CHAPTER 1

Getting Started with the Cisco Multicast Manager SDK

Cisco Multicast Manager (CMM) 3.2 provides a Web Services Definition Language (WSDL)-based Application Programming Interface (API) that allows you to develop document-oriented web services that send WSDL messages to CMM and receive WSDL messages in response. Using the CMM API, you can develop an application that communicates with CMM to perform tasks such as:

- Querying a layer 3 device about the currently available S, G on the device.
- Querying CMM about the available multicast streams in a specified domain.
- Adding layer 2 and layer 3 devices or video probes to CMM in bulk.
- Adding, querying, and deleting S,G polling configuration.
- Adding, querying, and deleting Layer 2 polling configuration.
- Adding, querying, and deleting interface polling configuration.
- Adding, querying, and deleting Reverse Path Forwarding (RPF) polling configuration.
- Adding, querying, and deleting tree polling configuration.
- Adding, querying, and deleting video probe polling configuration.
- Adding, querying, and deleting VidMon polling configuration.
- Adding, querying, and deleting Source Specific Multicast (SSM) polling configuration.
- Adding and querying S, G time-based polling configuration.
- Adding, querying, and deleting Rendezvous Point (RP) polling configuration.

This chapter contains the following sections:

WSDL-Based Services

The CMM API allows you to create WSDL-based Web Services. The CMM API is based on:

- **JAXB**—The Java Architecture for XML Binding (JAXB), which is provided by Oracle.

- **The Spring Framework**—The sample code included in the SDK uses the Spring Framework, provided by SpringSource, as an example of a framework that you can use to develop Web Services to communicate with CMM. The Spring Framework provides a Java beans package that includes a "bean factory," which allows development of beans that enable configuration of virtually any Java object. The CMM API includes a set of beans, developed using the Spring Framework, that allow you to:
  - Marshall and un-marshall SOAP messages from XML format to Java objects and back to XML format.
  - Define WS-Security policies

Note You can use any framework based on the SOAP with Attachments API for Java (SAAJ) for creating Web services applications to work with CMM.

- **MTOM**—SOAP Message Transmission Optimization Mechanism (MTOM) is a W3C recommendation that defines standards for efficient and optimal encoding of SOAP messages.

CMM also uses basic WS-Security policies for password-based authentication.

Using JAXB, the CMM programming interface invokes a binding compiler that generates Java classes and interfaces that are derived from the CMM WSD schema—*cmmWebServices.xsd*.

Since the JAXB web services are document-oriented web services, there is no concept of making a remote function call. Your application creates a SOAP message and sends it to the CMM server, and in response, receives a SOAP response message or a SOAP fault.

JAXB

JAXB is a Java API that allows you to use Java routines to access and maintain data in XML documents without using an XML parser.

Using JAXB, the CMM programming interface invokes a binding compiler that generates Java classes and interfaces that are derived from the CMM WSD schema—*cmmWebServices.xsd*. Using the Java classes, you develop document-oriented Web services as opposed to remote procedure call (RPC)-based applications. You use the JAXB-derived classes to marshall and unmarshall SOAP messages and perform the tasks for implementing an application that communicates with CMM.

For detailed information on JAXB, go to the JAXB web site at the following location:

http://www.oracle.com/technetwork/java/index.html

Since one of the CMM APIs enables you to upload an image file uploaded to a client, CMM uses MTOM (http://www.w3.org/TR/soap12-mtom/) for efficient and optimal encoding of SOAP messages. Basic WS-Security policies are used for password based authentication.

Web services APIs can be used either on HTTP or on the secure HTTPS protocol.

Since CMM uses JAXB for marshalling and un-marshalling, all APIs are described using classes generated by JAXB after compiling the WSDL file.

Framework

The CMM API uses several beans developed using the Spring Framework. The Spring Framework provides a Java beans package that includes a "bean factory," which allows development of beans that enable configuration of virtually any Java object.

For detailed information on the Spring Framework, go to the Spring website at the following locations:

http://www.springsource.org/

The CMM 3.2 SDK uses Spring Framework 2.5.6, which can be downloaded from the Spring download site:

http://www.springsource.org/spring-community-download

Note Although the CMM API includes beans derived from the Spring Framework, you can use any framework that is based on the SOAP with Attachments API for Java (SAAJ) for creating and sending SOAP messages. For basic information on SAAJ, see:

http://java.sun.com/developer/EJTechTips/2005/tt0425.html#1

SOAP Message Transmission Optimization Mechanism

SOAP Message Transmission Optimization Mechanism (MTOM) is a W3C recommendation that defines standards for efficient and optimal encoding of SOAP messages.

Summary of the API Operations and Java Beans

For a list of the API operations in the CMM SDK, see Table 2-1, "CMM API Operations" in Chapter 2, "Cisco Multicast Manager API Reference."

Table 1-1 describes the Java beans provided with the SDK.

Table 1-1 CMM SDK—Java Beans

Bean	Description
Message Factory Bean	Initializes the message factory object for using SAAJ SOAP messages. See Message Factory Bean, page 2-46.
JAXB Marshaller Bean	Converts SOAP messages from XML format to Java objects or from Java objects to XML format. See JAXB Marshaller Bean, page 2-46.
Security Interceptor Bean	Allows you to define WS-Security policies. See Security Interceptor Bean, page 2-47

Contents of the SDK and Web Services Sample Code Structure

This section contains the following:

SDK Source Files

The SDK source code and supporting files are contained in a zip file called *testWeb.zip/TibcoJar.zip*. This file is provided on the Software Download Area for Cisco Multicast Manager at this location:

http://www.cisco.com/en/US/products/ps6337/index.html

Downloading the SDK

The source code for the CMM 3.2 SDK is provided on the software download area for CMM on cisco.com.

Complete these steps to download the SDK files:

Step 1 Go to the product page for CMM 3.2 at the following URL:

http://www.cisco.com/web/go/cmm

Step 2 Click the **Software Download** link.

Step 3 Log in to Cisco.com.

Step 4 Locate the directory for CMM 3.2 and download the distribution file for the SDK (*testWeb.zip)*.

Step 5 Copy the zip file containing the distribution to your client machine at one of the following locations:

- **Linux:**

  /usr/local/netman/testWeb

- **Solaris:**

  /opt/RMSMMT/testWeb

Step 6 Unzip the *testWeb.zip* distribution file.

SDK Directory Structure and Key Files

The following directory contains the SDK files:

testWeb/client/jaxws/src/com/cisco/nm/cmm/ws/client/jaxws/

Table 1-2 describes the key files of interest in the SDK.

Table 1-2 Key Files in the CMM SDK

Directory	Filename	Description
testWeb/client/jaxws	*build.xml*	This is an Ant build script that you can use if you are compiling the SDK in a standard Ant environment. This *build.xml* file contains the *wsimport* task. The *wsimport* task downloads a WSDL file from specified URL. You can change the URL to point to the IP address of the CMM server. In addition, ensure that you use the correct port depending on your choice of HTTP or HTTPS The *wsimport* task downloads the CMM WSDL file (*cmmNB.wsdl)* and schema file (*cmmWebServices.xsd*) and generates code using the JAXB compiler. The generated code is written to the *testWeb/client/jaxws/gen* directory. **Note** If you are using another compiler, such as an IDE compiler, the compiler will generate its own *build.xml* file.
testWeb.zip/	*TibcoJar.zip*	Java class files for the SDK.
testWeb/client/jaxws/src/com/cisco/nm/cmm/ws/client/jaxws	*CmmWebServicesClient.java*	Contains the source code defined in this document. This is working code that you can use as a sample to build your application
testWeb/client/jaxws/src/com/cisco/nm/cmm/ws/client/jaxws	*applicationContext.xml*	This is the SpringSource configuration file for configuring several beans to be used by client application, in this case *CmmWebServicesClient.java* is sample client application.
testWeb/client/jaxws/gen	Various	If you are using the Ant compiler, this directory contains he code for your application generated by the Ant compiler.

Setting Up to Use the API

Before you can use the CMM SDK, you must set up your system to compile and run the SDK code.

Complete these steps to set up your system to use the CMM API:

1. If you will run your CMM client application on HTTPS, complete the setup tasks for HTTPS.

   See Required Setup for HTTPS, page 1-6.
2. If you will run your CMM client application on HTTP, complete the setup tasks for HTTP.

   See Required Setup for HTTP, page 1-9
3. Configure your compiler for the SDK. You can compile the SDK in several different ways:

- By using the setup files provided with the SDK for the Ant task and running the Ant task to generate the Java code for the SDK.

  See Running the Ant Task to Compile the SDK, page 1-10.

- By setting up another compiler of your choice; for example the Netbeans IDE.

  See Compiling with an IDE, page 1-12.

Required Setup for HTTPS

If you are using HTTPS for your application, then in order to use CMM web services on HTTPS, you must complete the following preliminary steps on the CMM server:

Step 1 On your development server download a Java source file from this location:

https://sectool.dev.java.net/source/browse/sectool/

This is an open source GPL2 license project. It lets you create certificates with *SubjectAlternativeName*.

If your organization does not accept GPL2 based development in that case you can request certificate from certificate generation authority for your organization.

Step 2 Click on **Documents and Files,** and then click on the **src** link in the content area.

You will see a link called **SecTool.java**.

Step 3 Right-click on this link and save it under the filename *SecTool.java* on your development desktop.

Step 4 Verify that you have JDK 1.6 installed.

Step 5 Go to the directory where you have downloaded the file *SecTool.java* and run the Java compiler command to compile this file:

```
javac SecTool.java
```

You will see several warnings, which you can ignore.

The process generates three class files:

- *Pair.class*
- *PassphraseSecretKey.class*
- *SecTool.class*

Step 6 Copy these three class files into the *sectool* directory on the server machine.

Note Configure FTP to use binary mode.

- On Solaris, enter:

  ```
  /opt/RMSMMT/sectool
  ```

- On Linux, enter:

  ```
  /usr/local/netman/sectool
  ```

  ```
  On the server machine, set up a keystore with the IP address of the server and the DNS
  name if a DNS name exists.
  ```

Step 7 Run the **sectool** script.

- On Linux, enter:

```
cd /usr/local/netman/sectool
./sectool.sh
```

- On Solaris enter:

```
cd /opt/RMSMMT/sectool
./sectool.sh
```

The following prompt appears:

```
Does this server have DNS name  [y/n]
```

Step 8 Enter **y**.

Step 9 Enter the DNS name for the server.

For example, enter *cmm-le4-07.cisco.com*.

Step 10 Enter the IP address for the server.

For example, enter *172.20.111.242*.

- The operating system is Linux, and the DNS name is cmm-le4-07.cisco.com (IP address: 172.20.111.242).
- The DNS name is cmm-le4-07.cisco.com.
- The IP address is IPAddress: 172.20.111.242.

The script processes the backup version of the existing keystore and creates a new keystore file. The keystore file is located in the following directory:

- On Linux:

  /usr/local/netman/mmtsys/apache-tomcat/conf/keystore

- On Solaris:

  /opt/RMSMMT/mmtsys/apache-tomcat/conf/keystore

Step 11 Copy the keystore file to the *client machine* and remember the location of the file, as you will need this file path to edit the file build.

Step 12 Make sure that on the *server machine*, the *acegi* security configuration file has a secure channel for web services.

Step 13 Edit the following file:

- On Linux:

  /usr/local/netman/cmm/WEB-INF/conf/acegi/security-config.xml

- On Solaris:

  /opt/RMSMMT/cmm/WEB-INF/conf/acegi/security-config.xml

Step 14 Locate the following entries:

```
/j_acegi_security_check*=REQUIRES_INSECURE_CHANNEL
       /**=REQUIRES_INSECURE_CHANNEL
```

Step 15 Change the entries to read:

```
/j_acegi_security_check*=REQUIRES_SECURE_CHANNEL
                   /**=REQUIRES_SECURE_CHANNEL
```

Step 16 Locate the following entries:

```
/service/*=REQUIRES_INSECURE_CHANNEL
                                                    /*.xsd=REQUIRES_INSECURE_CHANNEL
```

Step 17 Change the entries to read:

```
/service/*=REQUIRES_SECURE_CHANNEL
                                                    /*.xsd=REQUIRES_SECURE_CHANNEL
```

Step 18 Save the *security-config.xml* file.

Step 19 Edit the following file:

- On Linux:

  /usr/local/netman/cmm/WEB-INF/spring-ws-servlet.xml

- On Solaris:

  /opt/RMSMMT/cmm/WEB-INF/spring-ws-servlet.xml

Step 20 Locate the following entry:

```
<ref bean="wsSecurityInterceptor"/>
```

Step 21 Comment out this line:

```
<!-- <ref bean="wsSecurityInterceptor"/> -->
```

Step 22 Save the *spring-ws-servlet.xml* configuration file.

Step 23 Restart the server.

Step 24 On the *client machine***,** navigate to the t*estWeb/client/jaxws* directory.

Step 25 Edit the *build.xml* file.

Step 26 Update the location of the keystore file (copied to the client machine) and enter the default keystore password *cmm_dev* as arguments to JVM in the build.xml file on the client machine, as follows (you need to update this information at two places):

```
-Djavax.net.ssl.trustStore=C:/ftp/testWeb/keystore
-Djavax.net.ssl.trustStorePassword=cmm_dev
```

Step 27 Verify that the server URLs have been changed from http to https and that the port on these URLs has been changed from 8085 to 8080.

For example:

```
<arg value="https://172.20.111.233:8080/cmm/service/cmmNB.wsdl"/>
```

Step 28 Execute the following commands:

```
ant clean
ant generate
```

Step 29 Verify that the code is being generated.

If no code is being generated, an error code appears on the console.

Step 30 Edit the client java file and enter.

```
ant run
```

Required Setup for HTTP

If you are using HTTP for your application, then in order to use CMM web services on HTTP, you must complete the following preliminary steps on the CMM server:

Step 1 Verify that the *server machine* acegi security configuration file has an *insecure* channel configured for web services.

Step 2 Edit the following file:

- On Linux:

  /usr/local/netman/cmm/WEB-INF/conf/acegi/security-config.xml

- On Solaris:

  /opt/RMSMMT/cmm/WEB-INF/conf/acegi/security-config.xml

Step 3 Locate the following entries:

```
/j_acegi_security_check*=REQUIRES_SECURE_CHANNEL
                        /**=REQUIRES_SECURE_CHANNEL
```

Step 4 Change them to read:

```
/j_acegi_security_check*=REQUIRES_INSECURE_CHANNEL
                        /**=REQUIRES_INSECURE_CHANNEL
```

Step 5 Save the *security-config.xml* file.

****

Step 6 Edit the following file:

- On Linux:

  /usr/local/netman/cmm/WEB-INF/ spring-ws-servlet.xml

- On Solaris:

  /opt/RMSMMT/cmm/WEB-INF/ spring-ws-servlet.xml

Step 7 Locate the following entry:

```
<ref bean="wsSecurityInterceptor"/>
```

Step 8 Comment out this line:

```
<!-- <ref bean="wsSecurityInterceptor"/> -->
```

Step 9 Save the *spring-ws-servlet.xml* configuration file.

Step 10 Restart the server.

Step 11 On the *client machine* go to the to *testWeb/client/jaxws* directory.

Step 12 Edit the *build.xml* file.

Step 13 Make sure that server URLs have been changed from https to http and that the port on these URLs has been changed from 8080 to 8085.

For example:

```
<arg value="http://172.20.111.233:8085/cmm/service/cmmNB.wsdl"/>
```

Step 14 Enter the following commands:

```
ant clean.
ant generate
```

You should see some code being generated; if no code is generated, you will see an error on the console.

Step 15 Edit the client java file and execute the following command:

```
ant run
```

Running the Ant Task to Compile the SDK

If you are using an Ant environment, complete these steps to run the Ant task and compile the SDK:

Step 1 Copy the zip file *testWeb.zip* to a client machine and unzip it.

Step 2 Verify that the client machine has access to the internet.

Example 1-1 shows the Ant build script:

Example 1-1 Ant Build Script for the CMM SDK

```
<?xml version="1.0"?>
<project name="spring-ws-airline-sample-spring-ws-client" default="build"
xmlns:artifact="urn:maven-artifact-ant">
    <property name="bin.dir" value="bin"/>
    <property name="src.dir" value="src"/>
    <property name="gen.dir" value="gen"/>

    <target name="init">
    <mkdir dir="${bin.dir}"/>
        <mkdir dir="${gen.dir}"/>
        <typedef resource="org/apache/maven/artifact/ant/antlib.xml"
uri="urn:maven-artifact-ant">
            <classpath>
                <pathelement location="${basedir}/../../maven-ant-tasks-2.0.7.jar"/>
            </classpath>
        </typedef>

        <artifact:remoteRepository id="main" url="http://repo1.maven.org/maven2/"/>
        <artifact:remoteRepository id="java.net"
url="https://maven-repository.dev.java.net/nonav/repository" layout="legacy"/>
    <artifact:remoteRepository id="Spring-repo-ext"
url="https://springframework.svn.sourceforge.net/svnroot/springframework/repos/repo-ext"
/>

      <artifact:dependencies pathId="generate.classpath">
            <remoteRepository refid="main"/>
            <remoteRepository refid="java.net"/>
            <remoteRepository refid="Spring-repo-ext"/>
            <dependency groupId="com.sun.xml.ws" artifactId="jaxws-tools" version="EA3"/>
            <dependency groupId="javax.xml.crypto" artifactId="xmldsig" version="1.0"/>
            <dependency groupId="com.sun.xml.bind" artifactId="jaxb-xjc"
version="2.1-EA2"/>
            <dependency groupId="com.sun.xml.stream.buffer" artifactId="streambuffer"
version="0.3"/>
            <dependency groupId="com.sun.xml.stream" artifactId="sjsxp" version="1.0"/>
            <dependency groupId="org.jvnet.staxex" artifactId="stax-ex" version="1.0"/>
            <dependency groupId="com.sun.xml.messaging.saaj" artifactId="saaj-impl"
version="1.3"/>
            <dependency groupId="javax.activation" artifactId="activation"
version="1.1.1"/>
```

```
    </artifact:dependencies>

        <artifact:dependencies pathId="classpath">
            <remoteRepository refid="main"/>
            <remoteRepository refid="java.net"/>
            <remoteRepository refid="Spring-repo-ext"/>
           <dependency groupId="org.springframework.ws" artifactId="spring-ws-core-tiger"
version="1.5.4"/>
            <dependency groupId="org.springframework.ws" artifactId="spring-ws-security"
version="1.5.4"/>
            <dependency groupId="org.springframework.ws" artifactId="spring-oxm-tiger"
version="1.5.4"/>
            <dependency groupId="com.sun.xml.messaging.saaj" artifactId="saaj-impl"
version="1.3"/>
            <dependency groupId="commons-httpclient" artifactId="commons-httpclient"
version="3.0.1"/>
            <dependency groupId="javax.xml.bind" artifactId="jaxb-api" version="2.1"/>
            <dependency groupId="javax.xml.crypto" artifactId="xmldsig" version="1.0"/>
            <dependency groupId="com.sun.xml.stream.buffer" artifactId="streambuffer"
version="0.3"/>
            <dependency groupId="com.sun.xml.stream" artifactId="sjsxp" version="1.0"/>
            <dependency groupId="org.jvnet.staxex" artifactId="stax-ex" version="1.0"/>
            <dependency groupId="log4j" artifactId="log4j" version="1.2.13"/>
            <dependency groupId="javax.activation" artifactId="activation"
version="1.1.1"/>
        </artifact:dependencies>

    </target>

    <target name="generate" depends="init">
        <taskdef name="wsimport" classname="com.sun.tools.ws.ant.WsImport"
classpathref="generate.classpath"/>
    <wsimport fork="true" wsdl="https://172.20.111.233:8080/cmm/service/cmmNB.wsdl"
destdir="${bin.dir}"
                  package="com.cisco.nm.cmm.ws.client.jaxws" sourcedestdir="${gen.dir}"
verbose="true" debug="true">
         <produces dir="${bin.dir}/com/cisco/nm/cmm/ws/client/jaxws" includes="**/*" />
         <jvmarg line="-Djavax.net.ssl.trustStore=C:/ftp/keystore
-Djavax.net.ssl.trustStorePassword=cmm_dev"/>
        </wsimport>
    </target>


    <target name="build" depends="generate">
        <mkdir dir="${bin.dir}"/>
        <javac destdir="${bin.dir}">
            <src path="${src.dir}"/>
            <src path="${gen.dir}"/>
            <classpath refid="classpath"/>
        </javac>
        <copy todir="${bin.dir}">
            <fileset dir="${src.dir}">
                <exclude name="**/*.java"/>
            </fileset>
        </copy>
    </target>

    <target name="clean">
        <delete dir="${bin.dir}"/>
         <delete dir="${gen.dir}"/>
    </target>

    <target name="run" depends="cmmWebServicesClient"/>
```

```
    <target name="cmmWebServicesClient" depends="build">
        <java classname="com.cisco.nm.cmm.ws.client.jaxws.CmmWebServicesClient"
fork="true"
              failonerror="true">
            <classpath refid="classpath"/>
            <classpath location="${bin.dir}"/>
        <jvmarg line="-Djavax.net.ssl.trustStore=C:/ftp/keystore
-Djavax.net.ssl.trustStorePassword=cmm_dev"/>
       <arg value="https://172.20.111.233:8080/cmm/service/cmmNB.wsdl"/>
        </java>
    </target>

</project>
```

Step 3 Edit the Ant build script as follows:

a. Edit the taskdef for the *wsimport* task to specify the IP address of the CMM server.

- Locate the line that says:

```
wsimport fork="true" wsdl="http://172.20.110.23:8085/cmm/service/cmmNB.wsdl"
destdir="${bin.dir}"
```

- Specify the IP address of the CMM server.
- Specify the protocol used:

  If you are using HTTP, leave the protocol set to the default (wsdl = http://(ip_adddress)

  If you are using HTTP, change the protocol to *https*.

Step 4 Enter the following commands:

```
ant clean
ant generate
```

You should see code being generated. If no code is generated, you will see an error message on the console.

Step 5 Edit the client java file and enter the following command:

```
ant run
```

The Ant build script runs the **wsimport** task., which downloads the WSDL file and the schema file and generates code using the JAXB compiler.

The generated code is written to the *testWeb/client/jaxws/gen* directory.

Note The *gen* directory is defined as a property in the beginning on Ant script file.

Compiling with an IDE

If you are compiling with an IDE such as the Netbeans IDE, the compilation steps will be determined by the tool that you are using.

Here is a list of steps that you would use to compile using a sample compiler—the Netbeans IDE.

Step 1 On a Windows machine where the SDK is installed (*testWeb.zip*), log in to the compiler for example, Netbeans IDE.

Step 2 Create a project file.

a. Select **File > New Project.**

b. In the Choose Project dialog, select **Java Application**.

c. Uncheck the Create Main Class check box.

d. Click **Next.**

Step 3 Copy the contents of the */src* directory to the */src* directory in your TestWeb directory structure.

This is the files in the */com* directory.

Step 4 Go to the Springsource website and download Springsource 2.5.6.

Step 5 Add the Springsource */jar* files to your test project (under the Libraries folder).

Step 6 Go to the *applicationContext.xml* file in the CMM.

Step 7 Locate the following line:

```
<property name="defaultUri" value="http://172.20.110.42:8080/cmm/service/cmmNB.wsdl"/>
```

Step 8 Change the IP address in the *defaultUri* property line to the IP address of your CMM client.

Step 9 Run the compile (select **Run > Clean** and **Build Main Project).**

Step 10 Edit the application Java file (*CmmWebServicesClient.java*) and change the IP addresses in the generated code as required, to refer to your actual CMM device addresses.

Running Client Code

Step 1 Verify that the client machine has internet access.

Step 2 Unzip the *testWeb.zip* file on the client machine.

Step 3 Open the *build.xml* file and update the URL for server IP address

Note If you are using HTTPS, the port will always be 8080; for HTTP, port will be 8085.

Step 4 Open the *CmmWebServicesClient.java* file and update the domain name, device IP addresses, and other required information.