CHAPTER 1

Overview of Web Services API

The Cisco IP Interoperability and Collaboration System (IPICS) 4.0(x) application programming interface (API) provides a web services-based API that enables the management and control of various Cisco IPICS operations through programmatic interfaces and custom applications. The API includes a set of functions to control Cisco IPICS operations such as VTG creation, policy invocation, user management, and more.

This chapter introduces general concepts that relate to the web services API. It also explains how to generate a client stub by using the Eclipse integrated development environment (IDE), and explains options for executing API requests.

This chapter includes these topics:

Generating a Web Services Client Stub

Before using the Cisco IPICS API, you must generate a client stub on the PC that is to be used for the API development (the *developer workstation*).

You can use a variety of tools to generate a client stub. The following sections describe how to generate the stub by using the Eclipse IDE.

Obtaining and Configuring the Eclipse IDE

Before you generate a web services client stub for the Cisco IPICS API, follow these steps to obtain and configure the Eclipse IDE:

Step 1 On the developer workstation, download the Eclipse IDE for Java Developers package from the Galileo (v 3.5.2) release of the Eclipse open-source IDE.

This IDE is available from the Eclipse website.

Step 2 On the developer workstation, download and extract Axis2 version 1.3 from the Apache website

Step 3 Take these actions to configure Axis2 runtime in the Eclipse IDE:

a. Launch the Eclipse IDE.

b. Choose **Windows > Preferences**.

c. In the Preferences window. expand **Web Services**.

d. Click **Axis2 Preferences**.

e. Click **Browse** and, in the Browse to a Folder window, click the folder in which you extracted Axis2.

f. Click **OK**.

g. Click **OK** to exit the Preference window.

Generating a Client Stub on a Developer Workstation

The following procedure describes how to use the Eclipse IDE to generate a client stub on a developer workstation. Before you perform this procedure, configure the Eclipse IDE on the developer workstation as described in the "Obtaining and Configuring the Eclipse IDE" section on page 1-2.

Procedure

Step 1 Launch the Eclipse IDE and take these actions:

a. Choose **File > New > Other... > Web Services > Web ServiceClient**.

b. Click **Next**.

The Web Services Client window appears.

Step 2 In the Web Services Client window, take these actions:

a. In the service definition field, enter the following URL, where *cisco_IPICS_server* is the host name or IP address of your server:

http://*cisco_IPICS_server*/ipics_server/services/IpicsWebService?wsdl

b. Make sure that the Client type field displays **Java Proxy**.

c. Click the Server: **Tomcat *version* Server** hyperlink.

The Client Environment Configuration window appears.

Step 3 In the Client Environment Configuration window, take these actions:

a. In the Server area, choose the server type that Eclipse uses when generating the stub (by default, the server type is Tomcat v*X.X* Server.

b. Click **OK**.

Step 4 In the Web Services Client window, take these actions, click the **Web service runtime: Apache Axis** hyperlink.

The Client Environment Configuration window appears.

Step 5 In the Client Environment Configuration window, take these actions:

a. Click **Apache Axis2** in the Web service runtime area.

b. Click **OK**.

Step 6 In the Web Services Client window, take these actions, click the **Client project** hyperlink.

Step 7 In the Specify Client Project Settings window, take these actions:

a. In the Client project field, enter a name for the client project.

b. Make sure that the Client Project field displays **Dynamic Web Project**.

c. Click **OK**.

d. Click **Next**.

The Web Service Client window appears.

Step 8 In the Web Service Client window, click **Finish**.

The client stub is generated in a folder with the name that you specified in the Specify Client Project Settings window.

Creating and Executing API Client Code

Each API service function requires a valid session ID to access the API functions. To start a new web services session, the client code must call the startSession function and provide the login name and password that are used to authenticate the user who executes the API code. After authentication completes, the startSession method returns a unique SessionId value, which should be used when executing subsequent API methods in a web service session. To end a session, the API client code must call the endSession function, which invalidates the current SessionId and is important for security.

For additional information about using the API, see the "Function Guidelines" section on page 2-1.

After you create API client code, follow these steps to execute it:

Procedure

Step 1 Launch the Eclipse IDE and take these actions:

a. Click the name of the client file in the left panel of the window.

b. Choose **Run > Run Configurations...**.

The Run Configurations window appears.

Step 2 In the Run Configurations window, take these actions:

a. Right-click **Java Application** in the left panel of the window.

b. Choose **New**.

c. Click **Run**.

API Client Code Example

The following is an example of Cisco API client code. In this example:

- The location of the keystore is specified. The keystore contains the SSL certificate that the Cisco IPICS server uses to verify its identity when a client attempts to connect to the server, and the URL for the web service endpoint.
- The startSession function is executed. This function returns a unique sessionId, which is needed to execute other web service functions.
- The getIpicsVersion is executed. This function returns the Cisco IPICS version that is running on the Cisco IPICS sever.
- These VTG functions are executed:
  - createVtg—Create a new VTG on the Cisco IPICS server and returns the ID of the VTG, activates VTG and returns list of all VTG's present in the system respectively.
  - activateVTG—Activates the newly-created VTG.
  - getAllVtgs—Retrieves a list of all VTGs that are present in Cisco IPICS.
- The endSession function is executed. This function terminate the web service session. A terminated session cannot be used again.

```
package com.example;
import java.rmi.RemoteException;
import com.example.IpicsWebServiceStub.*;

public class IpicsWebServiceTestClient {
    // The web service stub
    private static IpicsWebServiceStub ws = null;

    //
    // Modify the following parameters as needed for your environment
    //

    // Update with your server's address.
    // Note: The use of HTTPS/SSL is optional, but highly recommended.
    private static final String WEB_SERVICE_ENDPOINT_URL =
        "https://<my_server_address>/ipics_server/services/IpicsWebService";

    // If you use SSL to connect to the server, update with the location
    // of your trusted SSL certificate key store (sometimes also called
    // a "trust store").
    private static final String KEY_STORE_FILE =
        "c:\\client_truststore.jks";

    // Choose the client type: IDC or IPHONE
    // Affects how clients show up in the Active Users page found in the
    // Cisco IPICS Administration Console.
    private static final String CLIENT_TYPE = "IDC";
```

```
// Update with the IPICS user login used to connect to the server.
// Use of the default "ipics" user account is not recommended, but
// shown here for simplicity.
private static final String USER_LOGIN = "ipics";

// Update with the corresponding IPICS user password.
private static final String USER_PASSWORD = "secret";


public static void main(String[] args) {
    String sessionId = null;
    try {
        // Note: Use Java keytool to import the server's SSL certificate
        // into KEY_STORE_FILE before running this client application,
        // otherwise your SSL connection will fail.
        System.setProperty("javax.net.ssl.trustStore", KEY_STORE_FILE);

        // Create web service stub object using end-point URL
        ws = new IpicsWebServiceStub(WEB_SERVICE_ENDPOINT_URL);

        sessionId = testStartSession();
        System.out.println("Session ID : " + sessionId);
        testGetIpicsVersion(sessionId);

        int vtgId = testCreateVtg(sessionId);

        testActivateVTG(sessionId, vtgId);

        testGetAllVtgs(sessionId);

        testEndSession(sessionId);

    } catch (Exception e) {
        e.printStackTrace();
    }
}

private static String testStartSession() throws Exception {
    ClientDescriptorVO cd = new ClientDescriptorVO();
    cd.setClientType(CLIENT_TYPE);

    StartSession params = new StartSession();
    params.setClientDescriptor(cd);
    params.setUserLogin(USER_LOGIN);
    params.setPass(USER_PASSWORD);

    SessionVO result = ws.startSession(params).get_return();

    if (result.getErrorVO() != null) {
        System.out.println(result.getErrorVO().getErrorMessages());
        return null;
    }
    String sessionId = result.getSessionId();
    return sessionId;
}

private static String testGetIpicsVersion(String sessionId)
        throws RemoteException {
    System.out.println("\n---------- GetIpicsVersion ---------- ");

    GetIpicsVersion params = new GetIpicsVersion();
    params.setSessionId(sessionId);
```

```
    StringVO result = ws.getIpicsVersion(params).get_return();

    if (result.getErrorVO() == null) {
        System.out.println("\nIPICS Version: " + result.getValue());
        return result.getValue();
    } else {
        System.out.println("\nIPICS Version: "
                + result.getErrorVO().getErrorMessages());
        return null;
    }
}

private static int testCreateVtg(String sessionId) {
    System.out.println("\n----------- Create VTG ----------- ");
    CreateVtg params = new CreateVtg();
    params.setSessionId(sessionId);
    params.setVtgName("vtg1");
    params.setDescription("Test VTG 1");

    IntegerVO result = new IntegerVO();
    try {
        result = ws.createVtg(params).get_return();
    } catch (RemoteException e) {
        e.printStackTrace();
    }
    if (result.getErrorVO() != null) {
        System.out.println(result.getErrorVO().getErrorMessages());
        return 0;
    }
    System.out.println("\n VTG Id : " + result.getValue());
    return result.getValue();
}

private static void testActivateVTG(String sessionId, int vtgId)
        throws Exception {
    System.out.println("\n----------- Activate VTG ----------- ");

    ActivateVtg params = new ActivateVtg();
    params.setSessionId(sessionId);
    params.setVtgId(vtgId);

    BooleanVO result = ws.activateVtg(params).get_return();

    if (result.getErrorVO() != null) {
        System.out.println(result.getErrorVO().getErrorMessages());
        return;
    }
    System.out.println("\n Return Value : " + result.getValue());
}

private static VtgContainerVO testGetAllVtgs(String sessionId)
    throws Exception
{
    System.out.println("\n----------- Get All VTGs ----------- ");

    PaginationContextVO pc = new PaginationContextVO();
    pc.setPageNumber(1);
    pc.setPageRecords(100);
    pc.setTotalRecords(100);

    GetAllVtgs params = new GetAllVtgs();
    params.setSessionId(sessionId);
    params.setPc(pc);
```

```
        VtgContainerVO result =
            ws.getAllVtgs(params).get_return();

        if (result.getErrorVO() != null) {
            System.out.println(result.getErrorVO().getErrorMessages());
            return null;
        }
        System.out.println("VTG container length : "
                + result.getVtgVOs().length);
        System.out.println("VTG Total Records : "
                + result.getPaginationContextVO().getTotalRecords());
        System.out.println("VTG Total Pages : "
                + result.getPaginationContextVO().getTotalPages());
        return result;
    }

    private static void testEndSession(String sessionId) throws Exception {
        System.out.println("\n---------- End Session ------------ ");
        EndSession params = new EndSession();
        params.setSessionId(sessionId);
        ws.endSession(params);
        System.out.println("Session ID deleted successfully.");
    }
}
```

API Security

The Cisco IPICS server and the Cisco IPICS API use SSL for secure communication between the server and client systems. The server uses a X.509 certificate (also called an *SSL certificate*) to verify its identity when a client attempts to connect to the server.

By default, the Cisco IPICS server provides a self-signed certificate, which a client typically rejects. To prevent a client from rejecting this certificate, take one of the of the actions that Table 1-1 describes.

Table 1-1 Methods for Preventing a Client from Rejecting the Cisco IPICS Server Self-Signed Certificate

Method	Remarks
Replace the Cisco IPICS server self-signed certificate with a certificate that is signed by a well-known certificate authority (CA), such as VeriSign or a local CA in your organization.	For additional information, see the “Installing Third Party Certificates on the Cisco IPICS Server” section in *Cisco IPICS Server Installation and Upgrade Guide*.

Table 1-1 Methods for Preventing a Client from Rejecting the Cisco IPICS Server Self-Signed Certificate (continued)

Method	Remarks
When using a Java client, configure the SSL libraries for your clients to trust the self-signed certificate by using the Java keytool to import the certificate into the client truststore.	Procedure: 1. Use SSH to log in to the Cisco IPICS server as the root user. 2. Enter the following command: **cd /opt/cisco/ipics/security** 3. Download the self-signed certificate named server.cert.pem to the client system. 4. Enter the following command. Replace *name* with a unique name for the alias, replace *certificate_file* that you downloaded to the client, and replace *path* with the path to the client keystore: **keytool -import -alias *name* -file *certificate_file* -keystore *path***
If you are using a client other than Java, configure the SSL libraries for your clients to trust the self-signed certificate.	You may be able to copy the self-signed certificate to a special directory on the client system. See your library documentation for detailed information.
Modify the way in which your client code validates certificate trust chains.	Some languages provide configurable SSL files that let you change the default certificate validation behavior.

A client verifies its identity with a user name and password that are sent to the server by the client application.

API Logging

The Cisco IPICS server logs all web service invocations. Entries include the date and time of the invocation, the API function that was used, and the status of the response.

This information is maintained in the Cisco IPICS log file. For more information about the log file, see *Cisco IPICS Administration Guide*.