

Cisco CallManager 4.0(1) AXL API Programming Guide

NOTE: To access all AXL SOAP API downloads and AXL requests and responses found in this document, please refer to the following url:

http://www.cisco.com/warp/public/570/avvid/voice_ip/axl_soap/axl_api_down.html

1 Introduction

The AVVID XML Layer (AXL) Application Programming Interface (API) provides a mechanism for inserting, retrieving, updating, and removing data from the database using an eXtensible Markup Language (XML) Simple Object Access Protocol (SOAP) interface. This allows a programmer to access Cisco CallManager data using XML and receive the data in XML form, instead of using a binary library or DLL.

The AXL API methods, known as *requests*, are performed using a combination of HTTP and SOAP. SOAP is an XML remote procedure call protocol. Users perform requests by sending XML data to the Cisco CallManager server. The server then returns the AXL *response*, which is also a SOAP message.

1.1 Target Audience for this Guide

This programming guide is designed for fairly experienced developers who would like access to one or more of the following items:

- Cisco CallManager data
- Cisco CallManager data in XML format
- Cisco CallManager data in a platform-independent manner

This guide assumes the developer has knowledge of a high-level programming language such as C++, Java, or an equivalent language. The developer must also have knowledge or experience in the following areas:

- TCP/IP Protocol
- Hypertext Transport Protocol
- Socket programming
- XML

In addition, the users of the AXL API and this programming guide must have a firm grasp of XML Schema, which was used to define the AXL requests, responses, and errors. For more information on XML Schema, please refer to the following URL:

<http://www.w3.org/TR/xmlschema-0/>.

- **Caution**

The AXL API gives enormous power to developers to modify the Cisco CallManager system database. The developer must take caution when using AXL, since each API call impacts the system. Abuse of the API can lead to dropped calls and slower system performance. AXL is not intended as a real-time API, but as a provisioning and configuration API.

1.2 New and Changed Information

This section describes the new or changed API calls for Release 4.0(1).

1.2.1 Added API Calls

- listAARGroupByName
- listVoiceMailProfileByName
- listGatekeeperByName
- addGatekeeper
- getGatekeeper
- removeGatekeeper
- updateGatekeeper
- addH323Phone
- addH323Gateway
- addH323Trunk
- getH323Phone
- getH323Gateway
- getH323Trunk
- removeH323Phone
- removeH323Gateway
- removeH323Trunk
- updateH323Phone
- updateH323Gateway
- updateH323Trunk
- executeSQLQuery

1.2.2 Changed API Calls

These changes may require changes to existing user code that makes use of the following:

- addPhone
- updatePhone
- getPhone
- addLine
- updateLine
- addUser
- removeUser

- updateUser
- getUser
- addDeviceProfile
- updateDeviceProfile
- getDeviceProfile
- addTransPattern
- updateTransPattern
- getTransPattern
- addRoutePattern
- updateRoutePattern
- getRoutePattern
- addRouteList
- updateRouteList
- getRouteList
- listUserByName
- addDevicePool
- updateDevicePool
- getDevicePool
- addGatewayEndpoint
- updateGatewayEndpoint
- getGatewayEndpoint
- addLocation
- updateLocation
- getLocation
- updateRegionMatrix
- addCTIRoutePoint
- updateCTIRoutePoint
- getCTIRoutePoint
- addMGCP Endpoint
- addVoiceMailPort
- updateVoiceMailPort
- getVoiceMailPort

2 AXL API

Request methods are XML structures that are passed to the AXL API server. The server receives the XML structures and executes the request. If the request completes successfully, then the appropriate AXL response is returned. All responses are named identically to the associated requests, except that the word "Response" has been appended.

For example, the XML response returned from an *addPhone* request is called *addPhoneResponse*.

If an error occurs, then an XML error structure is returned wrapped inside of a SOAP Fault structure (see Section 2.2 AXL Error Codes).

2.1 AXL Methods

The Server Impact column is a sample measuring of the time (in milliseconds) it takes for the AXL server to handle the request. The tests were performed on a 7835-1000 MCS server. These figures should only be used as a guideline, and can drastically change due to cluster configuration, hardware configuration, and the size of the database.

The database used for these tests had the following entries:

- 1 Conference Bridge
- 2 CTI Route Points
- 2 Media Termination Points
- 1 Music On Hold
- 61 Phones
- 1 Route List
- 2 Call Pick Up Groups
- 2 Call Parks
- 1 Conference
- 74 Directory Numbers
- 1 Route Pattern
- All Cisco services running

2.2 AXL Error Codes

If an exception occurs on the server, or if any other error occurs during the processing of an AXL request, then an error is returned in the form of a SOAP Fault message:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <SOAP-ENV:Fault>
      <faultcode>SOAP-ENV:Client</faultcode>
      <faultstring>
        <![CDATA[
          An error occurred during parsing
          Message: End element was missing the character '>'.

          Source = Line : 41, Char : 6
          Code : c00ce55f, Source Text : </re
          ]]>
        </faultstring>
      </SOAP-ENV:Fault>
    </SOAP-ENV:Body>
```

```
</SOAP-ENV:Envelope>
```

SOAP Fault messages can also contain more detailed information. The following is an example of a detailed SOAP Fault.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <SOAP-ENV:Fault>
      <faultcode>SOAP-ENV:Client</faultcode>
      <faultstring>Device not found with name SEP003094C39708.</faultstring>
      <detail xmlns:axl="http://www.cisco.com/AXL/1.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.cisco.com/AXL/1.0
          http://myhost/CCMApi/AXL/V1/axlsoap.xsd">
        <axl:error sequence="1234">
          <code>0</code>
          <message>
<![CDATA[
Device not found with name SEP003094C39708.
]]>
          </message>
          <request>doDeviceLogin</request>
        </axl:error>
      </detail>
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Error codes are included in the <detail> element of a SOAP Fault. The errors are represented by the axl:Error element. If a response to a request contains an <error> element, the user agent can determine the cause of the error by looking at the sub-elements of the <error> tag.

The following list describes the <error> element.

- **code**

The <code> element is a numerical value that is used by the user agent to find out what type of error has occurred. The error codes are as follows:

Less than 5000: These are errors that directly correspond to DBL Exception error codes. Please refer to the documentation for the DBLException class for explanations of these errors.

5000: Unknown Error: An unknown error occurred while processing the request. This can be due to a problem on the server, but can also be caused by errors in the request.

5002: Unknown Request Error: This error occurs if the user agent submits a request that is unknown to the API.

5003: Invalid Value Exception: This error occurs if an invalid value is detected in the XML request.

5004: AXL Unavailable Exception: This error occurs if the AXL service is too busy to handle the request at that time. The request should be sent again at a later time.

5005: Unexpected Node Exception: This error occurs if the server encounters an unexpected element. For example, if the server expects the next node to be <name>, but encounters <protocol>, then this error is returned. These errors are always caused by malformed requests that do not adhere to the latest AXL Schema.

- **message**

The <message> element is provided so that the user agent gets a detailed error message explaining the error.

- **request**

The <request> element is provided so that the user agent can determine what type of request generated this error. This element is optional; therefore it may not always appear.

3 Examples

The following examples describe how to make an AXL request and read back the response to the request. Each SOAP request must be sent to the web server via an HTTP POST. The endpoint URL is an ISAPI Extension DLL. The following list contains the only four required HTTP headers.

- POST /CCMApi/AXL/V1/soapisapi.dll

The first header identifies that this particular POST is intended for the AXL SOAP API. The AXL API only responds to the POST method.

- content-type: text/xml

The second header confirms that the data being sent to AXL is XML. If this header is not found then an HTTP 415 error is returned to the client.

- Authorization: Basic <some Base 64 encoded string>

The third header is the Base64 encoding of the user name and password for the administrator of the AXL Server. If authentication of the user fails, then an HTTP 401 Access Denied error is returned to the client.

- content-length: <a positive integer>

The fourth header is the length (in bytes) of the AXL request.

Note: Currently, the content-length cannot exceed 40 kilobytes. If a request is received, which is greater than 40 kilobytes, then an HTTP 413 error message is returned.

The following example contains an HTTP header for an AXL SOAP request:

```
POST /CCMApi/AXL/V1/soapisapi.dll
Host: axl.myhost.com:80
Accept: text/*
Authorization: Basic bGFycnk6Y3VybHkgYW5kIG1vZQ==
Content-type: text/xml
Content-length: 613
```

The following AXL request will be used in the code examples displayed in the following sections. This is an example of a getPhone request:

```
POST /CCMApi/AXL/V1/soapisapi.dll
Host: axl.myhost.com:80
Accept: text/*
Authorization: Basic bGFycnk6Y3VybHkgYW5kIGlvZQ==
Content-type: text/xml
Content-length: 613

<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <SOAP-ENV:Body>
    <axl:getPhone xmlns:axl="http://www.cisco.com/AXL/1.0"
xsi:schemaLocation="http://www.cisco.com/AXL/1.0 http://gkar.cisco.com/schema/axlsoap.xsd"
sequence="1234">
      <phoneNumber>SEP222222222245</phoneNumber>
    </axl:getPhone>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

3.1 C/C++ Example

This code example uses a hard-coded AXL request and sends it to the AXL Server running on the local system (localhost). It then reads the response back, outputting the response to the screen.

```
#include <winsock2.h> // required for sockets
#include <iostream> // required for console I/O
#include <sstream>
#include <string> // required for std::string

using namespace std;

int main(int argc, char* argv[])
{
  // make connection to server

  WSADATA WSAData;

  // initialize sockets
  if (int iError = WSASStartup (MAKEWORD(2,0), &WSAData))
  {
    cout << "Windows Sockets startup error. Aborting." << endl;
    return -1;
  }

  SOCKET Socket = socket (AF_INET, SOCK_STREAM, IPPROTO_TCP);

  if (Socket == INVALID_SOCKET)
  {
    cout << "Socket creation error. Aborting." << endl;
    return -1;
  }

  SOCKADDR_IN sinRemote;

  sinRemote.sin_family = AF_INET;
  sinRemote.sin_port = htons (80);
  sinRemote.sin_addr.s_addr = inet_addr( "127.0.0.1" );

  cout << "connecting to service" << endl;

  int retval = connect(Socket, (SOCKADDR *)&sinRemote, sizeof (sinRemote));

  if (retval != 0)
  {
    cout << "Error ocured while connecting to socket. Aborting." << endl;
  }
}
```

```

        closesocket(Socket);
        return -1;
    }

    const int BUFSIZE = 2048;
    char buff[BUFSIZE];           // the temporary receive buffer
    string strHTTPHeader;        // the HTTP Header
    string strAXLRequest;        // the AXL SOAP request

    // The AXL request: getPhone
    strAXLRequest = "<SOAP-ENV:Envelope xmlns:SOAP- \
ENV=\\"http://schemas.xmlsoap.org/soap/envelope/\" \
xmlns:xsi=\\"http://www.w3.org/2001/XMLSchema-instance\" \
xmlns:xsd=\\"http://www.w3.org/2001/XMLSchema\"> \
<SOAP-ENV:Body> \
<axl:getPhone xmlns:axl=\\"http://www.cisco.com/AXL/1.0\" \
csi:schemaLocation=\\"http://www.cisco.com/AXL/1.0 \
http://gkar.cisco.com/schema/axlsoap.xsd\" sequence=\\"1234\"> \
<phoneNumber>SEP222222222245</phoneNumber> \
</axl:getPhone> \
</SOAP-ENV:Body> \
</SOAP-ENV:Envelope>";

    // temporarily use the buffer to store the length of the request
    sprintf(buff, "%d", strAXLRequest.length());

    // build the HTTP header
    strHTTPHeader = "POST /CCMApi/AXL/V1/soapisapi.dll\r\n \
Host: localhost:80\r\n \
Authorization: Basic bGFycnk6Y3VybhkgYW5kIGlvZQ==\r\n \
Accept: text/*\r\n \
Content-type: text/xml\r\n \
Content-length: ";

    strHTTPHeader += buff;
    strHTTPHeader += "\r\n\r\n";

    // put the HTTP header and SOAP XML together
    strAXLRequest = strHTTPHeader + strAXLRequest;

    // send these bytes to the socket
    retval = send (Socket, strAXLRequest.c_str(), strAXLRequest.length(), 0);
    if ( retval != SOCKET_ERROR)
    {
        // output response
        cout << "received response: " << endl;

        int iTotalRead = 0;
        // read BUFSIZE at a time, writing to another ostream
        do {
            iNumRead = recv (Socket, buff, BUFSIZE-1, 0);
            buff[iNumRead] = NULL;

            cout << buff;
            iTotalRead += iNumRead;

        } while (iNumRead == BUFSIZE-1);

        cout << "Read " << iTotalRead << " bytes." << endl;
    }
    else
    {
        cout << "An error occured while sending the data to socket." << endl;
    }

    // all finished, close socket
    closesocket(Socket);

    return 0;
}

```

3.2 Java Example

This code example uses a hard-coded AXL request and sends it to the AXL Server running on the local system (localhost). It then reads the response back, outputting the response to the screen.

```

import java.io.*;
import java.net.*;

public class main
{
public static void main(String[] args)
{
    //Declare references
    String sAXLSOAPRequest = null;          // will hold the complete request,
                                            // HTTP header and SOAP payload

    String sAXLRequest = null;             // will hold only the SOAP payload

    Socket socket = null;                  // socket to AXL server
    OutputStream out = null;               // output stream to server
    InputStream in = null;                 // input stream from server
    byte[] bArray = null;                  // buffer for reading response from server

    // Build the HTTP Header
    sAXLSOAPRequest = "POST /CCMApi/AXL/V1/soapisapi.dll\r\n";
    sAXLSOAPRequest += "Host: localhost:80\r\n";
    sAXLSOAPRequest += "Authorization: Basic bGFycnk6Y3VybhkgYW5kIGlvZQ==\r\n";
    sAXLSOAPRequest += "Accept: text/*\r\n";
    sAXLSOAPRequest += "Content-type: text/xml\r\n";
    sAXLSOAPRequest += "Content-length: ";

    // Build the SOAP payload
    sAXLRequest = "<SOAP-ENV:Envelope xmlns:SOAP-
ENV=\"http://schemas.xmlsoap.org/soap/envelope/\" ";
    sAXLRequest += "xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"
xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\"> ";
    sAXLRequest += "<SOAP-ENV:Body> <axl:getPhone
xmlns:axl=\"http://www.cisco.com/AXL/1.0\" ";
    sAXLRequest += " xsi:schemaLocation=\"http://www.cisco.com/AXL/1.0
http://gkar.cisco.com/schema/axlsoap.xsd\" ";
    sAXLRequest += "sequence=\"1234\"> <phoneNumber>SEP22222222245</phoneNumber> ";
    sAXLRequest += "</axl:getPhone> </SOAP-ENV:Body> </SOAP-ENV:Envelope>";

    // finish the HTTP Header
    sAXLSOAPRequest += sAXLRequest.length();
    sAXLSOAPRequest += "\r\n\r\n";

    // now add the SOAP payload to the HTTP header, which completes the AXL SOAP
request
    sAXLSOAPRequest += sAXLRequest;

    // now that the message has been built, we can connect to server and send it
try
{
    socket = new Socket("localhost", 80);

    out = socket.getOutputStream();
    in = socket.getInputStream();

    // send the request to the host
    out.write(sAXLSOAPRequest.getBytes());

    // read the response from the host
    StringBuffer sb = new StringBuffer(2048);
    bArray = new byte[2048];
    int ch = 0;
    int sum = 0;
    while ( (ch = in.read(bArray)) != -1 )

```

```

        {
            sum += ch;
            sb.append(new String(bArray, 0, ch));
        }

        socket.close();

        // output the response to the standard out
        System.out.println(sb.toString());

    } catch (UnknownHostException e)
    {
        System.err.println("Error connecting to host: " + e.getMessage());
        return;
    } catch (IOException ioe)
    {
        System.err.println("Error sending/receiving from server: " +
ioe.getMessage());

        // close the socket
        try
        {
            if (socket != null) socket.close();
        } catch (Exception exc)
        {
            System.err.println("Error closing connection to server: " +
exc.getMessage());
        }
        return;
    }
}

```

4 Throttling of Requests

The side-effects of updating the Cisco CallManager database can adversely affect system performance; therefore, the system administrator is capable of controlling how many AXL requests are allowed to update the database per minute. This value is controlled via the Database Layer service parameter “MaxAXLWritesPerMinute”.

AXL accommodates all requests until the “MaxAXLWritesPerMinute” value is reached. Subsequent attempts to modify the database with AXL are rejected with an HTTP 503 Service Unavailable response. Every minute, AXL resets its internal counter and begins to accept AXL update requests until the limit gets reached again.

The following AXL requests are considered “Reads” and do not count against the “MaxAXLWritesPerMinute” service parameter. All other AXL requests not in the following list count against the service parameter:

- executeSQLQuery
- doDeviceReset
- all AXL “get” requests
- all AXL “list” requests

5 The AXL Schema Documentation

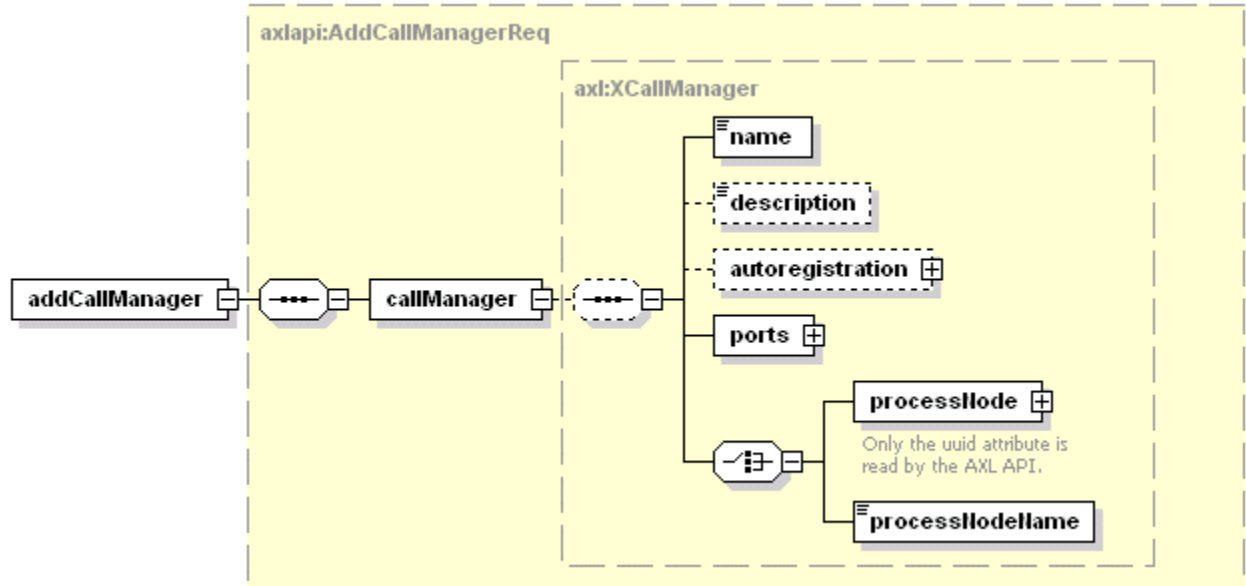
The complete AXL schema (including details of all requests, responses, XML objects, data types, etc.) is encapsulated in the following four files found on the Cisco CallManager server: axlsoap.xsd, axl.xsd, axlmessage.xsd, and AXLEnums.xsd. The default path is: C:\CiscoWebs\API\AXL\V1.

Standard XML handling IDEs and development environments can illuminate or auto-generate 'friendly' formatted documents based on the AXL schema files.

The following describes a complete auto-generated HTML document based on the schema that is available for download from the Developer Services website at the following URL:

<http://www.cisco.com/go/developersupport> (follow the 'Supported Products' link).

5.1 Example XML Structure



Request or Response	Element Name	Description
	Complex Element	A complex element can have child elements, as well as attributes. An element with a solid border is required to appear in an XML instance document.
	Simple Element	A simple element cannot have child elements but can have attributes. An element with a solid border is required to appear in an XML instance document.
	Optional Element	An optional element is not required to appear in an instance of the XML. Any type of element can be optional, including sequence and choice elements.
	Sequence Element	A sequence means that all children of this element must appear in the XML in the order that they are listed.
	Choice Element	A choice element means that only one of the children of this element can appear in the XML.

6 Authentication

Anonymous access to the AXL SOAP service should be deactivated to enforce user authentication. User authentication gets controlled via the HTTP Basic Authentication scheme. Therefore, you must include the Authorization header in the HTTP Header.

For example, if the user agent wishes to send the userid "larry" and password "curly and moe", it would use the following header field:

```
Authorization: Basic bGFycnk6Y3VybHkgYW5kIG1vZQ==
```

Where the string "bGFycnk6Y3VybHkgYW5kIG1vZQ==" is the Base 64 encoding of "larry:curly and moe".

6.1 Data Encryption

If the user agent wishes to encrypt the AXL SOAP message, then the user agent must use HTTP SSL.

SSL is not functional on the web server by default. The customer must request and install a SSL Certificate from a certified authority such as VeriSign. Once the SSL Certificate has been installed on the web server, AXL requests can be made using the "https" protocol in lieu of "http".

7 Integration considerations and Inter-operability

The AXL API gives enormous power to developers to modify the Cisco CallManager system database. The developer must take caution when using AXL, since each API call impacts the system. Abuse of the API can lead to dropped calls and slower system performance. AXL is not intended as a real-time API, but as a provisioning and configuration API.

If AXL is determined to be using too much of the CPU time, consider lowering the MaxAXLWritesPerMinute service parameter. If this does not solve the problem, consider purchasing a second server to be used only by applications using AXL.

Copyright © 2001-2004. Cisco Systems, Inc. All rights reserved.