September 22, 2004

Author: **Darin McBeath**

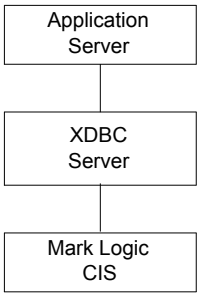
Summary

The goal of the SOAP main module is to provide a simple framework for accessing user defined XQuery functions[1] via SOAP. The SOAP style supported by this main module is Wrapped Document Literal[2]. This unofficial SOAP style, which is really a semantic approach based on Document Literal has become a de facto best practice for achieving interoperability across various platforms.

The SOAP main module is not 100% compliant with the WS-I[3] specification, but should provide sufficient capabilities for a majority of an application's needs. In particular, SOAP Headers, SOAP Header Faults, and SOAP with Attachments are currently not supported. The remainder of this note is not meant to be a tutorial on SOAP but rather highlight some key concepts that should allow the reader to quickly expose their user defined functions as Web Services[4].

The advantage of the SOAP main module approach is that it eliminates the need for an intermediate application server (i.e Tomcat), the requisite Java marshalling and unmarshalling, and the interaction with a Mark Logic XDBC Server. Instead, the client directly interacts with the HTTP Server and directly invokes the appropriate user defined function. The following diagrams should clarify the differences in the two approaches.

A traditional server side implementation of a Web Service (utilizing an application server) would have the following server side flow.



The application server receives the requests and will contain the SOAP runtime, the Java code which serializes and deserializes the XML messages, and the Java code that interacts with the Mark Logic XDBC Server.

The XDBC Server will process the XQuery (essentially executing the user-defined functions).

The Mark Logic CIS engine will be acessed as necessary for content.

This flow can be compared to the server side implementation of a Web Service (utilizing the SOAP main module).

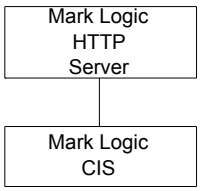
---

[1] A XQuery user defined function is synonymous with an operation defined for a Web Servie.

[2] http://www-106.ibm.com/developerworks/webservices/library/ws-whichwsdl provides a good overview of Wrapped Document Literal.

[3] http://www.ws-i.org

[4] The signature for user defined functions (exposed as Web Services) will need to follow a certain pattern. To expose an existing user defined function as a Web Service, it might be necessary to create a 'wrapper' user defined function which invokes the original function.



The Mark Logic HTTP server receives the requests, determines the appropriate user defined function to invoke, and returns the response..

The Mark Logic CIS engine will be acessed as necessary for content.

Regardless of the server side implementation chosen, this is completely transparent to the client. Of course, one would expect better performance with the latter approach since a bulk of processing has been removed.

SOAPRouter.xqy

This SOAP main module provides the foundation for exposing the user defined functions as Web Services. If one is familiar with the original RPCRouter concept provided by the Apache SOAP implementation, several similarities will be observed. For example, there is a soap-deployment-descriptor which identifies the user defined functions that will be made available via SOAP.

The SOAPRouter.xqy main module should be installed in the root directory for a corresponding Mark Logic HTTP server. For example, setting up a HTTP Server with the name of SOAP would likely be an excellent choice.

There is also the soap-deployment-descriptor which must be loaded into the database accessible from the previously created HTTP Server. The uri for this file should be 'soap-deployment-descriptor' and it should have the following sample structure.[5]

```
<?xml version="1.0" encoding="UTF-8"?>
<soap-deployment-descriptor>
<entry>
<namespace-uri>http://types.mcbeath.com/math/service/v1/</namespace-
uri>
<method-name>addition</method-name>
<database>Math</database>
<module-location>Math/operations</module-location>
</entry>
</soap-deployment-descriptor>
```

The above example contains a single user defined function that would be accessible via SOAP. The URI for the library module containing this user defined function is contained in the namespace-uri element. The function name for the user defined function is contained in the method name. The Mark Logic database where this query should be evaluated is contained in the database element.[6] Lastly, the module-location element defines the location of the library module relative to the root for the previously defined

[5] The schema for the soap-deployment-descriptor is contained in the soap.zip. The specified URI should be used since the SOAPRouter.xqy is dependent on this value when looking for the descriptor in the database.
[6] Ensure that the 'Math' database has been created in Mark Logic. Since there is no content pulled from the database in this example, the SOAP database could also be used. This will eliminate the need for creating the Math database.

HTTP Server. So, if the root for the HTTP Server is SOAP, then the above library module would be located at SOAP/Math/operations.

Some of the interesting bits of code in the SOAPRouter.xqy main module deal with using try/catch blocks, the error function, and the Mark Logic defined exception element. These are used throughout the code to assist with the determination for whether an HTTP Fault or a SOAP Fault should be returned to the client. There is also a programmatic attempt at validation of the SOAP request since Mark Logic does not currently support the validate function. Other than that, the XQuery code is really quite straightforward and fairly well documented.

One limitation with the current approach is that the signature for the user defined function is required to be of type string (in fact the quoted version of the SOAP Body immediate child will be passed to a user-defined function). It is then the responsibility of the function to parse the string and extract the various elements and values. This is essentially a limitation of the current xdmp:eval family of functions provided by Mark Logic. It's possible that this restriction could be alleviated by using the external variable feature provided with the xdmp:eval-in, but this has not thoroughly been investigated.

A User Defined Function

The following user defined function is a very simple example of a function that could be exposed as a Web Service. Note how the URI for the library module and the function name are identified in the previously discussed soap-deployment-descriptor. Also, based on the above descriptor, the file containing the library module would be named operations contained in the SOAP/Math directory[7].

```
module "http://types.mcbeath.com/math/service/v1/"

declare namespace math="http://types.mcbeath.com/math/service/v1/"

default function namespace="http://www.w3.org/2003/05/xpath-functions"

define function math:addition($payload as xs:string) as element() {

      let $parms := xdmp:unquote($payload)
      let $parm1 := xs:integer(data($parms/math:operand1))
      let $parm2 := xs:integer(data($parms/math:operand2))

      return

            <math:result>{$parm1 + $parm2}</math:result>

}
```

But, to allow this function to be easily accessed as a Web Service, a supporting WSDL file must be created. The following set of files (the WSLD was separated into multiple

[7] It should also be possible to store the library modules and the SOAPRouter.xqy in the modules database associated with the SOAP Http server. This would require some minimal changes to the SOAPRouter.xqy.

files) were created for this user defined function and could be easily consumed by a Java client (using WSDL2Java) to access the user defined function[8]. One important note is that the message payload described for the operation does not match the signature for the user defined function. Remember, the user defined function is defined with a type string for the payload. However, for the clients of the Web Service, they should use the structure truly expected by the function (not the current string workaround).

This is the WSDL 'types' schema file that defines the various elements and types used by the Web Service.

```
<?xml version = "1.0" encoding = "UTF-8"?>
<xsd:schema xmlns:xsd = "http://www.w3.org/2001/XMLSchema"
            targetNamespace = "http://types.mcbeath.com/math/service/v1/"
            xmlns = "http://types.mcbeath.com/math/service/v1/"
            elementFormDefault="qualified"
                        attributeFormDefault="unqualified">

        <!-- The addition operation input payload -->
    <xsd:element name="addition">
                <xsd:complexType>
                        <xsd:sequence>
                                <xsd:element name="operand1" type="operandType" />
                                <xsd:element name="operand2" type="operandType"  />
                        </xsd:sequence>
                </xsd:complexType>
        </xsd:element>

        <!-- The addition operation output payload -->
        <xsd:element name="additionResponse">
                <xsd:complexType>
                        <xsd:sequence>
                                <xsd:element name="result" type="operandType"  />
                        </xsd:sequence>
                </xsd:complexType>
        </xsd:element>

        <xsd:simpleType name="operandType">
                        <xsd:restriction base="xsd:integer"/>
        </xsd:simpleType>

</xsd:schema>
```

This is the WSDL 'bindings' file which defines the various messages and encoding style.

```
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
        targetNamespace="http://bindings.mcbeath.com/math/service/v1/"
        xmlns:tns="http://bindings.mcbeath.com/math/service/v1/"
        xmlns:types="http://types.mcbeath.com/math/service/v1/">
        <types>
                <xsd:import namespace="http://types.mcbeath.com/math/service/v1/"
schemaLocation="./mathtypes.xsd"/>
        </types>

        <message name="additionRequest">
                <part name="payload" element="types:addition"/>
        </message>
        <message name="additionResponse">
                <part name="payload" element="types:additionResponse"/>
```

[8] Relevant files have been included in a soap.zip to assist with experimentation.

```
        </message>

        <portType name="MathServicePortType">
                <operation name="addition">
                        <input name="additionRequest" message="tns:additionRequest"/>
                        <output name="additionResponse" message="tns:additionResponse"/>
                </operation>
        </portType>

        <binding name="MathServiceSoapBinding" type="tns:MathServicePortType">
                <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http/"/>
                <operation name="addition">
                        <soap:operation soapAction=""/>
                        <input>
                                <soap:body parts="payload" use="literal"/>
                        </input>
                        <output>
                                <soap:body parts="payload" use="literal"/>
                        </output>
                </operation>
        </binding>

</definitions>
```

Lastly, this is the WSDL service definition that actually ties the Web Service to a specific endpoint. The SOAP end point for all user defined functions will be the SOAPRouter.xqy … in the example below, this implies that this script is accessible on port 8020 for the respective machine. So, remember to use the appropriate domain and port for the SOAP Mark Logic HTTP server.

```
<definitions name="MathService"
        xmlns="http://schemas.xmlsoap.org/wsdl/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
        targetNamespace="http://wsdls.mcbeath.com/math/service/v1/"
        xmlns:tns="http://wsdls.mcbeath.com/math/service/v1/"
        xmlns:bind="http://bindings.mcbeath.com/math/service/v1/" >

        <import namespace="http://bindings.mcbeath.com/math/service/v1/"
location="./mathbindings.wsdl"/>

        <service name="MathService">
                <port name="MathPort" binding="bind:MathServiceSoapBinding">
                        <soap:address location="http://localhost:8020/SoapRouter.xqy"/>
                </port>
        </service>

</definitions>
```

Sample SOAP Payloads

The following is the SOAP request sent from a client to be processed by the SOAPRouter.xqy script.

```
<?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>
  <addition xmlns="http://types.mcbeath.com/math/service/v1/">
   <operand1>1</operand1>
   <operand2>2</operand2>
```

```
  </addition>
 </soapenv:Body>
</soapenv:Envelope>
```

The following is the SOAP response returned from the SOAPRouter.xqy script after invoking the appropriate user defined function.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <additionResponse
xmlns="http://types.mcbeath.com/math/service/v1/"><result>3</result></additionResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Other Thoughts

The SOAPRouter main module is only the beginning of many tasks that could be done to expose user defined functions as web services. An Admin Tool could easily be created to control the deployment of web services and the population of values in the soap-deployment-descriptor. Access control could be incorporated in the SOAPRouter to restrict access to specific user defined functions. A tool similar to WSDL2Java could be developed that would generate skeletal XQuery code based on a WSDL (or generate the WSDL from a user defined function). The opportunities are really endless.

Begin with the sample user defined function (and supporting files and documentation) contained in soap.zip and begin having fun. Keep in mind that the code is dependent on functionality provided with the Mark Logic CIS 2.2.1 release.