Dialogic®

Tech Note

Using XMLBeans to Simplify Dialogic® PowerMedia™ Extended Media Server (XMS) RESTful Application Programming in Java

NETWORK FUEL™

Using XMLBeans to Simplify Dialogic® PowerMedia™ Extended Media Server (XMS) RESTful Application Programming in Java

Tech Note

Introduction

A Dialogic® PowerMedia™ Extended Media Server (XMS) application, as RESTful, web-based software, relies on an Extensible Markup Language (XML) payload to send and receive information to/from the server side. XML payloads are often defined using the XML Schema Document (XSD) formal schema description language. This is the case with PowerMedia XMS. Representational State Transfer (RESTful) payloads are described by an XSD document on the server side, and this same document may be used to generate XML language bindings for the client/application side. This article will describe one specific XSD processor, XMLBeans, and how it can be used as part of a Java PowerMedia XMS RESTful applicaton, the PowerMedia XMS Verification Demo. These techniques can be applied to other PowerMedia XMS Java applications as well.

XMLBeans

Apache XMLBeans is a technology for accessing XML by binding it to Java types. Using XMLBeans, XML schema can be compiled to generate Java types that represent the schema. In this way, you can access instances of the schema through JavaBeans-style accessors after the fashion of "getFoo" and "setFoo". While a simple command line processor is used for creating the bindings and the Java JAR file, getting the options correct can be a hit or miss process. For this reason, scripts to run the processor (for both Windows and Linux) are included in this article.

Installation

A system with PowerMedia XMS installed is not necessary to create the xmsrest Java classes; only the latest xmsrest.xsd file from the XMS download is needed. It is located in the /etc/xms directory on an installed PowerMedia XMS system. See the the Dialogic PowerMedia XMS Download page to get started downloading, installing and configuring PowerMedia XMS.
Installing XMLBeans is covered in its README.txt file in its top level directory. However, a synopsis of what you need to do for processing an PowerMedia XMS schema is included here (release numbers current as of October 2012):

- Java - make sure you have Java Development Kit (JDK) 1.6.x installed, that java[.exe] is on your path and that the JAVA_HOME environment variable is set
- Download and uncompress the latest XMLBeans binary distribution, version 2.5.0. It may be found here.
- Set the environment variable XMLBEANS_LIB to the installations lib directory. For example, in Linux, the .bash_profile file would have a line similar to "export XMLBEANS_LIB=/usr/java/xmlbeans-2.5.0/lib
- Copy a current xmsrest.xsd, from (/etc/xms on an installed XMS system) xmsrest.xsdconfig and the Windows (.bat) or Linux (.sh) script to xmlbeans-2.5.0 directory. All except for the current xmsres.xsd may be found at the end of this article.

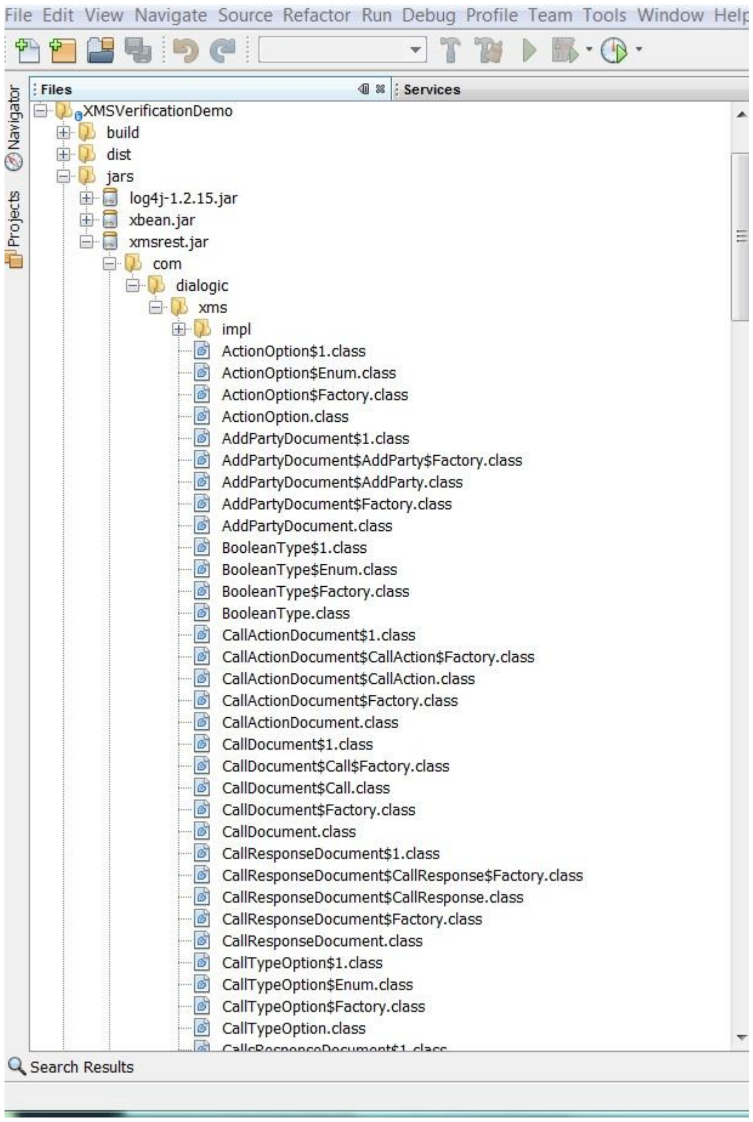
Producing the JAR File

Simply run the script on the Windows or Linux command line:

```
> make_jar[.bat or .sh]
```

A file by the name of xmsrest.jar will be produced. Copy this into your Java project under a jars or libs directory. In addition, there are dependencies on a number of classes in xbean.jar. This jar file is found under the "libs" directory of the XMLBeans distribution. It should also be copied into the same location as xmsrest.jar.

Further steps, depending on the IDE, will be needed to add the jar files to your build path. Here is the jar file embedded in the PowerMedia XMS Verification Demo, using the NetBeans IDE:


File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Navigator
Projects
Files
Services
Search Results

```
XMSVerificationDemo
  build
  dist
  jars
    log4j-1.2.15.jar
    xbean.jar
    xmsrest.jar
      com
        dialogic
          xms
            impl
            ActionOption$1.class
            ActionOption$Enum.class
            ActionOption$Factory.class
            ActionOption.class
            AddPartyDocument$1.class
            AddPartyDocument$AddParty$Factory.class
            AddPartyDocument$AddParty.class
            AddPartyDocument$Factory.class
            AddPartyDocument.class
            BooleanType$1.class
            BooleanType$Enum.class
            BooleanType$Factory.class
            BooleanType.class
            CallActionDocument$1.class
            CallActionDocument$CallAction$Factory.class
            CallActionDocument$CallAction.class
            CallActionDocument$Factory.class
            CallActionDocument.class
            CallDocument$1.class
            CallDocument$Call$Factory.class
            CallDocument$Call.class
            CallDocument$Factory.class
            CallDocument.class
            CallResponseDocument$1.class
            CallResponseDocument$CallResponse$Factory.class
            CallResponseDocument$CallResponse.class
            CallResponseDocument$Factory.class
            CallResponseDocument.class
            CallTypeOption$1.class
            CallTypeOption$Enum.class
            CallTypeOption$Factory.class
            CallTypeOption.class
```

Using the Java Classes

Several examples of using the Java classes to build new XML requests and parsing out elements in responses will now be reviewed. Further generic examples can be found in the "samples" directory in the XMLBeans distribution.

Making an Outbound Call with PowerMedia XMS

An outbound call is done using an HTTP POST method. The HTTP document can be easily created and formatted. Then the XML payload can also be created and set up easily in terms of the actual numbers of lines of code and the actual formatting of the message itself:

```
// Create the HTTP document
WmsDoc = WebServiceDocument.Factory.newInstance();
WebServiceDocument.WebService wms= WmsDoc.addNewWebService();

XmlNMTOKEN  version = XmlNMTOKEN.Factory.newInstance();
verion.setStringValue("1.0");

// Create and setup a new call resource
Call newCall = wms.addNewCall();
newCall.setDestinationUri(toURI);
newCall.setCpa(BooleanType.NO);
newCall.setSdp("");
newCall.setMedia(MediaType.AUDIOVIDEO);
newCall.setSignaling(BooleanType.YES);
newCall.setDtmfMode(DtmfModeOption.RFC_2833);
newCall.setAsyncDtmf(BooleanType.YES);
newCall.setAsyncTone(BooleanType.YES);
newCall.setRxDelta("+0dB");
newCall.setTxDelta("+0dB");
newCall.setSourceUri(fromURI);
wms.xsetVersion(version);

// Convert the class to a string of XML for the HTTP request
XML=WmsDoc.toString();

// Send the HTTP request to media server and get the response
response = XMSInterface.SendHTTPRequestAndGetResponse(XMSInterface.getURL() + "calls",
"POST", XML);
```

Parsing a PlayRecord Response

Requests to play and record media will have a transaction ID for the media operation embedded in the response. This allows the play/record to be programatically terminated. The transaction ID is needed to identify the operation and will usually be parsed out of the HTTP response when it is received. The reason that the operation terminated is also retrieved. Here is an example of a response to a PlayRecord request:

```
// Send the HTTP request to XMS
response = XMSInterface.SendHTTPRequestAndGetResponse(XMSInterface.getURL() +
fullPathOfPlayRecord, "PUT", XML);

WebServiceDocument wsDoc;
try {
         wsDoc = WebServiceDocument.Factory.parse(response) ;
         WebServiceDocument.WebService wS = wsDoc.getWebService();
         // Information on the result of the call action, including the transaction ID and
         // the reason the record terminated can now be parsed out
        CallResponse cR = wS.getCallResponse();
        CallAction cA = cR.getCallAction();
        Playrecord playRecordStatus = cA.getPlayrecord();
        String playTransactionID = playRecordStatus.getTransactionId();
} catch (XmlException ex) {
        logger.error(ex);
}
```

Links

The following are links regarding the major components used in this article:

- Dialogic® PowerMedia™ Extended Media Server (XMS) –
    - http://www.dialogic.com/products/media-server-software/xms.aspx
- XMLBeans - http://xmlbeans.apache.org
- NetBeans - http://netbeans.org
- Java - http://www.java.com

Open Source

This article discusses one or more open source products, systems and/or releases. Dialogic is not responsible for your decision to use open source in connection with Dialogic products (including without limitation those referred to herein), nor is Dialogic responsible for any present or future effects such usage might have, including without limitation effects on your products, your business, or your intellectual property rights.

Using XMLBeans to Simplify Dialogic® PowerMedia™ Extended Media Server (XMS) RESTful Application Programming in Java

Tech Note

www.dialogic.com

Dialogic Inc
1504 McCarthy Boulevard
Milpitas, California 95035-7405
USA