



CHAPTER 1

AXL Configuration Programming

The Administrative XML Layer (AXL) Application Programming Interface (API) provides a mechanism for inserting, retrieving, updating, and removing data from the Cisco Unified Communications Manager database by using an eXtensible Markup Language (XML) Simple Object Access Protocol (SOAP) interface. This allows a programmer to access the database 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 HTTPS and SOAP. SOAP is an XML remote procedure call (RPC) protocol. Users perform requests by sending XML data to the Cisco Unified Communications Manager server. The server then returns the AXL response, which is also a SOAP message.

The AXL-SOAP web service is enabled by default on all Cisco Unified Communications Manager servers. It requires no other installation or configuration.

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

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 \(specifically HTTPS\)](#)
- Socket programming
- [XML](#)



Tip

Users of the AXL API must have a firm grasp of XML syntax and 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/>.

For more information on XML syntax/grammar, refer to <http://www.w3.org/TR/rdf-syntax-grammar/>.



Caution

The AXL API allows you to modify the Cisco Unified Communications Manager system database. 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. AXL should act as a provisioning and configuration API, not as a real-time API.

This chapter contains the following sections:

- [New and Changed Information](#), page 1-2
- [AXL API](#), page 1-10
- [Example AXL Requests](#), page 1-11
- [Throttling of Requests](#), page 1-20
- [AXL Schema Documentation](#), page 1-20
- [Authentication](#), page 1-21
- [Data Encryption](#), page 1-21
- [Integration Considerations and Interoperability](#), page 1-21
- [Impact on Backward Compatibility](#), page 1-22
- [Post-Installation Steps and Troubleshooting on the Linux Platform](#), page 1-22
- [Using the AXL API with AXIS](#), page 1-28
- [Using the AXL API in a .NET Environment](#), page 1-29
- [Returned Namespace for AXIS and .NET Applications](#), page 1-33

New and Changed Information

Release 6.0(1) makes the following major changes in the AXL APIs for :

- Introduced AXL schema versioning (see [AXL Versioning Support](#)).
- Added the following three fields into the AXL device/line create APIs:
 - ASCII Display (internal call ID)
 - Secondary Calling Search Space for Forward All
 - Unattended Port
- Added new AXL APIs and attributes that are needed to support Mobility provisioning and added optional attributes to existing AXL objects.
- Added the new tag cfaCSSPolicy in the Line API.
- Added Credential Policy support APIs.
- Added information about AXL versioning.
- Changed the default value of the AXL service parameter, so it allows a valid namespace to be returned in AXL responses.
- In Cisco Unified Communications Manager Release 6.0(1), the AXL API now considers PKID with or without curly brackets as a valid input. For example, in an addLine request for the DevicePool tag, PKID can be mentioned either as “{xxx-xxx...}” or “xxx-xxx...”; both represent acceptable inputs for the AXL request.

AXL Versioning Support

To improve backward compatibility, Cisco Unified Communications Manager Release 6.0(1) introduces AXL schema versioning. The system duplicates the previous AXL 1.0 schema as the AXL 6.0(1) schema, and, in upcoming releases, the AXL schema will be numbered the same as the corresponding Cisco Unified Communications Manager release. This approach maintains AXL backward compatibility for one full release cycle.

Developers have the option in Cisco Unified Communications Manager Release 6.0(1) to request a specific AXL schema version (either 1.0 or 6.0(1)); however, both these schema versions are identical. If a specific schema version is not requested, the AXL 1.0/6.0(1) schema automatically gets used and an appropriate response gets returned, which will have no backward compatibility issues for several releases.

Cisco highly recommends that developers include the version of AXL schema on which an AXL request is based because support for unversioned requests might be removed in future releases of Cisco Unified Communications Manager.

For those developers who are using the AXL APIs `executeSQLQuery` and `updateSQLQuery`, changes have occurred to the Cisco Unified Communications Manager 6.0(1) database schema that affect the direct SQL query approach. Refer to the Cisco Unified Communications Manager *Database Dictionary, Release 6.0(1)*, which describes the specific changes in the database schema.

To help developers plan for AXL versioning, [Table 1-1](#) provides the approach that Cisco will follow in supporting upcoming releases.

- Cisco will support AXL requests **without** version information for only three releases following the 6.0(1) release; after that, requests without version information might be rejected.
- AXL requests **with** version information will have the corresponding schema applied for up to three subsequent releases; after that, the specified version might not be available.

Table 1-1 AXL Versioning and Schema Plan for Future Releases

	AXL Request no version specified	AXL Request with Version Specified					
		Release 6.0	Plus 1 release	Plus 2 releases	Plus 3 releases	Plus 4 releases	
Cisco Unified Communications Manager Release	Release 6.0	6.0 schema applied	6.0 schema applied				
	Plus 1 release	6.0 schema applied	6.0 schema applied	Plus 1 schema applied	Not Applicable		
	Plus 2 releases	6.0 schema applied	6.0 schema applied	Plus 1 schema applied	Plus 2 schema applied		
	Plus 3 releases	6.0 schema applied	6.0 schema applied	Plus 1 schema applied	Plus 2 schema applied	Plus 3 schema applied	
	Plus 4 releases	Request rejected		Plus 1 schema applied	Plus 2 schema applied	Plus 3 schema applied	Plus 4 schema applied
	Plus 5 releases	Request rejected	Schema no longer available		Plus 2 schema applied	Plus 3 schema applied	Plus 4 schema applied

The following is a sample AXL request carries version information:

```
POST /axl/ HTTP/1.0
Host:10.77.31.194:8443
Authorization: Basic Q0NNQWRtaW5pc3RyYXRvcjppjaXNjb19jaXNjbw==
Accept: text/*
Content-type: text/xml
SOAPAction: "CUCM:DB ver=6.0"
Content-length: 427

<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:getUser 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"> <userid>tttt</userid> </axl:getUser>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

Sample AXL Response:

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Set-Cookie: JSESSIONIDSSO=950805DE5E10F32C5788AE164EEC4955; Path=/
Set-Cookie: JSESSIONID=151CF94ACF20728B1D47CC5C3BECC401; Path=/axl; Secure
SOAPAction: "CUCM:DB ver=6.0"
Content-Type: text/xml; charset=utf-8
Content-Length: 728
Date: Mon, 22 Jan 2007 06:51:42 GMT
Connection: close
```

AXL APIs Added for Cisco Unified Communications Manager 6.0(1)

The following list provides AXL API calls that are new in Release 6.0(1):

- addAARGroup
- removeAARGroup
- updateAARGroup
- getAARGroup
- updateAARGroupMatrix
- addApplicationToSoftkeyTemplate
- removeApplicationToSoftkeyTemplate
- addCallerFilterList
- removeCallerFilterList
- updateCallerFilterList
- getCallerFilterList
- getCCMVersion
- addCommonDeviceConfig
- removeCommonDeviceConfig
- updateCommonDeviceConfig
- getCommonDeviceConfig
- addCredentialPolicy
- removeCredentialPolicy
- updateCredentialPolicy
- getCredentialPolicy
- addIVRUserLocale
- removeIVRUserLocale
- updateIVRUserLocale
- getIVRUserLocale
- updateLicenseCapabilities
- getLicenseCapabilities
- addMeetMe
- removeMeetMe
- updateMeetMe
- getMeetMe
- addMobileVoiceAccess
- removeMobileVoiceAccess
- updateMobileVoiceAccess
- getMobileVoiceAccess
- addMobility
- removeMobility
- updateMobility

- getMobility
- addMOHServer
- removeMOHServer
- updateMOHServer
- getMOHServer
- addPhoneTemplate
- removePhoneTemplate
- updatePhoneTemplate
- getPhoneTemplate
- addPhysicalLocation
- removePhysicalLocation
- updatePhysicalLocation
- getPhysicalLocation
- addRecordingProfile
- removeRecordingProfile
- updateRecordingProfile
- getRecordingProfile
- addRemoteDestination
- removeRemoteDestination
- updateRemoteDestination
- getRemoteDestination
- addRemoteDestinationProfile
- removeRemoteDestinationProfile
- updateRemoteDestinationProfile
- getRemoteDestinationProfile
- addSoftKeyTemplate
- updateSoftKeyTemplate
- getSoftKeyTemplate
- removeSoftKeyTemplate
- updateSoftKeySet
- getSoftKeySet
- addTranscoder
- updateTranscoder
- getTranscoder
- removeTranscoder
- addTransformationPattern
- removeTransformationPattern
- updateTransformationPattern
- getTransformationPattern

**Note**

The getCCMVersion API will return the Cisco Unified Communications Manager Version based on the Node Name that is specified in the request. If no Node Name is specified, you will get the Cisco Unified Communications Manager Version of the lowest node ID Cisco Unified Communications Manager.

Cisco always advises running the AXL API on a completely upgraded cluster. When run on a cluster that is not upgraded completely, the response of the AXL API will be correct when executed on a server that already has been upgraded. However, if you execute the AXL API on a server that has not yet been upgraded, then it will return the Cisco Unified Communications Manager Version of the lowest node ID Cisco Unified Communications Manager per the server local database information.

AXL APIs Changed for Cisco Unified Communications Manager 6.0(1)

The following AXL API calls changed since the previous release. These changes might require changes to existing user code that is making use of them:

- updateAppUser
- addCallPickupGroup
- updateCallPickupGroup
- getCallPickupGroup
- addConferenceBridge
- updateConferenceBridge
- getConferenceBridge
- addCSS
- updateCSS
- getCSS
- addDevicePool
 - In axl.xsd, the tag name aarNeighborhood and the annotation for the revertPriority tag in XDevicePool changed to match the AXL response.
 - In axlsoap.xsd, the tag name aarNeighborhood and the annotation for the revertPriority tag in updateDevicePoolReq changed.
- updateDevicePool
 - In axl.xsd, the tag name aarNeighborhood and the annotation for the revertPriority tag in XDevicePool changed to match the AXL response.
 - In axlsoap.xsd, the tag name aarNeighborhood and the annotation for the revertPriority tag in updateDevicePoolReq changed.
- getDevicePool
 - In axl.xsd, the tag name aarNeighborhood and the annotation for the revertPriority tag in XDevicePool changed to match the AXL response.
 - In axlsoap.xsd, the tag name aarNeighborhood and the annotation for the revertPriority tag in updateDevicePoolReq changed.
- addDeviceProfile
- updateDeviceProfile
- getDeviceProfile

- addGatewayEndpoint
- updateGatewayEndpoint
- getGatewayEndpoint
- addH323Phone
- updateH323Phone
- getH323Phone
- addH323Trunk
- updateH323Trunk
- getH323Trunk
- addLine
- updateLine
- getLine
- addMGCP
- getMGCP
- addPhone
- updatePhone
- getPhone
- addRegion
- updateRegion
- getRegion
- updateRegionMatrix
- addRoutePartition
- updateRoutePartition
- getRoutePartition
- addSIPTrunk
- updateSIPTrunk
- getSIPTrunk
- addUser
- updateUser
- getUser
- addVoiceMailPort
- updateVoiceMailPort
- getVoiceMailPort
- doAuthenticateUser
- getCMCInfo, removeCMCInfo, and updateCMCInfo

In axlsoap.xsd

- A new option tag “code” along with the “uuid” tag for these three requests exists. The user can send either the uuid or code tag.

- This release renames the existing tag “code” to “newCode” in the updateCMCInfo request. Users can send the new code to be updated as the “newCode” tag instead of “code” in updateCMCInfo requests.
- This release removes the invalid authorizationLevel tag in the updateCMCInfo request.
- addFACInfo, getFACInfo, updateFACInfo, and removeFACInfo

In axl.xsd

- The existing tag “description” changed to “name.” This makes the addFACInfo, getFACInfo, and updateFACInfo request match the database. In previous releases, the value that was supplied for the “description” tag updated “name” in the database.

In axlsoap.xsd

- A new option tag “name” along with the “uuid” tag for getFACInfo, updateFACInfo, and removeFACInfo exist.
- The existing tag “description” changed to “newName” in the updateFACInfo request.
- Users can send either uuid or name in the getFACInfo, updateFACInfo, and removeFACInfo requests.

This release deprecated some of the fields that were removed from the Cisco Unified Communications Manager 6.0(1) database in AXL. This release adds annotation for such fields.

Service Parameters Added or Changed

The Cisco Unified Communications Manager 6.0(1) release adds a new service parameter, EnableAXLEncodingInfo, to the Cisco Unified Communications Manager Administrator windows under the Cisco Database Layer Monitor service. This parameter allows the user to decide whether AXL responses should contain the encoding information. Consider encoding information as important if an AXL request has non-English characters in it.

The Cisco Unified Communications Manager 5.1(1) release added a new service parameter, Send Valid Namespace in the AXL response, under the Cisco Database Layer Monitor service. This parameter determines the namespace that is sent in the AXL response from the Cisco Unified Communications Manager. In the 6.0(1) release, the default value of this parameter changed from False to True.

- When this parameter is True, Cisco Unified Communications Manager sends the valid namespace (<https://www.cisco.com/AXL/API/1.0>) in the AXL response, so the namespace matches the AXL schema specification.
- If the parameter is False, Cisco Unified Communications Manager sends an invalid namespace (<https://www.cisco.com/AXL/1.0>) in the AXL response, which does not match the AXL schema specification.

To maintain backward compatibility with older applications, you might need to change the value to False. Cisco recommends that you set this parameter to True, so the Cisco Unified Communications Manager sends a valid namespace.

AXL APIs Added for Cisco Unified Communications Manager 5.1(1)

The following list provides AXL API calls that were added in Release 5.1(1):

- addSIPRealm
- updateSIPRealm

- `getSIPRealm`
- `removeSIPRealm`

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

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 system returns the appropriate AXL response. All responses get named identically to the associated requests, except that the word “Response” is appended.

For example, the XML response that is returned from an `addPhone` request is 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 1-10).

AXL Compliance

The Cisco Unified Communications Manager 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 Communications Manager 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, the system returns an error 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/"
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>
      </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 elements represent 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
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.
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.
5007	Item Not Valid Error —The system identified the specified item 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 Communications Manager Release 6.0(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 Unified Communications Manager 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, the system returns an HTTP 415 error 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, the system returns an HTTP 401 Access Denied error 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, the system returns an HTTP 413 error message.

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

```
POST :8443/axl/
Host: axl.myhost.com:8443
Accept: text/*
Authorization: Basic bGFycnk6Y3VybHkgYW5kIG1vZQ==
Content-type: text/xml
SOAPAction: "CUCM:DB ver=6.0"
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:8443
Accept: text/*
Authorization: Basic bGFycnk6Y3VybHkgYW5kIG1vZQ==
Content-type: text/xml
SOAPAction: "CUCM:DB ver=6.0"
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">
      <phoneNumber>SEP222222222245</phoneNumber>
    </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 the user name and "cisco_cisco" is the 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\"";
    strAXLRequest += " 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 ";
strHTTPHeader += getAuthorization() + "\r\n";
strHTTPHeader += "Content-type: text/xml\r\n";
strHTTPHeader += "SOAPAction: \"CUCM:DB ver=6.0\"\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.*;
import javax.net.ssl.*;
import java.security.cert.CertificateException;
import java.security.cert.X509Certificate;

public class AXLJavaClient {
    public static void main(String[] args) {
        byte[] bArray = null; // buffer for reading response from
        Socket socket = null; // socket to AXL server
        OutputStream out = null; // output stream to server
        InputStream in = null; // input stream from server

        String sAXLSOAPRequest = "";
        // HTTPS header and SOAP payload
        String sAXLRequest = null; // will hold only the SOAP payload
        //username=CCMAdministrator and password=cisco_cisco
        String authorization = "CCMAdministrator" + ":" + "cisco_cisco";
        // base64 encoding of the username and password
        authorization = new sun.misc.BASE64Encoder().encode(authorization.getBytes());
        // Form the http header
        sAXLSOAPRequest = "POST /axl/ HTTP/1.0\r\n";
        sAXLSOAPRequest += "Host:localhost:8443\r\n";
        sAXLSOAPRequest += "Authorization: Basic " + authorization + "\r\n";
        sAXLSOAPRequest += "Accept: text/*\r\n";
        sAXLSOAPRequest += "Content-type: text/xml\r\n";
        sAXLSOAPRequest += "SOAPAction: \"CUCM:DB ver=6.0\"\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>SEP000000000009</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;
        try {
            AXLJavaClient axl = new AXLJavaClient();
            // Implement the certificate-related stuffs required for sending request via https
            X509TrustManager xtm = axl.new MyTrustManager();
            TrustManager[] mytm = { xtm };
            SSLContext ctx = SSLContext.getInstance("SSL");

```

```

    ctx.init(null, mytm, null);
    SSLSocketFactory sslFact = (SSLSocketFactory) ctx.getSocketFactory();
    socket = (SSLSocket) sslFact.createSocket("10.77.31.203", Integer.parseInt("8443"));
    in = socket.getInputStream();
    // send the request to the server
    // read the response from the server
    StringBuffer sb = new StringBuffer(2048);
    bArray = new byte[2048];
    int ch = 0;
    int sum = 0;
    out = socket.getOutputStream();
    out.write(sAXLSOAPRequest.getBytes());
    while ((ch = in.read(bArray)) != -1) {
        sum += ch;
        sb.append(new String(bArray, 0, ch));
    }
    socket.close();
    // output the response to the standard output
    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
} catch (Exception ea) {
    System.err.println("Unknown exception " + ea.getMessage());
    return;
}
}
finally{
    try {
        if (socket != null)
            socket.close();
    } catch (Exception exc) {
        exc.printStackTrace();
        System.err.println("Error closing connection to server: "+ exc.getMessage());
    }
}
}

public class MyTrustManager implements X509TrustManager {

    MyTrustManager() {
        // create/load keystore
    }

    public void checkClientTrusted(X509Certificate chain[], String authType)
        throws CertificateException {

    }

    public void checkServerTrusted(X509Certificate chain[], String authType)
        throws CertificateException {

    }

    public X509Certificate[] getAcceptedIssuers() {

        return null;
    }
}

```

}

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

Using executeSQLUpdateAXL

This example illustrates the use of the executeSQLUpdateAXL request:

```
POST :8443/axl/
Host: axl.myhost.com:8443
Accept: text/*
Authorization: Basic bGFycnk6Y3VybHkgYW5kIG1vZQ==
Content-type: text/xml
SOAPAction: "CUCM:DB ver=6.0"
Content-length: 613

<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <axlapi:executeSQLUpdate sequence="1"
xmlns:axlapi="http://www.cisco.com/AXL/API/1.0" 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/API/1.0 axlsoap.xsd">
      <sql>
        insert into device (fkPhoneTemplate,fkDevicePool,tkclass, tkpreemption,
tkdeviceprofile, tkmodel, tkdeviceprotocol, tkproduct, description,
tkstatus_mlppindicationstatus, name, pkid) values ('Standard 7941 SCCP','default',1, 2, 2,
115, 0, 115, '', 0, 'Cisco 7941', newid())
      </sql>
    </axlapi:executeSQLUpdate>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Using executeSQLQuery

This example illustrates the use of the executeSQLQuery request:

```
POST :8443/axl/
Host: axl.myhost.com:8443
Accept: text/*
Authorization: Basic bGFycnk6Y3VybHkgYW5kIG1vZQ==
Content-type: text/xml
SOAPAction: "CUCM:DB ver=6.0"
Content-length: 613

<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <axlapi:executeSQLQuery sequence="1"
xmlns:axlapi="http://www.cisco.com/AXL/API/1.0"
xmlns:axl="http://www.cisco.com/AXL/API/1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.cisco.com/AXL/API/1.0 axlsoap.xsd">
      <sql>SELECT * from numplan</sql>
    </axlapi:executeSQLQuery>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Throttling of Requests

The side effects of updating the Cisco Unified Communications Manager 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 Monitor advanced service parameter “MaxAXLWritesPerMinute.”

AXL accommodates all requests until the “MaxAXLWritesPerMinute” value is reached. The system rejects subsequent attempts to modify the database with AXL by sending an HTTPS 503 Service Unavailable response. Every minute, AXL resets its internal counter and begins to accept AXL update requests until the limit is reached again.

The following AXL requests, which are considered “Reads,” do not count against the preset “MaxAXLWritesPerMinute” service parameter value. All other AXL requests that are not in the following list count against the service parameter value:

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

The execSQLUpdate request is throttled.

AXL Schema Documentation

The axlsqltoolkit.zip plug-in contains the following five AXL schema files:

- AXLAPI.wsdl
- AXLEnums.xsd
- axlmessage.xsd
- axlsoap.xsd
- axl.xsd

These files encapsulate the complete AXL schema, including details of all requests, responses, XML objects, and data types.

In addition to these schema files, two folders exist:

- WSDL-AXIS
- WSDL-NET

Each of these folders contains AXLAPI.wsdl and AXLSOAP.xsd files to be used for application development in AXIS or .NET client environments, respectively. The plug-in also contains version specific AXL in folders 1.0 and 6.0.

You can obtain complete documentation of all available AXL messages from the Cisco Developer Services web site: http://www.cisco.com/cgi-bin/dev_support/access_level/product_support. This website requires a Cisco.com login.

From the Cisco Unified Communications Manager server administration interface, you can use the *Application > Plugins > Cisco Unified Communications Manager AXL SQL Toolkit* command to obtain

- AXL schema (.xsd) files
See also [AXL Schema Documentation, page 1-20](#).
- The WSDL file

Authentication

The system controls user authentication via the HTTPS Basic Authentication scheme; therefore, you must include the Authorization header in the HTTPS header. 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.

Ensure users are authorized to access AXL. For help with configuring authorization, see [Post-Installation Steps and Troubleshooting on the Linux Platform, page 1-22](#).

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

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 Secure Sockets Layer (SSL). SSL remains functional on the web server by default. Ensure AXL requests are made by using the “https” protocol.

Integration Considerations and Interoperability

The AXL API gives much power to developers to modify the Cisco Unified Communications Manager 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 Database Layer Monitor advanced service parameter MaxAXLWritesPerMinute (see the [“Throttling of Requests” section on page 1-20](#)). If this does not solve the problem, consider employing an additional 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.

Impact on Backward Compatibility

For Cisco Unified Communications Manager Release 6.0(1), be aware that the defined AXL APIs are backward compatible; however, the `executeSQLQuery` request, which lets the user run a database query directly on the Cisco Unified Communications Manager database, is **not** backward compatible. Necessary changes in the Cisco Unified Communications Manager 6.0(1) database schema make this true.

- Release 6.0(1) splits some of the tables in the Cisco Unified Communications Manager database. Feature-related information moved to the corresponding dynamic tables. If the direct SQL query to which the 'executeSQLQuery' API referred uses any of the changed tables, you may need to rewrite the query according to the new database schema.
- The columns `enduser.password` and `enduser.pin` from the `enduser` table and the `applicationuser.password` column from the `applicationUser` table moved to the `credential` table as `credential.credentials`. A direct SQL query that refers to these columns will not work in Cisco Unified Communications Manager Release 6.0(1).



Note Be aware that Cisco Unified Communications Manager password and pin fields are encrypted. Applications should not write to those fields using `<executeSQLUpdate>`. Instead, update passwords and pins by using the appropriate `<updateXXXUser>` request.

- The phone API for extension mobility-related parameters has the new tag `CurrentConfig`. This tag is valid only for the `getPhone` response. The tag lets AXL provide the original device configuration as well as the logged-in profile information:
 1. If a user has logged in to a device by using a device profile, the `CurrentConfig` tag contains the values for the extension mobility-related parameters from that device profile.
 2. If no user has logged in, the `CurrentConfig` tag contains the values of the extension mobility-related parameters for the actual device.
- Schema changes for the `CMCInfo` and `FACInfo` APIs help maintain consistency with other AXL APIs. See the description of the changes in [AXL APIs Changed for Cisco Unified Communications Manager 6.0\(1\), page 1-7](#).

For further details about the Cisco Unified Communications Manager database schema changes, refer to the Cisco Unified Communications Manager *Data Dictionary for Release 6.0(1)*.

Post-Installation Steps and Troubleshooting on the Linux Platform

The system implements AXL as a Java servlet. The Java implementation provides platform independence. AXL accesses the Cisco Unified Communications Manager database by using `DBL2`, which is a JDBC wrapper implementation. AXL gets packaged as a WAR file. Linux RPM installs the war file for AXL on Cisco Unified Communications Manager server.

Follow the procedures in [Post-Installation Steps, page 1-23](#), to start the AXL service and set up user permissions. Next, follow the procedures in [Post-Installation Troubleshooting Checklist, page 1-23](#), to check the installation.

Post-Installation Steps

You can start or stop the AXL web service from Cisco Unified Communications Manager Serviceability. The service is disabled by default. You should start the service before using the AXL APIs.

Starting the AXL Service

-
- Step 1** From the Cisco Unified Communications Manager Administration window, choose **Navigation > Cisco Unified Communications Manager Serviceability**.
- Step 2** Choose **Tools > Service Activation**.
- Step 3** From the **Server** box, choose the server and click **GO**.
- Step 4** From **Database and Admin Services**, select **Cisco AXL Web Service** and save the changes.

Upon starting the AXL service, AXL gets deployed as a web application within Apache Tomcat. The WAR file gets deployed to Tomcat under `/usr/local/thirdparty/jakarta-tomcat/webapps/axl`.

Setting AXL API Access Permissions

-
- Step 1** From the Cisco Unified Communications Manager Administration window, choose **User Management > UserGroup > Add New**.
- Step 2** To add AXP API access for the new UserGroup
- Choose **User Management > User Group**.
 - Choose **Role > Assign Role to Group**.
 - Select **Standard AXL API Access**.
 - Click **Add Selected**.
 - On the main page, click **Save**.
- Step 3** To add a user to the new UserGroup
- Choose **User Management > User Group**.
 - Choose **UserGroup > Add End Users to Group**.
 - Select the user and click **Add Selected**.

Post-Installation Troubleshooting Checklist

Use the following checklist to avoid some common problems by fine-tuning your configuration before proceeding with the troubleshooting process:

-
- Step 1** If the AXL client application cannot connect to the AXL service, check the following
- Is the AXL application configured with the correct IP address for the AXL server?
 - Is the AXL application configured with the appropriate AXL user credentials?
 - Does the application server have HTTPS connectivity to the AXL server?
- Use this URL for accessing AXL: <https://system-name:port/axl/> (port is 8443).

- Is HTTPS (secure) configured for AXL?
- Step 2** If the AXL functions or requests are failing, check the following
- AXL logs for AXL or SOAP error responses. See [Error Codes, page 1-26](#).
 - For further debugging, you can view the AXL log files with the RTMT application.
- Step 3** Check that applications in a cluster configuration are connected to the AXL service only on the Cisco Unified Communications Manager Publisher server if the application needs to modify the database.
- Step 4** Verify basic AXL functionality by performing the procedure that follows:
1. Go to the AXL API URL via a web browser.
For instance, enter <https://system-name:8443/axl/> in the address text box.
 2. When prompted for user name and password, use the standard administrator login, or use the user name that is associated with a user group that is assigned the AXL role.
 3. Look for a plain page that states the AXL listener is working and accepting requests but only communicates via POST.
- This verifies functionality and user access.
- Step 5** Check the Cisco Database Layer service parameter MaxAXLWritesPerMinute:
- Updating the Cisco Unified Communications Manager database can have the side effect of adversely affecting system performance. To prevent this effect, the system administrator can control how many AXL requests are allowed to update the database per minute through the Database Layer service parameter MaxAXLWritesPerMinute.
 - AXL accommodates all requests until the MaxAXLWritesPerMinute value is reached. The default value of this parameter is 50, and the maximum configurable value is 999; however, the maximum Cisco supported value for production systems is 60.
 - After the MaxAXLWritesPerMinute value is reached, the system rejects attempts to modify the database with AXL with an HTTP 503 Service Unavailable response. Every minute, AXL resets its internal counter and begins to accept AXL update requests until the MaxAXLWritesPerMinute value is reached.
- Step 6** If the AXL functions or requests are failing with error as User Authorization error: “Access to the requested resource has been denied,” check whether the user has the permission to the Standard AXL API Access. You can check this from the Permission Information section of the EndUser configuration window.
-

AXL Trace Logs

AXL trace logs contain the text of every AXL request and response, along with user and origination IP information. Trace logs prove useful for identifying who is making AXL requests, inspecting the AXL XML request for format or syntax errors, and determining the actual AXL service response or errors.

The system writes the AXL trace logs to the `/var/log/active/tomcat/logs/axl/log4j` directory. You can view them with RTMT. File names are `axl####.log`, where # represents a number from 0000 (zero) to the maximum number of files allowed. The maximum file size is 1 MB by default. The maximum number of stored files defaults to 10. You can change these settings through the Serviceability windows .

**Note**

If an AXL login request contains a <password> tag with an xmlns attribute value, versions of the AXL API prior to 6.0(1) or 5.1(2) log the password in clear text. In later versions of the API, the system replaces the password with an "*" character.

For .NET WSDL applications, including the xmlns attribute in the <password> tag would be typical behavior, and, in earlier releases, this could represent a security issue. If the SOAP message body contains a namespace tag, you do not need to specify the xmlns attribute for each individual tag.

While analyzing the log files,

- Determine which log file is currently active by timestamp.
- Look for Exception traces that indicate processing errors.
- If no traces are being added, verify that Tomcat is running, AXL is currently activated, and a client application is attempting to communicate with the AXL API.

The following sample shows the AXL trace log output:

```
2007-03-17 05:32:26,512 INFO [http-8443-Processor21] axl.AxlListener - Received request
1173323669700 from CCMAdministrator at IP 10.77.31.203
2007-03-17 05:32:26,513 INFO [http-8443-Processor21] axl.AxlListener - <!-- edited with
XMLSPY v5 rel. 4 U (http://www.xmlspy.com) by Jerry Vander Voord (Cisco Systems)
--><SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"><SOAP-ENV:Body>
<axlapi:addGatekeeper sequence="1" xmlns:axlapi="http://www.cisco.com/AXL/API/1.0"
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/API/1.0 axlsoap.xsd">
<gatekeeper>
  <name>AXL-Sample-GK1</name>
  <description>This is a sample gatekeeper</description>
  <rrqTimeToLive>30</rrqTimeToLive>
  <retryTimeout>30</retryTimeout>
  <enableDevice>>false</enableDevice>
</gatekeeper>
</axlapi:addGatekeeper></SOAP-ENV:Body></SOAP-ENV:Envelope>
2007-03-17 05:32:26,668 INFO [http-8443-Processor21] axl.Handler - Handler initializing
2007-03-17 05:32:26,788 INFO [http-8443-Processor21] axl.AxlListener - <?xml
version="1.0" encoding="UTF-8"?><SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"><SOAP-ENV:Header/><SOAP
-ENV:Body>
<axl:addGatekeeperResponse xmlns:axl=http://www.cisco.com/AXL/1.0
xmlns:xsi="http://www.cisco.com/AXL/1.0" sequence="1">
<return>{7EB31958-C24A-3E63-3E4B-A8446365D35}</return>
</axl:addGatekeeperResponse></SOAP-ENV:Body></SOAP-ENV:Envelope>
2007-03-17 05:32:26,789 INFO [http-8443-Processor21] axl.AxlListener - Request
1173323669700 was process in 356ms
```

The AXL trace log contains

- Time that the request was received - 05:32:26,512
- Client IP address - IP 10.77.31.203
- Client user ID - CCMAdministrator
- Request ID number - 1173323669700
- Request contents – addGatekeeper, <gatekeeper>, <name>, and so on
- Response contents - <name>
- Time taken for the response - 356 ms

Follow these steps to set the AXL trace level from the Serviceability window:

- Step 1** From the Cisco Unified Communications Manager Administration window, choose **Application > Cisco Unified Communications Manager Serviceability**.
- Step 2** Choose **Trace > Configuration**.
- Step 3** From the **Servers** column, select the server and click **GO**.
- Step 4** From the **Service Group** box, select **Database And Admin Services** and click **GO**.
- Step 5** From the **Services** box, select the **Cisco AXL Web Service** and click **GO**.
- Step 6** Check the **Trace On** check box.
- Step 7** If you want the trace to apply to all Cisco Unified Communications Manager servers in the cluster, select the **Apply to All Nodes** check box.
- Step 8** From the **Debug Trace Level** field, select **Debug**.

201549

- Step 9** You can set trace output options from the same window.

**Note**

You should enable AXL logs only on request from Cisco TAC or Cisco Developer Services.

Error Codes

If an exception occurs on the server, or if any other error occurs during the processing of an AXL request, the system returns an error in the form of a SOAP Fault message, such as in the following example:

```
<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 shows a detailed 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>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>
        </axl:error>
      </detail>
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

The <detail> element of a SOAP Fault contains the 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> element.

- The <code> element is a numerical value that the user agent uses to find out what type of error occurred. The next table describes the error codes:

Error Code	Description
5000	Unknown Error: an unknown error occurred while the request was processed. A problem on the server or mistakes in the request can cause this error.
5002	Unknown Request Error: the user agent submitted a request that is unknown to the API.
5003	Invalid Value Exception: the system detected an invalid value in the XML request.

5004	AXL Unavailable Exception: the AXL service was too busy to handle the request at that time. The request should be sent again at a later time.
5005	Invalid Value Exception: the system detected an invalid value in the XML request.

- The <message> element gives the user agent a detailed error message explaining the error.
- The <request> element lets the user agent determine what type of request generated this error. Because this element is optional, it may not always appear.

Using the AXL API with AXIS

This section explains how to use the AXLAPI.wsdl file in a Java programming environment.

Cisco has verified the AXLAPI.wsdl file in the WSDL-AXIS folder in the AXIS environment.

Cisco provides the associated schema file, AXLSOap.xsd, only for code generation. Cisco has modified this file to minimize the backwards compatibility impact of future changes in the database schema. Use the WSDL and the schema files in the parent directory for reference and for validation of responses.

When you run the wsdl2Java utility to create the Java source code by using AXLAPI.wsdl, the utility throws two errors that are specific to AXIS_1_4. For further details on these errors, refer to <http://issues.apache.org/jira/browse/AXIS-2545> and <http://issues.apache.org/jira/browse/AXIS-1280>.

The incorrect ordering of the parameters that are passed to the constructor causes the first AXIS jira error. A code example follows:

```
public class XNPDirectoryNumberShareLineAppearanceCSS extends
com.cisco.www.AXL.API._6_0.XCallingSearchSpace implements java.io.Serializable {
>
>
super(
    uuid,
    name,
    description,
    clause,
    dialPlanWizardGenId,
    members);
```

However, the parent constructor is defined as

```
public XCallingSearchSpace(
    org.apache.axis.types.Name name,
    java.lang.String description,
    java.lang.String clause,
    org.apache.axis.types.NonNegativeInteger dialPlanWizardGenId,
    com.cisco.www.AXL.API._6_0.XCallingSearchSpaceMember[] members,
    java.lang.String uuid) {
    this.name = name;
    this.description = description;
    this.clause = clause;
    this.dialPlanWizardGenId = dialPlanWizardGenId;
    this.members = members;
    this.uuid = uuid;
}
```

You need to change either the constructor or the constructor calling as shown below:

```
super(
    name,
    description,
    clause,
    dialPlanWizardGenId,
    members,
    uuid);
```

The second AXIS jira error relates to having a string constructor for simple types; for example

```
// Simple Types must have a String constructor
public XLoadInformation(java.lang.String _value) {
    super(_value);
}
```

For such cases, the corresponding schema file (axl.xsd) in the parent schema folder must be referred and must implement the string class that these classes can inherit.

Using the AXL API in a .NET Environment

To integrate the AXL API with a .NET client, you must modify the Cisco-provided WSDL and XSD files.

Cisco provides the associated schema file, AXLSOAP.xsd, only for code generation. Cisco has modified this file to minimize the backwards compatibility impact of future changes in the database schema. Use the WSDL and the schema files in the parent directory for reference and for validation of responses.

The inability of .NET to handle complex schemas necessitates some of the changes that are described below.

Required Changes to the Generated Code

After running WSDL.exe, you must make several changes to the generated code. Run WSDL.exe by using the following command:

```
wsdl.exe AXLAPI.wsdl axlsoap.xsd
```

This command generates the file AXLAPIService.cs. The class AXLAPIService in AXLAPIService.cs requires at least three changes:

1. Create an ICertificatePolicy-derived class, which will later be associated with our service. This class represents a brute-force approach to policy and certificate management. You need to use this method in 5.x and 6.x AXL due to the use of HTTPS.

```
public class BruteForcePolicy : System.Net.ICertificatePolicy
{
    public bool CheckValidationResult(System.Net.ServicePoint sp,
        System.Security.Cryptography.X509Certificates.X509Certificate cert,
        System.Net.WebRequest request, int problem)
    {
        return true;
    }
}
```

2. Modify the service constructor to take username/password credentials, with the Cisco Unified Communications Manager IP address as an argument, and associate the BruteForcePolicy class with the static CertificatePolicy manager.

```
public AXLAPIService(string ccmIp, string user, string password)
{
    System.Net.ServicePointManager.CertificatePolicy = new BruteForcePolicy();

    this.Url = "https://" + ccmIp + ":8443/axl/";
    this.Credentials = new System.Net.NetworkCredential(user, password);
}
```

3. The .NET framework uses the *expects* header differently (http://issues.apache.org/bugzilla/show_bug.cgi?id=31567). Several possible workarounds exist to this problem:

- a. Override the GetWebRequest method to use HTTP 1.0 due to an error between TOMCAT/AXIS and the .NET HTTP 1.1 Web Service request mechanism.

```
protected override System.Net.WebRequest GetWebRequest(Uri uri)
{
    System.Net.HttpWebRequest request = base.GetWebRequest (uri) as
        System.Net.HttpWebRequest;
    request.ProtocolVersion = System.Net.HttpVersion.Version10;

    return request;
}
```

- b. Override the GetWebRequest method to manually embed the authentication string. If you do this, do not use the line

```
this.Credentials = new System.Net.NetworkCredential(user, password);
```

from the constructor that is provided in point 2 earlier in this section.

```
protected override System.Net.WebRequest GetWebRequest(Uri uri)
{
    System.Net.HttpWebRequest request = (System.Net.HttpWebRequest)base.GetWebRequest(uri);
    if (this.PreAuthenticate)
    {
        System.Net.NetworkCredential nc = this.Credentials.GetCredential(uri, "Basic");
        if (nc != null)
        {
            byte[] credBuf = new System.Text.UTF8Encoding().GetBytes(nc.UserName + ":" + nc.Password);
            request.Headers["Authorization"] = "Basic " + Convert.ToBase64String(credBuf);
        }
    }
    return request;
}
```

- c. If you use wsdl2wse (WSE library) instead of wsdl.exe, you cannot override the HTTP version or supply HTTP headers manually. To use WSE, you must set the keepalive header to false for the generated class, or set the user-agent to restricted. This technique will work as an alternative to steps a and b.

Resolving JIT Errors

When you compile and attempt to instantiate the `AXLAPIService` class, you should expect one error to appear while the types are inspected and loaded.

Class `XUserUserGroup` includes a field that was generated incorrectly. You must remove one of the '[' from the two '[' brackets after the `XUserUserGroupUserRolesUserRole` field:

```
public XUserUserGroupUserRolesUserRole[] userRoles;
```

Backward Compatibility Issues

When you add the definitions for new Cisco Unified IP Phone devices to the Cisco Unified Communication Manager database, the original WSDL that was sent out for that Cisco Unified Communication Manager becomes outdated. For example, the `XModel` enumeration in Cisco CallManager Release 4.1.3 does not contain the Cisco Unified IP Phone 7961G-GE.

However, if you install the latest device pack that contains that device information into your release 4.1.3 environment, that value will be returned if you use the `listAllDevices` or `getPhone` commands for that device name. This causes .NET to throw an exception when it encounters the new model because the definition does not contain the mode.

More generally, almost all enumerations in `AXLEnums.xsd` could change in some future release, which in turn might create backward incompatibility with your code. To address this issue, Cisco has changed the type of all of the tags that use any of these enumerations to `String` and added an annotation to that tag that specifies where to look for the correct value (`AXLEnums.xsd`).

Tag Serialization Issues

If you generate the client stub by using `wsdl.exe`, you may find that some fields that have default values that are defined in the schema would not work if passed in the AXL request. For example, in the `updatePhoneReq` class of the generated client stub, a field named "ignorePresentationIndicators" has a default value of "False" defined in the schema.

```
[System.Xml.Serialization.XmlTypeAttribute(Namespace="http://www.cisco.com/AXL/6.0")]
    public class UpdatePhoneReq : APIRequest {
        .
        .
        .

        [System.Xml.Serialization.XmlElementAttribute(Form=System.Xml.Schema.XmlSchemaForm.Unqualified)]
        [System.ComponentModel.DefaultValueAttribute(false)]
        public bool ignorePresentationIndicators = false;
        .
        .
    }
```

When this tag is sent with a value of false, `XmlSerializer` does not serialize this tag because of a design restriction in Microsoft .NET Framework 1.0. Refer to <http://support.microsoft.com/kb/325691>.

To work around this problem, comment out all instances of

```
[System.ComponentModel.DefaultValueAttribute(XXX)]
```

in the generated client stub as shown:

```
[System.Xml.Serialization.XmlTypeAttribute(Namespace="http://www.cisco.com/AXL/6.0")]
    public class UpdatePhoneReq : APIRequest {
        .
        .
        .
        [System.Xml.Serialization.XmlElementAttribute(Form=System.Xml.Schema.XmlSchemaForm.Unqualified)]
        // Comment this line below
        // [System.ComponentModel.DefaultValueAttribute(false)]
        public bool ignorePresentationIndicators = false;
        .
        .
        .
    }
```

A second issue that is found when you are using the version of wsdl.exe that comes with .NET 1.0 is that some tags, including `fkcallingsearchspace_autoregistration`, do not get updated to null/none in the database.

This appears to be an issue in which .NET does not serialize tags that are defined as `nillable=true` in the schema.

For example, to work around this limitation for the tag `callingSearchSpace` in `updatePhoneReq` in the generated stub, you can remove the `"Form=System.Xml.Schema.XmlSchemaForm.Unqualified"` from

```
        [System.Xml.Serialization.XmlElementAttribute("name", typeof(string),
Form=System.Xml.Schema.XmlSchemaForm.Unqualified)]
        [System.Xml.Serialization.XmlElementAttribute("uuid", typeof(string),
Form=System.Xml.Schema.XmlSchemaForm.Unqualified)]
        [System.Xml.Serialization.XmlChoiceIdentifierAttribute("ItemElementName")]
        public string Item;
```

With this change, the serializer will serialize the tags. Passing the tag value as "" will set the `callingSearchSpace` to null/None. The same workaround applies to other such tags.

Names Containing Special Characters

Using the version of wsdl.exe that comes with .NET 1.0, Cisco has found that when attempting to add elements like `gatewayEndpoint`, `MGCP endpoint` or `CSS` where the name contains special characters, the elements do not get updated in the database properly.

For example, a `gatewayEndpoint` with `name="AALN@SAA000011114444"` sent as `name="AALN_x0040_SAA000011114444"` in the AXL request.

This appears to be a limitation in .NET serialization of tags that are defined as type `xsd:name` in the schema.

In the XML specification, the type `xsd:name` is defined as a token that begins with a letter or one of a few punctuation characters and continues with letters, digits, hyphens, underscores, colons, or periods, together known as name characters. Thus, `xsd:name` does not allow any special characters such as '@' or '/'.

One workaround involves changing the data type from "Name" to "string" in the generated stub:

Original Code

```
public class XDevice {
    [System.Xml.Serialization.XmlElementAttribute
    (Form=System.Xml.Schema.XmlSchemaForm.Unqualified, DataType="Name")]
    public string name;
```

Modified Code

```
public class XDevice {
    [System.Xml.Serialization.XmlElementAttribute(typeof(string),
        Form=System.Xml.Schema.XmlSchemaForm.Unqualified)]
    public string name;
```

With this modification, the special characters in the name will be updated in the database without any conversion.

Returned Namespace for AXIS and .NET Applications

By default, the AXLAPI.wsdl carries the namespace `http://www.cisco.com/AXL/API/6.0`. The generated client stubs also have this namespace. In some situations, you must change the namespace in AXLAPI.wsdl before creating the client stubs.

The namespace that is returned in the AXL response depends on two factors:

1. Whether the SOAPAction attribute in the HTTP header had the value "CUCM:DB ver=6.0."
2. The value of the "Send Valid Namespace in AXL Response" service parameter in the Cisco Unified Communications Manager Administration Service Parameter window.

If the SOAPAction attribute in the HTTP header has the value "CUCM:DB ver=6.0":

- The AXL response will have the namespace value that the "Send Valid Namespace in AXL Response" service parameter specifies: either `http://www.cisco.com/AXL/API/6.0` or `http://www.cisco.com/AXL/6.0`.
- If you set the service parameter "Send Valid Namespace in AXL Response" to true, the namespace that is returned in the AXL response will be `http://www.cisco.com/AXL/API/6.0`, which will match the namespace that is specified in AXLAPI.wsdl.
- If you set this service parameter to False, the namespace that is returned in the AXL response will be `http://www.cisco.com/AXL/6.0`.

If the SOAPAction attribute has any other value, the AXL response will have the namespace `http://www.cisco.com/AXL/API/1.0` or `http://www.cisco.com/AXL/1.0`, depending on the value of the service parameter.

