Java EE application server.

```
HTTP/1.1 200 OK
Content-Type: text/xml;charset=utf-8
Transfer-Encoding: chunked
Date: Date

length
<?xml version='1.0' encoding='utf-8'?><soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
        <soapenv:Body>
        <ns:authenticateResponse xmlns:ns="labware_weblims_authenticate">
            <ns:return>TokenString</ns:return>
        </ns:authenticateResponse>
        </soapenv:Body>
</soapenv:Envelope>
0
```

SOAP Authentication request for user with roles is similar, with the SOAP body as follows:

```
<soapenv:Body>
   <authenticateWithRole soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns="labware_weblims_authenticate">
      <username>username</username>
      <password>password</password>
      <role>role</role>
      <limsDSName>DataSource</limsDSName>
      <limsServiceName>LIMSService</limsServiceName>
   </authenticateWithRole>
</soapenv:Body
```

CAUTION: The first Authentication executed after the Java EE application server is started may require a long time to complete (up to 30 seconds), as all the function classes and routine classes are rebuilt as a result of this first request. Subsequent requests will be much quicker

TIP: Create a separate Web Service consume procedure to be run once immediately on restarting the Java EE application server, which simply authenticates with a long timeout period (say 60 seconds), then closes. Then normal authentication calls may be set with shorter timeout periods.

26.2.2 ListFunctions

The *ListFunctions* web service retrieves the list of available LIMS Basic functions, with an option to refresh them. The refresh option ("reload" = true) rebuilds the function classes.

NOTE: The function classes are automatically refreshed on the first Authentication request after the Java EE application server is restarted.

A SOAP *ListFunctions* request is as follows. The ***placeholders*** need to be replaced with actual values.

```
POST /LabWare-8/services/labware_weblims_listFunctions HTTP/1.1
SOAPAction:
Host: <server_address>
Content-Type: application/soap+xml; charset=utf-
Content-Length: length

<?xml version="1.0" encoding="UTF-8"?><soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><soapenv:Body>
   <listFunctions soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" xmlns="labware_weblims_listFunctions">
      <authToken>TokenString</authToken>
      <reload>true_or_false</reload>
   </listFunctions></soapenv:Body></soapenv:Envelope>
```

If WS-Security is enabled, the namespace definition in this request is replaced by
xmlns="http://www.labware.com/webservices"

and the modifications described in the Using WS-Security section are required.

The SOAP *ListFunctions* response contains details of all the web functions. As installed, this return string is too long to reproduce here (it would take over 100 pages). The structure is the same as the *ListRoutines* return string (refer to the example response in *ListRoutines* for details).

26.2.3 ListRoutines

The *ListRoutines* web service retrieves the list of available LIMS Basic subroutines, with an option to refresh them. The refresh option ("reload" = true) rebuilds the routine class files.

NOTE: The routine classes are automatically refreshed on the first Authentication request after the Java EE application server is restarted.

A SOAP *ListRoutines* request is as follows. The ***placeholders*** need to be replaced with actual values.

```
POST /LabWare-8/services/labware_weblims_listRoutines HTTP/1.1
SOAPAction:
Host: <server_address>
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="UTF-8"?><soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><soapenv:Body>
   <listRoutines soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" xmlns="labware_weblims_listRoutines">
      <authToken>TokenString</authToken>
      <reload>true_or_false</reload>
   </listRoutines></soapenv:Body></soapenv:Envelope>
```

If WS-Security is enabled, the namespace definition in this request is replaced by
xmlns="http://www.labware.com/webservices"

and the modifications described in the Using WS-Security section are required.

An example of a SOAP *ListRoutines* response follows, with web subroutine ***variables*** highlighted. This example illustrates the response with two Web Subroutines defined, as follows:

1. LSAM, which invokes a LIMS Basic subroutine LOGSAM, and returns an integer called *sampleNumber*. It has 3 arguments, *templateName* (a required string), *FieldsArray* (an optional array), and *ValuesArray* (an optional array).
2. A_2DARRAY, which invokes a LIMS Basic subroutine A_2DARRAY and returns a 2-dimensional array called *anArray*. It has one required argument, *arsize*, an integer with a default value of 1.

There may be additional headers, depending on the Java EE application server

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: text/xml;charset=UTF-8
Transfer-Encoding: chunked
Date: Date

a54
<?xml version='1.0' encoding='utf-8'?><soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"><soapenv:Body><ns:listRoutinesResponse
xmlns:ns="labware_weblims_listRoutines" xmlns:ax25="http://ws.web.labware/xsd"
xmlns:ax26="http://connector.labware/xsd"><ns:return type="labware.web.ws.WebFunction"><ax25:args
```

type="labware.connector.Argument"><ax26:defaultValue ***xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:nil***="true" /><ax26:description>***Login Template***</ax26:description><ax26:name>***templateName***</ax26:name><ax26:optional>***false***</ax26:optional><ax26:type>***STRING***</ax26:type></ax25:args><ax25:args type="labware.connector.Argument"><ax26:defaultValue ***xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:nil="true"*** /><ax26:description>***fields***</ax26:description><ax26:name>***FieldsArray***</ax26:name><ax26:optional>***true***</ax26:optional><ax26:type>***ARRAY***</ax26:type></ax25:args><ax25:args type="labware.connector.Argument"><ax26:defaultValue ***xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:nil="true"*** /><ax26:description>***values***</ax26:description><ax26:name>***ValuesArray***</ax26:name><ax26:optional>***true***</ax26:optional><ax26:type>***ARRAY***</ax26:type></ax25:args><ax25:description ***xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:nil="true"*** /><ax25:functionName>***LOGSAM***</ax25:functionName><ax25:limsReturnType>***INTEGER***</ax25:limsReturnType><ax25:name>***LSAM***</ax25:name><ax25:numArguments>***3***</ax25:numArguments><ax25:retVal xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:nil="true" /><ax25:returnDescription>***sample number***</ax25:returnDescription><ax25:returnType>***java.lang.Long***</ax25:returnType><ax25:returnVariable>***sampleNumber***</ax25:returnVariable></ns:return><ns:return type="labware.web.ws.WebFunction"><ax25:args type="labware.connector.Argument"><ax26:defaultValue>***1***</ax26:defaultValue><ax26:description>***Enter 1 to get 3x2 example, 2 to get 2x3 example, 3 to get 3x3 example***</ax26:description><ax26:name>***arsize***</ax26:name><ax26:optional>***false***</ax26:optional><ax26:type>***INTEGER***</ax26:type></ax25:args><ax25:description>***Examples illustrating return of a 2 dimensional array***</ax25:description><ax25:functionName>***A_2DARRAY***</ax25:functionName><ax25:limsReturnType>***2D_ARRAY***</ax25:limsReturnType><ax25:name>***A_2DARRAY***</ax25:name><ax25:numArguments>***1***</ax25:numArguments><ax25:retVal xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:nil="true" /><ax25:returnDescription>***Example of 2-D array***</ax25:returnDescription><ax25:returnType>***labware.connector.Array2D***</ax25:returnType><ax25:returnVariable>***anArray***</ax25:returnVariable></ns:return></ns:listRoutinesResponse></soapenv:Body></soapenv:Envelope>
0

26.2.4 Close

The *Close* web service closes the web session. LIMS releases the authentication token and performs a logout for the user.

A SOAP *Close* request is as follows. The ***placeholders*** need to be replaced with actual values.

```
POST /LabWare-8/services/labware_weblims_close HTTP/1.1
SOAPAction:
Host: <server_address>
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="UTF-8"?><soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><soapenv:Body>
   <close soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" xmlns="labware_weblims_close">
      <authToken>TokenString</authToken>
   </close></soapenv:Body></soapenv:Envelope>
```

The SOAP *Close* response is as follows, for a successful request. There may be additional headers, depending on the Java EE application server.

```
HTTP/1.1 200 OK
Content-Type: text/xml;charset=utf-8
Transfer-Encoding: chunked
Date: Date

103
<?xml version='1.0' encoding='utf-8'?><soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"><soapenv:Body><ns:closeResponse
xmlns:ns="labware_weblims_close"><ns:return>true</ns:return></ns:closeResponse></soapenv:Body></soapenv:Envelope>
0
```

26.2.5 Web Functions and Web Subroutines

There is a web service for each entry in the Web Function and Web Subroutine tables for which the Private flag is set to false.

- Web Function services invoke LIMS Basic functions. The base system is delivered with approximately 700 Web Functions. For security reasons, only the 200 or so web functions that do not access the database have the Private flag set to false and are thus exposed as web services.
- Web Subroutine services invoke LIMS subroutines. These are defined as part of the LIMS configuration. There are no Web Subroutines in the base system.

For Web Functions and Web Subroutines, the Service Path is LabWare-8/services/NAME, where NAME is the Web Function or Web Subroutine name in uppercase. In all cases, the request operation (SOAPAction) is "invoke".

CAUTION: Web Subroutines must not have the same name as any Web Function. Unpredictable results may occur if a Web Subroutine and a Web Function have the same name.

NOTE: 1) The name of the invoke operation can be modified if necessary (for example, if consuming Web Services using a language for which "invoke" is a reserved word). The operation is specified in the tag <ws-operation-name> in the file connector-config.xml.
2) The namespace http://www.labware.com/webservice can be modified to be unique for each function and routine if necessary. Change the value of the tag <append-servicename-in-namespace> in the file connector-config.xml from false to true to append the function or routine name to the namespace definition. The connector-config.xml file is in the subdirectory webapps\LabWare-8\WEB-INF\classes of the Java EE application server. The application server must be stopped and restarted for any change to take effect.

A SOAP request for the LSAM Web Subroutine example in the *ListRoutines* section is as follows. The ***placeholders*** need to be replaced with actual values. In this example, the Sample Template is SAM_TEMPLATE, and the two fields FIELD_TEXT and FIELD_NUMBER are populated with "example of Web Service" and 3.1416. This example was generated by consuming the Web Service from LIMS, using LIMS arrays.

```
POST /LabWare-8/services/LSAM HTTP/1.1
Content-Type: text/xml; charset=utf-8
SOAPAction: invoke
Content-Length: length
Host: <server_address>

<?xml version="1.0" encoding="UTF-8"?><soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><soapenv:Body>
   <ns:LSAM xmlns:ns="http://www.labware.com/webservice">
      <authToken> TokenString </authToken>
      <templateName>SAM_TEMPLATE</templateName>
      <FieldsArray>OrderedDictionary(&apos;FIELD_TEXT&apos; &apos;FIELD_NUMBER&apos; )</FieldsArray>
      <ValuesArray>OrderedDictionary(&apos;example of Web Service&apos; &apos;3.1416&apos; )</ValuesArray>
   </ns:LSAM></soapenv:Body></soapenv:Envelope>
```

Alternatively, the array tags (*FieldsArray* and *ValuesArray*) in this example could be replaced by the following. Note repeated tag names indicate multiple array elements.

```
<FieldsArrayelem>FIELD_TEXT</FieldsArrayelem>
<FieldsArrayelem>FIELD_NUMBER</FieldsArrayelem>
<ValuesArrayelem>example of Web Service</ValuesArrayelem>
```

```
<ValuesArrayelem>3.1416</ValuesArrayelem>
```

The SOAP response is as follows for a successful call to this LSAM Web Subroutine. The ***placeholders*** are replaced by actual values. There may be additional headers, depending on the Java EE application server.

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: text/xml;charset=UTF-8
Transfer-Encoding: chunked
Date: Date

length
<?xml version='1.0' encoding='UTF-8'?><soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"><soapenv:Body><ns:invokeResponse
xmlns:ns="http://www.labware.com/webservice"><ns:return>Sample_Number</ns:return></ns:invokeResponse></soapenv:Body>
</soapenv:Envelope>
0
```

If the Web Subroutine returns an error, the SOAP response is as follows. The ***placeholders*** are replaced by actual values. There may be additional headers, depending on the Java EE application server.

```
HTTP/1.1 500 Internal Server Error
Server: Apache-Coyote/1.1
Content-Type: text/xml;charset=UTF-8
Transfer-Encoding: chunked
Date: Date
Connection: close

length
<?xml version='1.0' encoding='utf-8'?><soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"><soapenv:Body><soapenv:Fault><faultcode>soapenv:Server</faultc
ode><faultstring>Error while invoking routine LSAM: error details</faultstring><detail
/></soapenv:Fault></soapenv:Body></soapenv:Envelope>
0
```

If the error is a LIMS error (with a value for the variable *lasterror*), the ***error details*** string contains the contents of *lasterror*.

26.2.5.1 Using WS-Security

If WS-Security is enabled, the *Authenticate* and *Close* Web Services are not used. HTTPS is required unless the LabWare debug environment variable has been enabled (refer to the Web Reference chapter for more information on Environment Variable Configuration). If not defined, the Test Web Services app will return the error "HTTP transport is not permitted for WS-Security. Please contact administrator." and SOAP requests will return a faultstring "Expected transport is 'https' but income transport found: 'http'." The SOAP requests described in the Web Functions and Web Subroutines section of this manual are modified as follows:

A SOAP header must be added, containing the username and password parameters, following the WS UsernameToken profile, and a timestamp element (NOTE: if the LABWARE_WS_SEC_INCL_TIMSESTAMP environment variable is defined as FALSE, the timestamp element is not required). The ***placeholders*** need to be replaced with actual values. For users with roles, the "u***sername***" value must be username,role. The token ID "ExampleTS" and "ExampleUN" can have any value.

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:web="http://www.labware.com/webservice"><soapenv:Header><wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"><wsu:Timestamp wsu:Id="**ExampleTS**"><wsu:Created>**DateTimeCreated**</wsu:Created><wsu:Expires>**DateTimeExpires**</wsu:Expires></wsu:Timestamp><wsse:UsernameToken wsu:Id="ExampleUN"><wsse:Username>**username**</wsse:Username><wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#PasswordText">**password**</wsse:Password></wsse:UsernameToken></wsse:Security></soapenv:Header>

The LIMS Datasource and LIMS Service may be specified as arguments of the HTTP POST command. The ***placeholders*** need to be replaced with actual values.

POST /LabWare-8/services/***NAME***?lims_datasource=***Datasource***&lims_service=***Service*** HTTP/1.1

If not specified, the default Datasource and LIMS Service, as configured in the LIMS Admin Console, are used. If specified, both must be present.

26.2.6 Invoking Web Services by URL

Web Functions and Web Subroutines may be invoked by URL reference, provided the Call web service option in the Web Configuration record is enabled.

The general format is as follows (THE ***placeholders*** need to be replaced with actual values):

http://***<server_address>***/LabWare-8/call-web-service.htm?lims_datasource=***datasource***&lims_service=***service***&functionname=***FN{&parameters}***

and

http://***<server_address>***/LabWare-8/call-web-service.htm?lims_datasource=***datasource***&lims_service=***service***&routinename=***RTN{&parameters}***

The names call-web-service, lims_datasource, lims_service, functionname, and routinename are case-sensitive and must all be lower case. The parameter lims_service is optional; if omitted, the default service is used. Parameters must be in the order defined for the Web Function or Web Subroutine (thus empty parameters may be required). Arrays must be enclosed in square brackets [ ]. Numbers in arrays that are to be treated as a string must be put into quotes. For example, ["4"] will be treated as a string, and [4] will be treated as a number.

NOTE: URL calls do not support users with Roles.

The following examples illustrate the requirements for URL calls (assuming a local Java EE application server, e.g. Tomcat, on port 8080, a DataSource named LABWAREDB, with the default service).

Web Functions:

http://localhost:8080/LabWare-8/call-web-service.htm?lims_datasource=LABWAREDB&functionname=ave&array=[6,3,2]

http://localhost:8080/LabWare-8/call-web-service.htm?lims_datasource=LABWAREDB&functionname=log_sample&templatename=SAM_TMP&fieldsArray=[DESCRIPTION,PRODUCT]&valuesArray=[logged+from+URL,LOG_PROD]

(This example assumes the web function log_sample has been made private=false. The + signs in the DESCRIPTION field are because spaces are always encoded in URL's)

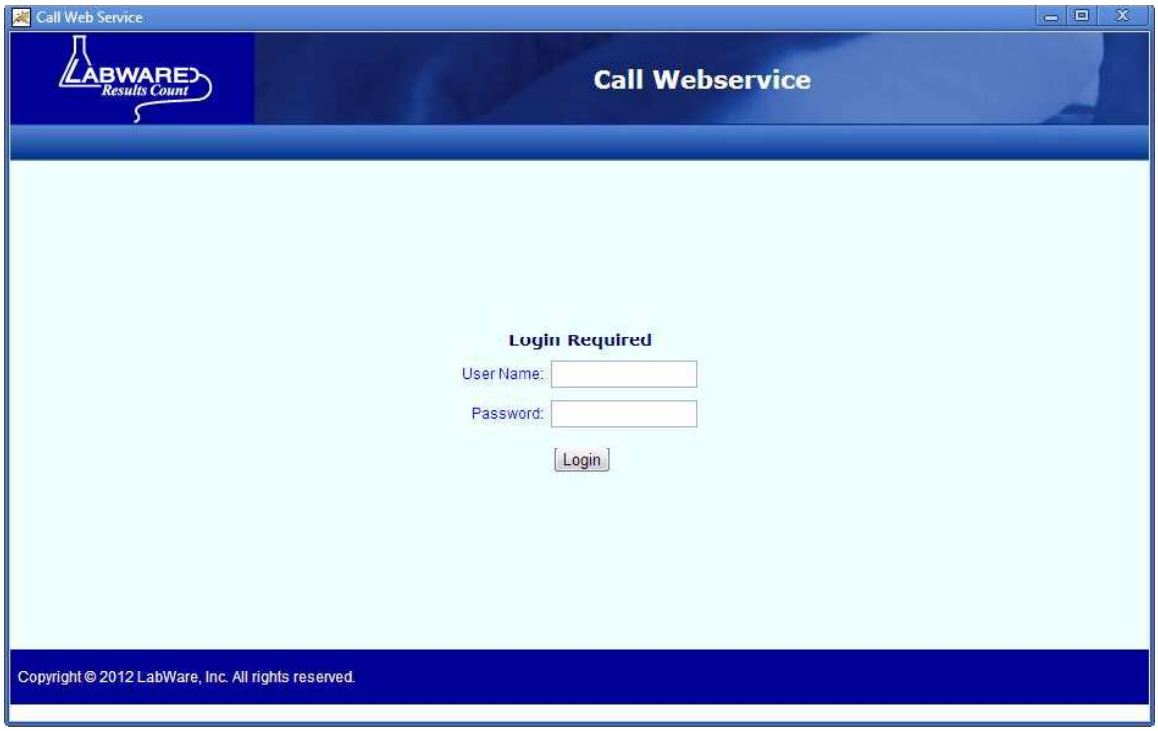
Web Routines:

http://localhost:8080/LabWare-8/call-web-service.htm?lims_datasource=LABWAREDB&routinename=A_2DARRAY&arsize=3

http://localhost:8080/LabWare-8/call-web-service.htm?lims_datasource=LABWAREDB&routinename=WebSub1

When a web service is called from a URL, an authentication screen is displayed.

Figure 26-1 Authentication Screen



On successful authentication, the specified web service is invoked, with the specified arguments, and the web session is then closed.

The Remember me option will be available if this functionality is enabled for the requested datasource. If the Remember me option is checked, then further calls to the same LABWARE datasource invoke the web service directly, bypassing the login page.

NOTE: The Remember me function works by saving a browser cookie containing an encrypted string with the datasource name, username and password. If this cookie is deleted, or a different browser is used, then the authentication screen will be displayed on the next call.

26.2.7 Invoking a Web Subroutine from a PDF form

PDF forms may be used to invoke Web Subroutines, provided the PDF Submit option in the Web Configuration record has been enabled. Web Subroutines which are to be invoked from PDF forms must be defined with two array arguments, a fields array and a values array. On successful completion, the return variable will be returned to the program that submits the form.

26.2.7.1 PDF form requirements

The PDF form must contain the following fields (the names are case-sensitive) in addition to data fields:

username

password

role (this field may be hidden and left blank, if roles not used)

lims_datasource

LIMSservice (optional; if omitted, the default service will be invoked)

websubroutinename

The form must have an element, for example a Submit button, defined so that the Action **Submit a form** will be taken when requested (for example, on Mouse Up, i.e., when the button is clicked). This Submit Form Action must be configured as follows:

Link to the URL	http://<server_address:port>/LabWare-8/pdfsubmit.jsp
Export format	HTML.
Field Selection	as desired, must include the fields listed above.

26.2.7.2 PDF Form Submit

When the Submit button (or similar) is clicked, the Web Subroutine in *websubroutinename* will be invoked, with the fields and values as arguments. The fields such as username, password, etc are not submitted to the web subroutine. The return variable will be returned to the program that submits the form.

For example, a form normally opened in a web browser may be defined to invoke a web subroutine that returns an HTML string. Then the HTML string will be returned to the browser on submission of the form.

26.3 Return Types and Argument Types

The return types and argument types for a web function or web routine can be one of the following: String, Number, Integer, Boolean, Date, DateTime, Time (return type only), One-Dimensional Array, Two-Dimensional Array, Any, and One-Way. All types except the arrays, Any, and One-Way, type are returned as single values. The format of the types is the same for returns and for arguments.

26.3.1 Date, DateTime, and Time types

Date types are in the format YYYY-MM-DD. DateTime types are in the format YYYY-MM-DDThh:mm:ss.sss, with no Time Zone indication (for example 2016-01-31T13:29:35.000). Time retrun types are in the format HH:mm:ss.sss.

26.3.2 Array types

One-Dimensional Arrays are a set of values matching the contents of the array. Two-Dimensional Arrays consist of a column length, a row length, followed by values by row (eg row1 column1, row1 column2, row1 column3, row2 column1, etc).

26.3.3 Any type

The type Any is used if the type is not known in advance (for example, the return type of the web function RESULT_FLD, or the argument types of the web function IS_EQUAL). The Any type consists of two Booleans “array” and “array2D”, followed by a single value for all non-Array types, or a set of values if the “array” Boolean is true. For return type Any, Two-Dimensional Arrays are also supported. In this case, Boolean values false for array and true for array2D are followed by values in the Two-Dimensional Array format.

26.3.4 One-Way type

The type of One-Way is used if the request should not return any response content. An HTTP 202 Accepted response will be returned to inform the remote application that the request has been accepted for processing.

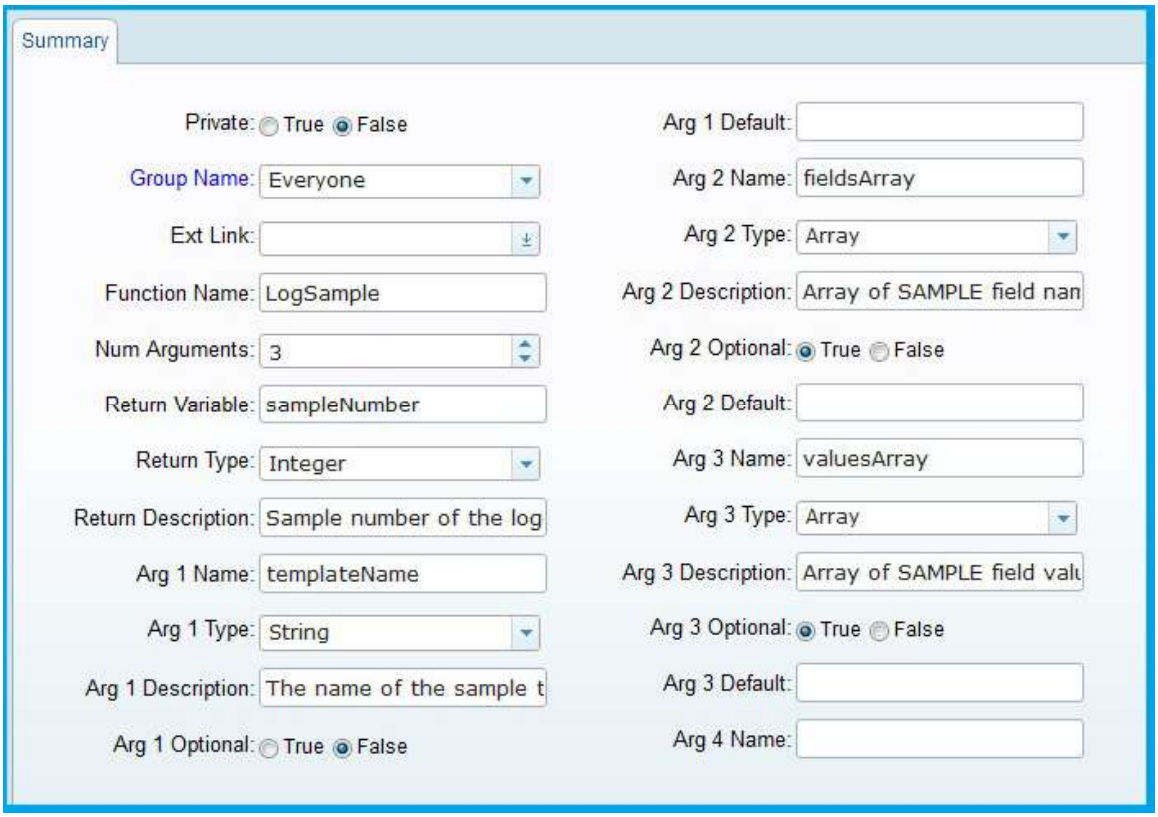
26.3.5 Files as String Arguments

String Argument type which are files, for example, External Link values, have a special format. The input string must have have the format file^filename^base64 encoded contents of file. For example, to input a file named data.txt with contents equal to “file contents”, the input string would be file^data.txt^ZmlsZSBjb250ZW50cw==.

26.4 Web Function Table

The Web Function table is used to expose LIMS Basic functions to the web.

Figure 26-2 Web Functions table



The *Private* field is used to determine if the function is to be visible on the web. The function will be enabled on the web only if *Private* is False.

The *Group Name* field is used to define to which security group the Web Function belongs. The *Group Name* field will display a list of valid group names.

The *Ext Link* field is used to link the Web Function to an external file. The *Ext Link* field will display a file browser.

The *Function Name* field is used to specify the name of the LIMS Basic function to be invoked by this Web Function.

The *Num Arguments* field is used to define the number of arguments of the function.

The *Return Variable* is used to define the name of the variable returned by the function.

The *Return Type* is used to define the data type of the *Return Variable*. The *Return Type* field will display a list of valid return types.

The *Return Description* is used to define a description of the *Return Variable*.

The *Arg 1 Name* field is used to define the name of the first argument of the function.

The *Arg 1 Type* field is used to define the data type of the first argument. The *Arg 1 Type* field will display a list of valid argument types.

The *Arg 1 Description* field is used to define a description of the first argument.

The *Arg 1 Optional* field is used to specify if the first argument is optional. True indicates that the argument is optional, and false indicates it is mandatory.

The *Arg 1 Default* field is used to specify a default value for the first argument.

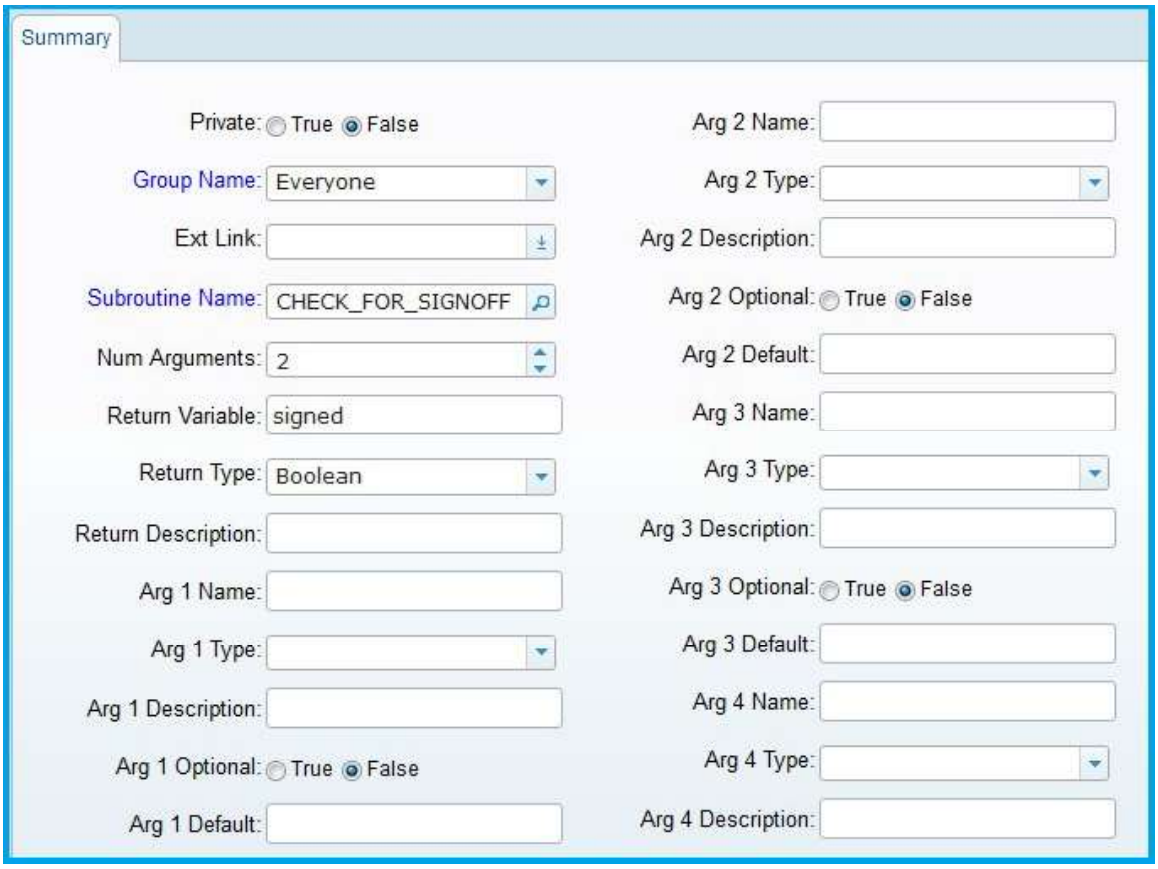
In like manner, the set of five *Arg 2* fields are used for the second argument, and so on, to the number of arguments defined by *Num Arguments*.

NOTE: LIMS supports up to 20 arguments, with 12 available in the base database. If any LIMS Basic functions are added which have more than 12 arguments, five fields need to be added to the WEB_FUNCTION table for each additional argument. For example, for one additional argument, the fields ARG_13_NAME, ARG_13_TYPE, ARG_13_DESCRIPTION, ARG_13_OPTIONAL, and ARG_13_DEFAULT must be added, with the same configuration as the similar fields for other arguments.

26.5 Web Subroutine Table

The Web Subroutine table is used to expose Subroutines to the web.

Figure 26-3 Web Subroutine table



The *Private* field is used to determine if the subroutine is to be visible on the web. The subroutine will be available on the web only if *Private* is False.

The *Group Name* field is used to define to which security group the Web Subroutine belongs. The *Group Name* field will display a list of valid group names.

The *Ext Link* field is used to link the Web Subroutine to an external file. The *Ext Link* field will display a file browser.

The *Subroutine Name* field is used to specify the name of the LIMS Subroutine to be invoked by this Web Subroutine.

The *Num Arguments* field is used to define the number of arguments of the subroutine.

The *Return Variable* is used to define the name of the variable returned by the subroutine.

The *Return Type* is used to define the data type of the *Return Variable*. The *Return Type* field will display a list of valid return types.

The *Return Description* is used to define a description of the *Return Variable*.

The *Arg 1 Name* field is used to define the name of the first argument of the subroutine.

The *Arg 1 Type* field is used to define the data type of the first argument. The *Arg 1 Type* field will display a list of valid argument types.

The *Arg 1 Description* field is used to define a description of the first argument.

The *Arg 1 Optional* field is used to specify if the first argument is optional. True indicates that the argument is optional, and false indicates it is mandatory.

The *Arg 1 Default* field is used to specify a default value for the first argument.

In like manner, the set of five *Arg 2* fields are used for the second argument, and so on, to the number of arguments defined by *Num Arguments*.

NOTE: LIMS supports up to 20 arguments, with 10 available in the base database. If any LIMS Basic subroutines with more than 10 arguments are to be invoked, five fields need to be added to the WEB_ROUTINE table for each additional argument. For example, for an 11th argument, the fields ARG_11_NAME, ARG_11_TYPE, ARG_11_DESCRIPTION, ARG_11_OPTIONAL, and ARG_11_DEFAULT must be added, with the same configuration as the similar fields for other arguments.

27. Web Service Consume

Web Service Consume may be used to issue HTTP POST or HTTP GET messages, using the *InvokeWebService* LIMS Basic function. The function specifies the contents (e.g., SOAP Body, XML message, etc) of the message. The type and parameters of the message are determined by entries in the Web Services Table. Refer to the Web Services Table section for details on the configuration parameters.

To consume a web service, the *CreateWebService*, *InvokeWebService*, and *CloseWebService* LIMS Functions are called. A number of LIMS Basic functions are provided to facilitate building XML messages, such as functions to automatically create XML tags and tag values, and functions to escape and unescape characters in XML strings. Refer to the Examples section below.

Additionally, the Libcurl library has been exposed via CurlEasy LIMS Basic functions, which gives full control of the HTTP request message structure. Refer to the Libcurl Support section in the Managing LIMS Basic chapter for more information on using these functions.

27.1 Message Types

All message types except GET generate HTTP POST messages, with different content types and message formatting.

27.1.1 SOAP 1.1

For SOAP 1.1 messages, the content type is specified as `text/xml`, with `charset=utf-8`. An HTTP SOAP Action header is always included. Namespaces *`xsd`* and *`xsi`*, as well as the ws-addressing namespace *`wsa`* are defined, so the SOAP Envelope is as follows.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:wsa="http://www.w3.org/2005/08/addressing"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
```

A SOAP Header may be specified, if required.

27.1.2 SOAP 1.2

For SOAP 1.2 messages, the content type is specified as `application/soap+xml`, with `charset=utf-8`. An action parameter is included, if specified. The namespace *`xs`*, and the ws-addressing namespace *`wsa`* are defined, so that the SOAP Envelope is as follows:

```
<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope"
xmlns:xs="http://www.w3.org/2001/XMLSchema"> xmlns:wsa="http://www.w3.org/2005/08/addressing"
```

A SOAP Header may be specified, if required.

27.1.3 XML

For XML messages, the content type is specified as `text/xml`, with `charset=utf-8`. The only message formatting included is the XML declaration `<?xml version="1.0" encoding="UTF-8"?>`.

27.1.4 Multipart (Multipart/form-data)

For Multipart messages, the content type is specified as `multipart/form-data`, with a boundary value as specified. No messaging formatting is included.

27.1.5 GET

For GET messages, an HTTP Get command is issued. If a message is present, it is included as the URL query string (i.e, prepended with ?).

27.1.6 JSON

For JSON messages, the content type is specified as `application/json`, with `charset=utf-8`. No message formatting is included.

27.2 Authentication

Windows Authentication of Web Service messages is supported, either using the Windows user where the LIMS instance is running, or using a specified Username and Password.

The use of Client Certificates (e.g., X.509 certificates) is also supported.

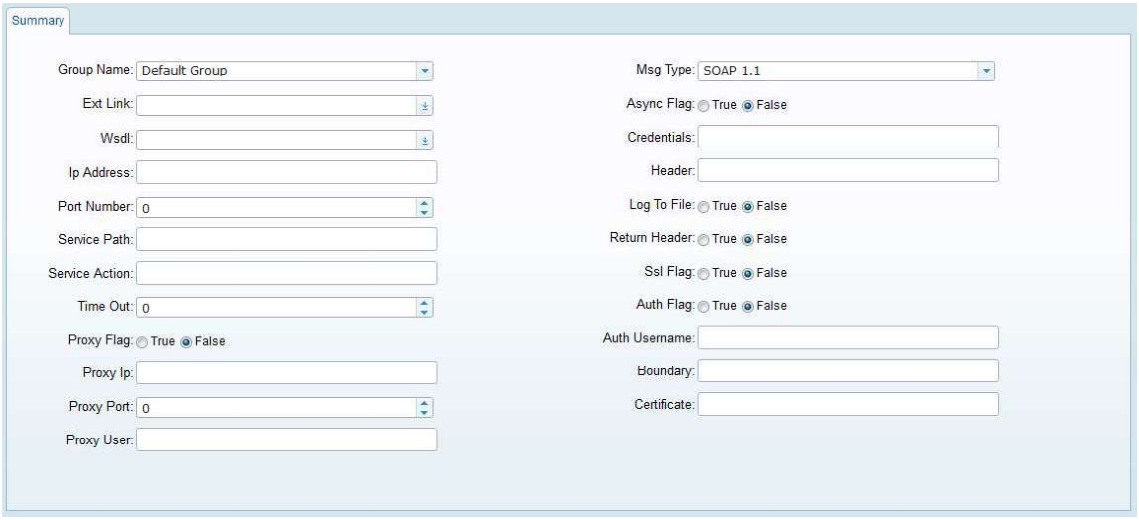
In addition, specification of the HTTP Authorization Header is supported, typically for HTTP Basic Authentication.

27.3 Other Message Features

Security	Plain text and SSL transport mechanisms are supported.
Proxy Server	Web Service calls may be made via a proxy server. If needed, HTTP Basic Authentication may be specified for the proxy server.
Logging	Log files containing the outgoing and incoming messages may be created, if desired.
No Wait for Reply	An option is provided to not wait for a reply to a Web Service call.
Static or Dynamic Parameters	Some parameters (Windows authentication parameters, SOAP Header, Service Path, and Service Action) may be defined statically as database field values or dynamically as function parameters.

27.4 Web Services Table

Figure 25-1 Web Services Table Fields



The *Group Name* field is used to define the security group that the Web Service belongs to. The *Group Name* field will display a list of valid group names.

The *Ext Link* field is used to link the Web Service to an external file. The *Ext Link* field will display a file browser.

The *Wsdl* field is used to link the Web Service to an external WSDL file, for purposes of documentation. The *Wsdl* field will display a file browser.

The *Ip Address* field is used to specify the IP Address of the Web Service to be consumed.

The *Port Number* field is used to define the Port Number of the Web Service to be consumed. For standard http addresses, the Port Number should be set to 80; for standard https addresses, the Port Number should be set to 443.

The *Service Path* field is used to define the Service Path of the Web Service, as specified by the Web Service Publisher.

The *Service Action* field is used to define the Service Action (SOAP Action), if any, of the Web Service, as specified by the Web Service Publisher. This field is only used for the SOAP message types.

The *Time Out* field is used to define the time out, in seconds, to wait for a response to the Web Service call. If *Time Out* is 0, the default value of 30 seconds is used.

The *Proxy Flag* field is used to indicate if the Web Service is accessed through a Proxy Server. The Web Service is accessed through a Proxy Server if the Proxy Flag radio button is True.

The *Proxy IP* field is used to define the IP Address of the Proxy Server. This field is required if Proxy Flag is True, and is ignored if Proxy Flag is false.

The *Proxy Port* field is used to define the Port Number of the Proxy Server. This field is required if Proxy Flag is True, and is ignored if Proxy Flag is false.

The *Proxy User* field is used to define the User ID for Authentication on the Proxy Server. The password is defined in a separate dialog, see the Passwords section below. This field is ignored if Proxy Flag is false.

The *Msg Type* field is used to define the type of message. The *Msg Type* field will display a list of valid Message Types.

The *Async Flag* field is used to specify if LIMS should continue without waiting for a response after executing a Web Service call. LIMS waits for a response if the *Async Flag* radio button is False. The flag is normally set to False.

The *Credentials* field is used to set a value for HTTP Authorization credentials for the web service, typically Basic Authentication, if needed.

The *Header* field is used to specify a SOAP header for the Web Service call. If blank, no SOAP header is sent. This field is ignored if a SOAP header is specified in the *InvokeWebService* LIMS Basic function call, or if the message type is not one of the SOAP types.

The *Log To File* field is used to specify if a log file is generated. A log file is generated containing the sent and received messages if the *Log To File* radio button is set to True. The log file is named WebService-NAME.log, where NAME is the name of the Web Service, and is saved in the LIMS working directory.

The *Return Header* field is used to specify if the SOAP header from the reply message is to be included in the reply from a Web Service call. The SOAP header is included if the *Return Header* radio button is set to True. This field is ignored if the message type is not one of the SOAP types.

The *Ssl Flag* field is used to specify if the message is going to be sent in plain text, or as an SSL message. SSL is used if the *Ssl Flag* radio button is set to True.

The *Auth Flag* field is used to specify if the message is going to be authenticated using Windows Authentication. Windows Authentication is used if the *Auth Flag* radio button is set to True. If a username or password are specified in the *CreateWebService* LIMS Basic function call, this field is ignored.

The *Auth Username* field is used to specify a user name for Windows Authentication. If this field is left blank, the current Windows user is used for Windows Authentication. The password is defined in a separate dialog. Refer to the Passwords section below. This field is ignored if Auth Flag is false, or if a username or password is specified in the *CreateWebService* LIMS Basic function call.

The *Boundary* field is used to specify a boundary for the Multipart message type. If this field is left blank, the default boundary of ***** is used. This field is ignored for other message types.

The *Certificate* field is used to specify a X.509 certificate, if required. Refer to the section on Client Certificates below for more information.

27.4.1 Passwords

Pressing the **Configure** button displays a Password dialog.

Figure 25-2 Passwords Dialog

The *Auth Password* field is used to define the Password for Windows Authentication. This field is ignored if *Auth Flag* is false, or if a username or password is specified in the *CreateWebService* LIMS Basic function call.

The *Proxy Pw* field is used to define the Password for Authentication on the Proxy Server. This field is ignored if the *Proxy Flag* is set to false.

27.4.2 Client Certificates

If a client certificate (e.g., a X.509 certificate) is required, the *Certificate* field is used. The contents of this field are a set of key:value pairs, used to define the certificate, separated by semicolons. Supported keys, with possible values, are as follows:

Provider:

SYSTEM
MEMORY
FILE
REG
PKCS7
SERIALIZED
FILENAME
MSG
COLLECTION
SYSTEM_REGISTRY
PHYSICAL
SMART_CARD
LDAP

Location:

CURRENT_USER
LOCAL_MACHINE
CURRENT_SERVICE
SERVICES
USERS
CURRENT_USER_GROUP
LOCAL_MACHINE_GROUP
LOCAL_MACHINE_ENTERPRISE

Store:

My
AddressBook
TrustedPeople
TrustedPublisher

etc. (other stores may exist, the above are windows system certificate stores)

Subject:

(as desired)

If a key is not present, then the first value listed is the default. Key names are not case-sensitive, however, values may be. If defaults are used for all keys except Subject, this key may be omitted. Blanks are trimmed from the field contents. The first certificate (in the specified Provider, Location, Storage) that contains the string specified in the Subject key is used.

Custom user certificates are typically stored in the "My" store, either for the user or for the machine. Typical examples of Certificate fields for a certificate containing the string "LabWare" are as follows: (The first four values produce equivalent behavior.)

Provider:SYSTEM;Location:CURRENT_USER;Store:My;Subject:LabWare
Store:My;Subject:LabWare
Subject:LabWare
LabWare

Location:CURRENT_MACHINE;Subject:LabWare

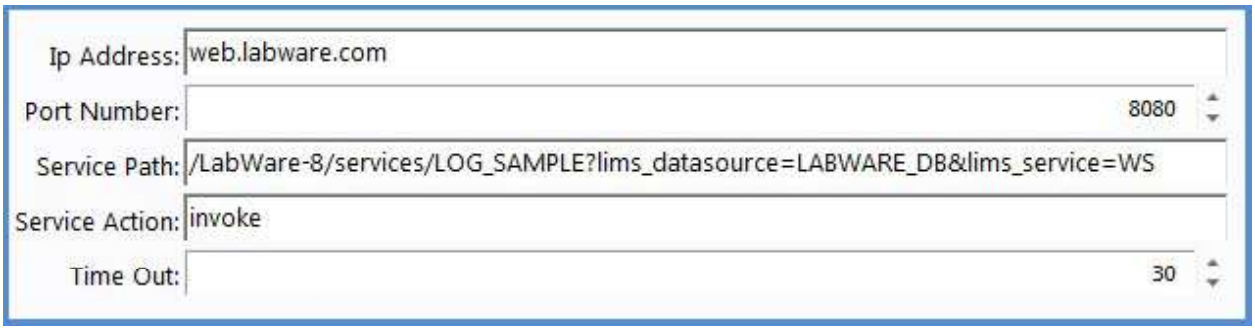
27.5 Using the Web Service Consume - Examples

27.5.1 Consuming LOG_SAMPLE function, as published by LabWare

The following example illustrates invoking the web function LOG_SAMPLE published by another instance of LabWare 8.

An entry (for this example, LW_LSAM) is needed in the Web Service Table for the LOG_SAMPLE web function. This entry, for LabWare 8 at web.labware.com, with datasource LABWARE_DB and LIMS Service WS, is as follows:

Figure 25-3 Invoke LOG_SAMPLE function

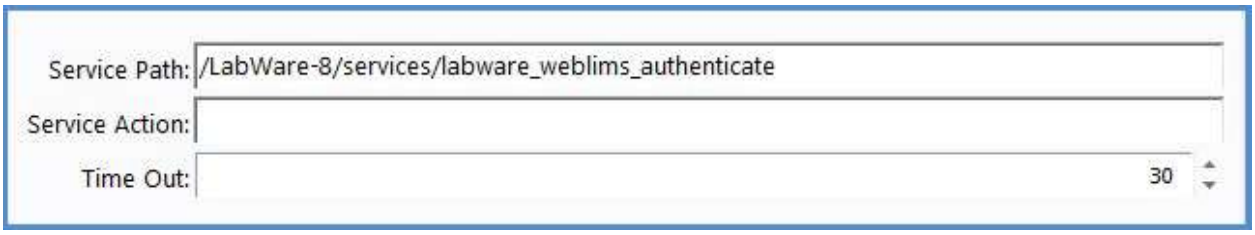


The Msg Type field must be SOAP 1.1 or SOAP 1.2. Proxy fields must be configured if Proxy authentication is required.

NOTE: The complete sequence for consuming the LOG_SAMPLE web service published by LabWare 8 normally involves invoking the Authentication, LOG_SAMPLE, and finally Close web services. Refer to the chapter on Web Service Publishing for more information.

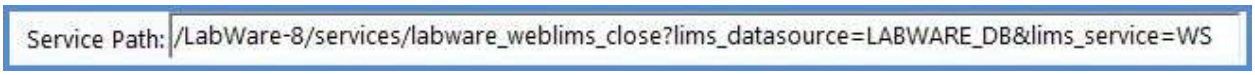
The Web Service table entries for the *Authentication* and *Close* Web Services are similar, with differences as follows.

Figure 25-4 Authentication Web Services



The *datasource* and *service* values are specified in the Authenticate call. Time Out value for Authentication is typically larger to allow for possible redeployment of web services.

Figure 25-6 Close Web Service



A LIMS Basic code snippet to invoke LOG_SAMPLE and return the logged sample number is as follows:

```
' token = value from Authentication
tmplate = "SAM_TEMP"
fldarr[1] = "PRODUCT"
fldarr[2] = "DESCRIPTION"
valarr[1] = "LOG_PROD"
valarr[2] = "Example of log sample via Web Services"

'****Build up XML Message to Send by attaching Required Arguments as Tags which are supplied in the WSDL
handle = XMLCreateMessage()
tagName = "invoke"
props = "xmlns=" + chr(34) + "http://www.labware.com/webservice" + chr(34)
status = XMLAddTagToMessage(handle, tag,props)
status = XMLAddTagValueToMessage(handle, "authToken" ,token)
status = XMLAddTagValueToMessage(handle, "templateNameC", tmplate)
status = XMLAddTagValueToMessage(handle, "fieldsArrayC", fldarr)
status = XMLAddTagValueToMessage(handle, "valuesArrayC", valarr)
status = XMLCloseTagInMessage(handle)
soapMessage = XMLMessageContents(handle)

'***Call the Web Service that has the target end point for the Service
serviceHandle = CreateWebService("LW_LSAM")
serviceResponse = InvokeWebService(serviceHandle, soapMessage)

'***Parse Data
status = XMLParse(PARS2, serviceResponse)
qry = "select ns:return from ns:invokeResponse"
status = XMLQuery(PARS2, qry, "SampArr")
sampleNumber  = SampArr[1,1]
return sampleNumber
```

NOTE: Error checking has not been included in this example.

An example of a HTTP POST command generated by this example follows.

```
POST http://web.labware.com:8080/LabWare-8/services/LOG_SAMPLE?lims_datasource=LABWARE_DB&lims_service=WS
HTTP/1.1
Content-Type: application/soap+xml; charset=utf-8; action=invoke
User-Agent: LabWare LIMS
Connection: Keep-Alive
Content-Length: 676
Host: web.labware.com:8080

<?xml version="1.0" encoding="UTF-8"?><soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:wsa="http://www.w3.org/2005/08/addressing"><soapenv:Body>
   <tag xmlns="http://www.labware.com/webservice">
      <authToken>CLUST_MAN-02_1cba857c16aea8b7dc3e809550749683</authToken>
      <templateNameC>SAM_TEMP</templateNameC>
      <fieldsArrayC>OrderedDictionary(&amp;apos;PRODUCT&amp;apos; &amp;apos;DESCRIPTION&amp;apos; )</fieldsArrayC>
      <valuesArrayC>OrderedDictionary(&amp;apos;LOG_PROD&amp;apos;  &amp;apos;Example of log sample via Web Services&amp;apos;
)</valuesArrayC>
   </tag></soapenv:Body></soapenv:Envelope>
```

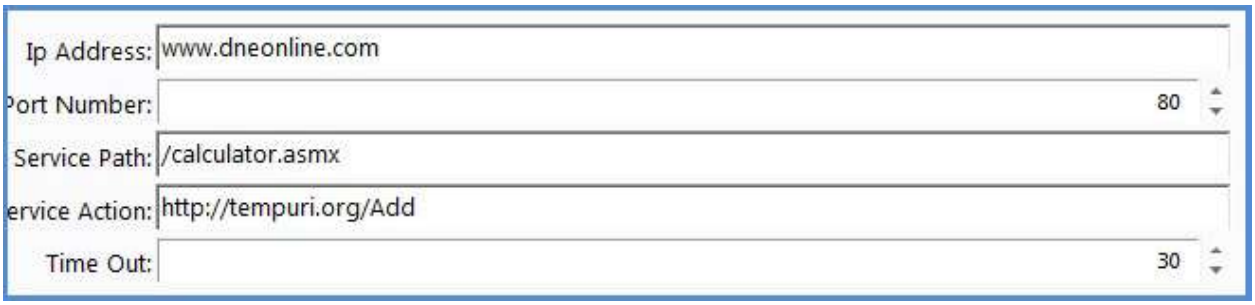
27.5.2 A Public Calculator Web Service

This example illustrates consuming a public web service. It may be used to confirm that the LIMS web service consume is working correctly. This example invokes the web service Add from www.dneonline.com, a free web service which returns mathematical operations for Add, Divide, Multiply, and Subtract. It is defined at http://www.dneonline.com/calculator.asmx?op=Add .

> **NOTE:** This web service works as described as of the writing of this manual. It is recommended to check the definition URL shown here to confirm that the web service is working.

The entry (for this example, ADD) in the Web Service Table for this service is configured as follows:

Figure 25-7 Entries in Web Service Table (Weather)



The Term Msg Type must be SOAP 1.1 or SOAP 1.2. Proxy fields must be configured if Proxy authentication is required.

LIMS Basic code to invoke Add and display the return is as follows.

```
intA = 1
intB = 2
handle = XMLCreateMessage()
propertiesString = "xmlns=" + chr(34) + "http://tempuri.org/" + chr(34)
status = XMLAddTagToMessage(handle, "Add", propertiesString)
status = XMLAddTagValueToMessage(handle, "intA", intA)
status = XMLAddTagValueToMessage(handle, "intB", intB)
status = XMLCloseTagInMessage(handle)
xmlString = XMLMessageContents(handle)

serviceHandle = CreateWebService("ADD")
serviceResponse = InvokeWebService(serviceHandle, xmlString)
status = CloseWebService(serviceHandle)

status = XMLParse("ADDRESPONSE", serviceResponse)
queryString = "select addresult from addresponse"
status = XMLQuery("ADDRESPONSE", queryString, "returnArrayName")
addresult = returnArrayName[1,1]
MsgBox(addresult)
```

> **NOTE:** When successful, the message box will display a SOAP message containing the present weather conditions. Error checking has not been included in this example.