CHAPTER 36

AXL Programming

To access all AXL SOAP API downloads and AXL requests and responses found in this chapter, refer to http://www.cisco.com/pcgi-bin/dev_support/access_level/product_support

This chapter contains the following sections:

Introduction

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

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

Target Audience for this Chapter

This chapter targets somewhat experienced developers who need to access one or more of the following items:

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

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

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

In addition, users of the AXL API must have a firm grasp of XML Schema, which is used to define the AXL requests, responses, and errors. For more information on XML Schema, refer to http://www.w3.org/TR/xmlschema-0/.

Caution The AXL API allows you to modify the Cisco Unified CallManager system database. AXL acts as a provisioning and configuration API, not as a real-time API. You must use caution when using AXL, because each API call impacts the system. Misuse of the API can lead to dropped calls and slower performance.

New and Changed Information

AXL APIs

The following list provides AXL API calls that are new in Release 5.1:

- addSIPRealm
- updateSIPRealm
- getSIPRealm
- removeSIPRealm

These APIs add and update credentials (passwordreserve) in siprealm.

New Service Parameter

Cisco Unified CallManager Administration 5.1 release adds a new service parameter, “Send Valid Namespace in AXL Response,” under the Cisco Database Layer Monitor service. This parameter determines the namespace that gets sent in the AXL response from Cisco Unified CallManager.

When this parameter specifies True, Cisco Unified CallManager sends the valid namespace (that is, http://www.cisco.com/AXL/API/1.0) in the AXL response so the namespace matches the AXL schema specification.

If the parameter specifies False, Cisco Unified CallManager sends an invalid namespace (that is, http://www.cisco.com/AXL/1.0) in the AXL response, which does not match the AXL schema specification.

The default service parameter value specifies False to maintain backward compatibility with the AXL response in the Cisco Unified CallManager 5.0 release. Cisco recommends that you set this parameter to True so Cisco Unified CallManager sends the valid namespace.

AXL API

Request methods represent 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, the appropriate AXL response gets returned. All responses get named identically to the associated requests, except that the word “Response” is appended.

For example, the XML response returned from an addPhone request gets named addPhoneResponse.

If an error occurs, an XML error structure gets returned wrapped inside a SOAP Fault structure (see the “AXL Error Codes” section on page 36-3).

AXL Compliance

The Cisco Unified CallManager AXL implementation complies with XML Schema 1.0, which was tested for XML Schema compliance with a third-party application that is called XML Spy version 4.x. Early versions of the MSXML schema validator did not support enough of the XML Schema 1.0 recommendation to be used.

The Cisco Unified CallManager AXL implementation also complies with SOAP 1.1 as defined by the World Wide Web Consortium as well as HTTPS 1.1. The AXL API runs as an independent service that can be accessed only via HTTPS.

AXL Error Codes

If an exception occurs on the server, or if any other error occurs during the processing of an AXL request, an error gets 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 example depicts a detailed SOAP Fault.

```
<SOAP-ENV:Envelope  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" S
OAP-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/CMApi/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>
```

The <detail> element of a SOAP Fault includes error codes. The axl:Error element represents the errors. If a response to a request contains an <error> element, the user agent can determine the cause of the error by looking at the subelements of the <error> tag.

The following list describes the <error> elements:ws The user agent uses the <code> element, a numerical value, to find what type of error occurred. The error codes follow:

Error Code	Description
Less than 5000	These errors directly correspond to DBL Exception error codes. Refer to the documentation for the DBLException class for explanations of these errors.
5000	**Unknown Error**—An unknown error occurred while the request was processed. This can occur due to a problem on the server but can also be due to errors in the request.
5001	**Parser Error**—An error occurred while the XML request was being parsed.
5002	**Unknown Request Error**—The user agent submitted a request that is unknown to the API.
5003	**Invalid Value Exception**—The API detected an invalid value in the XML request.
5004	**AXL Unavailable Exception**—This error indicates that the AXL service is too busy to handle the request at this time. The request should be sent again at a later time.

Error Code	Description
5005	**Unexpected Node Exception**—The server encountered an unexpected element. For example, if the server expects the next node to be <name>, but encounters <protocol>, this error gets returned. Malformed requests that do not adhere to the latest AXL Schema will cause these errors.
5007	**Item Not Valid Error**—The specified item gets identified as invalid, which means that it does not exist or that it was specified incorrectly at input.

message

The system provides the <message> element so the user agent gets a detailed error message that explains the error.

request

The system provides the <request> element so the user agent can determine the type of request that generated this error. Because this element is optional, it may not always appear.

Example AXL Requests

No platform considerations exist in Cisco Unified CallManager Release 5.1. The client must be able to send an HTTPS request to the AXL endpoint.

The following examples describe how to make an AXL request and read back the response to the request.

Ensure each SOAP request is sent to the web server via an HTTPS POST. The endpoint URL represents the AXL web service that is running on a Cisco CallManager server. The following list contains the only four required HTTPS headers.

- POST :8443/axl/

  The first header specifies that this particular POST is intended for the Cisco AXL Web Service. The AXL API only responds to the POST method.

- content-type: text/xml

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

- Authorization: Basic <some Base 64 encoded string>

  The third header gives the Base64 encoding of the user name and password for the administrator of the AXL Server. Because Base64 encoding takes three 8-bit bytes and represents them as four printable ASCII characters, if the encoded header does not contain an even multiple of four ASCII characters (16, 20, 24, and so on), you must add padding characters (=) to complete the final group of four as in the following examples.

  If authentication of the user fails, an HTTP 401 Access Denied error gets returned to the client.

- content-length: <a positive integer>

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

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

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

```
POST :8443/axl
Host: axl.myhost.com:80
Accept: text/*
Authorization: Basic bGFycnk6Y3VybHkgYW5kIG1vZQ==
Content-type: text/xml
Content-length: 613
```

The following AXL request gets used in the code examples that display in the following sections. This example shows a getPhone request:

```
POST :8443/axl
Host: axl.myhost.com:80
Accept: text/*
Authorization: Basic bGFycnk6Y3VybHkgYW5kIG1vZQ==
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://ccmserver/schema/axlsoap.xsd"
sequence="1234">
            <phoneName>SEP222222222245</phoneName>
        </axl:getPhone>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

C or C++ Example

This code example uses a hard-coded AXL request and sends it to the AXL Server that is running on the local system (localhost).

It then reads the response and outputs the response to the screen.

```
#include <sys/socket.h>
#include <sys/types.h>
#include <stdlib.h>
#include <openssl/ssl.h>
#include <stdio.h>
#include <unistd.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <strings.h>
#include <openssl/x509.h>
#include <openssl/crypto.h>
#include <iostream>
#include <string>
using namespace std;
typedef unsigned char byte;

void encodeBase64( const string& inBuf, string &outBuf )
{
    unsigned int i;
    unsigned int j;
    bool hiteof = false;
    byte dtable[256];
```

```
    outBuf.erase();

    for(i= 0;i<9;i++)
    {
       dtable[i]= 'A'+i;
       dtable[i+9]= 'J'+i;
       dtable[26+i]= 'a'+i;
       dtable[26+i+9]= 'j'+i;
    }
    for(i= 0;i<8;i++)
    {
       dtable[i+18]= 'S'+i;
       dtable[26+i+18]= 's'+i;
    }
    for(i= 0;i<10;i++)
    {
       dtable[52+i]= '0'+i;
    }

    dtable[62]= '+';
    dtable[63]= '/';

    j = 0;
    while(!hiteof)
    {
       byte igroup[3],ogroup[4];
       int c,n;

       igroup[0]= igroup[1]= igroup[2]= 0;
       for(n= 0;n<3;n++){
          if( j < inBuf.size() )
          {
             c = inBuf[j++];
          } else
          {
             hiteof = true;
             break;
          }
          igroup[n]= (byte)c;
       }
       if(n> 0)
       {
          ogroup[0]= dtable[igroup[0]>>2];
          ogroup[1]= dtable[((igroup[0]&3)<<4)|(igroup[1]>>4)];
          ogroup[2]= dtable[((igroup[1]&0xF)<<2)|(igroup[2]>>6)];
          ogroup[3]= dtable[igroup[2]&0x3F];

          if(n<3)
          {
             ogroup[3]= '=';
             if(n<2)
             {
                ogroup[2]= '=';
             }
          }
          for(i= 0;i<4;i++)
          {
             outBuf += ogroup[i];
          }
       }
    }
}
```

```
string getAuthorization()
{
  string m_encode64,name;
  //You should change name to your own axl server user name and passwd
  //in this example,"CCMAdministrator" is user name, "cisco_cisco" is passwd.
  name="CCMAdministrator:cisco_cisco";
  encodeBase64(name,m_encode64);
  return m_encode64;
}

void
BuildDeviceNameSQL(string &buf,  // Buffer to build AXL
                                  string& deviceNumber, // DN
                                  string& seqNum )
{
    const int BUFSIZE = 2048;
    char buff[BUFSIZE];           // Temp buffer
    string strHTTPHeader;         // HTTP/AXL Header
    string strAXLRequest;         // AXL Request

    strAXLRequest = "<SOAP-ENV:Envelope xmlns:SOAP-ENV=";
    strAXLRequest += "\"http://schemas.xmlsoap.org/soap/envelope/\"";
    strAXLRequest += " xmlns:SOAP-ENC=\"http://schemas.xmlsoap.org/soap/encoding/\"";
    strAXLRequest += " xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"";
    strAXLRequest += " xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\"> ";
    strAXLRequest += "<SOAP-ENV:Body> ";
    strAXLRequest += "<m:executeSQLQuery xmlns:m=\"http://www.cisco.com/AXL/API/1.0\"
sequence=\"" + seqNum + "\"> ";
    strAXLRequest += "<m:sql> ";
    strAXLRequest += "SELECT * FROM Device ";
    strAXLRequest += "</m:sql> ";
    strAXLRequest += "</m:executeSQLQuery> ";
    strAXLRequest += "</SOAP-ENV:Body> ";
    strAXLRequest += "</SOAP-ENV:Envelope>";

    strHTTPHeader = "POST /axl/ HTTP/1.1\r\n";

    strHTTPHeader += "Host: localhost:8443\r\n";
    strHTTPHeader += "Accept: text/*\r\n";
    //strHTTPHeader += "Authorization: Basic YWRtaW5pc3RyYXRvcjpjaXNjbw== \r\n";

    strHTTPHeader += "Authorization: Basic ";
    strHTTPHeader += getAuthorization() + "\r\n";

    strHTTPHeader += "Content-type: text/xml\r\n";
    strHTTPHeader += "Content-length: ";

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

    strHTTPHeader += buff;
    strHTTPHeader += "\r\nConnection: Keep-Alive";
    strHTTPHeader += "\r\n\r\n";

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

    return;

}
```

```
int main(int argc, char** argv)
{
  struct sockaddr_in saddr;
  SSL_METHOD *meth;
  SSL_CTX *sslctx;
  SSL *ssl;
  X509* server_cert;
  string buff,line,seqnum;
  char buffer[2048];
  int status,error;
  char *str;

  if( argc!=3 )
  {
     printf("Usage : ssltest <ip> <port> \n");
     printf("Usage : the default port is 8443 \n");
     printf("Usage : the ip is the ip of ccm5.0 \n");
     printf("Example: ssltest 10.77.31.168 8443 \n");

     exit(2);
  }

  int sock=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);
  if(sock<0)
  {
    printf("create socket failed\n");
    exit(1);
  }
  saddr.sin_family=AF_INET;
  saddr.sin_port=htons(atoi(argv[2]));
  saddr.sin_addr.s_addr =inet_addr(argv[1]);
  status=connect(sock,(struct sockaddr *)&saddr,sizeof(saddr));
  if(status<0)
  {
    printf("connect to %s failed\n",argv[1]);
    exit(2);
  }
  SSL_library_init();
  meth=TLSv1_client_method();
  sslctx=SSL_CTX_new(meth);
  if(!sslctx)
  {
     printf("SSL_CTX_new failed\n");
     close(sock);
     exit(3);
  }
  SSL_CTX_set_verify(sslctx,SSL_VERIFY_NONE,NULL);
  ssl =SSL_new(sslctx);
  if(!ssl)
  {
    printf("SSL_new failed\n");
    close(sock);
    exit(4);
  }
  status=SSL_set_fd(ssl,sock);
  if(!status)
  {
    printf("SSL_set_fd failed\n");
    close(sock);
    exit(5);
  }
  SSL_set_mode(ssl,SSL_MODE_AUTO_RETRY);
  status=SSL_connect(ssl);
  error=SSL_get_error(ssl,status);
```

```
  switch(error)
  {
    case SSL_ERROR_NONE:
      printf("connect successful\n");
      break;
    case SSL_ERROR_ZERO_RETURN:
      printf("peer close ssl connection \n");
      break;
    default:
      printf("connect error is %d\n",error);
   }

   server_cert = SSL_get_peer_certificate (ssl);
   if(!server_cert)
   {
     printf("get server certificate failed!\n");
     SSL_shutdown(ssl);
     close(sock);
     exit(6);
   }
   str= X509_NAME_oneline(X509_get_subject_name (server_cert),0,0);
   if(str)
   {
     printf("subject :%s\n",str);
   }
   else
     printf("subject is empty\n");
   str = X509_NAME_oneline (X509_get_issuer_name  (server_cert),0,0);
   if(!str)
     printf("issuer name is :%s\n",str);
   else
     printf("issuer name is empty \n");
   line="12";
   seqnum="1234";
   BuildDeviceNameSQL(buff,line,seqnum);
   SSL_write(ssl,buff.c_str(),buff.length());
   printf("\n");
   printf("\n");
   printf("Request sent is:\n");
   printf(buff.c_str());
   printf("\n");
   printf("\n");
   SSL_read(ssl,buffer,sizeof(buffer));

   printf("Response from server is: \n%s\n",buffer);
   status=SSL_shutdown(ssl);
   if(status==1)
     printf("shutdown successful\n");
   else
     printf("\nshutdown error code is %d\n",status);
   close(sock);
}
```

Java Example

This code example uses a hard-coded AXL request and sends it to the AXL server that is running on the local system (localhost). It then reads the response and outputs 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,
                                       // HTTPS 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 HTTPS Header
    sAXLSOAPRequest = "POST :8443/axl\r\n";
    sAXLSOAPRequest += "Host: localhost:80\r\n";
    sAXLSOAPRequest += "Authorization: Basic bGFycnk6Y3VybHkgYW5kIG1vZQ==\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://ccmserver/schema/axlsoap.xsd\" ";
    sAXLRequest += "sequence=\"1234\"> <phoneName>SEP222222222245</phoneName> ";
    sAXLRequest += "</axl:getPhone> </SOAP-ENV:Body> </SOAP-ENV:Envelope>";

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

    // now add the SOAP payload to the HTTPS 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;
       }
```

In addition to these examples, refer to the AXL Sql Toolkit, which is available for download from the Cisco Unified CallManager server at https://ccmserver:8443/plugins/axlsqltoolkit.zip.

Throttling of Requests

The side effects of updating the Cisco Unified CallManager database can adversely affect system performance; therefore, the system administrator can control how many AXL requests are allowed to update the database per minute. You can control this value by using 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 HTTPS 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, which are considered “Reads,” do not count against the “MaxAXLWritesPerMinute” service parameter. All other AXL requests that are not in the following list count against the service parameter:

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

The AXL Schema Documentation

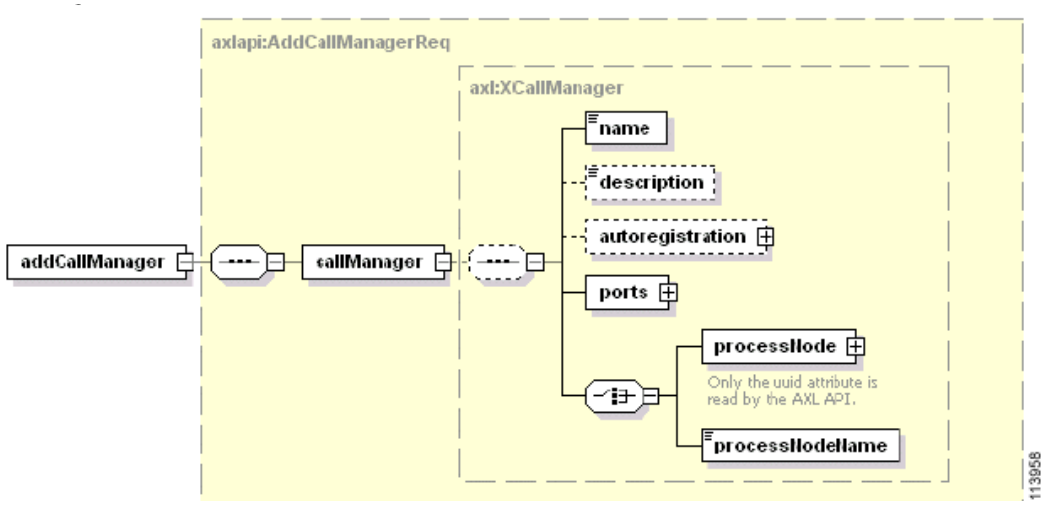
The axlsqltoolkit.zip plugin contains the following six AXL schema files: AXLAPI.wsdl, AXLEnums.xsd, axlmessage.xsd, axlsoap.xsd, axl.xsd, and SoapEnvelope.xsd. These files encapsulate the complete AXL schema (including details of all requests, responses, XML objects, data types, and so on).

The Cisco developer support program also provides the AXL schema files at http://www.cisco.com/pcgi-bin/dev_support/access_level/product_support.

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

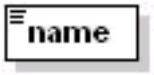
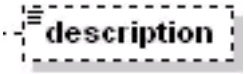
The following example describes a complete auto-generated HTML document based on the schema.

Example XML Structure



The AXL schema, which is provided as an HTML document, graphically describes each request and response. (See http://www.cisco.com/pcgi-bin/dev_support/access_level/product_support.)

The following legend explains the graphics that are used in the AXL schema document.

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 must 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 must appear in an XML instance document.
	Optional Element	An optional element need not appear in an instance of the XML. Any type of element can be optional, including sequence and choice elements.

Request or Response	Element Name	Description
	Sequence Element	A sequence element means that all children of this element must appear in the XML in the order in which they are listed.
	Choice Element	A choice element means that only one child of this element can appear in the XML.

Authentication

Deactivate anonymous access to the AXL SOAP service to enforce user authentication. User authentication gets controlled via the HTTPS Basic Authentication scheme; therefore, you must include the Authorization header in the HTTPS header.

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

Authorization: Basic bGFycnk6Y3VybHkgYW5kIG1vZQ==

where the string “bGFycnk6Y3VybHkgYW5kIG1vZQ==” provides the Base64 encoding of “larry:curly and moe.”

Note The two “equals” characters (=) at the end of the string act as padding characters for Base64 encoding.

Data Encryption

Encrypt AXL SOAP messages by using HTTP SSL. SSL remains functional on the web server by default. AXL requests get made by using the “https” protocol.

Integration Considerations and Interoperability

The AXL API gives much power to developers to modify the Cisco Unified CallManager system database. The developer must use caution when using AXL, because each API call impacts the system. Abuse of the API can lead to dropped calls and slower system performance. AXL acts as a provisioning and configuration API, not as a real-time API.

If AXL is determined to be using too much 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 that are using AXL.

Note Because AXL is not a real-time API, the autologout function of Extension Mobility does not work when the user is logged in/out of EM via the AXL interface.