Paper 174-28

Extending SAS® Data Services via XML and Java™

Scott E. Chapal, Ichauway, Inc., Newton, GA

ABSTRACT

Designing for interoperability requires the choice of key technologies for the software components of a Web Service. SAS provides very powerful data management and analysis capabilities which can be enhanced and extended in a distributed service-oriented architecture. As XML has become the de facto standard for structured data representation on the Web, Java has become the programming language of choice when considerations of portability and device-independence are paramount. The confluence of SAS, Java and XML creates a symbiotic relationship enabling the existence of “Data Services” which are conceptually metadata-driven, analytically capable, and universally accessible. The continued development and emphasis of Java and XML capabilities in SAS underscores the natural alignment of these technologies. SAS Data Services, distributed via XML and Java, will provide interoperability not only for business problems, but also for scientific, governmental, academic and non-profit institutions.

INTRODUCTION

A persistent challenge for information management is the necessity to integrate disparate applications. One of the most problematic issues for integration is choosing (or creating) and enforcing a data transport standard that inter-operates with processing software (like SAS), but is also vendor independent. In recent years, it has become clear that XML serves this unique role of a “universal data” representation layer. The capacity to develop data driven content models in XML schema that are validate-able and which can be interpreted by any capable software, is propelling the success of XML. By understanding that XML is now the de facto data interchange medium, it is a simple shift in perspective to re-factor data management applications as services, with their access mechanisms formally defined in XML.

Although the Web Services idea has been criticized as “just another distributed computing model”, such as CORBA, RMI or DCOM, there appears to be genuine potential for ’loosely coupled’ application integration. Much of this optimism concerns the fact that the Web Services model is based on core Internet technologies for document transport (XML) over standard protocols (like HTTP SMTP, FTP, etc.). The ability to achieve remote object access in a web service model is profound because it simplifies the interoperability conundrum by standardizing the protocol(s) and the data-representation language. The ubiquity of HTTP makes it ideal for data transfer between clients and services, while XML, as an extensible (meta) language, specifies the data structure and interactions between the parties.

Additionally, the web-services “stack” - SOAP, WSDL, and UDDI provides standardized layers of functionality to ensure interoperability. The Simple Object Access Protocol defines a uniform way to pass XML and perform remote procedure calls. Universal Description, Discovery and Integration provides a registry of SOAP-encoded messages. The UDDI registry uses the Web Services Description Language to describe what a client needs to know to bind to the service. A broad array of support is growing for these ideas and technologies.

Even if wholesale adoption of the web services model is not immediately feasible, there is merit in understanding the concepts and incorporating relevant ideas into future project plans. The proposed architecture of the web services style of distributed computing promises greatly enhanced deployment flexibility. A service can be used by a certain client, by many different clients or by another service. Along with this flexibility comes the increased capability to partition and modularize applications into components. The programming languages in which those components are built is a matter of choice.

JAVA

Java is an ideal language to create distributed enterprise applications. Because of it’s object-oriented design and platform independence, it can be used for scalable, distributed systems, and is an obvious platform for web services, which may be accessed by many different clients. Additionally, the motivation to use Java is enhanced by the fact that a coalition of organizations support the Java 2 Enterprise Edition platform, which includes EJB’s, Servlets, JSP’s, Java Message Services, etc. This coalition has formally evolved into the Java Community Process which oversees the development of Java technology.

The success of Java is evidenced in application server as well as on clients and information appliances like PDA’s. JVM’s are supported on many operating systems including Windows, Linux, MacOS X, Solaris, SymbianOS, PalmOS, and PocketPC. This wide-spread availability makes it possible to address multiple platforms with the same or similar code bases, and eases the effort involved in porting. The concept of modularity can then be extended to the choice of platform(s), creating greater choice and potentially simplifying development and migration.

XML

The Extensible Markup Language may owe its pedigree to SGML, but it owes its success to the web. In its key role of data interchange, XML-based information has come to be referred to as 'Universal Data'. XML provides interoperability because it is structured, extensible and open. The structure is embodied in the concepts of well-formedness and validation. Extensibility is inherent in the design of XML, giving the authors of XML documents and schema control over the choice and implementation of the structure. XML is intrinsically open, in that XML documents are readable text, often with associated DTD's (Document Type Definition) or Schema, providing all the metadata needed to use, or transform, the data.
The benefits of XML have been detailed extensively, but a typical list of features would include:

- Plain Text
- Data Identification
- Stylability
- Inline Reusability
- Linkability
- Easily Processed
- Hierarchical

Notably, the hierarchical nature of XML data representation contrasts distinctly with the 'rectangular' nature of SAS data sets. Thus the difference between hierarchical and rectangular data representation illustrates the need for efficient and automate-able means of 'transformation'.

Transformation of XML data concerns principally three XML specifications: XSL, XSLT and XPath. The Extensible Stylesheet Language (XSL) is both a language for transforming XML documents and a vocabulary for formatting XML documents (McLaughlin 2001). XSL Transformations (XSLT) specifies the textual conversion from one document type to another. XPath (Clark & DeRose 1999) provides a mechanism to identify and refer to an arbitrary element or attribute names and values. XPath and XSLT work together to define what data to use, and how to transform it.

Of course, an XML parser[1] is required to read XML data into memory. XML can be read from a file, although Java XML parsers can read from an `InputStream` or a URL. A validating parser uses the documents' DTD or Schema to validate the data during the parsing process, which simplifies the Java validation code required.

COOPERATING TECHNOLOGIES

Java has deep and multi-faceted XML support. In Java, the Simple API for XML (SAX) is the most commonly used parsing method. Current parsers, (Apache Xerces for example) now implement SAX and DOM (Document Object Model) interfaces and are adding support for Namespaces, Schema and JAXP (see below).

Java, XML, and XSLT are suitable for web applications because of the high degree of modularity they offer (Burke 2001). The Java API for XML Processing (JAXP) supports processing of XML documents using DOM, SAX and XSLT. It is essentially an abstraction which allows the substitution of different parser in an application without changing the application code. This achieves vendor-independence of parsers: "It encapsulates differences between various XML parsers, allowing Java programmers to use a consistent API regardless of which parser they use"(Burke 2001).

SAS has been steadily acquiring the ability to process XML and to expose SAS data as XML documents. These functions are incorporated in the Output Delivery System and the LIBNAME XML facility

The ODS MARKUP statement can be given various tagset= options to define several markup languages including several XML types. Notably, the XML tagsets generate corresponding XML data, XSL stylesheets and DTD (Document Type Definition) schema. The tagset definitions define ODS MARKUP and LIBNAME XML destinations.

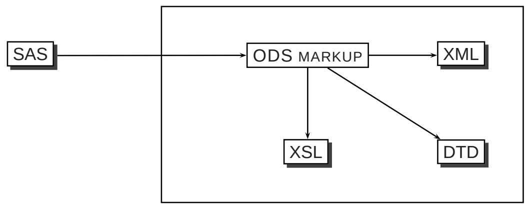


Figure 1: ODS MARKUP can be used to create XML data, stylesheets and schema (DTDs) from procedural and data step output. Using XML to represent content relegates the presentation another (client) component.

Conversely, there is also the ability to import XML using the XML LIBNAME engine, providing that the XML is "very regular". As of version 9.0, there is also the ability to import XML into a SAS dataset using the XMLMap[2]

[1]Parsers are available in many languages including Java, both commercial and open-source.

[2]Using XPath methodology to define and retrieve XML elements.

TAGSET	XML Type
DEFAULT	ODSXML (Generic XML)
DOCBOOK	OASIS DocBook
SASIOXML	Generic XML
SASXMOG	Oracle8iXML[a]
SASXMOIM	Open Information Model XML
SASXMOR	Oracle8iXML[b]

[a]XML LIBNAME XMLTYPE=GENERIC
[b]XML LIBNAME XMLTYPE=ORACLE

Table 1: ODS MARKUP in SAS v9.0 has TAGSET options to support several XML 'dialects'.

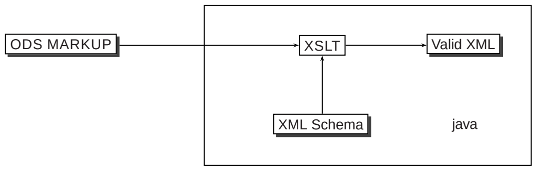


Figure 3: Automating the conversion of XML with Schema-based validation can be accomplished very well by using cooperating technologies. ODS MARKUP is quite adept at generating metadata for sas data structures. XSL stylesheets can then be used to manipulate the XML based on transformation logic and Schema requirements.

option with an explicitly defined XML file containing syntax to 'map' variables, observations and variable attributes.

In cases where the XML is not sufficiently regular, transformation using XSL/XSLT could be used to re-shape the XML to a format SAS can accommodate. However, this highlights a perplexing issue for XML processing with SAS, because the XML capabilities in SAS (ODS MARKUP, LIBNAME XML) overlap with (but don't completely duplicate) the functionality of XSLT, XPATH, and Java technologies such as JAXB[3] and JDOM[4]. In this regard, the SAS (Java-based) Atlas[5] product may help in the creation of XMLMAP files, however the need to automate (XML $\Leftrightarrow$ SAS) transformation is a requirement beyond just an interactive interface.

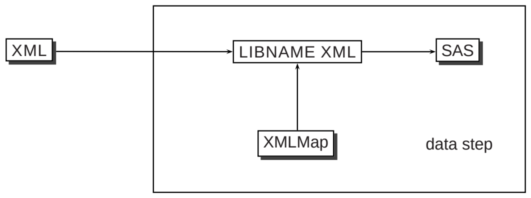


Figure 2: LIBNAME XML uses XMLMap directives (XPath) to map data from XML input to a rectangular SAS data set.

A combination of Java and SAS technologies can be used to achieve XML processing and validation. Using the java `ant` build system, the xalan XSL Stylesheet processor and the xerces validating parser, structured metadata can be auto-generated from SAS data. There is a simplified XML transformation example in Appendix 1.

SERVICE ORIENTED ARCHITECTURE

Extending the capabilities of SAS and designing for interoperability are increasingly important objectives but implementation of these goals is often difficult. Multiple platforms, languages and access methods all contribute to implementation headaches. The promise of the Service Oriented Architecture [SOA] conceptual framework, is to simplify many of these difficulties by providing common standards-based technologies to support web services. The benefits of the SOA approach are rapidly becoming appreciated by data managers and IT architects everywhere. A Service Oriented Architecture [SOA] is "an architecture that uses a distributed, discovery-based environment to expose and manage a collection of service-oriented software assets."(Chappell & Jewell 2002).

The design intent in these emerging Web services aspires to a loose coupling approach to the integration of systems. Although some detractors fault the model as being little more than basically cross-platform RPC, this loose coupling is advantageous insofar as the components used can be modular and therefore replaceable. The SOA is thus built on independent, interdependent, replaceable components.

The service oriented architectural effect, which web services is supposed to achieve (and about which there is so much hype), envisions a 'semantic web' wherein self-describing information is produced, advertised, delivered and consumed. This result as a wide spread phenomenon is probably still a long way off, for a variety of technical and non technical reasons. But the vision is a compelling one, especially for data managers and other IT professionals who spend seemingly inordinate effort massaging data to get it "in the right format".

A SAS DATA SERVICE

SAS has long provided cross-platform connectivity and distributed processing via SAS/Connect and SAS/Access as well as concurrent update access via SAS/Share. The evolution of these products has resulted in a feature rich set of distributed processing technologies which comprises one of SAS' core strengths[6]. Increasingly, SAS has provided software to extend it's data

[3]Java API for XML Binding.
[4]A java-centric XML processing approach (JSR-102).
[5]An interactive XMLMAP creation utility released in v9.0

[6]SSL is supported in v9 SAS/Share and is on the SASWare Ballot for SAS/Connect

management capabilities through "information delivery" by facilitating access via the web.

As these SAS tools and abilities have evolved, presentation of information in SAS has graduated from delivery of static HTML to dynamic content and interactivity. SAS recognizes the need for application integration and is increasingly providing the tools to support interoperability. SAS is now poised to provide the functionality envisioned by the service oriented architecture in the form of 'SAS Data Services'.

Product	Technology niche
IntrNet	CGI Java HTML
ODS	XML HTML PDF RTF WML etc.
LIBNAME XML	XML, markup
Integration Tech.	Middleware
WebEIS	OLAP Java
WebAF	Java Servlet JSP WAP/WML

Table 2: SAS Web Technologies.

INTEGRATION TECHNOLOGIES

SAS Integration Technologies, which is available under separate license, provides support for communication with SAS via standards-based mechanisms, as well as a publishing framework. The Integrated Object Model (IOM) component provides interfaces to distributed object standards such as COM, DCOM and CORBA. The object interfaces include:

- SAS Workspace
- LanguageService
- DataService
- FileService
- Utilities

create a single server-side implementation which can be used by many clients. Because the IOM interfaces are standards-based, almost any programming language can be used for client development. The extensibility and modularity of this approach is obvious; .NET components, J2EE application servers or Java applets can all reference the same set of IOM interfaces.

STORED PROCESSES: A stored process is simply a SAS program stored on a server which can be executed by requesting applications. The stored process can return result packages[7] or can produce streaming results[8]. Maintaining stored processes on the server facilitates change control management and enhances security and application integrity. But furthermore, provides a distribution mechanism which can potentially support a model-view-controller application architecture, strictly. Thus an application can request a result object which would be returned according to the needs of that particular client application.

[7]IOM Direct Interface Stored Processes

[8]Introduced in v9 as "SAS Stored Processes".

SAS DATA SERVICE: The idea of using SAS in a Web Service is addressed in the recent SAS Whitepaper "How to Implement a Web Service with SAS - A Business Scenario". The approach is demonstrated wherein a Stored Process can be executed as a simple "data service" and return the result back to the requesting client. As previously mentioned, an architectural feature is that the client is loosely coupled to the Service via SOAP and the Web Service is loosely coupled to SAS via IOM. So the client and the web service need not be written in the same language nor on the same platform. In the whitepaper, two approaches are outlined to deploying a service which can utilize the stored process; one using VB.NET and the other using the Java API for XML-based RPC (JAX-RPC).

In a web service, if data presentation considerations are handled by the client and data processing (business logic) is handled by SAS, what is left to do? Well, there are a number of requirements to satisfy including: 1) Publishing service metadata to a registry; 2)Finding the appropriate service; 3) Determining how to bind to the service (HTTP, MIME, SMTP, etc.), 4) Facilitate the transactions between the requester and the provider. Many of these capabilities are becoming available in SAS Integration Technologies software.

The JAX-RPC approach described in the whitepaper is followed in the "Climate Service" example. The service requires coding the following components:

- `ClimateServiceIF.java` - The service definition interface.
- `ClimateServiceImpl.java` - The service implementation class, which implements the `ClimateServiceIF` interface.
- `ClimateServiceClient.java` - Simple client (for testing) which contacts the service to invoke the `getClimateData()` method.

The example was developed and tested using:

- SAS v9.0
- Integration Technologies
- Platform Services 1.0
  - sas.core.jar
  - sas.svc.connection.jar
- Java Development Kit 1.4.1
- Java Web Services Developer Pack

The client initiates the transaction and communicates to the Web Service via SOAP over HTTP. The WSDL describing the "Climate Service" is generated during the build process and is included in the Appendix. It includes definitions for <message>, <portType>, <binding> and <service>. JAX-RPC generates java classes built from the WSDL definitions.

In the service implementation class (`ClimateServiceImpl`), the `getClimateData()` method connects to the SAS IOM server to execute a Stored Process (`GetClimateData.sas`) and return the result to the caller.

```
/* GetClimateData.sas                */
*ProcessBody;

libname climate '/data/climate';
data &outdata (keep=datetime outparam);
    set climate.&indata;
    where datetime = "&datetime"dt;
        outparam = &requestedparam;
    run;
```

In a Stored Process, the `*ProcessBody;` comment is a marker interpreted by the Server to end the prologue in which the default values of necessary parameters can be declared (essentially providing functionality equivalent to the %macro variables):

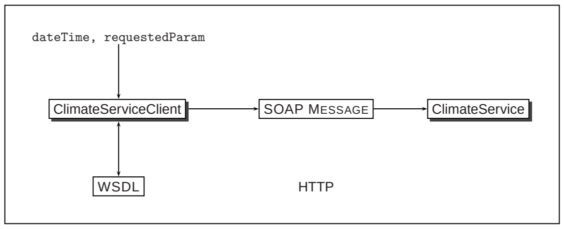
```
%let indata = ;
%let outdata = ;
%let datetime = ;
%let requestedparam = ;
```

But, importantly, when the stored process is executed, the IOM StoredProcessServer substitutes the values for these parameters provided by the `getClimateData()` method in `ClimateServiceImpl` as described below. Using the IOM bridge for Java, a reference to a Workspace object can be created to connect to a remote object on the IOM server, such as in the `getClimateData()` method:

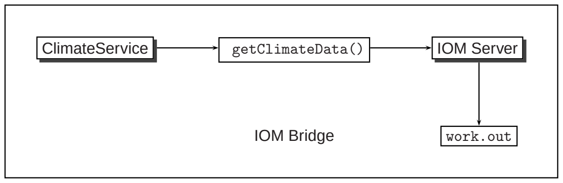
```
/* ClimateServiceImpl.java                    */
/* getClimateData() method                    */
WorkspaceFactory climf = new WorkspaceFactory();
...
IWorkspace climWorkspace =
   climf.createWorkspaceByServer(climprop);
ILanguageService climLang =
   climWorkspace.LanguageService();
```

Once the `ILanguageService` object is created, then the location of the Stored Process on the server can be defined using the `Repository` method and run using parameters passed via the `Execute` method[9]. So the parameter list which the stored process expects is a series of name/value pairs which correspond to macro-type variables that occur in and are resolved by the stored process.

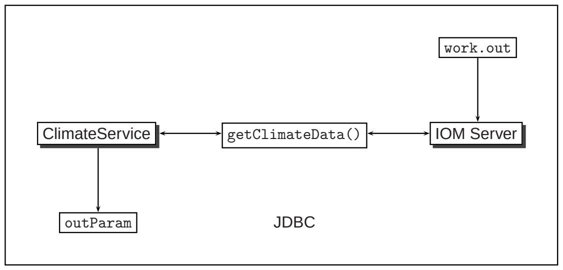
[9]There is also the `ExecuteWithResults` method which returns a `ResultPackage` from the Stored Process that can be published and transported



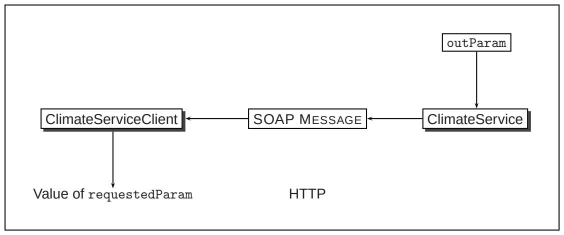
(a) Client initiates a request, by specifying a `dateTime` value and the `requestedParameter` to query, which is transported via SOAP.



(b) The Climate Service invokes the Stored Process via the `getClimateData()` method



(c) `outParam` value returned via the `getClimateData()` method



(d) Value returned to client via SOAP.

Figure 4: Climate Data Service web service example which uses JAX-RPC implementation to invoke a Stored Process via IOM on SAS server. "Loose coupling" of the components is accomplished by the JAX-RPC runtime which handles SOAP, WSDL interpretation and processing.

```
IStoredProcessService climSP =
            climLang.StoredProcessService();
climSP.Repository("file: /stored_processes");
climSP.Execute("GetClimateData",
             "libname=climate"
             + " indata=alldates"
             + " outdata=work.out"
             + " datetime=" + dateTime
             + " requestedparam=" + requestedParam);
```

For this simple example, the result is a single number which is returned by the `getClimateData()` method. In order to do so, a `DataService` object is used to allow an MVAConnection to read the result via JDBC.

```
IDataService climDataService =
              climWorkspace.DataService();
java.sql.Connection climconnect =
    new MVAConnection(climDataService,
                      new Properties());
java.sql.Statement statement =
    climconnect.createStatement();
java.sql.ResultSet rs =
    statement.executeQuery("Select * from work.out");
if( rs.next()) {
    String climData = rs.getString("outParam");
    _retVal = Double.parseDouble(climData);
```

The method then closes the connection, ends the use of the workspace and shuts down the WorkSpaceFactory.

```
climconnect.close();
climWorkspace.Close();
climf.shutdown();
```

Although this example highlights a very simple operation of returning a value, the approach could be used to return objects of any sort. So a statistical operation or an ODS generated report could be the result of a service request. Another design objective of a web service is to provide a 'coarse grained' functionality. This means that the actual service would not consist of a single fine grained method such as `getClimateData()`, but would rather be a carefully designed package of functions which would provide access to the appropriate logic. For example, these types of operations might be bundled as a service:

- `getClimateData`
- `getClimateStat`
- `getClimateReport`
- `getClimateGraph`
- `getClimateMetadata`
- `updateClimateLog`
- `updateClimateUsageHistory`

SAS STORED PROCESSES: Introduced in version 9, “SAS Stored Processes” have the additional capability to stream results to a PIPE via a SAS fileref. The stream is then usable by the calling program via the `StoredProcessExecutionInterface`. A web service could be written to expect a known, validatable XML stream as the result of SAS Stored Process. This is obviously a powerful combination of analytic and data delivery capabilities when using SAS in a web service. It also enhances the ability to partition the application logic in appropriate ways. For example, XSLT or some binding mechanism may be more adept at transforming complex XML than SAS methodologies; whereas analytical capabilities available via stored processes are likely more capable, maintainable and scalable.

The idea of web services is compelling in many ways. Not least of which is the potential for components of the system to evolve independently as long as they continue to conform to understood specifications and interfaces. In this respect, a problem to which the web service model seems to provide an attractive solution, is the deployment of Handheld computers for data collection. The expectation is that it will be simpler to manage a diversity of data collection devices that, inevitably, undergo continuous hardware and software changes. Java client software on these devices is attractive because of it's availability on many platforms. A SAS-based 'data service' (in development), is intended to simplify the deployment of handheld computers in a 'field' situation by standardizing the access method and data delivery. Although not yet complete, a prototype of this service exchanges information between the handheld and the 'data service'. In connected-mode, a message defined on the handheld will be delivered to the server (via a SOAP request), and the server responds with the requested form definition in XML. The handheld application then parses the XML encoded form-definition to build a data entry form and its behaviors. In disconnected mode data are entered into the form, after which it is reconnected. Finally, the XML-encoded data are transferred to the 'data service' which transforms the incoming data for processing and updates to the data repository.

Other aspects of a data entry web-service are planned to achieve:

- Description: To name and define forms and their data models.
- Discovery: To acquire form definitions from unknown sources.

CONCLUSIONS

The motivation for delivering information as a 'service' is very compelling and web services are now being positioned to provide a Service Oriented Architecture. XML increasingly provides the 'data transportation layer' for inter application messaging and presentation-neutral content markup to such an extent that it is now a de-facto standard. XML supporting technologies, however, are still evolving rapidly and it remains to be seen which of them will be successfully and garner support into the future.

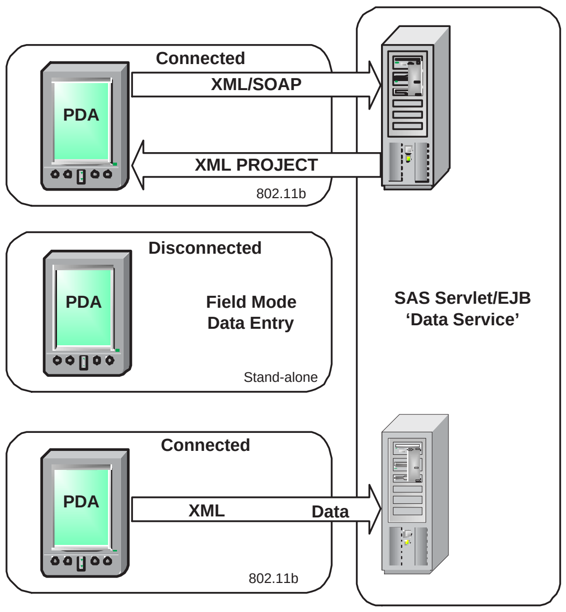


Figure 5: Modeling a Data Service for Data Entry: Distribute form definitions; Data Entry in stand-alone mode; Subsequent data synchronization event after data collection.

Software legacy systems, which perform data processing and storage functions, obviously perpetuate themselves by their inherent utility and through an unwillingness to abandon the investment in that technology. But increasingly, these systems are required to interoperate more extensively and flexibly. SAS has focused technology development to satisfy the requirement for interoperability with products like IntrNet, AppDevStudio and Integration Technologies. The commitment by SAS to support Java and XML is beginning to yield compelling alternatives including web service capabilities. Using SAS, Java and XML in cooperation, allows for partitioning the design of information access into components that interoperate as a loosely coupled data service.

WEB REFERENCES

http://www.jcp.org
http://xml.apache.org

REFERENCES

Burke, E. M. (2001). Java and XSLT, first edn, O'Reilly.

Chappell, D. A. & Jewell, T. (2002). Java™ Web Services, first edn, O'Reilly.

Clark, J. & DeRose, S. (1999). Xml path language (xpath) version 1.0, "http://www.w3.org/TR/xpath".

How to Implement a Web Service with SAS - A Business Scenario (2002). Technical report, SAS Institute.

McLaughlin, B. (2001). Java and XML, second edn, O'Reilly.

ACKNOWLEDGMENTS

I would like to thank Myla Jordan, Information Systems Developer, and John Merritt, System Administrator, for Java development and software configuration management support, respectively. I would also like to thank the SAS Institute for providing a Demo version of Integration Technologies without which the testing and development of the "Climate Service" example code would not have been possible.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Scott Chapal
Ichauway, Inc.
Rt. 2 Box 2324
Newton GA. 31770
scott.chapal@jonesctr.org

APPENDIX 1

Using SAS XML in cooperation with XSLT.

Metadata generated in SAS:

```
libname contents xml "../metadata/datasets/&DSNAME._contents.xml";
proc contents
    data=library.&DSNAME
    out=work.&DSNAME
    varnum
    details
    directory;
run;
data contents.&DSNAME;
    set work.&DSNAME;
run;
```

A resulting XML fragment, can then be transformed by XSLT to...

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<TABLE>
  <GC>
     <LIBNAME>LIBRARY</LIBNAME>
     <MEMNAME>GC</MEMNAME>
     <MEMLABEL Missing="" />
     <TYPEMEM Missing="" />
     <NAME>commitd</NAME>
     <TYPE>1</TYPE>
     <LENGTH>8</LENGTH>
     <VARNUM>4</VARNUM>
     <LABEL>Commit Date</LABEL>
     <FORMAT>YYMMDDD</FORMAT>
     <FORMATL>10</FORMATL>
     <FORMATD>0</FORMATD>
     <INFORMAT>YYMMDD</INFORMAT>
     <INFORML>8</INFORML>
     <INFORMD>0</INFORMD>
     <JUST>1</JUST>
     <NPOS>0</NPOS>
     <NOBS>5</NOBS>
     <ENGINE>V8</ENGINE>
....
  </GC>
```

Satisfy the requirements of a particular Schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<dataTable
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="eml://jonesctr.org/dataTable-2.0.0">
  <entityName xmlns="">LIBRARY.GC</entityName>
    <entityDescription xmlns="">LIBRARY.GC
       </entityDescription>
    <attributeList xmlns="">
      <attribute>
        <attributeName>commitd</attributeName>
        <attributeLabel>Commit Date</attributeLabel>
        <attributeDefinition>Commit Date
           </attributeDefinition>
        <storageType>date</storageType>
        <measurementScale>
          <datetime>
            <formatString>YYYY-MM-DD</formatString>
            <dateTimePrecision>1</dateTimePrecision>
            <dateTimeDomain/>
         </datetime>
       </measurementScale>
     </attribute>
...
</dataTable>
```

APPENDIX 2

WSDL example for the Web Service “Climate Service”:
`http://slash:8080/climateservice-jaxrpc/climateservice?WSDL`

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
             xmlns:tns="http://jonesctr.org/ClimateService"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
             name="ClimateService"
             targetNamespace="http://jonesctr.org/ClimateService">
  <types/>
  <message name="ClimateServiceIF_getCliateData">
    <part name="String_1" type="xsd:string"/>
    <part name="String_2" type="xsd:string"/></message>
  <message name="ClimateServiceIF_getClimateDataResponse">
    <part name="result" type="xsd:double"/></message>
  <portType name="ClimateServiceIF">
    <operation name="getClimateData" parameterOrder="String_1 String_2">
      <input message="tns:ClimateServiceIF_getClimateData"/>
      <output message="tns:ClimateServiceIF_getClimateDataResponse"/>
    </operation>
   </portType>
  <binding name="ClimateServiceIFBinding" type="tns:ClimateServiceIF">
    <operation name="getClimateData">
      <input>
        <soap:body
                   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                   use="encoded"
                   namespace="http://jonesctr.org/ClimateService"/>
      </input>
      <output>
        <soap:body
                   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                   use="encoded"
                   namespace="http://jonesctr.org/ClimateService"/>
      </output>
      <soap:operation soapAction=""/></operation>
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
                  style="rpc"/>
    </binding>
  <service name="ClimateService">
    <port name="ClimateServiceIFPort" binding="tns:ClimateServiceIFBinding">
      <soap:address xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
                    location="http://slash:8080/climateservice-jaxrpc/climateservice"/>
    </port>
  </service>
</definitions>
```