



Cisco Collaboration Cloud Platform Services 3.0

Reference Guide

Document Version 1.2

Americas Headquarters

Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
<http://www.cisco.com>
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883

Copyright

THE SPECIFICATIONS AND INFORMATION REGARDING THE PRODUCTS IN THIS MANUAL ARE SUBJECT TO CHANGE WITHOUT NOTICE. ALL STATEMENTS, INFORMATION, AND RECOMMENDATIONS IN THIS MANUAL ARE BELIEVED TO BE ACCURATE BUT ARE PRESENTED WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. USERS MUST TAKE FULL RESPONSIBILITY FOR THEIR APPLICATION OF ANY PRODUCTS.

THE SOFTWARE LICENSE AND LIMITED WARRANTY FOR THE ACCOMPANYING PRODUCT ARE SET FORTH IN THE INFORMATION PACKET THAT SHIPPED WITH THE PRODUCT AND ARE INCORPORATED HEREIN BY THIS REFERENCE. IF YOU ARE UNABLE TO LOCATE THE SOFTWARE LICENSE OR LIMITED WARRANTY, CONTACT YOUR CISCO REPRESENTATIVE FOR A COPY.

The Cisco implementation of TCP header compression is an adaptation of a program developed by the University of California, Berkeley (UCB) as part of UCB's public domain version of the UNIX operating system. All rights reserved. Copyright © 1981, Regents of the University of California.

NOTWITHSTANDING ANY OTHER WARRANTY HEREIN, ALL DOCUMENT FILES AND SOFTWARE OF THESE SUPPLIERS ARE PROVIDED "AS IS" WITH ALL FAULTS. CISCO AND THE ABOVE-NAMED SUPPLIERS DISCLAIM ALL WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THOSE OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OR ARISING FROM A COURSE OF DEALING, USAGE, OR TRADE PRACTICE.

IN NO EVENT SHALL CISCO OR ITS SUPPLIERS BE LIABLE FOR ANY INDIRECT, SPECIAL, CONSEQUENTIAL, OR INCIDENTAL DAMAGES, INCLUDING, WITHOUT LIMITATION, LOST PROFITS OR LOSS OR DAMAGE TO DATA ARISING OUT OF THE USE OR INABILITY TO USE THIS MANUAL, EVEN IF CISCO OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

CCDE, CCENT, Cisco Eos, Cisco HealthPresence, the Cisco logo, Cisco Lumin, Cisco Nexus, Cisco StadiumVision, Cisco TelePresence, Cisco WebEx, DCE, and Welcome to the Human Network are trademarks; Changing the Way We Work, Live, Play, and Learn and Cisco Store are service marks; and Access Registrar, Aironet, AsyncOS, Bringing the Meeting To You, Catalyst, CCDA, CCDP, CCIE, CCIP, CCNA, CCNP, CCSP, CCVP, Cisco, the Cisco Certified Internetwork Expert logo, Cisco IOS, Cisco Press, Cisco Systems, Cisco Systems Capital, the Cisco Systems logo, Cisco Unity, Collaboration Without Limitation, EtherFast, EtherSwitch, Event Center, Fast Step, Follow Me Browsing, FormShare, GigaDrive, HomeLink, Internet Quotient, IOS, iPhone, iQuick Study, IronPort, the IronPort logo, LightStream, Linksys, MediaTone, MeetingPlace, MeetingPlace Chime Sound, MGX, Networkers, Networking Academy, Network Registrar, PCNow, PIX, PowerPanels, ProConnect, ScriptShare, SenderBase, SMARTnet, Spectrum Expert, StackWise, The Fastest Way to Increase Your Internet Quotient, TransPath, WebEx, and the WebEx logo are registered trademarks of Cisco Systems, Inc. and/or its affiliates in the United States and certain other countries.

All other trademarks mentioned in this document or website are the property of their respective owners. The use of the word partner does not imply a partnership relationship between Cisco and any other company. (0812R)

Any Internet Protocol (IP) addresses and phone numbers used in this document are not intended to be actual addresses and phone numbers. Any examples, command display output, network topology diagrams, and other figures included in the document are shown for illustrative purposes only. Any use of actual IP addresses or phone numbers in illustrative content is unintentional and coincidental.

Cisco Collaboration Cloud Platform Services Reference Guide

© 2011 Cisco Systems, Inc. All rights reserved



CONTENTS

List of Figures **i**

List of Tables **iii**

About This Reference Guide v

Audience for this Guide 1-v

What this Guide Contains 1-v

CHAPTER 1

Introduction 1-1

CHAPTER 2

CCCPS Server API 2-1

Client-Server Model 2-1

GZIP Encoding 2-2

API Request Format 2-2

REST API Requests 2-3

SOAP API Requests 2-3

API Responses: <wbxapi> 2-3

Security Context Stanza: <securityContext></securityContext> 2-3

Status Stanza: <response></response> 2-4

Results Stanza: <return></return> 2-4

Login Process 2-5

Commands: cmd=<theCmd> 2-7

Login Command: cmd=login 2-8

Logout Command: cmd=logout 2-8

Create an Object Command: cmd=create 2-9

Delete Command: cmd=delete 2-10

Retrieve an Object: cmd=get 2-11

Update Object Attributes: cmd=set 2-12

Add an Element: cmd=add 2-13

Remove an Element: cmd=remove 2-14

Retrieve Definition: cmd=shape 2-15

Retrieve Value List: cmd=enum 2-16

Validate an Object: cmd=validate 2-17

Commit an Object: cmd=commit 2-18

Revert Changes: cmd=rollback 2-19

Extend the API: cmd=execute 2-20

- Pre-Configured Operations 2-21
 - CCCPS Pre-Configured Actions 2-21
 - CCCPS Pre-Configured Privileges 2-21
 - CCCPS Pre-Configured Roles 2-23
- Selection Methods 2-24
 - Conditional Identifiers: ‘where’ Parameter 2-24
 - Terminal Elements 2-25
 - Conditional Elements 2-29
 - Field Selector Format (XPATH) 2-31
 - Updated Data Transfer Format 2-32

CHAPTER 3

- Login Processes 3-1**
 - CAS and CCCPS 3-1
 - CAS login 3-1
 - CAS response 3-1
 - CCCPS login 3-2
 - CCCPS response 3-2
 - SAML2 and CCCPS 3-3

CHAPTER 4

- Objects 4-1**
 - XML Representation 4-1
 - Object Extensions 4-2
 - Object Validation 4-3
 - Recurring Elements 4-3
 - Templates 4-4
 - Email Templates 4-4
 - Reference Elements 4-5
 - Namespace Objects 4-6
 - Namespace Object: type=nspc 4-6
 - Affiliation Objects 4-7
 - User Object: type=user 4-7
 - Organization Object: type=org 4-9
 - Role Object: type=role 4-12
 - Privilege Object: type=priv 4-13
 - Policy Object: type=polc 4-13
 - Information Object Types 4-14
 - Document Object: type=doc 4-14
 - Tags Object: type=tags 4-14
 - Communication Objects 4-16

Session Info Object:type=ses	4-16
IntgServType Object:type=ist	4-17
UserIntgServ Object:type=uis	4-17

CHAPTER 5**Policies 5-1**

Policy Evaluation	5-1
Policy Definition	5-2
Privileges	5-3
Privilege Evaluation	5-3
Roles	5-3
Inheritance	5-4
Expressions	5-4

CHAPTER 6**Filter Plug-ins 6-1**

General Information	6-1
DirectorySearch	6-1

CHAPTER 7**Execute Plug-ins 7-1**

General Information	7-1
ActivationSetUserEmail	7-2
AssertPrivilege	7-4
CreatePersonalLibrary	7-5
FTSearch (FTSearch2)	7-5
GetMinUserProfile	7-7
GetUserProfile	7-8
ListTimeZones	7-9
ManageUserProfileImage	7-9
ProbeAccess	7-10
ProvisionUserComplete	7-12
Report	7-12
SendEmail	7-15
Tag	7-15
TagDensitySearch	7-16
TimeConversion	7-18
UpdateEmailTemplate	7-19

Standard Cisco Collaboration Cloud Platform Policies A-1

Path Based Policies	A-19
---------------------	------

GLOSSARY

INDEX



LIST OF FIGURES

- Figure 1-1: CCCPS Network **1-1**
- Figure 1-2: CCCPS Services **1-2**
- Figure 2-1: Typical CCCPS Interaction Example **2-2**



LIST OF TABLES

Table 2-1: Response Elements	2-4
Table 2-2: Cross-Reference of Commands and Parameters	2-7
Table 2-3: Logout Command Parameters	2-9
Table 2-4: Create Command Parameters	2-9
Table 2-5: Delete Command Parameters	2-10
Table 2-6: Get Command Parameters	2-11
Table 2-7: Set Command Parameters	2-13
Table 2-8: Add Command Parameters	2-13
Table 2-9: Remove Command Parameters	2-14
Table 2-10: Shape Command Parameters	2-15
Table 2-11: Enum Command Parameters	2-17
Table 2-12: Validate Command Parameters	2-18
Table 2-13: Commit Command Parameters	2-18
Table 2-14: Rollback Command Parameters	2-19
Table 2-15: Execute Command Parameters	2-20
Table 2-16: Pre-Configured Actions	2-21
Table 2-17: Pre-Configured Privileges	2-21
Table 2-18: Pre-Configured Roles	2-23
Table 2-19: Session Values for <senv>	2-26
Table 4-1: Objects Supported by the CCCPS	4-1
Table 4-2: Standard Elements of the Namespace Object	4-6
Table 4-3: Standard Elements of the User Object	4-7
Table 4-4: Standard Elements of the Organization Object	4-9
Table 4-5: Standard Elements of the Group Object	4-10
Table 4-6: Standard Elements of the Role Object	4-12
Table 4-7: Standard Elements of the Privilege Object	4-13
Table 4-8: Standard Elements of the Privilege Object	4-14
Table 4-9: Standard Elements of the Tag Object	4-15
Table 4-10: Standard Elements for the Session Info Object	4-16
Table 4-11: Standard Elements for the IntgServType Object	4-17
Table 4-12: Standard Elements for the UserIntgServ Object	4-17
Table 7-1: Execute Plug-ins	7-2
Table 7-2: Excerpt Definitions	7-6
Table 7-3: Sub-tree Return Elements	7-7
Table A-1: Default WebEx Platform Policies	A-1
Table A-2: Path Based Policies	A-19



About This Reference Guide

The Cisco Collaboration Cloud Platform Services (CCCPS) Reference Guide provides the primary reference materials to enable developers to use Cisco Collaboration Cloud Platform Services. It describes the overall CCCPS, the Server API with request and response examples, a detailed description of the CCCPS extensible object model, and the principles and procedures of the CCCPS policy model for access control.

Audience for this Guide

This document is intended for product designers, developers, partners and system integrators.

What this Guide Contains

The Cisco Collaboration Cloud Platform Services Reference Guide includes the following chapters:

Introduction, on page 1-1,
provides an overview of the CCCPS architecture.

CCCPS Server API, on page 2-1,
provides a description of the client-server model.

Login Processes, on page 3-1,
provides a description of the login process.

Objects, on page 4-1,
describes the extensible object model with diagrams, definitions and examples.

Policies, on page 5-1,
describes a flexible model that defines the roles and privileges of users, groups, and organizations.

Filter Plug-ins, on page 6-1,
provides descriptions of the various filter plug-ins for the Collaboration Cloud Server.

Execute Plug-ins, on page 7-1,
provides descriptions of the various execute plug-ins for the Collaboration Cloud Server.

Standard Cisco Collaboration Cloud Platform Policies, on page A-1,
describes the default group policies.

Glossary, on page GL-1,
contains a list of terms and their definitions.

Index, on page IN-1,
contains cross-references of important sections.



CHAPTER 1

Introduction

The Cisco Collaboration Cloud Platform is a client-server environment for integrating partner and public web services, including traditional WebEx services, using an extensible object model and flexible access control mechanisms, as depicted in [Figure 1-1](#).

- The Cisco Collaboration Cloud Platform Server enables the CCCPS Client to interact via SOAP / REST protocols, supported by WSDL service definitions. An extensible object model provides an infrastructure for affiliating users into organizations, groups, and for creating and manipulating documents, calendars, and other digital entities.
- The Cisco Collaboration Cloud Platform supports robust and secure logging, monitoring, filtering, and compliance services.

The below figures illustrate the CCCPS Network. CCCPS Services are provided to clients based on policies that establish user privileges according to membership in organizations and groups. Clients manage user credentials provided by login.

Figure 1-1 CCCPS Network

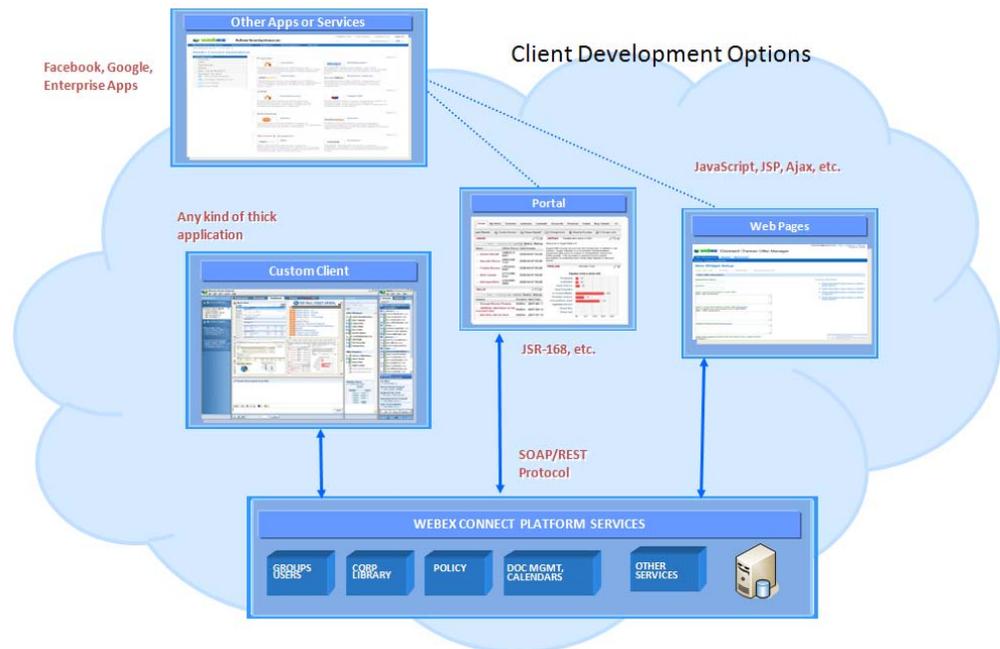
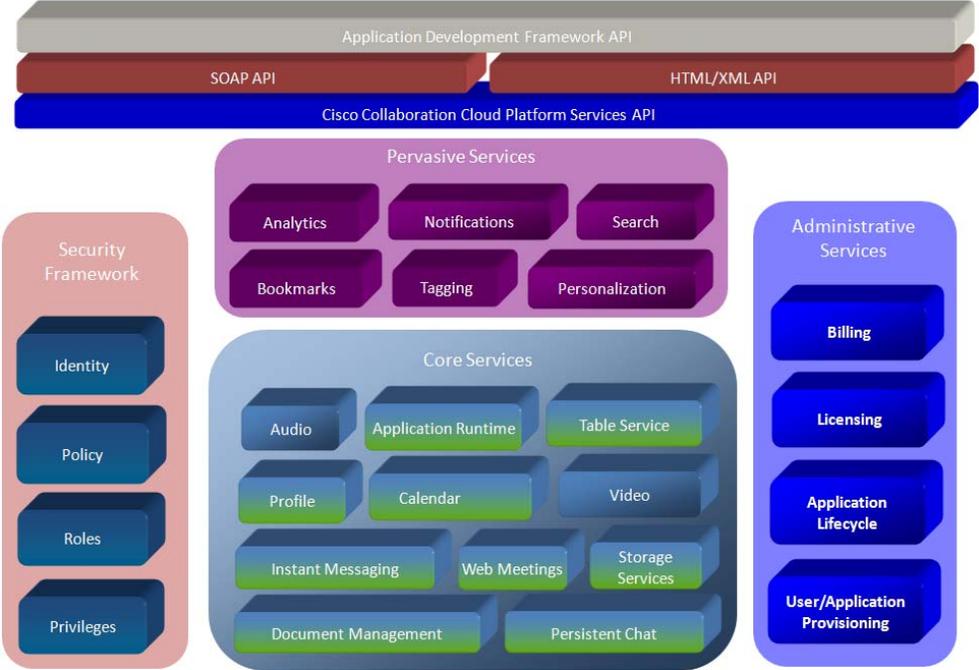


Figure 1-2 CCCPS Services





CHAPTER 2

CCCPS Server API

The Cisco Collaboration Cloud Platform Service (CCCPS) provides an Application Programming Interface (API) by which clients can obtain services. Key aspects of the API are:

- The API is exposed via Web Services, either SOAP or REST methods, allowing for a broad range of clients to connect and access the platform.
- Servers are able to accept gzip encoding.
- The interface is compact and simple. Using a few basic methods (essentially, session control, CRUD, object creation, and a custom execute method), the CCCPS provides stability to developers and partners.
- These simple API calls manipulate an extensible object model, allowing developers to customize and enhance features and functions.
- Access to the API requires a security credential in order to provide identity management. Access to objects and data elements is governed according to users privileges based on their roles within groups or organizations. See the User Provisioning & Single Sign-On Tech Note for more information.

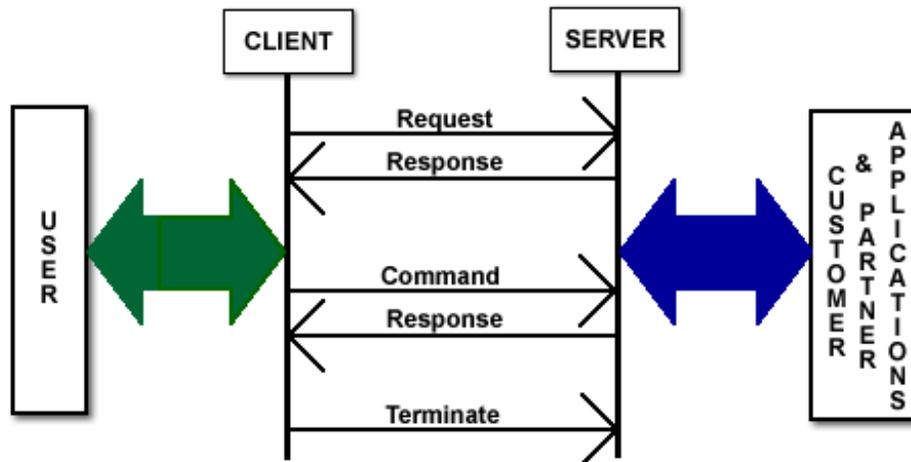
Client-Server Model

CCCPS Clients may be:

- Thick clients, running as executables in a traditional desktop environment.
- Web clients, running within web browsers or on servers that use Web Services.
- Mobile clients, such as phones or PDAs, that use 3G or other wireless Physical Layers to access the CCCPS, typically via a proxy.

A typical CCCPS interaction between client and server is illustrated in the message sequence chart below. Details about requests, responses, and commands are provided in the sections "GZIP Encoding" on page 2, "API Responses: <wbxapi>" on page 3, and "Commands: cmd=<theCmd>" on page 7.

Figure 2-1 Typical CCCPS Interaction Example



GZIP Encoding

In order to speed up the transfer of information between the server and the client, CCCPS allows files to be transferred using *gzip* encoding. This allows for the transfer of compressed files to the user, reducing the amount of time needed to transfer the file. As computers have increased their processing power, it has become faster to decode these encoded files, thereby providing a net time savings.

It is important that both the browser and the server know that it is alright to send a zipped file. The browser first sends a header to the server letting it know that it is able to accept compressed content. The server then sends a response if the content is actually compressed. If the server does not send the response header, then the file is available in a compressed format. An example of a sample browser header used to inform a server that it accepts *gzip* encoding follows:

```

User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.6; en-US; rv:1.9.1.2)
Gecko/20090729 Firefox/3.5.2
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Referer:
http://www.google.com/search?q=browser+headers&ie=utf-8&oe=utf-8&aq=t&rls=org.mozilla:en-US:official&client=firefox-a
Host: pgl.yoyo.org
X-Forwarded-For: 173.37.18.218
Connection: Keep-Alive
Cache-Control: bypass-client=173.37.18.218
Via: 1.1 Application and Content Networking System Software 5.5.13
  
```

API Request Format

CCCPS API calls are made via HTTP GET and / or POST requests.

REST API Requests

Access to the CCCPS API using REST uses either HTTP GET or POST methods with the following URL format:

```
https://swapi.webexconnect.com/wbxconnect/op.do?cmd=<theCmd>&
<otherParameters>
```

The URL for the WSDL is:

```
http://swapi.webexconnect.com/wbxconnect/services/WBXConnect?wsdl
```

SOAP API Requests

A SOAP message is an XML document. It consists of a mandatory SOAP envelope, an optional SOAP header, and a mandatory SOAP body. The envelope is the top element of the XML document and is represented by an envelope element.

Access to the CCCPS API using SOAP uses the following location:

```
http://swapi.webexconnect.com/wbxconnect/services/WBXConnect
```

API Responses: <wbxapi>

The response from an API call is always text / XML. Depending on the call, the response may have only the status and security context, or it may include data.

The root element for the response XML is `wbxapi`. For example:

```
<wbxapi>
  <securityContext>
    <cred>f383bb3bb65836202f0fff707b3d4ccc</cred>
  </securityContext>
  <response>
    <result>
      - status information -
    </result>
  </response>
  <return>
    ... more information based on the request ...
  </return>
</wbxapi>
```

Security Context Stanza: <securityContext></securityContext>

The `securityContext` encapsulates the Client's security credential (acquired at login) and uniquely identifies the API session. This credential expires at logout or after two hours of inactivity in order to prevent replay attacks and must be used in the subsequent API call. An example of a `securityContext` stanza is:

```
<securityContext>
  <cred>f383bb3bb65836202f0fff707b3d4ccc</cred>
</securityContext>
```

The `cred` element contains the unique credential for the current API session. Although this credential will be unchanged until its expiration or logout of the client, good practice is to use the response credential itself rather than a locally stored copy. This makes it possible to support multiple active credentials for a user with different organizations.

Status Stanza: <response></response>

Status is returned in a response stanza, for example:

```
<response>
  <result>FAILURE</result>
  <reason>User session has expired</reason>
  <exceptionID>wbxc.expired_credential</exceptionID>
</response>
```

A successful call will contain only a SUCCESS result, unless the response is to a GET command. In the case of a GET command, the elements `count` and `totalCount` are included. [Table 2-1](#) shows the elements used in the response stanza.

Table 2-1 Response Elements

Element	Description
result	Mandatory. SUCCESS or FAILURE.
reason	Only present on FAILURE.
exceptionID	Additional Failure information.
count	Returned with the GET command. The number of results returned per the users specifications. I.E. a page size of 20 would return <code>count = 20</code> if the page was full of found items.
totalCount	Returned with the GET command. The total number of results satisfying the GET command. <code>totalCount</code> is returned only if it differs from <code>count</code> .

Results Stanza: <return></return>

Many successful API responses return results other than just the status. In these cases the requested data is an immediate child of the `return` element.

In the following example, a site ID is returned in response to a request:

```
<wbxapi>
  <securityContext>
    <cred>f383bb3bb65836202f0fff707b3d4ccc</cred>
  </securityContext>
  <response>
    <result>SUCCESS</result>
  </response>
  <return>
    <site>
      <siteID>13342</siteID>
    </site>
  </return>
</wbxapi>
```

The following example shows multiple returned meeting domain names.

```
<wbxapi>
  <securityContext>
    <cred>f383bb3bb65836202f0fff707b3d4ccc</cred>
  </securityContext>
  <response>
    <result>SUCCESS</result>
    <count>3</count>
  </response>
  <return>
    <meetingDomain>
      <domainID>1234</domainID>
    </meetingDomain>
    <meetingDomain>
      <domainID>1235</domainID>
    </meetingDomain>
    <meetingDomain>
      <domainID>1236</domainID>
    </meetingDomain>
  </return>
</wbxapi>
```

Login Process

The login process is a two phase process that has the user logging into the Central Authentication Server (CAS) and then logging into the CCCPS server. Users use their username and password when signing into CAS and their username and a token generated by CAS to sign into CCCPS. Once signed into CAS users are able to create multiple CCCPS sessions using the token generated by CAS. Each CCCPS session will have its own set of distinct credentials.

CAS login example

Users first have to login into CAS and generate a token to be used during the CCCPS login process. Below is an example showing the initial login and the response from CAS.

CAS login

```
https://login.webexconnect.com/cas/auth.do?cmd=getwebextoken&username=xxxx&password=xxx&isp=WBX
```

CAS response

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
  <LoginResponse>
    <response>
      <result>SUCCESS</result>
    </response>
    <returnmsg>
      <username>jonathan.mi@bts4.webex.com</username>
      <token>
AAABLYVIqHoAEnUAB0cqMJ6K2TSTIara9YIgoSFX7xAD1ADHXbj7Luz4HXJv5EAAAADLJjgcuVd2uL8LwnMnhTw+iR1
ofLH060KkHT510hj5devGWQB2rI5IkCbWchDU6XzX9YqJ56R4qEFI2BpMXUncXhqUMiHEqcjZUoOLdIuLKPJ9+nnTo
cMuINM2t2BUksPbx4Sjdc4NOaGI0gs1wCvDPxHiJ6/RFaEwaCXBRZfZGJfvEQGS+CMc1hRnlf7zC1k8JZ9LtcU/OZL
cCA8cx7mYz4Pq4umwHGUP8NWx+v1ucdoKm9W4XYnkM1dfXNhvffIf6rggP3mLS3j9UmdnumqnmZVJs49PCkuEmtb5K
G5EaXA1SuXf04j/bz5fRXUmNKNre4ODIAHJWTK13WYUtYYR1NBqy1/tk7ZaL/EIT/RzDun3V//DH/cLdNaal40Ijf
Gq0jtpttrjMbcE6ahPIqYZZ
```

```

    </token>
    <serviceurl>
      wapipts.webexconnect.com/wbxconnect
    </serviceurl>
    <createtime>1295021287546</createtime>
    <timetolive>1209600</timetolive>
    <jabberToken>
zw1T9HbRFFcu1xex4kLj7jUGONMOZmOGGXXXgzB7o35pI0f/EO9eh72EeRaDYtE09gLSzn75AmTHrDqKthAkwlSfm4
ukCOnXmpEoZMz8Qq86NlUp9VebTXtuimnuIsnS+jVlu8Bk6/0YTWU3cXJtcaRDwDTAt1AZ/02wo4YaOFwYTYNMf9gy
TqRHqKodoAZE/vKm3f+Ldwxhb159D2Fq5cxZUvze6SFot5uXvp7RxxRwDEIzP/pAyx0P3mgtFqVfKkbD3ubzHQXbWe
kWQIy8uR93IgcBPjtSWX4HN3bNLI/s/tvVSYRJK2sMQ1bysbnnULDxWyS4S+BaXCeuf9xKLI3ofNsxEpND8gxkZVjw
1AqLNHh192U750fP/JQhVavU
    </jabberToken>
    <jabberCluster>
      ibt1cmx-gw.webexconnect.com
    </jabberCluster>
    <jabberName>ibt1</jabberName>
  </returnmsg>
</LoginResponse>

```

CCCPS login example

Once the user has generated a token by signing into CAS, they are able to sign into CCCPS. Below is an example showing the login process and CCCPS response.

CCCPS login

```

https://[<serviceurl>content]/op.do?cmd=login&username=xxxx&token=[<token>
content]&isp=WBX

```

CCCPS response

```

<wbxapi>
  <response>
    <result>SUCCESS</result>
  </response>
  <securityContext>
    <cred>W0112nZLsySuiwYpj5KVU8SPG00</cred>
    <additionalInfo>
      <jabberToken>
+FPuplQ4m23LL5IbD67lRnt9j9jwxddd6kv2FzDbttC9y/6tj+0dk129UkdZJ3QfXiH7jNtTf9ZCJvsh73iKwYdRg9
8D1VxVGMkaoWlSogGo0aiMCI4MQOOVB1heGWCEvJvF2jfTjOs20HMs03B2CQmrK00tSEyGeWlV1uEfy+tPjwG1/qTp
JuB16juf0Aan9PtAlx0uY01wr+wGg9/teDgjl1XhDnBR95A6i7PRnmToBWMlyBMSvjt5htXVB2UVTzot5Qn3qVMfke
U5qeF1fDwsyvg6F0Vp48m+6RHCUuUGZH7IZZokXOPO+MTVhU6hELZLCL0JjvxxbzdY5K+z6tiEvJj17Sr+6osQm9qw
CWuZuU7/4drZ4yTOEFB+kWW0
      </jabberToken>
    </additionalInfo>
  </securityContext>
  <return>
    <jabberCluster>ibt1cmx-gw.webexconnect.com</jabberCluster>
  </return>
</wbxapi>

```

After the response, use the same <serviceurl> content to call CCCPS with the <cred> content.

Jabber content is accessed by using <jabberCluster> content to call the Jabber XCP server using <jabberToken> auth or by using 'http://wwwim.ciscowebex.com/<jabberName> content' to call the Jabber BOSH server using <jabberToken> auth.

Commands: cmd=<theCmd>

The REST and SOAP API requests include a command parameter identifying the action to take. Commands are further modified by parameters specifying where and how the action will be performed.

The following actions are supported by the CCCPS API:

- login, see "Login Command: cmd=login" on page 8.
- logout, see "Logout Command: cmd=logout" on page 8.
- create, see "Create an Object Command: cmd=create" on page 9.
- delete, see "Delete Command: cmd=delete" on page 10.
- get, see "Retrieve an Object: cmd=get" on page 11.
- set, see "Update Object Attributes: cmd=set" on page 12.
- add, see "Add an Element: cmd=add" on page 13.
- remove, see "Remove an Element: cmd=remove" on page 14.
- shape, see "Retrieve Definition: cmd=shape" on page 15.
- enum, see "Retrieve Value List: cmd=enum" on page 16.
- validate, see "Validate an Object: cmd=validate" on page 17.
- commit, see "Commit an Object: cmd=commit" on page 18.
- rollback, see "Revert Changes: cmd=rollback" on page 19.
- execute, see "Extend the API: cmd=execute" on page 20.

Table 2-2 shows the parameters supported by these commands.

Table 2-2 Cross-Reference of Commands and Parameters

Parameter	login	logout	create	delete	get	enum	set	add	remove	validate	commit	rollback	shape	execute
username	X													
WBXToken	X		X	X	X	X	X	X	X				X	X
cred		X	X	X	X	X	X	X	X	X	X	X	X	X
type			X	X	X	X	X	X	X	X			X	
use			X											
id				X	X	X	X	X	X	X				
where				X	X		X	X						
order					X									
page					X									
pageSize					X									
select					X	X			X				X	
xml			X				X	X						X

Table 2-2 Cross-Reference of Commands and Parameters

Parameter	login	logout	create	delete	get	enum	set	add	remove	validate	commit	rollback	shape	execute
tzid			X		X		X	X						
locale														
task														X
ns	X													
autocommit														

Login Command: cmd=login

The `login` command request attempts to establish a session using credentials provided in its parameters. Login credentials must include a `username` parameter, but may use alternative approaches for asserting identity. These approaches are listed below.

- WBXToken plus `namespaceID`.

WBXToken plus namespaceID

This approach is a conventional login using the `username` and `WBXToken` elements, with the additional specification of a `namespaceID`, so that the user names are bound to a namespace. However, the `namespaceID` element is optional. For example:

```
cmd=login&username=<username>&token=<WBXToken>&ns=<namespaceID>
```

Login Command Response

The response to a successful `login` command is a session credential. For example:

```
<wbxapi>
  <securityContext>
    <cred>f383bb3bb65836202f0fff707b3d4ccc</cred>
  </securityContext>
  <response>
    <result>SUCCESS</result>
  </response>
</wbxapi>
```

Logout Command: cmd=logout

The `logout` command makes a session credential invalid and releases all resources allocated to the session. All saved context will be deleted.

```
cmd=logout&cred=<theCred>
```



Note

Session credentials expire after two hours of inactivity.

Table 2-3 Logout Command Parameters

Parameter	Definition
cred	Required. The credential for the current session.

Logout Command Request

The following is an example of a logout request.

```
cmd=logout&cred=f383bb3bb65836202f0fff707b3d4ccc
```

Logout Command Response

The response to a successful logout command includes a session credential. For example:

```
<wbxapi>
  <response>
    <result>SUCCESS</result>
  </response>
</wbxapi>
```

Create an Object Command: cmd=create

The create command generates a new object, such as a user, group, policy, etc.



Note

Objects are described in detail in Chapter 4.

```
cmd=create&type=<what>&use=<prototype>&xml=<initialData>&tzid=<tzid>
&cred=<theCred>
```

Table 2-4 Create Command Parameters

Parameter	Definition
type	Required. Type of object.
use	Optional. The ID of an existing object of the same type, which can be used to initialize the newly created object.
xml	Optional. The initial values of an object may be specified in an XML wrapper.
tzid	Optional. The timezoneID.
cred	Required. The session credential.

Create Command Request

The following example request creates a group having a PST time zone.

```
cmd=create&type=grp&tzid=america/los%20angeles&cred=f383bb3bb65836202f0fff707b3d4ccc
```

Create Command Response

The response to a successful `create` command contains a temporarily assigned object ID. The object must be committed, see "Commit an Object: cmd=commit" on page 18, to be made permanent. Otherwise it will be deleted upon logout or session expiration.

```
<wbxapi>
  <securityContext>
    <cred>f383bb3bb65836202f0fff707b3d4ccc</cred>
  </securityContext>
  <response>
    <result>SUCCESS</result>
  </response>
  <return>
    <group>
      <groupID>TEMP_1234</groupID>
    </group>
  </return>
</wbxapi>
```

Delete Command: cmd=delete

The `delete` command destroys an object. Users may only delete objects in the current namespace at this time.

```
cmd=delete&type=<what>&[id=<theId>|where=<theWhere>]&cred=<theCred>
```

Table 2-5 Delete Command Parameters

Parameter	Definition
type	Required. Type of object.
cred	Required. The session credential.
id	Optional. The ID of an existing object.
where	Optional. A <code>condElement</code> specifying the characteristics of objects to be deleted. See "Conditional Identifiers: 'where' Parameter" on page 24.

Instead of specifying a single element `id`, the optional `where` parameter can be used to specify a set of objects to delete. If both the `id` and `where` elements are specified then the `where` element is ignored.

Delete Command Request

The following is an example request to delete.

```
cmd=delete&type=grp&id=GSFILJE20842IFEFE&cred=f383bb3bb65836202f0fff707b3d4ccc
```

Delete Command Response

The following example response demonstrates a successful delete command.

```
<wbxapi>
```

```

<securityContext>
  <cred>f383bb3bb65836202f0fff707b3d4ccc</cred>
</securityContext>
<response>
  <result>SUCCESS</result>
</response>
</wbxapi>

```

Retrieve an Object: cmd=get

The get command retrieves attribute information from the specified object.

```
cmd=get&type=<what>&id=<theId>&select=<updatedData>&tzid=<tzid>&cred=
<theCred>
```

Table 2-6 Get Command Parameters

Parameter	Definition
type	Required. Type of object.
id	Optional. The ID of an existing object.
select	Specifies the fields to be retrieved. Refer to "Field Selector Format (XPath)" on page 31.
tzid	Optional. Time zone ID to which retrieved times are to be converted. If omitted UTC is assumed.
cred	Required. The current session credential.
where	Optional. A condElement specifying the characteristics of objects to be deleted. See "Conditional Identifiers: 'where' Parameter" on page 24.
order	Optional. Only valid in conjunction with where.
page	Optional. Only valid in conjunction with where.
pageSize	Optional. Only valid in conjunction with where.

Instead of specifying a single element id, the optional where parameter can be used to specify a set of objects to retrieve. If both the id and where elements are specified then the where element is ignored.

Optional 'where' Parameters

Additional optional parameters may be specified for the get command instead of the id parameter. When the where parameter is used, you can also provide order, page, and pageSize. These parameters are ignored if the where parameter is not specified.

The order parameter allows you to specify the order of the returned DOM trees. The value of the order parameter must be one or more DOM paths formatted as follows:

```
<domPath>, [ASC | DESC]: <domPath>, [ASC | DESC] ...
```

- If you have more than one DOM path, separate them with colons.

- For each DOM path, you can append the literal ASC or DESC to specify that the returned elements are to be in ascending or descending order respectively. If you append a literal, it must be separated from the path part by a comma. If you do not append a literal, ascending sort is assumed.

The `page` and `pageSize` parameters paginate the return response. The `pageSize` parameter specifies the number of items desired on each return. The `page` parameter specifies which block of items is desired. Page is base 1 (i.e. first indexed page is 1, not 0).

**Note**

When using pagination, the total number of objects satisfying the `where` parameter is returned in the `count` parameter of the `<response>` section.

Get Command Request

The following example request retrieves a list of users with a `userName` beginning with the letter 'A' and sorts them in descending email order:

```
cmd=get&where=<eq><path>/user/userName</path><value>A</value></eq>
&order=/user/email,DESC&select=/user/*
```

Get Command Response

The following example response is a successful get command.

```
<wbxapi>
  <securityContext>
    <cred>f383bb3bb65836202f0fff707b3d4ccc</cred>
  </securityContext>
  <response>
    <result>SUCCESS</result>
    <count>2</count>
    <totalCount>2</totalCount>
  </response>
  <return>
    <user>
      <userName>Ashley</userName>
      <email>Ashley@email.com</email>
    </user>
    <user>
      <userName>Amber</userName>
      <email>Amber@email.com</email>
    </user>
  </return>
</wbxapi>
```

Update Object Attributes: cmd=set

The `set` command updates the attributes of an object.

Table 2-7 *Set Command Parameters*

Parameter	Definition
type	Required. Type of object.
id	Optional. The ID of an existing object.
xml	Required. A partial XML tree of the data to be updated.
tzid	Optional. Time zone ID to which retrieved times are to be converted.
cred	Required. The session credential.
where	Optional. A condElement specifying the characteristic of objects to be deleted. See "Conditional Identifiers: 'where' Parameter" on page 24.

**Note**

Instead of specifying a single element `id`, the optional `where` parameter can be used to specify a set of attributes to update. If both the `id` and `where` elements are specified then the `where` element is ignored.

Add an Element: cmd=add

The `add` command inserts an element into a group of recurring elements.

```
cmd=add&type=<what>&id=<theId>&xml=<updatedData>&tzid=<tzid>&cred=<theCred>
```

Table 2-8 *Add Command Parameters*

Parameter	Definition
type	Required. Type of object.
id	Optional. The ID of an existing object.
xml	Required. A partial XML tree of the data to be updated.
tzid	Optional. Time zone ID to which retrieved times are to be converted. If omitted, UTC is assumed.
cred	Required. The session credential.
where	Optional. A condElement specifying the characteristic of objects to be added. See "Conditional Identifiers: 'where' Parameter" on page 24.

**Note**

Instead of specifying a single element `id`, the optional `where` parameter can be used to specify a set of objects whose elements can be added. If both the `id` and `where` elements are specified then the `where` element is ignored. The behavior of the `add` command falls back to `set` when applied to single value elements.

**Note**

Duplicate key errors are not generated by add. Instead, the add command behaves as an update (the same as the set command) when all of the key elements match. These key elements are contained in the xml parameter of the add command.

Add Command Request

The following example request adds a group name.

```
cmd=add&type=grp&id=GSFILJE20842IFEFE&xml=<groupName>My Group</groupName>
&tzid=america/los%angeles&cred=f383bb3bb65836202f0fff707b3d4ccc
```

Add Command Response

The following example is the response to a successful add command.

```
<wbxapi>
  <securityContext>
    <cred>f383bb3bb65836202f0fff707b3d4ccc</cred>
  </securityContext>
  <response>
    <result>SUCCESS</result>
  </response>
  <return>
    <group>
      <groupID>GSFILJE20842IFEFE</groupID>
      <groupName>My Group</groupName>
    </group>
  </return>
</wbxapi>
```

Remove an Element: cmd=remove

The remove command deletes a recurring element from an object. Users may only remove objects in the current namespace at this time.

```
cmd=remove&type=<what>&id=<theId>&select=<fieldSelector>&cred=<theCred>
```

Table 2-9 Remove Command Parameters

Parameter	Definition
type	Required. The type of the parent object.
id	Required. The parent object ID of an existing object.
select	Required. Defines the entries to be deleted, using the fieldSelector method. See "Field Selector Format (XPath)" on page 31.
cred	Required. The session credential.

**Note**

Remove only operates on recurring elements.

Remove Command Request

The following is an example request to remove the group name.

```
cmd=remove&type=grp&id=GSFILJE20842IFEFE&select=/group/*&cred=f383bb3bb65836202f0fff707b3d4ccc
```

Remove Command Response

The following is an example response from a successful `remove` command.

```
<wbxapi>
  <securityContext>
    <cred>f383bb3bb65836202f0fff707b3d4ccc</cred>
  </securityContext>
  <response>
    <result>SUCCESS</result>
  </response>
  <return>
    <group>
      <groupID>GSFILJE20842IFEFE</groupID>
    </group>
  </return>
</wbxapi>
```

Retrieve Definition: cmd=shape

The `shape` command retrieves definitions of the elements and attributes which comprise an object. This allows the developer to dynamically obtain object definitions.

```
cmd=shape&type=<what>&select=<fieldSelector>&cred=<theCred>
```

Table 2-10 *Shape Command Parameters*

Parameter	Definition
type	Required. The type of the parent object.
select	Required. Defines the entries to be retrieved, using the <code>fieldSelector</code> method. See "Field Selector Format (XPath)" on page 31.
cred	Required. The session credential.

Shape Command Request

The following is an example request to retrieve the definition of a `groupID`.

```
cmd=shape&type=grp&select=/group/[groupID='GSFILJE20842IFEFE']&cred=f383bb3bb65836202f0fff707b3d4ccc
```

Shape Command Response

The following is an example response from a successful `shape` command.

```
<wbxapi>
  <securityContext>
```

```

    <cred>f383bb3bb65836202f0fff707b3d4ccc</cred>
  </securityContext>
</response>
<result>SUCCESS</result>
</response>
</return>
  <group>
    <groupID>
      <type>X</type>
      <minlength>0</minlength>
      <maxlength>26</maxlength>
      <clearable>>false</clearable>
      <enum>>false</enum>
      <readonly>>true</readonly>
    </groupID>
    <namespaceID>
      <type>X</type>
      <minlength>0</minlength>
      <maxlength>26</maxlength>
      <clearable>>false</clearable>
      <enum>>false</enum>
      <readonly>>false</readonly>
    </namespaceID>
    <namespaceName>
      <type>X</type>
      <minlength>0</minlength>
      <maxlength>2147483647</maxlength>
      <clearable>>false</clearable>
      <enum>>false</enum>
      <readonly>>true</readonly>
    </namespaceName>
    <groupName>
      <type>X</type>
      <minlength>0</minlength>
      <maxlength>256</maxlength>
      <clearable>>false</clearable>
      <enum>>false</enum>
      <readonly>>false</readonly>
    </groupName>
    <groupType>
      <type>9</type>
      <clearable>>false</clearable>
      <enum>>true</enum>
      <readonly>>false</readonly>
      <low>0</low>
      <high>3</high>
    </groupType>
    .....
  </group>
</return>
</wbxapi>

```

Retrieve Value List: cmd=enum

The enum command retrieves a list of values and definitions that would be used in a selection list.

```
cmd=enum&type=<what>&id=<theID>&select=<fieldSelector>&cred=<theCred>
```

Table 2-11 Enum Command Parameters

Parameter	Definition
type	Optional. Suggests the type of control expected.
id	An identifier. Only required if the enumeration could be different based on the setting of certain fields in an object.
select	Required. Defines the entries to be retrieved, using the <code>fieldSelector</code> method. See "Field Selector Format (XPath)" on page 31.
cred	Required. The session credential.

**Note**

The type entry in the return from an enum request is optional and only suggests the type of control expected.

Enum Command Request

The following is an example request to retrieve a list of userNames.

```
cmd=enum&type=user&select=users/user/userName&cred=f383bb3bb65836202f0fff707b3d4ccc
```

Enum Command Response

The following is an example response from a successful enum command.

```
<wbxapi>
  <securityContext>
    <cred>f383bb3bb65836202f0fff707b3d4ccc</cred>
  </securityContext>
  <response>
    <result>SUCCESS</result>
  </response>
  <return>
    <userName>
      <type>user</type>
      <enum>
        <display>Amber</display>
        <value>Amber@email.com</value>
      </enum>
      <enum>
        <display>Ashley</display>
        <value>Ashley@email.com</value>
      </enum>
    </userName>
  </return>
</wbxapi>
```

Validate an Object: cmd=validate

The `validate` command tests that an object conforms to its definition. The `validate` command is called automatically by the `commit` command or it can be called on its own by the user. It is designed to be run before committing changes in order to locate any potential problems.

```
cmd=validate&type=<what>&id=<theId>&cred=<theCred>
```

Table 2-12 Validate Command Parameters

Parameter	Definition
type	Required. The type of the parent object.
id	Required. The ID of a specific object.
cred	Required. The session credential.

Validate Command Request

The following example request validates a group object with id GSFILJE20842IFEFE.

```
cmd=validate&type=grp&id=GSFILJE20842IFEFE&cred=f383bb3bb65836202f0fff707b3d4ccc
```

Validate Command Response

The following is an example response from a successful validate command.

```
<wbxapi>
  <securityContext>
    <cred>f383bb3bb65836202f0fff707b3d4ccc</cred>
  </securityContext>
  <response>
    <result>SUCCESS</result>
  </response>
</wbxapi>
```

Commit an Object: cmd=commit

The commit command writes all changes to the CCCPS database.

```
cmd=commit&cred=<theCred>
```

Table 2-13 Commit Command Parameters

Parameter	Definition
cred	Required. The session credential.



Note

If the commit results in the creation of one or more objects, the permanent ID(s) will be returned.



Note

All of the data manipulation API commands, (set, add, remove, etc.), create a session copy of the object. These changes are not applied to the database nor are they visible to other sessions until the commit command has been called.

Commit Command Request

The following is an example request to commit all changes.

```
cmd=commit&cred=f383bb3bb65836202f0fff707b3d4ccc
```

Commit Command Response

The following is an example response from a successful commit command.

```
<wbxapi>
  <securityContext>
    <cred>f383bb3bb65836202f0fff707b3d4ccc</cred>
  </securityContext>
  <response>
    <result>SUCCESS</result>
  </response>
  <return>
    <group>
      <groupID>GSFILJE20842IFEFE</groupID>
    </group>
  </return>
</wbxapi>
```

Revert Changes: cmd=rollback

The rollback command removes all changes to the current session.

```
cmd=rollback&cred=<theCred>
```



Note

There is no partial rollback.

Table 2-14 Rollback Command Parameters

Parameter	Definition
cred	Required. The current session credential.

Rollback Command Request

The following is an example request to revert all changes.

```
cmd=rollback&cred=f383bb3bb65836202f0fff707b3d4ccc
```

Rollback Command Response

The following is an example response from a successful rollback command.

```
<wbxapi>
  <securityContext>
    <cred>f383bb3bb65836202f0fff707b3d4ccc</cred>
  </securityContext>
  <response>
    <result>SUCCESS</result>
  </response>
</wbxapi>
```

Extend the API: cmd=execute

The `execute` command invokes an executable (shell script or program, called a plug-in) to accomplish a task outside of the published API.

Because of the complex nature of the plug-ins, some of the operations may fail, even though others succeed. Generally, if one or more operations fail, the entire plug-in has failed, and intermediate changes are not committed. Usually the originator should use the `rollback` command in this case. However, in some cases, errors are sent back with the operations that failed, allowing the originator to decide upon the appropriate action or actions. If a plug-in sends errors back, its description will indicate that.

```
cmd=execute&cred=<theCred>&task=<taskName>&xml=<taskData>
```

Table 2-15 *Execute Command Parameters*

Parameter	Definition
cred	Required. The session credential.
task	Required. The name of a task to be invoked.
xml	Optional. A partial XML tree containing input data for the task.

Execute Command Request

The following is an example request to execute a plug-in which will create an organization.

```
cmd=execute&cred=AD08SFEJ907FS&task=provisionOrg&xml=<org params>
```

The above command assumes the following DOM structure (with values), serialized. The only required parameter is the `orgName`.

```
<org>
  <orgName/>
  <description/>
  <namespaceID/>
  <policies>
    <policy>
      <policyID/>
    </policy>
  </policies>
  <initialUserFile/>
  <useOrgID/>
</org>
```

Execute Command Response

The following is an example response from a successful `execute` command using the above `execute` request.

```
<wbxapi>
  <securityContext>
    <cred>AD08SFEJ897FS</cred>
  </securityContext>
  <response>
    <result>SUCCESS</result>
  </response>
  <return>
    <org>
```

```

<orgID>OSFEILJ2343FILJSEE</orgID>
<topLevelGroupID>GSEFI234SFE32</topLevelGroupID>
<newUserGroupID>GFSEIFJE34234</newUserGroupID>
<namespaceID>NSIJLSE24234FLIE</namespaceID>
  </org>
</return>
</wbxapi>

```

Pre-Configured Operations

Pre-configured actions, privileges and roles that are built into the CCCPS are detailed in this section.



Note

Pre-configured operations are constantly changing. The tables provided below show operations that are included in the server application at the time of publication. However, these operations are subject to change at any time.

CCCPS Pre-Configured Actions

Table 2-16 is a list of pre-configured actions.

Table 2-16 *Pre-Configured Actions*

Action	Description
get	Retrieve one or more fields from an object.
set	Set one or more fields in an object.
create	Create a new object.
delete	Delete an object.
<taskName>	Execute the task defined by taskName.
OpenDocument	Retrieve a document from DMS.

To determine if a policy exception will occur, the execute plug-in `RunPolicy` is run. `RunPolicy` runs the policy engine on one or more scenarios in order to determine if the user has the access to execute the operation. This is done before the user attempts to execute the operation.

CCCPS Pre-Configured Privileges

Table 2-17 is a list of pre-configured privileges.

Table 2-17 *Pre-Configured Privileges*

Privilege	Description
WBX:AutomaticUpdates	Allows auto-updates of partner apps or client.
WBX:CreateGroup	Create a group.
WBX:DesktopShareExt	Allow desktop share with users external to the domain.

Table 2-17 Pre-Configured Privileges (continued)

Privilege	Description
WBX:DesktopShareInt	Allow desktop share with internal users only.
WBX:DisableIMCatchExt	Remove the ability to catch IM from users external to domain.
WBX:DisableIMCatchInt	Remove the ability to catch IM from internal users.
WBX:FileTransferExt	Allow File Transfer to users external to domain.
WBX:FileTransferInt	Allow File Transfer to internal users.
WBX:FileXferVirusScan	Automatically scan the file to be transferred for viruses.
WBX:GroupCreateEvent	Create a group calendar event.
WBX:GroupDeleteEvent	Delete a group calendar event.
WBX:GroupReadEvent	Read a group calendar event.
WBX:GroupUpdateEvent	Update a group calendar event.
WBX:IMExt	Allow IM with users external to domain.
WBX:IMInt	Allow IM with internal users.
WBX:InviteExt	Allow invitations to users external to domain.
WBX:InviteInt	Allow invitations to internal users.
WBX:LocalArchive	Coming Soon.
WBX:ManageGroup	Gives user the privilege to manage groups.
WBX:ManageOrg	Gives user the privilege to manage org.
WBX:ManageRoles	Gives user the privilege to manage roles, privilege, and policy.
WBX:ManageUsers	Gives user the privilege to manage users.
WBX:PremiumServices	Org has purchased Premium services.
WBX:ReadDocument	Allow user to read shared files.
WBX:RemoteControlExt	Allow user to share remote control with external users.
WBX:RemoteControlInt	Allow user to share remote control with internal users.
WBX:UpdateDocument	Allows user to access and update document.
WBX:UserCreateEvent	Create a calendar event.
WBX:UserDeleteEvent	Delete a calendar event.
WBX:UserReadEvent	Read a calendar event.
WBX:UserUpdateEvent	Update a calendar event.

Table 2-17 *Pre-Configured Privileges (continued)*

Privilege	Description
WBX:VideoExt	Allows user to create a video conference with external users.
WBX:VideoInt	Allows user to create a video conference with internal users.
WBX:VoIPExt	Allows user to create a VoIP conference with external users.
WBX:VoIPInt	Allows user to create a VoIP conference with internal users.

**Note**

While a description may imply that a privilege confers a certain capability, it is really a policy that decides whether the capability is available or not.

CCCPS Pre-Configured Roles

[Table 2-18](#) is a list of pre-configured roles.

Table 2-18 *Pre-Configured Roles*

Role	Description
WBX:Creator	Role assigned to a user signifying that this user is the creator of the context (group). It does not confer any other capabilities.
WBX:GroupAdmin	Roll assigned to the administrator of the group. The creator receives this role by default. There can be more than one group admin per group.
WBX:GroupMember	Default role assigned to a user when being added to a group. This is only used if there is no default role associated with new group members in the role object.
WBX:OrgAdmin	Role assigned to the administrator of an organization. This role is able to manage users and organizations.
WBX:OrgMember	Assigned to a user in the group that corresponds to their organization group in the company org chart. It does not confer any other capabilities.
WBX:UserAdmin	Role assigned to the administrator of an organization. This role is able to manage users.

Roles are generally associated with privileges. In fact, you can consider a role as simply a privilege container. Having said that, some of the above roles only have an identification role and are not associated with privileges. These are:

- **WBX:Creator.** This marks the current user as the creator in the context. It does not provide any additional capabilities; those have to be provided by other roles.

Selection Methods

Some API commands provide flexible methods to specify the objects and / or elements on which a command is to operate. These include:

- A `where` parameter, similar to a SQL `WHERE` clause, that logically matches elements against a set of conditions.
- A `fieldSelector` for XML that uses `XPATH` expressions to select the elements to act on.

Conditional Identifiers: ‘where’ Parameter

The `where` parameter is used instead of the `id` parameter when the user wants to apply an API call against a set of objects. The `where` parameter can result in zero or more objects being selected. The total number of objects selected is returned in the response section as `element count`.

The value of the `where` parameter is a `condElement` as defined below in the section *Conditional Logic*.

Caching

The results of applying the `where` parameter are cached in the session. Subsequent calls using the exact same `where` parameter will result in the exact same list of returned objects. This indicates that the return list is stable and not subject to random injections or deletions (which makes pagination unstable).

If the user specifies a command which changes the `where` or `order` parameter or issues a `commit` command against any object type, a new result list is generated. This may result in injections or deletions relative to the prior list.

Cached result lists are discarded after a period of non-use and / or after a `commit`.

Conditional Logic

Conditional logic is used whenever a test is required. Currently, conditional logic is available in the following circumstances:

- Internal configuration. Conditional logic is available to code prerequisite, security and validate tests. These tests are specified in the configuration of the CCCPS environment prior to startup. They can be changed during an operation only through the service manager (which implies a WebEx operations person).
- `where` parameter. API callers can set up the selection of target objects using the `where` parameter. The value of the `where` parameter is a conditional.
- Policies. Administrators, who have the privileges necessary to create or update policies, specify the conditions of the policies using conditionals.

A conditional is an XML tree. An example of how to convert a conditional expression into a parse tree is shown below.

```
(/user/UserName = 'abc') & (/user/namespaceID != 'wbx')
```

Once parsed the above conditional would look like:

```
<and>
  <eq>
    <path>/user/userName</path>
    <value>abc</value>
  </eq>
```

```

    <ne>
      <path>/user/namespaceID</path>
      <value>wbx</value>
    </ne>
  </and>

```

In the above, the operational elements (`and`, `eq`, `ne`) are called conditional elements. The source and targets (`path`, `value`) are called terminal elements. Both of these are described below (i.e., `termElement` and `condElement` are not real element names. Everywhere you see one of these entries in the examples, mentally replace them with a terminal element or conditional element).

Terminal Elements

The following elements can occur at a number of locations, but are used in the same manner; to return one or more values.

<value>

The text content of this entry is a literal value. This literal value is returned.

<path>

The text content of this entry is an absolute XPath into the same object DOM. The text content of the XPath target is the returned value. More information on XPath can be found at "Field Selector Format (XPATH)" on page 31. Note that for a `set` command, the text content is the value after the `set` has been applied (see `<priorvalue>` below).

<this>

This is a variation on `path`. In this case, the text content of the current element is returned. The current element is defined to be the element that is the parent of the configuration entry that is enclosing this `termElement`. For example, if this `termElement` were part of a `condElement` that was invoked on a security check, the current element would be the element that contains the security specification. For example, in the following structure:

```

<site>
  <metaData>
    <siteName>
      <security>
        <put>
          <eq>
            <this/>
            <value>DogBite</value>
          </eq>
        </put>
      </security>
    </siteName>
  </metaData>
</site>

```

The `this` entry refers to the current contents of the element defined by XPath: `/site/metaData/siteName`. It would then compare it to the string "DogBite" and allow an update if the two were equal.

The text value for `this` is ignored, so you would almost always use the `<this/>` variation.

In many instances, this doesn't make any sense. For this to make sense, there needs to be a focus element. A focus element will generally occur in the configuration, security or policy checks that target a specific field. For example, checking to see if a user has 'set' access to /user/userName. In this case, the focus element would be userName and this would be equated to this element.

In where parameters, this never applies.

<priorvalue>

This is a variation on this (which itself is a variation on path). The original text content of the current element is returned. This is useful for comparing the prior value to the current value. Note that during a set command, both path and this return the value after the set has been applied. Priorvalue returns the text content before any sets have been applied (that is, if you do multiple sets to the same field, priorvalue will still return the original value).

The following example is a prerequisite that will allow a set to proceed only if the prior value of a field was 0 or empty:

```
<domainID>
  <wbx>
    <prereq>
      <or failmsg='Changing web domain after initial set is not allowed'>
        <eq>
          <priorvalue/>
          <value>0</value>
        </eq>
        <eq>
          <priorvalue/>
          <value/>
        </eq>
      </or>
    </prereq>
  </wbx>
</domainID>
```

As with this, priorvalue does not apply in all circumstances.

<senv>

The text content of this entry is a constant that identifies an authorization variable for the current user (on whose behalf, this call is made). For example: adminLevel. This is a way to get the privileges of the current user for use in tests, etc. The list of constants is provided below. The term senv is short for session environment and is used to refer to values in the current user session.

senv is actually a shortcut for senvpath where the target DOM is the current superadmin user. senvpath has access to more data, but senv is faster since the requested data values were fetched beforehand.

Table 2-19 Session Values for <senv>

Text Value	Returns
cmd	The current API command.
action	The current action. This is a more general value than cmd in that it reflects the overall operation being carried out.

Table 2-19 Session Values for <sehv> (continued)

Text Value	Returns
userID	The ID of the current session user.
userName	The name of the current session user.
appName	The application name if this is an application login.
/user/...	Any element from the current session user's user object.
ISProviderID	An indicator of how the user entered the system.
namespaceID	The namespace ID for the current session.
orgID	The current active orgID for the session user. A user can belong to more than one organization.
groupID	The groupID for the current active session group. This value may be null.

<sehvpath>

This is similar to sehv except that a path is provided. The path is an absolute XPath into the sessInfo object. The sessInfo object contains information about the current user's session.

<call>

This element is actually not a terminal as it has one or more child elements. However, since this element returns a value, it is included here with the terminal elements. The two elements, class and param, are described below.

- <class> Contains the fully specified java class that implements the java interface com.webex.webapp.wapi.condexec.CallInterface. This class is expected to be in the class path as the config file logic will attempt to load it (as well as any where or policy operation). The class implements boolean, string and a java.util.List return methods. The method called depends on the circumstance.
- <param> There may be zero or more param entries within a call element. The param entry defines a named parameter that is passed to the object defined by the class element. All of the parameter entries are placed in a Map prior to the method call, so the method call signature contains only one Java parameter. The parameter entry consists of two sub-elements:
 - <name> The text value of this element is the name of the parameter. It will become the key value in the parameter map.
 - <value> The value portion of the parameter can be any of the common terminal elements: value, path or call. When using the call element, beware of endless recursion. The result is placed in the parameter map as the value of the parameter.

<case>

This is a wrapper around one or more conditional expressions. The first one that evaluates to true is used. Each expression is structured as follows:

```
<cond>
```

```

    <condElement>...see condElement below...</condElement>
    <termElement>...yes, this can be recursive...</termElement>
</cond>

```

So the entire test wrapper would look like:

```

<case>
  <cond>...conditional expression 1...</cond>
  <cond>...conditional expression 2...</cond>
  <else>...when all else fails...</else>
</case>

```

The else tag is optional. It is used if all of the conditional expressions fail.



Note

case is a terminal element, not a conditional element.

<objref>

Similar to path except it refers to a field in a different object. Since the user needs to know which object to fetch the data from, several child nodes are required:

```

<objref>
  <sourceType>...the source object type...</sourceType>
  <keyPath>...path to the key in the current object...</keyPath>
  <sourcePath>
    ...path to the desired data in the source object...
  </sourcePath>
</objref>

```

An object reference, in order to be successful, needs three pieces of information:

- The type of object to be referenced. This is either the object prefix or the object root element name.
- The key used to load the reference object. This key is gotten by following the keyPath to the key value in the current object.
- The location in the reference object of the data that is desired. If there are multiple source paths, then the content is concatenated.

Beside sourcePath, you can also have a value node. This is generally used to inject whitespace or separators when concatenating multiple source fields together.

<auxpath>

Similar to path except that it refers to a field in the auxiliary data input and is only present on ProbeAccess calls. The text associated with auxpath is an xpath expression that is evaluated relative to the aux element. For example, if the input contains the following aux entry:

```

<inputData>
  ... some other data ...
  <aux>
    <meeting>
      <meetingID>SJSE89073978742</meetingID>
      <meetingHost>U2342380423SAJFSES</meetingHost>
    </meeting>
  </aux>
</inputData>

```

Then the contents of meetingHost is tested as follows:

```
<eq>
  <send>userID</send>
  <auxpath>meeting/meetingHost</auxpath>
</eq>
```

Conditional Elements

In this document, the term “conditional element” is abbreviated to `condElement`.

<eq>

Sub-elements are two or more `termElements`. Result is true if all of the `termElements` equal each other. False otherwise.

<ne>

Sub-elements are a pair of `termElements`. Result is true if the values of the `termElements` are not equal. False otherwise.

<lt>

Sub-elements are a pair of `termElements`. Result is true if the value of the first `termElement` is less than the value of the second. False otherwise.

<gt>

Sub-elements are a pair of `termElements`. Result is true if the value of the first `termElement` is greater than the value of the second. False otherwise.

<le>

Sub-elements are a pair of `termElements`. Result is true if the value of the first `termElement` is less than or equal to the value of the second. False otherwise.

<ge>

Sub-elements are a pair of `termElements`. Result is true if the value of the first `termElement` is greater than or equal to the value of the second. False otherwise.

<not>

The single sub-element is another `condElement`. Negates the sense of the sub-element conditional.

<and>

Two or more sub-elements that are themselves `condElements`. Returns true if all of the sub-elements evaluate to true. Otherwise returns false.

<or>

Two or more sub-elements that are themselves `condElements`. Returns true if any of the sub-elements evaluate to true. Otherwise returns false.

**Note**

This is a short-cut evaluation. The first true case will cause all of the rest of the conditional elements to not be evaluated. This may have consequences if you expected a call to always be evaluated. Note: you can simulate an `if..then` conditional by negating the first conditional expression and using `or`.

<exists>

The content is a path `termElement`. The result is true if one or more nodes in the current object DOM match the path. Otherwise, false.

<required>

Same as `exists` except that the path element checked is the current element. In other words, the path element in which the `required` conditional is embedded. This element does not have any sub-elements.

<requires>

The text value for this element is a privilege. This condition check returns true if the session user has the specified privilege. It does not have any sub-elements.

<like>

There are two sub-elements. The first is generally a path `termElement`. The second evaluates to a string that contains the SQL wild card character '%'. The first and second elements are compared. If a match occurs, then the result is true, otherwise false.

As part of a `where` clause, this is rewritten to be part of a dynamic SQL call (i.e. the first sub-element is converted to a table column name and the second part is evaluated to a string which contains wild card characters). As part of any other conditional, it is evaluated by wild-card matching.

<in>

There are two or more sub-elements. The first `termElement` is compared against the second and, if present, subsequent `termElements`. Returns true if any match, false if none match. This is the equivalent of OR-ing a series of `<eq>` tests.

In the `where` parameter, the second and subsequent `termElements` are literals. The first `termElement` is a path.

<contains>

`contains` provides access to full-text searching where available.

This element can have one or two sub-elements. If one sub-element is present, then it is assumed to be a search criteria. The search is applied against all full-text indexed fields in a document.

If two sub-elements are present, the first sub-element is a path in the object to be searched. This restricts the search to the field in the target object only. The second sub-element is the search criteria.

<call>

See the <call> termElement above. This is a case where the boolean return method will be called. The result is the value of the call return.

Numeric Compares

Between pair-wise comparisons, if both values are fully numeric, a numeric compare will be done. Otherwise a non-local string compare will be done.

Field Selector Format (XPATH)

The Field Selector is an XPATH expression that specifies a set of elements within an object.

The field selector parameter is:

- One or more matching patterns separated by colons.
- Each matching pattern is an exact relative XPATH to the desired element.
- Path is relative to the root element for the object; there is no need to specify the root of the path.
- Recurring elements are specified with a selector of the form [subelement=value]. The subelement is an element that occurs within the recurring element and identifies a target element. In some complex cases, there may be multiple recurring selectors (i.e. recurrence within a recurrence).
- All patterns are unqualified; the colon separator doesn't conflict.

Below is a sample field list for returning one or more sets of userID, userName and roleName from a group object:

```
group/users/user/userID:group/users/user/userName:group/users/
user/roles/roleName
```

Because the type of the object is almost always known, it is often possible to omit the root element name.

```
users/user/userID:users/user/userName:users/user/roles/roleName
```

Below is a sample field list that selects all the information for a particular user (within a group object):

```
group/users/user[userID='U1989323AFASCIWL']/**
```

In the above, a predicate locates the desired user, instead of returning all of the users in the group object. A wildcard (/**) then returns all of the elements subsidiary to the user.

While the trailing wild-card (/**) is not necessary (because it is always implicitly present) the interior wild-cards are necessary. The following achieves the same result.

```
/**user[userID='U1989323AFASCIWL']
```

Data returned via the get operation is always some subset of the object tree. While the field selector locates the elements to return, what is actually returned is the portion of the object DOM tree containing all of the selected elements. This includes the parent elements of the selected elements and, for recurrence elements, it includes the key elements that distinguish the recurrence.

The field selector format is also used in the `remove` operation. However, with `remove`, the selected elements are those that are to be chosen to be removed.

Updated Data Transfer Format

To send element (or field) updates, the originator must code that portion of the object model tree that contains the data that is to be changed. The server performs a merge of the tree that you send and the current DOM of the object that you want to modify. This allows for full specification of the fields to be updated.

The serialized update tree is HTML encoded and placed in the XML parameter. If the update tree could potentially be large, it would be preferable to do a POST instead of a GET as in many browsers the information sent by the GET command is limited in length.

For recurring fields, you must provide the selector element in the tree structure, even though you won't be changing that element. The selector element is used to determine which occurrence is to be modified. Here is an example:

```
<group>
  <ext>
    <N124324>
      <keyValuePair>
        <key>1234</key>
        <value> new value </value>
      </keyValuePair>
    </N124324>
  </ext>
</group>
```

This would change the value of the `keyValuePair` whose key was equal to '1234' to "new value".

To add or remove a recurring field, you can use the `add` or `remove` commands, or the `set` command if the key field does not duplicate a key field in an existing recurring element.

Preceding and trailing whitespace will be removed; interior whitespace is left as is.



Note

Even though the root element name of the object was optional in the field selector list, it is required here.



CHAPTER 3

Login Processes

There are two separate login processes for the Cisco Collaboration Cloud. The first process is for users that have to go through a CAS before logging into the CCCPS server. The second process is for users that use SAML 2.0 sites.

CAS and CCCPS

The login process is a two phase process that has the user logging into the Central Authentication Server (CAS) and then logging into the CCCPS server. Users use their username and password when signing into CAS and their username and a token generated by CAS to sign into CCCPS. Once signed into CAS users are able to create multiple CCCPS sessions using the token generated by CAS. Each CCCPS session will have its own set of distinct credentials.

Users first have to login into CAS and generate a token to be used during the CCCPS login process. Below is an example showing the initial login and the response from CAS.

CAS login

```
https://loginp.webexconnect.com/cas/auth.do?cmd=getwebextoken&username=xxxx&password=xxx&isp=WBX
```

CAS response

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
  <LoginResponse>
    <response>
      <result>SUCCESS</result>
    </response>
  </returnmsg>
  <username>jonathan.mi@bts4.webex.com</username>
  <token>
AAABLYVIqHoAEnUAB0cqMJ6K2TSlAra9YIgoSFX7xADlADHXbj7Luz4HXJv5EAAAADLJjgcuvD2uL8LwnMnhTw+iRl
ofLH06OKkHT51Ohj5devGWQB2rI5IkCbwchDU6XzX9YqJ56R4qEFI2BpMXUncXhqUMiHEqcjZUoOLdTuLKPJ9+nnTo
cMuINM2t2BUksPbx4Sjdc4NOaGI0gs1wCvDPxHiJ6/RFaEwaCXBRZfZGJfvEQGS+CMc1hRnlf7zC1k8JZ9LtcU/OZL
cCA8cx7mYz4Pq4umwHGUP8Nwx+v1ucdoKm9W4XYnkM1dfXNhvffIff6rggP3mLS3j9UmdnumqnmZVJs49PCkuEmtb5K
G5EaXa1SuXf04j/bz5fRXUmNKNre4ODIAHJWTK13WYUtyYR1NBqy1/tk7ZaL/EIT/RzDun3V//DH/cLdNaal4OIJf
Gq0jtptrjMbcE6ahPIqYZZ
  </token>
  <serviceurl>
wapibts.webexconnect.com/wbxconnect
```

```

</serviceurl>
<createtime>1295021287546</createtime>
<timetolive>1209600</timetolive>
<jabberToken>
zw1T9HbRFFcu1xex4kLj7jUGONMOZmOGGXXXgzB7o35pI0f/EO9eh72EeRaDYtE09gLSzn75AmThRDqKthAkwlSfm4
ukConXmpEoZMZ8Qq86NlUp9VebTXtuimnuIsnS+jVlu8Bk6/0YTWU3cXJtcaRDWDTAt1AZ/02wo4YaOFwYTYNMf9gy
TqRHqKodoAZE/vKm3f+Ldwxhb159D2Fq5cxZUvze6SFot5uXvp7RxxRwDEIzP/pAyx0P3mgtFqVfKkbD3ubzHQXbWe
kWQIy8uR93IqcBPjtSWX4HN3bNLI/s/tvVSYRJK2sMQ1bysbnnULDxWys4S+BaXCeuf9xKLI3ofNsxEpND8gxkZVjw
1AqLNHh192U750fP/JQhVavU
</jabberToken>
<jabberCluster>
    ibt1cmx-gw.webexconnect.com
</jabberCluster>
<jabberName>ibt1</jabberName>
</returnmsg>
</LoginResponse>

```

Once the user has generated a token by signing into CAS, they are able to sign into CCCPS. Below is an example showing the login process and CCCPS response.

CCCPS login

```

https://[<serviceurl>content]/op.do?cmd=login&username=xxxx&token=[<token>
content]&isp=WBX

```

CCCPS response

```

<wbxapi>
  <response>
    <result>SUCCESS</result>
  </response>
  <securityContext>
    <cred>W0112nZLsySuiwYpj5KVU8SPG00</cred>
    <additionalInfo>
      <jabberToken>

+FPuplQ4m23LL5IbD671Rnt9j9jwxddd6kv2FzDbttC9y/6tj+0dk129UkDzJ3QfXiH7jNtTf9ZCJvsh73iKwYdRg9
8D1VxVGMkacWlSogGo0aimCI4MQOOVBlheGWceVjVf2jfTjOs20HMs03B2CQmrK00tSEyGeWlVluEfy+tPjwGl/qTp
JuB16juf0Aan9PtAlx0uY01wr+wGg9/teDgjl1XhDnBR95A6i7PRnmToBWM1yBMSvjt5htXVB2UVTzot5Qn3qVMfke
U5qeFlfDwsyvg6F0Vp48m+6RHCUuUGZH7I ZZokXOPO+MTVhU6hELZLC1OJJVxxbzDy5K+z6tiEvJj17Sr+6osQm9qw
CWuZuU7/4drZ4yTOEFB+kWW0
      </jabberToken>
    </additionalInfo>
  </securityContext>
</return>
  <jabberCluster>ibt1cmx-gw.webexconnect.com</jabberCluster>
</return>
</wbxapi>

```

After the response, use the same `<serviceurl>` content to call CCCPS with the `<cred>` content.

Jabber content is accessed by using `<jabberCluster>` content to call the Jabber XCP server using `<jabberToken>` auth or by using `'http://wwwim.ciscowebex.com/<jabberName>'` content to call the Jabber BOSH server using `<jabberToken>` auth.

SAML2 and CCCPS

SAML2 is used in IdP initiated requests only and uses a BOSH client. Connecting to the CCCPS server on SAML2 enabled sites is a four step process.

Step one: IdP verifies the user session. If no user session is found, then IdMS challenges the user.

Step two: IdP generates the SAML assertion and sends it to the CAS.

```
https://loginp.webexconnect.com/cas/SAML2AuthService?org=<Org Domain>&type=<Client Type>[&partner=<Partner Org Domain>]
```

The partner parameter is optional and is used in partner delegated authentication only.

Step three: Once the CAS has verified the assertion, it returns the CCCPS service URL, WebEx token, Jabber Cluster information, and the Jabber token.

```
<?xml version="1.0" encoding="UTF-8" ?>
<federatedSSO>
  <title>Connect Client Single Sign On</title>
  <status>SUCCESS</status>
  <screenname>jonathan.mi@bts4.webex.com</screenname>
  <createtime>1305577764556</createtime>
  <token>
AAABL/p/tswAEnUAB0f+x+phhWES4GtfV4UZ+BBg7YC+GRJ1+AoIAhNOEe7M9QAAADK4Le5uIJrnk5iboU6XfTFG72
+u1U6xsxk+qCy1UlaCfyWuQN9x2ebxpgAb6T4Wu3RA0FORByNpstrJLEAinehBt6v0uzNLZiDx2v/evE+DpH19BdVe
047qHAc1FY9zUIOzPDQY/bgVsobAmQAb1ok5v+J532WIITVAWPudCXOzZD7+nUeJ3ngydhB8hdeEOzaH/rvKNRKSQ5
G+McPXYZtRp2FVLoCZDqXGnaKd3hJOAfKjaHHj6OMks9vq+toRE+xCQC0bS3x3+noHQ/liPiZuRMsOH+Na040wkMmo
jX2/65ZuGuJjEdShIj86IZcC50vP62pSDUPOQ5fdHUFu51fO
  </token>
  <timetolive>1209600</timetolive>
  <serviceurl>wapi.bts.webexconnect.com/wbxconnect</serviceurl>
  <opisitem>WebExJabber</opisitem>
  <jabberToken>
q3jc3Iw0sAGqwqofOUzOa4QRweqKA+hPBy19acu9hGhwkjqgPihNxo+yE0IZmAQySpFSW36vNyNpELIim55uc+z/gp
u7RgDYF9jtbJIYUpqmBjMG14IEplFfW8bZZVBoQGd5jdQ1KwU0cYWrxyWtmIx4C3jAaQ9oEowNHCUbZMytMzyAUPLb
e9se4LJKKK1bEVImBIATSBT0wypkjlW3GKzLIDhaoxU5M9P+VyLqk7M2LrM7pcYOh5bEZOimaga8y6GNkdQczVvMO
cCessnVQk1gB6X+6vGNI7RIqTBMgVEEEROn68tiDzkSQdCKaCtRysdV0GsBugCijMA25Zpy10jHCwjF99txnOrAso0
dtWewuXZ0mqm6jTLV37QAvs
  </jabberToken>
  <jabberCluster>ibt1cmx-gw.webexconnect.com</jabberCluster>
  <jabberName>ibt1</jabberName>
</federatedSSO>
```

Step four: Once you have parsed out the token and the serviceurl, you must log into the CCCPS and the Jabber XCP. Use the WebEx token to log into the CCCPS using the service URL. Use the Jabber token to log into Jabber XCP using the Jabber Cluster information.



CHAPTER 4

Objects

The CCCPS provides an extensible Object model that allows developers to extend and enhance the underlying Cisco Collaboration Cloud Platform. It enables tight integration with existing applications and supports the development of completely new, independent applications. Objects are represented as XML entities and are classified into five categories.

- Namespace objects provide a method of distinguishing among identically named objects belonging to different organizations, groups, and users.
- Affiliation objects organize users into organizations and groups, and define the user's roles and privileges with the affiliated entities, typically based on policies.
- Information objects encapsulate knowledge in the form of documents, calendars and events.
- WebEx objects represent customary WebEx services, such as Meetings and Sessions.

XML Representation

Objects are represented as XML structures with a well-defined schema for each object type.

Object Types

The object types supported by the CCCPS are listed in [Table 4-1](#).

Table 4-1 Objects Supported by the CCCPS

Object Name	Type	Description
Namespace	nspc	Namespace objects provide a unique identifier to distinguish among identically named objects.
User	user	Information about a specific user.
Organization	org	The outermost group that corresponds to an organization. Unaffiliated users belong to a degenerate organization (possibly called 'unaffiliates'). The organization object is where organizational policies can be placed.
Group	grp	Information about a specific group. Groups are the general container object in the CCCPS system. They can contain other groups, users, documents, and custom objects.

Table 4-1 Objects Supported by the CCCPS (continued)

Object Name	Type	Description
Role	role	In the CCCPS system, a role is a container for one or more privileges. Roles are assigned to users in some context (such as a group). In making the assignment, you are essentially saying that this user, in this context, has these privileges. However, it is up to policy rules to determine what the privileges really mean.
Privilege	priv	A capability that a user can have. The interpretation of this capability is done by one or more policies executing in context.
Policy	polc	A policy (rule) is a set of directives that defines how access control and security is to be implemented. Policies can be associated with organizations, groups, documents, and custom objects. Policies evaluate a requested action based on the privilege in question and the context of the action.
Document	doc	A reference to a document. This document can be present in the CCCPS DMS, or it can be present in a 3rd party DMS.
Tag	tag	A way to tag objects in the CCCPS system.
SessionInfo	ses	Returns information on the state of the current user's session.
IntgServType	ist	Stores the integration service definition.
UserIntgServ	uis	Stores a integration service definition for a single user.

Object Extensions

Each object may be extended within an `<ext></ext>` stanza, allowing developers to customize objects in order to meet particular needs. All custom extensions are attached here. These extensions however are not returned via the `Shape` command.

Under the `ext` element is one or more DOM sub-trees. The root element of the sub-tree is the `namespaceID` of the owning organization.

Below is an example using three different extensions for the user object.

```
<user>
  <userID/>
  <email/>
  <name/>
  <homeGroupID/>
  <groups/>
  <ext>
    <N1234142568124526843985217>
      ... extension DOM for namespace N1234142
    </N1234142568124526843985217>
    <NAFAEIFE568124526843985217>
      ... extension DOM for namespace NAFAEIFE
    </NAFAEIFE568124526843985217>
```

```

    <N12AASQQ568124526843985217>
      ... extension DOM for namespace N12AASQQ
    </N12AASQQ568124526843985217>
  </ext>
</user>

```

**Note**

Depending on your context, you will probably only 'see' the custom extensions for your organization. Extensions for other organizations will generally be hidden unless they have exposed them for your use.

**Note**

The XSD for each standard object will accept any 'well-formed' XML within the `ext` stanza. The nature and structure of the extension and its content is entirely determined by the application that invoked the `ext` stanza, which is responsible for assuring the consistency and validity of the extension.

Object Validation

The CCCPS validates all object elements for type and length, except the `ext` element and its progeny, to assure that they conform to the schema defined for that type of object, specifically type and length.

Developer customizations are attached below the `ext` element. Although the developer is not required to declare a specific structure, the following apply:

- The developer may choose to describe valid structures for extensions and for custom objects.
- The developer may choose to describe certain fields to aid searches.
- The developer may choose to specify that certain fields are date fields. These fields are converted on input and output to UTC time.
- Validation is always done at the server when an update API call is made.

**Note**

The SOAP type for the children of the `ext` element is `XSD:ANY`.

For `set` and `add` calls, all text input is scanned for cross-site scripting attempts.

Recurring Elements

A consistent nomenclature is applied to recurring elements within objects: a plural wrapper name enclosing zero or more non-plural recurring elements, as illustrated in the following example:

```

<policies>
  <policy>
    <policyID/>
    <policyName/>
    ... more data about one policy
  </policy>
  ... more policy entries
</policies>

```

Recurring elements are path-specific. That is: `/user/policies/policy` is a different recurring element than `/user/ext/NS1313412/policies/policy`.

Recurring elements need to be identified by one or more key fields so that updates and deletes can be correctly applied. In the above example, the `policyID` is the key field. Updating an element by position is not supported.

Templates

Existing objects can be copied using the `create` command to make a new object. The new object has a copy of all the attributes of the original object. A new group could be created that has all the policies, child groups, parent groups, users and extensions of the source group. Or a role could be created with all the privileges of a source role. See “Create an Object Command: `cmd=create`” on page 9 for more detail on creating copies of existing objects.

Cisco WebEx also provides an Organizational Admin tool that helps organizations customize their templates. Information on this tool can be found at <http://www.webex.com/webexconnect/orgadmin/help/index.htm>.

Email Templates

Email templates are considered a ‘Thing’ object in CCCPS. The ‘Thing’ object is of type `emailTemplate` where the actual template is contained in the `ext` area.

```
<thing>
  <thingID/>
  <thingType>emailTemplate</thingType>
  <thingName/>
  <namespaceID/>
  <namespaceName>
  <ext>
    <emailTemplate>
      <displayName/>
      <locale/>
      <format/>
      <fromName/>
      <from/>
      <replyTo/>
      <subject/>
      <message>
        <![CDATA[
          [Actual email message]
        ]]>
      </message>
    </emailTemplate>
  </ext>
</thing>
```

The `subject` and `message` elements are able to contain variables in the format `%AAAAA%`. These variables are replaced by the appropriate values at run time.

The `thingType` has to be `emailTemplate` otherwise it will not be considered as an email template. The `thingName` is the email template name.

The `namespaceID` links the template to the organization and should hold the organization’s `namespaceID`.

The `locale` element is the standard locale notation and is in the format `languageID_countryID` (e.g. `en_US`). The `format` element determines the style of the email with possible values of `text` and `html`.

The `from` element holds the email address of the sender and `fromName` is the alias name displayed in the `from` field of the email. The reply email address is contained in the `replyTo` field. The subject and message elements can contain a combination of text and variables.

Default Email Templates

The system has the following default email templates. Whenever a new org is provisioned all these templates will be copied into the new org's namespace.

Welcome Message

The template name to reference this template is `welcomeUser`.

```
<emailTemplate>
  <displayName>Welcome Message</displayName>
  <locale>en_US</locale>
  <format>text</format>
  <fromName>Connect Admin</fromName>
  <from>connect@webex.com</from>
  <replyTo>connectAdmin@webex.com</replyTo>
  <subject>Welcome %USERNAME%</subject>
  <message><![CDATA[
You are now part of Cisco Collaboration Cloud user community!
To establish your password use the link %NEWPASSWORDURL%
To download client use the link %CLIENTDOWNLOADURL%
  ]]></message>
</emailTemplate>
```

Get/Reset Password

The template name to reference this template is `getResetPassword`.

```
<emailTemplate>
  <displayName>Get or Reset Password</displayName>
  <locale>en_US</locale>
  <format>text</format>
  <fromName>Connect Admin</fromName>
  <from>connect@webex.com</from>
  <replyTo>connectAdmin@webex.com</replyTo>
  <subject>Your password has been reset</subject>
  <message><![CDATA[
To establish a new password use the link %NEWPASSWORDURL%.
  ]]></message>
</emailTemplate>
```

Reference Elements

Some fields in the object model obtain their value from another object. These fields are called “object references”. In some cases, some of the other object elements may be included, in addition to the ID. For example, the policy object below contains both the `namespaceID` element and the `namespaceName` element.

```
<policy>
  <policyID/>
  <namespaceID/>
  <namespaceName/>
  <policyName/>
  <policyExpression/>
```

```
</policy>
```

Although the namespaceName is redundant (de-normalized in the SQL sense), since it can be obtained using the namespaceID, it is convenient to obtain it with the enclosing object to avoid an extra API call.

**Note**

Object reference elements, like namespaceName in the example, are not updateable within the enclosing object. In the example, the namespaceID could be altered within the policy object to apply the policy to a different namespace, but the namespaceName would not be changed by altering the policy object.

Namespace Objects

The CCCPS provides a Namespace Object to distinguish among identically named objects created by different organizations, groups, and users.

Namespace Object: type=nspc

The Namespace Object associates a Globally Unique Identifier (GUID) with a name, thereby uniquely identifying identically named spaces. It is used primarily to prevent name collisions.

**Note**

Previously, the namespace contained organization-wide policies based on the premise that namespace = organization. However, since a separate organization object has been added, this is no longer true.

Table 4-2 Standard Elements of the Namespace Object

Elements	Constraint	Description
namespaceID	Required. String.	A GUID or other valid and unique XML element name.
namespaceName	Required. String.	A 26 character string ('N' followed by 25 character GUID). Note Organizations are generally provisioned by WebEx with short, mnemonic, namespace Names.
ext		Custom extensions.

DOM structure for nspc:

```
<namespace>
  <namespaceID/>
  <namespaceName/>
  <allowNamespaces>
    <allowNamespace>
      <namespaceID/>
      <namespaceName/>
      <accessType/>
      <allowPath/>
    </allowNamespace>
  </allowNamespaces>
</namespace>
```

```

    <description/>
    <ext/>
</namespace>

```

The `xmlns:namespace` structure is used to allow users in another namespace access to the current namespace.

Affiliation Objects

The CCCPS provides a collection of objects pertaining to a user's affiliation with groups, organizations, and roles within their affiliations.

User Object: `type=user`

The User object represents a user of the Cisco Collaboration Cloud Platform. The user is not restricted to an organization and the design anticipates membership in more than one organization.

Table 4-3 Standard Elements of the User Object

Elements	Constraint	Description
<code>displayName</code>		Reader's name. Usually the same as the user.
<code>email</code>	Required.	User's email address.
<code>ext</code>		Custom extensions.
<code>groupID</code>	Required.	The GUID of a group to which the user belongs.
<code>groupName</code>	Object Reference.	The name of a group to which the user belongs.
<code>groupType</code>		The type of group.
<code>homeGroupID</code>		The GUID to the user's home group.
<code>isActive</code>		Boolean element that determines if a user is active or not.
<code>ISProviderID</code>		The GUID of a service provider by which the user can authenticate, such as Jabber.
<code>ISProviderName</code>	Object Reference.	The name of a service provider by which the user can authenticate.
<code>isTemplate</code>		Determines if the object is a template.
<code>namespaceID</code>		The GUID of a namespace in which the user can be located.
<code>namespaceName</code>	Object Reference.	The name of a namespace in which the user can be located.

Table 4-3 Standard Elements of the User Object (continued)

Elements	Constraint	Description
orgID		The GUID of the organization.
password	Write only field.	A password by which a user authenticates.
passwordChangedTime		When the password was last changed.
roleID		The GUID of a role which grants privileges to a user.
roleName	Object Reference.	The name of a role which grants privileges to a user.
ssoID		The GUID for the user's single sign on.
userID		The GUID of this user object.
userName		The name of the user.

DOM structure for user:

```

<user>
  <userID/>
  <email/>
  <displayName/>
  <userName/>
  <isActive/>
  <homeGroupID/>
  <groups>
    <group>
      <groupID/>
      <groupName/>
      <groupType/>
      <namespaceID/>
      <roles>
        <role>
          <roleID/>
          <roleName/>
        </role>
      </roles>
    </group>
  </groups>
  <namespaces>
    <namespace>
      <namespaceID/>
      <namespaceName/>
      <userName/>
      <password/>
      <orgID/>
      <groupID/>
      <ssoID/>
    </namespace>
  </namespaces>
  <ISProviders>
    <ISProvider>
      <ISProviderID/>

```

```

        <ISProviderName/>
        <accounts>
            <account>
                <userName/>
                <namespaceID/>
                <password/>
                <passwordChangedTime/>
            </account>
        </accounts>
    </ISProvider>
</ISProviders>
<subscribedGroups>
    <group>
        <groupID/>
        <groupName/>
    </group>
</subscribedGroups>
<ext/>
</user>

```

Organization Object: type=org

The Organization object represents a provisioned organization within the Cisco Collaboration Cloud Platform. It is the location where organizational level policies are recorded.

Table 4-4 Standard Elements of the Organization Object

Elements	Constraint	Description
orgID		The GUID of the organization.
orgName	Object Reference.	The name of the organization.
topLevelGroupID		The GUID of the topmost group within an organization. The organization may include subsidiary groups below the top level.
newUserGroupID		The GUID of a group to which new users are assigned if a group is not otherwise specified when a user object is created.
namespaceID		The GUID of a namespace associated with the organization. Note Organizations may have members with different namespaces (the unaffiliated organization for example), but in the typical case, there will be one namespace.
namespaceName	Object Reference.	The name of a namespace in which the user can be located.
policyID		The GUID of a policy object used by the organization.
policyName	Object Reference.	The name of a policy object.

Table 4-4 Standard Elements of the Organization Object (continued)

Elements	Constraint	Description
domainName		The name of the domain.
ext		Custom extensions.

DOM structure for org:

```
<org>
  <orgID/>
  <orgName/>
  <isActive/>
  <LCOrgName/>
  <orgType/>
  <isPartner/>
  <topLevelGroupID/>
  <newUserGroupID/>
  <namespaceID/>
  <namespaceName/>
  <policies>
    <policy>
      <policyID/>
      <policyName/>
    </policy>
  </policies>
  <domains>
    <domain>
      <domainName/>
    </domain>
  </domains>
  <ext/>
</org>
```

The group object represents a container of other objects within the Cisco Collaboration Cloud Platform. Groups can be arranged in a web with multiple parents and multiple children though typically they form normal hierarchies.

**Note**

Although `groupName` must be unique within its containing group, it is not necessarily unique within its namespace. A `groupName` may be reused by parents, children, uncles, and cousins, but not by siblings.

Table 4-5 Standard Elements of the Group Object

Elements	Constraint	Description
groupID	Required.	The GUID of the group.
groupName		The name of the group.
namespaceID		The GUID of a namespace to which this group belongs.
namespaceName	Object Reference.	The name of a namespace to which this group belongs.

Table 4-5 Standard Elements of the Group Object (continued)

Elements	Constraint	Description
groupType		The type of group. A number greater than 0 implies that the group is a library.
defaultRoleID		The GUID of a role which establishes the default privileges of members of this group.
defaultRoleName	Object Reference.	The name of the role which establishes the default privileges of members of this group.
userID		The GUID of a user belonging to this group.
userName	Object Reference.	The name of a user belonging to this group.
roleID		The GUID of a role which establishes the privileges of a user belonging to this group.
roleName	Object Reference.	The name of a role which establishes the privileges of a user belonging to this group.
policyID		The GUID of a policy which establishes privileges for this group.
policyName	Object Reference.	The name of a policy which establishes privileges for this group.
ext		An object extension.

DOM structure for grp:

```

<group>
  <groupID/>
  <groupName/>
  <namespaceID/>
  <namespaceName/>
  <groupType/>
  <defaultRoleID/>
  <defaultRoleName/>
  <parentGroups>
    <parentGroup>
      <groupID/>
      <groupName/>
    </parentGroup>
  </parentGroups>
  <childGroups>
    <childGroup>
      <groupID/>
      <groupName/>
    </childGroup>
  </childGroups>
  <members>
    <member>
      <userID/>
      <userName/>
      <roles>
        <role>
          <roleID/>
          <roleName/>
        </role>
      </roles>
    </member>
  </members>
</group>

```

```

        </role>
      </roles>
    </member>
  </members>
</policies>
  <policy>
    <policyID/>
    <policyName/>
  </policy>
</policies>
<ext/>
</group>

```

Role Object: type=role

The role object represents a collection of privileges associated with a user, and the user's membership in groups and organizations. The actions of users, for example to create or modify groups, are constrained by their privileges (see “*Privilege Object: type=priv*” on page 13). Roles may inherit privileges in a hierarchy.

Table 4-6 Standard Elements of the Role Object

Elements	Constraint	Description
roleID		The GUID of this role.
roleName		The name of this role.
namespaceID		The GUID of a namespace to which this role belongs.
namespaceName	Object Reference.	The name of a namespace to which this role belongs.
privilegeID		The GUID of a privilege held by this role.
privilegeName	Object Reference.	The name of a privilege held by this role.
inheritFrom	Optional.	Holds information on what role this role is inheriting privileges from.
negate	Boolean.	Explicitly denies this role from exercising its privileges.
ext		An object extension.

DOM structure for role:

```

<role>
  <roleID/>
  <namespaceID/>
  <namespaceName/>
  <roleName/>
  <inheritFrom>
    <roleID/>
    <roleName/>
    <namespaceID/>

```

```

        <namespaceName/>
    </inheritFrom>
    <privileges>
        <privilege>
            <privilegeID/>
            <namespaceID/>
            <namespaceName/>
            <privilegeName/>
            <level/>
        </privilege>
    </privileges>
    <ext/>
</role>

```

Privilege Object: type=priv

The privilege object represents the ability to act upon an object. Privileges are typically assembled into policies, and then applied to roles. For more information, see "Privileges" on page 3 in Chapter 5.

Table 4-7 Standard Elements of the Privilege Object

Elements	Constraint	Description
category		Determines who can act on an object: <ul style="list-style-type: none"> • 1 = access control • 2 = corporate policy • 3 = both
privilegeID		The GUID of a privilege.
privilegeName		The name of a privilege.
namespaceID		The GUID of a namespace to which this privilege belongs.
namespaceName	Object Reference	The name of a namespace to which this privilege belongs.

DOM structure for priv:

```

<privilege>
  <privilegeID/>
  <namespaceID/>
  <namespaceName/>
  <privilegeName/>
  <category/>
</privilege>

```

Policy Object: type=polc

The policy object represents a collection of privileges that create a framework of permitted actions. The possible actions are the CRUD primitives: Create, Retrieve, Update, and Delete. For more information, see Chapter 5.

Table 4-8 Standard Elements of the Privilege Object

Elements	Constraint	Description
inheritFrom		The object used to determine this object's privileges.
policyID		The GUID of the policy.
policyName		The name of the policy.
namespaceID		The GUID of a namespace to which this policy belongs.
namespaceName	Object Reference	The name of a namespace to which this policy belongs.
policyExpression		An expression that evaluates to a Boolean value.

DOM structure for polc:

```
<policy>
  <policyID/>
  <namespaceID/>
  <namespaceName/>
  <inheritFrom/>
  <policyName/>
  <policyExpression/>
</policy>
```

Information Object Types

The CCCPS provides a set of objects to facilitate the storage, retrieval and manipulation of information in the form of documents, events, or other resources.

Document Object: type=doc

For detailed information on the document object, see the *Cisco Collaboration Cloud Document Management Service Reference Guide*.

Tags Object: type=tags

Tags can be applied to any object in the Cisco Collaboration Cloud Platform system. Practically, only documents and custom objects will be tagged.

Table 4-9 Standard Elements of the Tag Object

Elements	Description
createTime	Time the object was created.
lastModifiedTime	Time the object was last modified.
namespaceID	The GUID of the namespace that contains this tag.
namespaceName	The name of the namespace.
private	Boolean that determines if the tag is private. TRUE = private, FALSE = public.
tag	Text of the tag.
tagID	The GUID assigned to each tag.
taggedObjectID	The object that the tag is placed on.
taggedObjectType	The object prefix of the tagged object.
tagNote	User's note about the tag. Max of 4000 chars.
tagText	Short text of the tag.
targetObjectID	GUID of the object to which the tag applies.
userID	GUID of the user.

DOM structure for tag:

```
<tag>
  <tagID/>
  <tagText/>
  <userID/>
  <userName/>
  <namespaceID/>
  <namespaceName/>
  <taggedObjectID/>
  <taggedObjectType/>
  <private/>
  <tagNote/>
  <createTime/>
  <lastModifiedTime/>
</tag>
```

A filter called “DistinctTagList” has been implemented in order to provide the user with a list of case-insensitive distinct tags. This is used in the where clause for list and get commands in order to fetch distinct tags.

```
<where>
  <filter name="DistinctTagList"/>
</where>
```

In CS6 if distinct tags specific to an object type or taggedObjectID are needed, they can be created using the param clause. These parameters are optional.

```
<where>
```

```

    <filter name="DistinctTagList">
      <param name="taggedObjectType">
        <value>doc/chrm/ ...</value>
      </param>
      <param name="taggedObjectID">
        <value> </value>
      </param>
    </filter>
  </where>

```

The filter should work in combination with other where clause elements too.

Communication Objects

Session Info Object:type=ses

The sessionInfo object is read-only. It returns information about the session state of the current user.

Table 4-10 Standard Elements for the Session Info Object

Elements	Constraint	Description
appName		The name of the application.
groupID		The GUID of the group.
id		The GUID of the session.
IsProviderID		The GUID of the provider.
locale		The locale of the user creating the session.
namespaceID		The name of the namespace.
orgID		The GUID of the organization.
sessionDuration		Holds how long the session has lasted.
timeZone		The time zone of the user creating the session.
userID		The GUID of the user.
userName		The name of the user.

DOM structure for ses:

```

<sessionInfo>
  <id/>
  <userID/>
  <userName/>
  <appName/>
  <namespaceID/>
  <IsProviderID/>
  <orgID/>
  <groupID/>
  <timeZone/>

```

```

    <locale/>
    <sessionDuration/>
</sessionInfo>

```

IntgServType Object:type=ist

The IntgServType object is used to store the integration service definition.

Structure of the IntgServType is:

Table 4-11 Standard Elements for the IntgServType Object

Elements	Constraints	Description
createTime		The time that the object was created.
description		A description of the object.
intgServTypeID		The GUID of the intgServType object.
intgServTypeName		The name of the intgServType object.
lastModifiedTime		The time that the object was last modified.

DOM structure for ist:

```

<intgServType>
  <intgServTypeID/>
  <intgServTypeName/>
  <description/>
  <createTime/>
  <lastModifiedTime/>
</intgServType>

```

UserIntgServ Object:type=uis

The UserIntgServ object is used to store the integration service for a single user. The structure of the UserIntgServ is:

Table 4-12 Standard Elements for the UserIntgServ Object

Elements	Constraints	Description
createTime		The time that the object was created.
identifier		Identifies the object.
intgServTypeID		The GUID of the intgServType object.
intgServTypeName		The name of the intgServType object.
lastModifiedTime		The time that the object was last modified.
status		Holds the status information.

Table 4-12 *Standard Elements for the UserIntgServ Object*

Elements	Constraints	Description
token		Holds the token information.
url		URL for the object.
userID		The GUID of the user.
userIntgServID		The GUID of the userIntgServ object.
userName		The name of the user.
webexUID		Holds the WebEx GUID.

DOM structure for uis:

```

<userIntgServ>
  <userIntgServID/>
  <intgServTypeID/>
  <intgServTypeName/>
  <userID/>
  <userName/>
  <url/>
  <identifier/>
  <token/>
  <status/>
  <webexUID/>
  <createTime/>
  <lastModifiedTime/>
</userIntgServ>

```



CHAPTER 5

Policies

Cisco Collaboration Cloud Platform policies determine whether or not a user may access the API, for example to Create, Retrieve, Update, or Delete (CRUD) elements or objects.

Policies are applied to each API request and are then evaluated to determine if the requesting client is permitted to perform the request. Privileges must be explicitly granted in a namespace before API requests can be performed. Typically, users obtain privileges by virtue of their role within a group or organization.

Roles may inherit privileges from other roles, which allows simple expressions to cascade into a comprehensive access control system across the Cisco Collaboration Cloud Platform.

The following information will be covered in this chapter.

- Policy application
- Privileges
- Roles
- Inheritance
- Expressions

Policies, roles, privileges and expressions are defined as XML objects to provide a flexible, extensible access control infrastructure.

- Users, groups, and organizations determine and tailor access control policy for the objects they control.
- Complex access control situations are satisfied, including ad hoc groups, and users and groups that span organizations.
- Access control is extended to object extensions.

Policy Evaluation

Each API request is evaluated against the policies to determine if it may proceed for the specific user and context. Policies may be specified for a range of contexts, such as:

- API request. Does the user have the privilege to access the API command? For example, a user might not have permission to change locale via the `setSession` API.
- Object type. Does the user have the privilege to access the object type specified in the request? For example, most users would not have permission to access the policy objects.

- **Object.** Does the user have the privilege to access a particular object, as specified by the object's namespaceID? For example, the user *Alice* has permission to access her own *user* object, but not *Bob's* user object.
- **Element.** Does the user have the privilege to access a particular element within an object? For example, the user *Alice* does not have the privilege to add the role *masterOfTheUniverse* to her user object.

A list of standard Cisco Collaboration Cloud Platform policies is provided in Appendix . Organizations and workgroups can inherit these standard policies to implement their own access control.

Policy Definition

Policies are defined using a rules grammar that logically specifies the conditions under which a privilege is granted. The rules are encapsulated in XML expressions which allow access to the fields in a document object only if all of the members of a group belong to the sponsoring organization.

```
<get-doc>
  <or>
    <eq>
      <send>extraOrgMembers</send>
      <value>0</value>
    </eq>
    <eq>
      <path>document/metaData/visibility</path>
      <value>public</value>
    </eq>
  </or>
</get-doc>
```

A more general way to write a policy that allows document access to members of the sponsoring organization, but deny it to non-members, is as follows:

```
<get-doc>
  <or>
    <eq>
      <send>orgID</send>
      <path>group/orgID</path>
    </eq>
    <eq>
      <path>document/metaData/visibility</path>
      <value>public</value>
    </eq>
  </or>
</get-doc>
```



Note

Users can be affiliated with multiple organizations (or no organization), and independently affiliated with groups that are affiliated with different organizations. To assure that this flexibility does not compromise an organization's data, privileges are denied unless affirmatively granted. Care must be taken when defining roles (see section “Roles” on page 3) to be certain that privileges are not inadvertently inherited.

Privileges

Users must have the necessary privileges for an API request to proceed. Privileges are labels that represent the permission to perform an action in the context of a particular namespace. For example, the user *Alice* might be granted the *get-doc* privilege, as in this example:

```
<privilege>
  <privilegeID>p777766</privilegeID>
  <privilegeName>get-doc</privilegeName>
  <namespaceID>n123456</namespaceID>
  <namespaceName>Acme organization</namespaceName>
</privilege>
```

A user can find their current privileges by using the `ProbeAccess` plug-in. By using this plug-in with the `dump-privileges` action, the user is able to retrieve a list of privileges for the current session.



Note

The labels are convenient for referring to privileges, but have no inherent meaning until placed into the context of a request.

Privilege Evaluation

Policies are evaluated in each namespace that applies to a request, to determine whether privileges are granted. Within each namespace, the policy expressions are evaluated as follows:

- If an explicit denial applies, the request is prohibited.
- If there are no explicit denials and an explicit allow applies, the request is permitted.

Roles

Roles are tools used to organize privileges. Privileges can be collected either explicitly via recurrence, or implicitly via inheritance. In the following example, a privilege is explicitly associated with a group.

```
<role>
  <roleid>987654zyx</roleid>
  <roleName>xxxxxx</roleName>
  <namespaceID>ch123456</namespaceID>
  <namespaceName>xxxxxx</namespaceName>
  <privileges>
    <privilege>
      <privilegeID>777766</privilegeID>
      <privilegeName>allowGetAPI</privilegeName>
      <namespaceID>n123456</namespaceID>
      <namespaceName>Acme organization</namespaceName>
    </privilege>
  </privileges>
</role>
```

Users are typically granted privileges by the roles they are assigned to. Essentially, roles give a user a set of privileges and contexts.

Inheritance

Roles and policies can use the attributes of another role or policy by inheritance. The new role or policy gets and uses their attributes from the parent role or policy. This differs from the creation by copy method in that any change made to the inherited object is immediately reflected in the inheriting object. This allows the user to change multiple inherited attributes by changing the root attribute only.

Expressions

The expression language is used to create new policies. In these implementations, the conditional expression returns true to allow an operation (i.e. explicit allow) or false otherwise. While the conditional expressions allow testing for a wide variety of conditions including the current group and user context, there is also the ability to call custom code (e.g. to call a 3rd party for clearance or to search a database).

An example of an expression is:

```
<groupCreation>
  <eq>
    <privilege>createGroup</privilege>
    <value>>true</value>
  </eq>
</groupCreation>
```

In the above, if the action equals `groupCreation`, then test for the privilege `createGroup`. If the privilege is true, then allow the operation to proceed.

A policy can have one or more of the above expressions. In any particular operation, the expressions that are evaluated are those that match the current action. Also there can be more than one policy associated with a group or user.

Another, more compact way, to write the above would be:

```
<groupCreation>
  <hasPrivilege>createGroup</hasPrivilege>
</groupCreation>
```



CHAPTER 6

Filter Plug-ins

General Information

Filter plug-ins are used to modify a search. They allow the user to easily define and implement their search routines.

Each filter plug-in is broken up into three sections: the section header, input subsection, and logic subsection.

The section header contains the name of the filter plug-in. The input subsection describes the format of the input parameters. The logic subsection describes the conditional logic of the filter.



Note

A filter is equal to a where clause, which only includes search conditions. The filter is not a separate call, it is used in the where clause of the get / list command.

DirectorySearch

The DirectorySearch filter is used by the Enterprise version of the Cisco Collaboration Cloud Platform client to perform a high performance search of an entire organization. It returns the users in the namespace of the currently logged user.

Input parameters

name

- Type = String
- Contains the name of the user.
- Is only used in SearchByName.
- This field is case insensitive.
- The name parameter takes precedence over all other name type parameters. Therefore, `firstname` and `lastname` if present will be ignored.

firstname

- Type = String
- Contains the first name of the user.

- Can be used in the SearchByField and SearchByFields cases.
- This field is case insensitive.

lastname

- Type = String
- Contains the last name of the user.
- Can be used in the SearchByField and SearchByFields cases.
- This field is case insensitive.

email

- Type = String
- Contains the email of the user.
- Can be used in the SearchByField and SearchByFields cases.
- Case insensitive in search.

isactive

- Type = String
- Indicates if the search result is an active user or an inactive user. Defaults to TRUE to indicate an active user.
- Can be used in the SearchByName, SearchByField, and SearchByFields cases.
- This is an optional parameter. If omitted, DirectorySearch behaves as if isactive is set to TRUE.

Logic

There are three types of search criteria that DirectorySearch can use: SearchByName, SearchByField, and SearchByFields.

SearchByName

SearchByName is the most commonly used search. It searches for users by name using the `firstname` and `lastname` fields. If either field matches the search criteria then that name is returned. An exception to this is when the `name` field is used. If the `name` field is used, then it takes precedence over the `firstname` and `lastname` fields so that any matches found by those fields are ignored.

SearchByField

SearchByField allows the user to search using only specific fields. For example if the user only wanted to search by first names, they would use SearchByField and the `firstname` field.

SearchByFields

SearchByFields allows the user to search multiple fields and return results that match all of the fields. For example, the user could use the `firstname` and `lastname` fields to find those users that match the input in both of those fields.

Examples

Search for active users whose first name matches xxx% or whose last name matches xxx%:

```
<filter name="DirectorySearch">
  <param name="name"><value>xxx%</value></param>
</filter>
```

Search for active users whose first name matches xxx%:

```
<filter name="DirectorySearch">
  <param name="firstname"><value>xxx%</value></param>
</filter>
```

Search for active users whose first name matches xxx% and whose last name matches yyy%:

```
<filter name="DirectorySearch">
  <param name="firstname"><value>xxx%</value></param>
  <param name="lastname"><value>yyy%</value></param>
</filter>
```

Search for inactive users whose first name matches xxx% or whose last name matches xxx%:

```
<filter name="DirectorySearch">
  <param name="name"><value>xxx%</value></param>
  <param name="isactive"><value>>false</value></param>
</filter>
```

Search for inactive users whose last name matches xxx% and whose email matches yyy%:

```
<filter name="DirectorySearch">
  <param name="lastname"><value>xxx%</value></param>
  <param name="email"><value>yyy%</value></param>
  <param name="isactive"><value>>false</value></param>
</filter>
```




CHAPTER 7

Execute Plug-ins

General Information

The execute plug-in descriptions are broken into three main parts: section header, input, and expected output.

The section header contains the plug-in's name and a description of the plug-in. The Input sub-section describes the format of the input data. It contains an XML structure (suitably URL encoded) to be put into the `xml` parameter of the API call. The Expected Output sub-section describes what the developer should expect to see on a successful response. For unsuccessful responses, you should see the usual CCCPS failure response.

Some execute plug-ins carry out complex operations, some of which may succeed and some of which may fail. Generally, if any sub-operation fails, the entire call fails and any intermediate changes are not committed. The developer should call 'rollback' in these cases.

However, in some cases, errors are sent back with the sub-operations that failed, allowing the developer to decide whether to re-issue those sub-operations or not. For the sub-operations that succeeded, the developer can assume that they were committed. In any case, if the execute plug-in exhibits this behavior, the description will say so.

Most execute plug-ins do not call commit on the changes, but some do. In particular, bulk insert operations will tend to commit all of the changes immediately. If the execute plug-in has this implicit commit behavior, the description will say so.

Table 7-1 *Execute Plug-ins*

Name	Page
ActivationSetUserEmail	page 2
AssertPrivilege	page 4
CreatePersonalLibrary	page 5
FTSearch (FTSearch2)	page 5
GetMinUserProfile	page 7
GetUserProfile	page 8
ListTimeZones	page 9
ManageUserProfileImage	page 9
ProbeAccess	page 10
ProvisionUserComplete	page 12
Report	page 12
SendEmail	page 15
Tag	page 15
TagDensitySearch	page 16
TimeConversion	page 18
UpdateEmailTemplate	page 19

ActivationSetUserEmail

The ActivationSetUserEmail plug-in sets the user's email, first and last names. On a successful execution, this information is updated and the activateEmailSent flag is set to FALSE as in the following example:

```
<user>
  <userID>JYW2q38DIF8KLKLS343US29JGK</userID>
  <email>test@webex.com</email>
  ...
  <ext>
    <WBX>
      <lastName>xxx</lastName>
      <firstName>xxx</firstName>
      <activateEmailSent>>false</activateEmailSent>
      ...
    </WBX>
  </ext>
</user>
```

Input

The token field is the nullSessionID returned by ActivationCodeRequest task while the p1 field is the ActivationCode entered by the user.

A GET example depicts the URL request below:

```
http://wiki.qa.webex.com/wbxconnect/nullsession.do?token=dced056d50ebef235c1fd196fc00e10db07
cbcd9ad5fe6b3e21823dd2f1712c&p1=AJ82CA
```

Expected Output

A successful request returns the following, where lastName and firstName are optional if they were entered by the user:

```
<?xml version="1.0" encoding="UTF-8" ?>
<wbxapi>
  <response>
    <result>SUCCESS</result>
  </response>
  <return>
    <user>
      <email>test@webex.com</email>
      <lastName>Test</lastName>
      <firstName>Test</firstName>
    </user>
  </return>
</wbxapi>
```

If a user enters an incorrect ActivationCode, a "wapi.retry" exception is thrown with the actual reason for the exception contained in the second message block. The Cisco Collaboration Cloud Platform Client needs to catch this exception and ask the user to re-enter the ActivationCode. The exception is shown as the following:

```
<?xml version="1.0" encoding="UTF-8" ?>
<wbxapi>
  <messageStrings>
    <message>
      wapi.retry - Retry last operation (nullsession).
    </message>
    <message>
      wapi.activationcode_mismatch - The activation code does not match the one in
the system.
    </message>
  </messageStrings>
  <response>
    <result>FAILURE</result>
    <exceptionID>wapi.retry</exceptionID>
    <reason>
      wapi.retry - Retry last operation (nullsession).
    </reason>
  </response>
</wbxapi>
```

AssertPrivilege

This plug-in allows the caller to give or remove a privilege from another user. This is typically used in the creation and application of privileges for TableService or similar applications. These applications can generate a potentially large number of targeted privileges (in the TableService case, several privileges will be created per user created table). The caller is assumed to be the person that is granting the privilege.

The following conditions must apply in order for this plug-in to succeed:

- The caller must have the right to grant the privilege. Typically, they would have gotten this by being the creator of the privilege (see NewPrivilege plug-in).
- The receiving user must be in the same namespace as the caller.
- If the privilege is to apply in the context of a particular group the caller must be a member of that group (or by inheritance a parent group).
- If the privilege is to apply in the context of an organization, the caller must be a member of the organization (namespace of the caller must equal the one of the namespaces of the organization).

Multiple operations can be made in one call. If an exception occurs, then all operations will fail. If no exceptions occur, all of the operations have succeeded and been committed.



Note

AssertPrivilege cannot be used to insert or delete standard WBX privileges (unless the caller is a user in the WBX namespace).

Input

```
<users>
  <user>
    <userID>...ID of the target user...</userID>
    <privilegeID>...ID of the privilege...</privilegeID>
    <privilegeName>...name of the privilege...</privilegeName>
    <op>'add' or 'remove'</op>
    <groupID>...groupID...</groupID>
    <orgID>...orgID...</orgID>
  </user>
  ... more users as necessary ...
</users>
```

Parameters are:

- `userID` - the ID of the receiving user. This user must be visible in some context (see conditions above).
- `privilegeID` - this is the ID of the privilege to be granted. Either `privilegeID` or `privilegeName` is required.
- `privilegeName` - this is the name of the privilege to be granted. The `privilegeName` is assumed to be in the current session's namespace. Either `privilegeID` or `privilegeName` is required.
- `op` - either the literal 'add' or the literal 'remove'. If not present, then 'add' is assumed.
- `groupID` - optional group ID of the group where the user is to receive the privilege.
- `orgID` - optional organization ID of the organization context where the user is to receive the privilege. In this case, insert the privilege and user combination in the top level group of the organization.

If none of the context entries (`groupID`, `orgID`) are specified, then the privilege is added for the user in the current session.

Expected Output

Returns either a normal success response or throws an exception. There is no commit call as the plug-in is internally transacted.

CreatePersonalLibrary

This plug-in creates a Personal Library for the context user. When provisioning a user, the personal library is also created then.

Input

The method needs no input XML.

Expected Output

This plug-in returns a normal success response. If any errors, it throws a `CCCPSEException`.

FTSearch (FTSearch2)

This plug-in executes a full-text search on document and thing object types. Full text searching is not field specific. All full-text indexed elements of the target objects are subject to search including tags (`FTSearch2`).



Note

Use `FTSearch2` instead of `FTSearch`. `FTSearch` is being deprecated.

The caller can constrain the search universe to the following:

- Current group only.
- Specific group identified by `groupID`.

Input

```
<FTSearch>
  <search>...the search criteria...</search>
  <restrictToGroup>...groupID...</restrictToGroup>
  <restrictToType>...type...</restrictToType>
  <select>...select list...</select>
  <page>...selected page (based 1)...</page>
  <pageSize>...returns per page...</pageSize>
  <useTags>...true or false...</useTags>
</FTSearch>
```

The element name of the outer most element (`FTSearch`) can be anything as it is ignored.

`restrictToGroup`, if specified, is either a `groupId` or the literal 'true'. If true, then the search is restricted to the current group (based on the user session). If a `groupId` is provided, then the search is restricted to the specific group. If the user does not have access to the group, an exception will be thrown.

`restrictToType`, if present, should be the type of the object expected on return, or it could also be a comma separated list of object types to restrict the result to multiple object types. If not present, then any type of object can be present in the response. For example, if you only wanted documents in the return, then you would code this parameter with 'doc' as the text value. For tags, this type restriction applies to the tagged object, not the tag itself (see `useTags` below, if you want to exclude tags).

`select` is a list of elements to be returned. It has the same function as the `select` parameter for the `get` or `list` commands, and is written the same way. Elements in the `select` list must be absolute paths, including the root element name as multiple object types can be returned. In the case that multiple object types are returned, the relative path is potentially ambiguous.

There are some elements that are always returned. See below (expected output) for these elements.

There is one special element that you can add to the `select` list: `excerpt`. The construction of the value for an excerpt is based on the object definition. The excerpt is determined as follows:

Table 7-2 *Excerpt Definitions*

Object Type	Excerpt Construction
Custom Object (xobj)	User defined excerpt. The fields used to construct the excerpt are defined in the corresponding custom object definition (xobjDef) located at path /xobjDef/ext/excerpt.
Document	The excerpt is returned as part of the search. The excerpt is the text surrounding the search match. There can be up to three such areas for each "hit".
Other built-in objects	Excerpt is defined in the object configuration at location /<objectRootElement>/excerpts.

`search` is the expression to be passed to the full-text search engine. The basic JCR search specifies the following:

- An expression consists of one or more search terms. Terms are single words or phrases within double quotes.
- Terms separated by whitespace are implicitly ANDed together.
- OR can be used between terms explicitly.
- AND has higher precedence than OR.
- Term exclusion is done by prefixing a - (minus sign) in front of a term.
- Single quotes, double quotes, hyphens and backslashes must be escaped by a backslash if they are to be treated as data. Special characters that require escaping include \ + - ! () : ^] { } ~ * ?

Terms are passed through a stemming filter before searching using the same filter as when originally creating the full text index. The stemming filter used is the 'Snowball' stemmer provided with Lucene. Stemming will remove capitalization and most incidental characters (i.e. punctuation). With stemming, the words `drive`, `driving`, `driver` and `driven` all stem to the same value and a search for any will find all of the others.

`page` and `pageSize` are optional parameters that are used to paginate the response. If provided, `page` is based 1 (i.e. the first page is 1, not 0). `pageSize` defaults to 1000, if not specified. The latter implies that absent a `pageSize` parameter, you will only get the first 1000 responses.

`useTags`, if present and set to true, applies the search criteria to tag text. If you are searching documents and do not want to include documents selected via the tags on the document, then code this element false (or leave it off). Doing so prevents unnecessary searching on the CCCPS full text index. In this case, the entire search can be passed to DMS. If, on the other hand, you want object types besides documents, then set this element to true. The inclusion of tag search will not materially affect the performance of the search in these cases.

Expected Output

For a successful search, a list of object sub-trees are returned. The sub-tree's root element is the object type (for example: doc or thing). The sub-elements of each sub-tree are the elements specified in the select list, but also include the following:

Table 7-3 Sub-tree Return Elements

Sub element name	Description
objectID	The ID of the returned object in anonymous form. The type of object can be determined by inspecting the parent element name. It is the root element name of the object type (e.g. doc or group). A specific object ID element name can be included. For those objects that match the type, the object ID element name will be returned (e.g. docID, groupID).
score	A value from 0 to 1000. This is based on the time of creation for most objects with 1000 for recently created object to 0 for objects over 30 days old. For DMS entries, this is the score returned from the opaque DMS search engine.
tagID	If the object was selected based on a tag, the ID of the tag is returned.
tagText	If the object was selected based on a tag, the tag text is returned.

All responses are sorted on descending score. This orders the returning objects on their creation times, except documents, which are ordered on search relevance. Different objects and documents are then interleaved based on this score. For tags, the relevant date is the tag creation date, not the tagged object creation date. This means that more recent tags can elevate the target object in the return list.

Most CCCPS search API calls return a `count` and a `totalCount` element. The `count` is the number of entries returned on the current call, while the `totalCount` is the total number of objects that meet the search criteria. If you enable `useTags` with `typeRestriction`, then the value of `totalCount` may be overestimated. Normally, FTSearch only transfers enough of the result list to format the current response page. To get an accurate count when the above restriction is applied, would require transferring the entire search result from the DB to the CCCPS server. This can potentially be millions of rows. So, we do not guarantee accuracy of the `totalCount` element in these cases.

Finally, policy is applied to each object being returned. This includes dynamic and specific object based policy. It is entirely possible that you will not be able to get any of the data elements of an object. However, you will always get the `objectID` and the `score`.

GetMinUserProfile

This plug-in retrieves a minimal profile of a user. This profile is comprised of `picURL`, `mobilePhone`, `homePhone`, and `businessPhone` elements.

Input

The information for a user can be retrieved in one of three ways as depicted below.

```
<users>
  <lastModifiedTime></lastModifiedTime>
  <tzid></tzid>
  <user>
    <userID></userID>
    <ISProviderID></ISProviderID>
  </user>
  <user>
    <userID></userID>
  </user>
  <user>
    <userName></userName>
    <ISProviderID></ISProviderID>
  </user>
</users>
```

Expected Output

```
<users>
  <user>
    <userID></userID>
    <userName></userName>
    <ISProviderID></ISProviderID>
    <picURL></picURL>
    <businessPhone></businessPhone>
    <homePhone></homePhone>
    <mobilePhone></mobilePhone>
  </user>
</users>
```

GetUserProfile

This plug-in is used to get a user's profile. Through this plug-in, any user can get the profile of another user.

Input

```
<code>
  <users>
    <user>
      <userID>Uxxxxx</userID>
      <userName/>
      <ISProviderID/>
    </user>
  </users>
</code>
```

Expected Output

The plug-in returns the first, middle and last name and the DMS URL.

```
<user>
  <userID/>
  <userName/>
```

```

    <email/>
  <ext>
    <WBX>
      <firstName/>
      <middleName/>
      <lastName/>
    </WBX>
  </ext>
</user>

```

ListTimeZones

Returns the list of time zones supported by CCCPS. This corresponds to the most recent `zoneInfo` tables. This plug-in is typically called to get the enumerated list of time zones, possibly for a user selection operation.

Input

This plug-in takes no input. If you provide any, it will be ignored.

Expected Output

```

<return>
  <zone>
    <zoneName>time zone name</zoneName>
    <wbxTzID>
  </zone>
  ... repeat <zone> as necessary ...
</return>

```

ManageUserProfileImage

This plug-in is executed to add a profile image by taking the following steps:

- The profile image is added to the DMS.
- The old profile image is deleted from the DMS.
- The url part of the extension is then made to point to the new profile image. The image file is stored in the DMS under the folder structure: `core/wbxc/U/Uwbxadmin/Profile/Photo/` where `Uwbxadmin` is the user ID of the user who has this profile image. The rest of the URL is the same for the other users.

Input

```

<code>
  <dmsURL/>
</code>

```

Expected Output

```

<return>
  <url>The URL of the stored profile image</url>

```

```
</return>
```

ProbeAccess

This execute plug-in allows the caller to provide a number of access tests and gets a response from the policy engine without invoking an exception. With ProbeAccess, the user provides a list of one or more 'tests'. Each requires at least an action (`cmdName` below).

There are a number of possible actions, but as a starting basis, the user can create actions by pairing an API call with an object model type. For example: `create-group` or `set-user`.

The best way to get the current list of built-in actions is to retrieve the policy with the id = 'WBX:Default'. This lists all of the actions and the conditions under which they are allowed.

There are two special actions:

- `dump-privileges` - This will return the current session state and a list of privileges that the current session user has. This list of privileges is based on the current context and represents the list of privileges that the user has in the current session, org and group environment. This context will be different if the user changes any of those variables.
- `has-privileges` - This will test a list of privileges and returns true if the user has all of the privileges in the current context. Having privileges is not the same as having the ability to carry out actions. However, for those cases where an action is tied directly to a privilege, this call can be used. It is most useful for auxiliary services such as `TableService`, that do their own enforcement.

For most actions, the current context includes the current backing group, parents of the backing group, and organization. However, for operations against a target group, the current context consists of the group itself, its parents and the organization containing the group. To `dump-privileges` in this case, the user has to provide the type and id of the group.

Input

```
<tests>
  <cmdName nsid="WBX">
    <type>objectType</type>
    <id>objectId</id>
    <path>pathInObject</path>
    <aux>...additional context data...</aux>
  </cmdName>
  ... more entries as necessary ...
</tests>
```

Where `cmdName` is the actual action to be tested. The following values are possible:

- `read` or `get`. Checks read access to the object by the current user in the current context.
- `write` or `set`. Checks update access to the object by the current user in the current context.
- `taskName` is the task name of an execute plug-in. Verifies that the current user can invoke this execute plug-in.
- `otherAction` as defined by customers. `otherAction` is a namespace specific action for which the customer has written a set of custom policy rules.

The `nsid` attribute is only required if the `cmdName` belongs to some other namespace than WBX. This only occurs on custom policy rules.

`type`, `id`, and `path` are optional unless you want to test access to a specific object (and if `path` is specified, to a specific path within the object).

aux is an optional means of passing additional information that might be required by a policy rule. The aux element can be single valued, or a complex XML structure. The values in this structure are accessed by the policy rule via the auxpath terminal element specification.

An alternative way of specifying the action uses the action element.

```
<tests>
  <test>
    <action>cmdName</action>
    <type>objectType</type>
    <id>objectId</id>
    <path>pathInObject</path>
  </test>
  ... more entries as necessary ...
</tests>
```

Dumping privileges, as mentioned above, can be done as follows:

```
<tests>
  <test>
    <action>dump-privileges</action>
  </test>
  ... more entries as necessary ...
</tests>
```

If you want to dump privileges in a group context, then:

```
<tests>
  <test>
    <action>dump-privileges</action>
    <type>group</type>
    <id>groupID</id>
  </test>
  ... more entries as necessary ...
</tests>
```

Checking for specific privileges can be done using the has-privileges action:

```
<tests>
  <test>
    <action>has-privileges</action>
    <privilegeID>WBX:GroupAdmin</privilegeID>
    <privilegeID>WBX:FileXferExt</privilegeID>
  </test>
</tests>
```

The above will return true if the user has both WBX:GroupAdmin and WBX:FileXferExt in the current context.

Expected Output

The return result will repeat the conditions of the test, but add an access and possibly a reason sub-element:

```
<return>
  <cmdName nsid="WBX">
    <type>objectType</type>
    <id>objectId</id>
    <path>pathInObject</path>
    <access>true or false</access>
    <reason>only if access=false</reason>
  </cmdName>
  ... more responses as necessary ...
```

```
</return>
```

ProvisionUserComplete

The ProvisionUserComplete plug-in provisions a user in the Cisco Collaboration Cloud Platform organization based on the user's email domain. It also creates accounts into Meeting Center if the organization has integration with it.

Input

```
<provisionUserComplete>
  <user>
    <groupID>groupID</group> ----- optional.
    <userName>screen name</userName> -----required
    <displayName>display name of the user</displayName> -- if not passed it will be
set to username.
    <email>email</email> ----- required
    <language>en</language>
    <country>US</country>
    <ssoID> optional. ID associated with the enterprise ID of the user.
    <isSendCPIPmail/> optional, Defaults to true. If false, don't send cpip mail. If
true, then decide if sending mail according to OPIS. If so, then the orgID provided is
used without checking if the username is one of the domains.
  </ext/>
-----required user ext info, it must include first and last name:
    /ext/WBX/firstName
    /ext/WBX/lastName
  </user>
</provisionUserComplete>
```

Report

The Report plug-in generates different reports asynchronously based on the parameters provided. The returned result displays if the report is started correctly, including a report ID for future reference. The report is only downloaded once a COMPLETED status is returned. A report is then generated and saved in CSV format. Only an org admin is able to submit the org admin reports.

There are three types of org admin reports that can be run via the type parameter:

- org_connect_usage_report
- org_connect_user_report
- org_connect_storage_consumption_report

Input

```
cmd=execute&task=Report&cred={cred}&xml=
<report>
  <type>org_connect_storage_consumption_report</type>
  <name>org_connect_storage_consumption_report</name>
  <params>
    <time>
      <startTime>2008-10-01 00:00:00</startTime>
      <endTime>2008-11-01 00:00:00</endTime>
    </time>
```

```

        <bucketSize>month</bucketSize>
        <outputFileName>StorageReport-yyyyymmdd.csv</outputFileName>
    </params>
</report>

```

The name of the report is specified in the name parameter. The startTime and endTime parameters use the format YYYY-MM-DD HH:MM:SS.mmm where SS.mmm is optional. The time zone used is GMT. The bucketSize parameter differs depending on what type of report is being produced. Both the org_connect_user_report and org_connect_storage_consumption_report types use the value 'month'. The org_connect_usage_report type can use the values 'hour', 'day', 'week', or 'month'. Use the extension .csv for the output file listed in the outputFileName parameter.

Expected Output

```

<wbxapi>
  <response>
    <result>SUCCESS</result>
  </response>
  <return>
    <job>
      <jobID>Z9P0UJ2DAQT42SRFY37YSTNQA7</jobID>
      <status>STARTING</status>
    </job>
    <report>
      <ID>Z9P0UJ2DAQT42SRFY37YSTNQA7</ID>
      <status>RUNNING</status>
    </report>
  </return>
</wbxapi>

```

Input

The following example returns the current status of any reports:

```

cmd=get&type=thing&cred={cred}&select=thingID:ext/WBX/jobID:ext/WBX/jobType:ext/WBX/jobName:ext/WBX/jobSubmissionTime_D:ext/WBX/status
&order=/thing/ext/WBX/jobSubmissionTime_D,DESC
&page=1&pageSize=10
&where=
<and>
  <and>
    <eq>
      <path>/thing/ext/WBX/orgID</path>
      <value>WEBEX</value>
    </eq>
    <eq>
      <path>/thing/ext/WBX/jobType</path>
      <value>report</value>
    </eq>
  </and>
  <eq>
    <path>/thing/ext/WBX/RunAsuserId</path>
    <value>U505WNEVNEDQX1NMA3NWWHZRAQ</value>
  </eq>
</and>
<gt>
  <path>/thing/ext/WBX/params/jobRunDate</path>
  <value>2008-10-28</value>
</gt>

```

```
</and>
```

Expected Output

```
<wbxapi>
  <response>
    <result>SUCCESS</result>
    <count>2</count>
    <totalCount>2</totalCount>
  </response>
  <securityContext>
    <cred>3D7vtsLclNVJ5rigVYi04VfsW00</cred>
  </securityContext>
  <return>
    <thing>
      <thingID>Z0U701TBOZDGLHCKNYX1Y3YBG</thingID>
      <ext>
        <WBX>
          <jobID>Z0U701TBOZDGLHCKNYX1Y3YBG</jobID>
          <jobType>report</jobType>
          <jobName>
            org_connect_storage_consumption_report
          </jobName>
          <jobSubmissionTime_D>
            2008-11-04 21:04:36
          </jobSubmissionTime_D>
          <status>COMPLETED</status>
        </WBX>
      </ext>
    </thing>
    <thing>
      <thingID>Z9P0UJ2DAQ42SRFY37YSTNQA7</thingID>
      <ext>
        <WBX>
          <jobID>Z9P0UJ2DAQ42SRFY37YSTNQA7</jobID>
          <jobType>report</jobType>
          <jobName>
            org_connect_storage_consumption_report
          </jobName>
          <jobSubmissionTime_D>
            2008-11-04 20:35:37
          </jobSubmissionTime_D>
          <status>COMPLETED</status>
        </WBX>
      </ext>
    </thing>
  </return>
</wbxapi>
```

Input

Shows how to cancel a running report job:

```
cmd=execute&task=Report&cred={cred}&xml=
<job>
  <jobID>Z9P0UJ2DAQ42SRFY37YSTNQA7</jobID>
  <jobType>report</jobType>
  <action>stop</action>
</job>
```

Input

Shows how to download a complete report file:

`https://swapi.webexconnect.com/wbxconnect/getfile.do?cmd=execute&task=Job&cred={cred}&xml=`

```
<job>
  <jobID>Z9P0UJ2DAQT42SRFY37YSTNQA7</jobID>
  <jobType>report</jobType>
  <action>getfile</action>
  <fileType>outputFile</fileType>
</job>
```

SendEmail

This plug-in sends email to a set of email address based on the specified org level template and a set of values.

Input

```
<email>
  <recipients>
    <to/>
    <to/>
  </recipients>
  <template/>
  <templateAttrs>
    <attribute>
      <name/>
      <value/>
    </attribute>
  </templateAttrs>
  <locale/>
</email>
```

The recipients /to element must contain valid email addresses of the recipients. The email template name is given in the template tag, the values of the variables in the template are specified by the templateAttrs/attribute/name and templateAttrs/attribute/value tags. The locale is optional but the format is languageID_countryID, for example en_US. The default value is en_US.

Expected Output

The plug-in returns the standard SUCCESS/FAILURE message.

Tag

This execute plug-in allows the caller to tag one or more objects. The objects can be of any type but, typically, they are documents or custom objects. The 'owner' of the tag is the caller. The user ID of the current context is used in this case. Tags can be marked as private, but by default they are public. This is a restriction that can be implemented by search. CCCPS does not try to enforce public / private access.

Input

```
<root>
  <tag>
    <objectID/>
    <objectType/>
    <objectText/>
    <url/>
    <tagText/>
    <tagNote/>
    <private/>
  </tag>
  ... more tags as required ...
</root>
```

The `tagNote` and `private` elements are optional.

Either the `objectID` and `objectType` needs to be provided, or `objectText` or `url` needs to be provided.

`objectID` and `objectType` are the ID and type of the object to be tagged. As an example, the object type of a document is 'doc'.

`objectText` or `url` can be plain text or a URL that needs to be tagged. This is for external entities (not stored in CCCPS). The tagged text or URL is maintained as a WBXCURL object within CCCPS.

`tagText` is a short descriptor (up to 255 characters) used to described the tag. The user tags similar objects with the same descriptor if he wants to group them via tagging. `tagText` is fully text indexed so you can structure a search using the contains operator.

`tagNote` is a longer descriptor, up to 4000 characters, that more fully describes the tag. Generally, one would not search on this.

`private` is a boolean, that if set to true, means the tag is only to be visible to the user. Default value is false (public).

Expected Output

If the tags are successfully applied, the minimal SUCCESS response is returned. If not, then an exception response is returned.

TagDensitySearch

This execute plug-in does a search for tag density records. Tag density is not a real object, nor is it associated directly with a tag. Tag density is recorded for the tagged object ID and for the words in the tag text. Since a number of different tags can refer to the same tagged object and since a number of different tags can contain the same tag words, tag density is orthogonal to either the object or the tag.

Tag density can be retrieved for any of the following:

- An explicit object ID. Caller provides a list of object IDs.
- A word in the tag text. Caller provides a list of words. The counts for each of these words will be summed.
- Objects that were tagged using a particular word. Caller provides a list of words. This is different than the prior option as in this case, the counts are on the objects that were tagged with various words.

- If a tag element is provided in the input, it can contain a where clause. The where is similar to what would have been provided with a standard "list" command. This plug-in will fetch all tagText matching the where clause, and then calculate word based densities for each of the tags and also provide back a relativeDensity for the tags.

Input

```
<reqs>
  <object>
    <objectID/>
    <word/>
  </object>
  ...and / or...
  <word>
    <word/>
  </word>
  <tag>
    <where></where>
  </tag>
  ...more search requests as necessary ...
</reqs>
```

Expected Output

The request list consists of one or more requests. The request element name is either 'object' or 'word'. The former signals that the search is expected to return object tagging counts. The latter signals that the search is expected to return word tagging counts.

For the object tagging counts, you can provide one or more objectIDs or you can provide one or more search words. Do not mix the two.

For word tagging counts, you can provide one or more search words.

The return consists of an entry for each search request. Within the entry is either the expected counts or an error. If the expected counts is returned, it will look as follows:

```
<object or word>
  ... original request values ...
  <today>xxx.xxx</today>
  <week>xxx.xxx</week>
  <twoWeek>xxx.xxx</twoWeek>
  <month>xxx.xxx</month>
  <quarter>xxx.xxx</quarter>
  <year>xxx.xxx</year>
  <cumulativeFactor>yyy.yyy</cumulativeFactor>
</object or word>
```

Where xxx.xxx is the count of the number of tags applied. The values are all adjusted based on the current GMT day. The other counts can be decimal values based on the results of the time adjustment.

Where yyy.yyy is a cumulative factor number derived by a weighted formula: $yyy.yyy = \text{dayCount} * 0.25 + \text{WeekCount} * 0.20 + \text{BiWeekly} * 0.10 + \text{Monthly}$. The weight factors can be tweaked by changing the static constants in the plug-in. If not present, then the namespace is used for the universe of tags.

In the case of a Tag based density search the output will be:

```
<tags>
  <tag>
    <tagtext/>
    <relativeDensity/>
  </tag>
```

```

</tags>
Example Call:
<reqs>
  <word>
    <word>__TAGTEXT__</word>
  </word>
</reqs>

```

The word element can contain the tagtext and this will return the counts for a tag. The output for this is:

```

<word>
  <word>__TAGTEXT__</word>
  <today>0.000</today>
  <week>0.000</week>
  <twoWeek>0.857</twoWeek>
  <month>1.484</month>
  <quarter>1.826</quarter>
  <year>1.956</year>
  <cumulativeFactor>0.821</cumulativeFactor>
</word>

```

The relativeDensity is a number between 0 and 1 and the highest/densest will have a density of 1.000.

TimeConversion

Converts a series of times from one time zone to another.

Input

```

<times>
  <time>
    <fromTime>the original time</fromTime>
    <fromTZID>the original timezone id</fromTZID>
    <toTZID>the timezone id of the converted time</toTZID>
  </time>
  ... additional entries as needed ...
</times>

```

The element name of the outermost element can be anything since it is treated as a wrapper but otherwise ignored. The following is a sample input:

```

<times>
  <time>
    <fromTime>2007-03-01 12:00:00</fromTime>
    <fromTZID>UTC</fromTZID>
    <toTZID>America/Los Angeles</toTZID>
  </time>
  <time>
    <fromTime>2007-03-01 12:00:00</fromTime>
    <fromTZID>4</fromTZID>
    <toTZID>5</toTZID>
  </time>
</times>

```

Expected Output

The return is a structure in the following format:

```

<return>

```

```

    <time> -- this repeats as necessary
    <fromTime>the original time</fromTime>
    <toTime>the converted time</toTime>
    <fromTZID>the original timezone id</fromTZID>
    <toTZID>the timezone id of the converted time</toTZID>
    </time>
  </return>

```

There will be one time entry for each corresponding entry in the input.

UpdateEmailTemplate

This execute plug-in updates the email template. This is needed because the normal "create" and "set" commands will not let HTML tags in the template. Using this plug-in writes directly to the DB.

Input

```

<code>
  <thing>
    <thingID/>
    <thingName/>
    <namespaceID/>
    <thingType/>
    <ext>
      <emailTemplate>
        <displayName/>
        <locale/>
        <format/>
        <fromName/>
        <from/>
        <replyTo/>
        <subject/>
        <message>
          <![CDATA[
            the mail text
          ]]>
        </message>
      </emailTemplate>
    </ext>
  </thing>
</code>

```

If the thingID is given, then it will update the template. If it is not given, it will create a new one in the user's namespace or it will use namespaceID if given. While creating a new template, thingName has to be given.

Expected Output

Returns a normal success message as output



Standard Cisco Collaboration Cloud Platform Policies

Table A-1 lists all of the current non calendar default policies as of August 2009. Users can get a list of current default policies by calling `policyID = WBX:Default`.

Table A-1 *Default WebEx Platform Policies*

Default Policy	XML Code
ActivationCodeRequest	<pre><ActivationCodeRequest> <true/> </ActivationCodeRequest></pre>
add_edit_autoupgrade_rules	<pre><add_edit_autoupgrade_rule> <false/> </add_edit_autoupgrade_rule></pre>
add-group	<pre><add-group> <requires>WBX:ManageGroup</requires> </add-group></pre>
add-groupThing	<pre><add-groupThing> <true/> </add-groupThing></pre>
add-policy	<pre><add-policy> <requires>WBX:ManageRoles</requires> </add-policy></pre>
add-privilege	<pre><add-privilege> <requires>WBX:ManageRoles</requires> </add-privilege></pre>
add-role	<pre><add-role> <requires>WBX:ManageRoles</requires> </add-role></pre>
add-thing	<pre><add-thing> <or> <requires>WBX:ManageGroup</requires> </or> </add-thing></pre>

Table A-1 Default WebEx Platform Policies (continued)

Default Policy	XML Code
add-user	<pre><add-user> <or> <requires>WBX:ManageUsers</requires> <eq> <send>userID</send> <path>/user/userID</path> </eq> </or> </add-user></pre>
add-user-role	<pre><add-user-role> <requires>WBX:ManageGroup</requires> </add-user-role></pre>
add-xobj	<pre><add-xobj> <call class="com.webex.webapp.wbxconnect.plugin.XObj DefIsMgr"> <param name="userID"> <send>userID</send> </param> <param name="xobjdefID"> <path>xobj/xobjDefID</path> </param> <param name="cmd"> <send>cmd</send> </param> <param name="access"> <value>write</value> </param> </call> </add-xobj></pre>
add-xobjDef	<pre><add-xobjDef> <call class="com.webex.webapp.wbxconnect.plugin.XObj DefIsMgr"> <param name="userID"> <send>userID</send> </param> <param name="xobjdefID"> <path>xobjDef/xobjDefID</path> </param> <param name="cmd"> <send>cmd</send> </param> <param name="access"> <value>mgr</value> </param> </call> </add-xobjDef></pre>
AddTabs	<pre><AddTabs> <true/> </AddTabs></pre>
AutomaticUpdates	<pre><AutomaticUpdates> <requires>WBX:AutomaticUpdates</requires> </AutomaticUpdates></pre>

Table A-1 Default WebEx Platform Policies (continued)

Default Policy	XML Code
CanDevelop	<CanDevelop> <true/> </CanDevelop>
CPIPPasswordRequest	<CPIPPasswordRequest> <true/> </CPIPPasswordRequest>
CPIPSetGIDPassword	<CPIPSetGIDPassword> <true/> </CPIPSetGIDPassword>
create-doc	<create-doc> <requires>WBX:UploadDocument</requires> </create-doc>
create-globalConfig	<create-globalConfig> <false/> </create-globalConfig>
create-group	<create-group> <requires>WBX:CreateGroup</requires> </create-group>
create-groupThing	<create-groupThing> <true/> </create-groupThing>
create-intgServType	<create-intgServType> <false/> </create-intgServType>
create-link	<create-link> <true/> </create-link>
create-objectPolicy	<create-objectPolicy> <true/> </create-objectPolicy>
create-offer	<create-offer> <requires>WBX:ManageOffers</requires> </create-offer>
create-org	<create-org> <requires>WBX:SuperAdmin</requires> </create-org>
create-orgIntgServ	<create-orgIntgServ> <requires>WBX:ManageOrg</requires> </create-orgIntgServ>
create-policy	<create-policy> <requires>WBX:ManageRoles</requires> </create-policy>
create-privilege	<create-privilege> <requires>WBX:ManageRoles</requires> </create-privilege>

Table A-1 Default WebEx Platform Policies (continued)

Default Policy	XML Code
create-role	<pre><create-role> <requires>WBX:ManageRoles</requires> </create-role></pre>
create-user	<pre><create-user> <requires>WBX:ManageUsers</requires> </create-user></pre>
create-userIntgServ	<pre><create-userIntgServ> <true/> </create-userIntgServ></pre>
create-xobj	<pre><create-xobj> <call class="com.webex.webapp.wbxconnect.plugin.XObj DefIsMgr"> <param name="userID"> <send>userID</send> </param> <param name="xobjdefID"> <path>xobj/xobjDefID</path> </param> <param name="cmd"> <send>cmd</send> </param> <param name="access"> <value>write</value> </param> </call> </create-xobj></pre>
create-xobjDef	<pre><create-xobjDef> <call class="com.webex.webapp.wbxconnect.plugin.XObj DefIsMgr"> <param name="userID"> <send>userID</send> </param> <param name="cmd"> <send>cmd</send> </param> <param name="access"> <value>mgr</value> </param> </call> </create-xobjDef></pre>
CreateMMNAVAccounts	<pre><CreateMMNAVAccounts> <true/> </CreateMMNAVAccounts></pre>
CreateUserGlobalID	<pre><CreateUserGlobalID> <true/> </CreateUserGlobalID></pre>

Table A-1 Default WebEx Platform Policies (continued)

Default Policy	XML Code
CSFEEEnable	<pre><CSFEEEnable> <call class="com.webex.webapp.wbxconnect.plugin.org. CheckOrgCSFSetting"> <param name="sourceOrgID"> <send>orgID</send> </param> </call> </CSFEEEnable></pre>
delete-countryList	<pre><delete-countryList> <false/> </delete-countryList></pre>
delete-doc	<pre><delete-doc> <requires>WBX:DeleteDocument</requires> </delete-doc></pre>
delete-globalConfig	<pre><delete-globalConfig> <false/> </delete-globalConfig></pre>
delete-group	<pre><delete-group> <requires>WBX:ManageGroup</requires> </delete-group></pre>
delete-groupThing	<pre><delete-groupThing> <true/> </delete-groupThing></pre>
delete-intgServType	<pre><delete-intgServType> <false/> </delete-intgServType></pre>
delete-link	<pre><delete-link> <true/> </delete-link></pre>
delete-objectPolicy	<pre><delete-objectPolicy> <true/> </delete-objectPolicy></pre>
delete-offer	<pre><delete-offer> <requires>WBX:ManageOffers</requires> </delete-offer></pre>
delete-org	<pre><delete-org> <requires>WBX:SuperAdmin</requires> </delete-org></pre>
delete-orgIntgServ	<pre><delete-orgIntgServ> <requires>WBX:ManageOrg</requires> </delete-orgIntgServ></pre>
delete-policy	<pre><delete-policy> <requires>WBX:ManageRoles</requires> </delete-policy></pre>

Table A-1 Default WebEx Platform Policies (continued)

Default Policy	XML Code
delete-privilege	<pre><delete-privilege> <or> <requires>WBX:ManageRoles</requires> <call class="com.webex.webapp.wbxconnect.plugin.privilege.AllowTableServiceDelete"> <param name="id"> <path>privilege/privilegeID</path> </param> </call> </or> </delete-privilege></pre>
delete-role	<pre><delete-role> <requires>WBX:ManageRoles</requires> </delete-role></pre>
delete-tag	<pre><delete-tag> <true/> </delete-tag></pre>
delete-thing	<pre><delete-thing> <true/> </delete-thing></pre>
delete-user	<pre><delete-user> <requires>WBX:ManageUsers</requires> </delete-user></pre>
delete-userIntgServ	<pre><delete-userIntgServ> <or> <requires>WBX:ManageUsers</requires> <eq> <send>userID</send> <path>/userIntgServ/userID</path> </eq> </or> </delete-userIntgServ></pre>
delete-xobj	<pre><delete-xobj> <call class="com.webex.webapp.wbxconnect.plugin.XObjDefISMgr"> <param name="userID"> <send>userID</send> </param> <param name="xobjdefID"> <path>xobj/xobjDefID</path> </param> <param name="cmd"> <send>cmd</send> </param> <param name="access"> <value>write</value> </param> </call> </delete-xobj></pre>

Table A-1 Default WebEx Platform Policies (continued)

Default Policy	XML Code
delete-xobjDef	<pre><delete-xobjDef> <call class="com.webex.webapp.wbxconnect.plugin.XObj DefIsMgr"> <param name="userID"> <send>userID</send> </param> <param name="xobjdefID"> <path>xobjDef/xobjDefID</path> </param> <param name="cmd"> <send>cmd</send> </param> <param name="access"> <value>mgr</value> </param> </call> </delete-xobjDef></pre>
DeleteCertificate	<pre><DeleteCertificate> <requires>WBX:ManageCertificate</requires> </DeleteCertificate></pre>
DesktopShareExt	<pre><DesktopShareExt> <requires>WBX:DesktopShareExt</requires> </DesktopShareExt></pre>
DesktopShareInt	<pre><DesktopShareInt> <requires>WBX:DesktopShareInt</requires> </DesktopShareInt></pre>
DisableIMCatchExt	<pre><DisableIMCatchExt> <requires>WBX:DisableIMCatchExt</requires> </DisableIMCatchExt></pre>
DisableIMCatchInt	<pre><DisableIMCatchInt> <requires>WBX:DisableIMCatchInt</requires> </DisableIMCatchInt></pre>
FileTransferExt	<pre><FileTransferExt> <requires>WBX:FileTransferExt</response> </FileTransferExt></pre>
FileTransferInt	<pre><FileTransferInt> <requires>WBX:FileTransferInt</response> </FileTransferInt></pre>
flush-autoUpgradeRules	<pre><flush-autoUpgradeRules> <false/> </flush-autoUpgradeRules></pre>
flushSiteTrackingCode	<pre><flushSiteTrackingCode> <requires>WBX:ManageOrg</requires> </flushSiteTrackingCode></pre>
ForceIndex	<pre><ForceIndex> <true/> </ForceIndex></pre>

Table A-1 Default WebEx Platform Policies (continued)

Default Policy	XML Code
FTSearch	<FTSearch> <true/> </FTSearch>
GenerateJabberCertificate	<GenerateJabberCertificate> <requires>WBX:ManageCertificate</requires> </GenerateJabberCertificate>
GenerateWebExCertificateCSR	<GenerateWebExCertificateCSR> <requires>WBX:ManageCertificate</requires> </GenerateWebExCertificateCSR>
GenerateWebExKeyAndCertificate	<GenerateWebExKeyAndCertificate> <requires>WBX:ManageCertificate</requires> </GenerateWebExKeyAndCertificate>
get-activityBucket	<get-activityBucket> <false/> </get-activityBucket>
get-activityDescriptor	<get-activityDescriptor> <false/> </get-activityDescriptor>
get-countryList	<get-countryList> <true/> </get-countryList>
get-doc	<get-doc> <requires>WBX:ReadDocument</requires> </get-doc>
get-globalConfig	<get-globalConfig> <true/> </get-globalConfig>
get-group	<get-group> <true/> </get-group>
get-groupThing	<get-groupThing> <true/> </get-groupThing>
get-intgServType	<get-intgServType> <true/> </get-intgServType>
get-link	<get-link> <true/> </get-link>
get-objectPolicy	<get-objectPolicy> <true/> </get-objectPolicy>
get-offer	<get-offer> <true/> </get-offer>

Table A-1 Default WebEx Platform Policies (continued)

Default Policy	XML Code
get-org	<pre><get-org> <true/> </get-org></pre>
get-policy	<pre><get-policy> <requires>WBX:ManageRoles</requires/> </get-policy></pre>
get-privilege	<pre><get-privilege> <requires>WBX:ManageRoles</requires/> </get-privilege></pre>
get-role	<pre><get-role> <requires>WBX:ManageRoles</requires/> </get-role></pre>
get-thing	<pre><get-thing> <true/> </get-thing></pre>
get-user	<pre><get-user> <or> <requires>WBX:ManageUsers</requires> <in> <send>namespaceID</send> <path> /user/namespaces/namespace/namespaceID </path> </in> <eq> <send>userID</send> <path>/user/userID</path> </eq> </or> </get-user></pre>
get-xobj	<pre><get-xobj> <call class="com.webex.webapp.wbxconnect.plugin.XObj DefIsMgr"> <param name="userID"> <send>userID</send> </param> <param name="xobjdefID"> <path>xobj/xobjDefID</path> </param> <param name="cmd"> <send>cmd</send> </param> <param name="access"> <value>read</value> </param> </call> </get-xobj></pre>

Table A-1 Default WebEx Platform Policies (continued)

Default Policy	XML Code
get-xobjDef	<pre><get-xobjDef> <call class="com.webex.webapp.wbxconnect.plugin.XObj DefIsMgr"> <param name="userID"> <send>userID</send> </param> <param name="xobjdefID"> <path>xobjDef/xobjDefID</path> </param> <param name="cmd"> <send>cmd</send> </param> <param name="access"> <value>read</value> </param> </call> </get-xobjDef></pre>
GetCertificate	<pre><GetCertificate> <requires>WBX:ManageCertificate</requires> </GetCertificate></pre>
GetCertificateList	<pre><GetCertificateList> <requires>WBX:ManageCertificate</requires> </GetCertificateList></pre>
GetOrgUserCount	<pre><GetOrgUserCount> <requires>WBX:ManageOrg</requires> </GetOrgUserCount></pre>
GetPrivilegeTree	<pre><GetPrivilegeTree> <true/> </GetPrivilegeTree></pre>
getSiteRegRequiredFields	<pre><getSiteRegRequiredFields> <requires>WBX:ManageOrg</requires> </getSiteRegRequiredFields></pre>
getSiteTrackingCode	<pre><getSiteTrackingCode> <true/> </getSiteTrackingCode></pre>
getSiteUser	<pre><getSiteUser> <requires>WBX:ManageOrg</requires> </getSiteUser></pre>
GetUserLibraries	<pre><GetUserLibraries> <true/> </GetUserLibraries></pre>
groupCreateEvent	<pre><groupCreateEvent> <requires>WBX:GroupCreateEvent</requires> </groupCreateEvent></pre>
groupDeleteEvent	<pre><groupDeleteEvent> <requires>WBX:GroupDeleteEvent</requires> </groupDeleteEvent></pre>

Table A-1 Default WebEx Platform Policies (continued)

Default Policy	XML Code
groupReadEvent	<pre><groupReadEvent> <requires>WBX:GroupReadEvent</requires> </groupReadEvent></pre>
groupUpdateEvent	<pre><groupUpdateEvent> <requires>WBX:GroupUpdateEvent</requires> </groupUpdateEvent></pre>
IM_NO_ENCODING	<pre><IM_NO_ENCODING> <true/> </IM_NO_ENCODING></pre>
IM_SSL_ENCODING	<pre><IM_SSL_ENCODING> <true/> </IM_SSL_ENCODING></pre>
IMExt	<pre><IMExt> <and> <requires>WBX:IM</requires> <eq> <senvpath> /org/ext/WBXCR/services/service[nam e='IM']/enabled </senvpath> <value>true</value> </eq> </and> </IMExt></pre>
IMInt	<pre><IMInt> <and> <requires>WBX:IMInt</requires> <eq> <senvpath> /org/ext/WBXCR/services/service[nam e='IM']/enabled </senvpath> <value>true</value> </eq> </and> </IMInt></pre>
IMIntWhitelist	<pre><IMIntWhitelist> <and> <requires>WBX:IMIntWhitelist</requires> <eq> <senvpath> /org/ext/WBXCR/services/service[nam e='IM']/enabled </senvpath> <value>true</value> </eq> </and> </IMIntWhitelist></pre>
ImportCertificate	<pre><ImportCertificate> <requires>WBX:ManageCertificate</requires> </ImportCertificate></pre>

Table A-1 Default WebEx Platform Policies (continued)

Default Policy	XML Code
ImportJabbercertificate	<pre><ImportJabberCertificate> <requires>WBX:ManageCertificate</requires> </ImportJabberCertificate></pre>
InitiateUserMigration	<pre><InitiateUserMigration> <requires>WBX:ManageUsers</requires> </InitiateUserMigration></pre>
ListTimeZones	<pre><ListTimeZones> <true/> </ListTimeZones></pre>
LocalArchive	<pre><LocalArchive> <requires>WBX:LocalArchive</requires> </LocalArchive></pre>
ManageProfile	<pre><ManageProfile> <ne> <senvpath> /org/ext/WBXCR/organizationAttributes/ enterpriseDirectory_B </senvpath> <value>true</value> </ne> </ManageProfile></pre>
ManageViewProfileSetting	<pre><ManageViewProfileSetting> <call class="com.webex.webapp.wapi.plugin.ComputeBoo lean"> <param name="value"> <senv> /org/ext/WBX/changeViewProfilePermi ssion_B </senv> </param> <param name="default"> <value>true</value> </param> </call> </ManageViewProfileSetting></pre>
PrecheckOrgCenterIntegration	<pre><PrecheckOrgCenterIntegration> <requires>WBX:ManageOrg</requires> </PrecheckOrgCenterIntegration></pre>
PremiumServices	<pre><PremiumServices> <requires>WBX:PremiumServices</requires> </PremiumServices></pre>
ProbeAccess	<pre><ProbeAccess> <true/> </ProbeAccess></pre>
ProvisionUser	<pre><ProvisionUser> <requires>WBX:ManageUsers</requires> </ProvisionUser></pre>
ProvisionUserComplete	<pre><ProvisionUserComplete> <requires>WBX:ManageUsers</requires> </ProvisionUserComplete></pre>

Table A-1 Default WebEx Platform Policies (continued)

Default Policy	XML Code
RedeemProxyCred	<pre><RedeemProxyCred> <true/> </RedeemProxyCred></pre>
remove-group	<pre><remove-group> <requires>WBX:ManageGroup</requires> </remove-group></pre>
remove-groupThing	<pre><remove-groupThing> <true/> </remove-groupThing></pre>
remove-org	<pre><remove-org> <requires>WBX:ManageOrg</requires> </remove-org></pre>
remove-policy	<pre><remove-policy> <requires>WBX:ManageRoles</requires> </remove-policy></pre>
remove-privilege	<pre><remove-privilege> <requires>WBX:ManageRoles</requires> </remove-privilege></pre>
remove-role	<pre><remove-role> <requires>WBX:ManageRoles</requires> </remove-role></pre>
remove-thing	<pre><remove-thing> <true/> </remove-thing></pre>
remove-user	<pre><remove-user> <or> <requires>WBX:ManageUsers</requires> <eq> <send>userID</send> <path>/user/userID</path> </eq> </or> </remove-user></pre>
remove-user-role	<pre><remove-user-role> <requires>WBX:ManageGroup</requires> </remove-user-role></pre>

Table A-1 Default WebEx Platform Policies (continued)

Default Policy	XML Code
remove-xobj	<pre><remove-xobj> <call class="com.webex.webapp.wbxconnect.plugin.XObj DefIsMgr"> <param name="userID"> <send>userID</send> </param> <param name="xobjdefID"> <path>xobj/xobjDefID</path> </param> <param name="cmd"> <send>cmd</send> </param> <param name="access"> <value>write</value> </param> </call> </remove-xobj></pre>
remove-xobjDef	<pre><remove-xobjDef> <call class="com.webex.webapp.wbxconnect.plugin.XObj DefIsMgr"> <param name="userID"> <send>userID</send> </param> <param name="xobjdefID"> <path>xobjDef/xobjDefID</path> </param> <param name="cmd"> <send>cmd</send> </param> <param name="access"> <value>mgr</value> </param> </call> </remove-xobjDef></pre>
RequestProxyCred	<pre><RequestProxyCred> <true/> </RequestProxyCred></pre>
RunScript	<pre><RunScript> <true/> </RunScript></pre> <p data-bbox="691 1520 1333 1612">Note RunScript is used for 2 - 15 commands only. For one command, call the command directly and not within the RunScript tag.</p>
set-countryList	<pre><set-countryList> <false/> </set-countryList></pre>
set-doc	<pre><set-doc> <requires>WBX:UpdateDocument</requires> </set-doc></pre>

Table A-1 Default WebEx Platform Policies (continued)

Default Policy	XML Code
set-globalConfig	<pre><set-globalConfig> <requires>WBX:SuperAdmin</requires> </set-globalConfig></pre>
set-group	<pre><set-group> <or> <requires>WBX:ManageGroup</requires> <eq> <send>action</send> <value>create</value> </eq> </or> </set-group></pre>
set-groupThing	<pre><set-groupThing> <true/> </set-groupThing></pre>
set-intgServType	<pre><set-intgServType> <false/> </set-intgServType></pre>
set-link	<pre><set-link> <true/> </set-link></pre>
set-objectPolicy	<pre><set-objectPolicy> <true/> </set-objectPolicy></pre>
set-offer	<pre><set-offer> <requires>WBX:ManageOffers</requires> </set-offer></pre>
set-org	<pre><set-org> <requires>WBX:ManageOrg</requires> </set-org></pre>
set-orgIntgServ	<pre><set-orgIntgServ> <requires>WBX:ManageOrg</requires> </set-orgIntgServ></pre>
set-policy	<pre><set-policy> <requires>WBX:ManageRoles</requires> </set-policy></pre>
set-privilege	<pre><set-privilege> <requires>WBX:ManageRoles</requires> </set-privilege></pre>
set-role	<pre><set-role> <requires>WBX:ManageRoles</requires> </set-role></pre>
set-thing	<pre><set-thing/> <true/> </set-thing></pre>

Table A-1 Default WebEx Platform Policies (continued)

Default Policy	XML Code
set-user	<pre> <set-user> <or> <requires>WBX:ManageUser</requires> <eq> <send>userID</send> <path>/user/userID</path> </eq> </or> </set-user> </pre>
set-userIntgServ	<pre> <set-userIntgServ> <or> <requires>WBX:ManageUsers</requires> <eq> <send>userID</send> <path>/userIntgServ/userID</path> </eq> </or> </set-userIntgServ> </pre>
set-xobj	<pre> <set-xobj> <call class="com.webex.webapp.wbxconnect.plugin.XObj DefIsMgr"> <param name="userID"> <send>userID</send> </param> <param name="xobjdefID"> <path>xobj/xobjDefID</path> </param> <param name="cmd"> <send>cmd</send> </param> <param name="access"> <value>write</value> </param> </call> </set-xobj> </pre>
set-xobjDef	<pre> <set-xobjDef> <call class="com.webex.webapp.wbxconnect.plugin.XObj DefIsMgr"> <param name="userID"> <send>userID</send> </param> <param name="xobjdefID"> <path>xobjDef/xobjDefID</path> </param> <param name="cmd"> <send>cmd</send> </param> <param name="access"> <value>mgr</value> </param> </call> </set-xobjDef> </pre>

Table A-1 Default WebEx Platform Policies (continued)

Default Policy	XML Code
SetPrivilegeTree	<SetPrivilegeTree> <requires>WBX:SuperAdmin</requires> </SetPrivilegeTree>
SetSiteConfigOptions	<SetSiteConfigOptions> <requires>WBX:ManageOrg</requires> </SetSiteConfigOptions>
setSiteRegRequiredFields	<setSiteRegRequiredFields> <requires>WBX:ManageOrg</requires> </setSiteRegRequiredFields>
ShowPresence	<ShowPresence> <requires>WBX:ShowPresence</requires> </ShowPresence>
SiteRegRequiredFields	<SiteRegRequiredFields> <requires>WBX:ManageOrg</requires> </SiteRegRequiredFields>
StoreAccess	<StoreAccess> <requires>WBX:StoreAccess</requires> </StoreAccess>
SyncConnectCenterLink Status	<SyncConnectCenterLinkStatus> <requires>WBX:ManageOrg</requires> </SyncConnectCenterLinkStatus>
SynchronizeDocIDs	<SynchronizeDocIDs> <true/> </SynchronizeDocIDs>
test_auto_upgrade_rules_task	<test_auto_upgrade_rules_task> <false/> </test_auto_upgrade_rules_task>
TimeConversion	<TimeConversion> <true/> </TimeConversion>
UpdateActiveCertificate	<UpdateActiveCertificate> <requires>WBX:ManageCertificate</requires> </UpdateActiveCertificate>
userCreateAudio	<userCreateAudio> <requires>WBX:UserCreateAudio</requires> </userCreateAudio>
userCreateEvent	<userCreateEvent> <requires>WBX:UserCreateEvent</requires> </userCreateEvent>
userCreateWeb	<userCreateWeb> <requires>WBX:UserCreateWeb</requires> </userCreateWeb>
userDeleteAudio	<userDeleteAudio> <requires>WBX:UserDeleteAudio</requires> </userDeleteAudio>

Table A-1 Default WebEx Platform Policies (continued)

Default Policy	XML Code
userDeleteEvent	<pre><userDeleteEvent> <requires>WBX:UserDeleteEvent</requires> </userDeleteEvent></pre>
userDeleteWeb	<pre><userDeleteWeb> <requires>WBX:UserDeleteWeb</requires> </userDeleteWeb></pre>
userReadAudio	<pre><userReadAudio> <requires>WBX:UserReadAudio</requires> </userReadAudio></pre>
userReadEvent	<pre><userReadEvent> <requires>WBX:UserReadEvent</requires> </userReadEvent></pre>
userReadWeb	<pre><userReadWeb> <requires>WBX:UserReadWeb</requires> </userReadWeb></pre>
userUpdateAudio	<pre><userUpdateAudio> <requires>WBX:UserUpdateAudio</requires> </userUpdateAudio></pre>
userUpdateEvent	<pre><userUpdateEvent> <requires>WBX:UserUpdateEvent</requires> </userUpdateEvent></pre>
userUpdateWeb	<pre><userUpdateWeb> <requires>WBX:UserUpdateWeb</requires> </userUpdateWeb></pre>
ValidateSession	<pre><ValidateSession> <true/> </ValidateSession></pre>
VideoExt	<pre><VideoExt> <requires>WBX:VideoExt</requires> </VideoExt></pre>
VideoInt	<pre><VideoInt> <requires>WBX:VideoInt</requires> </VideoInt></pre>
ViewProfile	<pre><ViewProfile> <call class="com.webex.webapp.wbxconnect.plugin.priv ilege.CheckUserProfilePermissions"> <param name="userID"> <auxpath>userID</auxpath> </param> <param name="orgID"> <auxpath>orgID</auxpath> </param> </call> </ViewProfile></pre>

Table A-1 Default WebEx Platform Policies (continued)

Default Policy	XML Code
VoIPExt	<pre><VoIPExt> <requires>WBX:VoIPExt</requires> </VoIPExt></pre>
VoIPInt	<pre><VoIPInt> <requires>WBX:VoIPInt</requires> </VoIPInt></pre>

Path Based Policies

Some policies are based on the path of an object. The following is a table of these policies:

Table A-2 Path Based Policies

Path	XML Code
/user/ext/WBX/randomBits	<pre><pathAction find="path"> <path>/user/ext/WBX/randomBits</path> <get> <eq> <send>userID</send> <path>/user/userID</path> </eq> </get> <set> <eq> <send>userID</send> <path>/user/userID</path> </eq> </set> </pathAction></pre>
/thing/ext/WBX/distribution	<pre><pathAction find="path"> <path>/thing/ext/WBX/distribution</path> <get> <requires>WBX:ManageUsers</requires> </get> </pathAction></pre>
/user/ext/WBX/profile	<pre><pathAction find="path"> <path>/user/ext/WBX/profile</path> <set> <or> <ne> <sendpath> /org/ext/WBXCR/organizationAttr ibutes/enterpriseDirectory_B </sendpath> <value>true</value> </ne> <requires>WBX:ManageUsers</requires> </or> </set> </pathAction></pre>

Table A-2 Path Based Policies (continued)

Path	XML Code
/user/ext/WBX/viewProfileSetting	<pre> <pathAction find="path"> <path>/user/ext/WBX/viewProfileSetting</pa th> <call class="com.webex.webapp.wapi.plugin.Comput eBoolean"> <param name="value"> <sendpath> /org/ext/WBX/changeViewProfileP ermission_B </sendpath> </param> <param name="default"> <value>true</value> </param> </call> </pathAction> </pre>
/user/groups	<pre> <pathAction find="path"> <path>/user/groups</path> <get> <or> <requires>WBX:ManageUsers</requires > <eq> <send>userID</send> <path>/user/userID</path> </eq> </or> </get> </pathAction> </pre>
/user/ext/WBX/buddies	<pre> <path>/user/ext/WBX/buddies</path> <get> <or> <requires>WBX:ManageUsers</requires > <eq> <send>userID</send> <path>/user/userID</path> </eq> </or> </get> </pathAction> </pre>



GLOSSARY

A

API Application Programming Interface. Details how computers can interact with each other.

C

Caching The act of storing information in memory so that future requests for the information can be performed more quickly.

callback functions In computer programming, a callback is executable code that is passed as an argument to other code. It allows a lower-level software layer to call a function defined in a higher-level layer.

CAS Central Authentication Server.

condElement An abbreviation used in the document to represent the term “conditional element”.

CCCPS An abbreviation for the Cisco Collaboration Cloud Platform Service.

CRM Customer Relationship Management.

CRUD An acronym that stands for Create, Retrieve, Update and Delete.

D

data nodes A node is an abstract basic unit used to build linked data structures, such as linked lists and trees, and computer-based representation of graphs. Nodes contain data and/or links to other nodes.

DMS Document Management System.

DOM Document Object Model (DOM) is a platform- and language-independent standard object model for representing HTML or XML and related formats.

E

element A node in a HTTP or XML format that holds a piece of information or data.

G

- group** A collection of users.
- GUID** Globally Unique Identifier.

H

- hashing** A way to store and retrieve information by transforming a literal into a specific key in order to facilitate manipulation of the information.
- HTTP** Hypertext Transfer Protocol. A communications protocol used to transfer hypertext documents over the World Wide Web.
- httpbind()** Using the HTTP Binding means, that the J2ME app connects to the gateway specified with “HTTPBind url” and routes the XMPP messages to the XMPP server (specified with “Host name”, “Port” and “SSL?”).

I

- inheritance** Inheritance is a way to form new classes (instances of which are called objects) using classes that have already been defined. The new classes, known as derived classes, take over (or inherit) attributes and behavior of the pre-existing classes, which are referred to as base classes (or ancestor classes). It is intended to help reuse of existing code with little or no modification.
- internationalization (i18n)** Internationalization most commonly refers to the addition of a framework for multiple language support, especially in software.
- ISO-639** A set of standards that uses codes to indicate languages.
- ISP** Internet Service Provider.

J

- J2ME** Java 2 Micro Edition. A version of Java used on devices with limited memory such as PDAs and cell phones.
- JSON** Javascript Object Notation is a method of representing data elements in a structure of name/value pairs.
- JSR-170** A set of standard APIs used to access content repositories in the Java 2 platform.

K

- key** The result of a hash transformation. This is used to look up information in a hash table.

L

- localization (l10n)** Software localization is a process of translating software user interfaces from one language to another and adapting it to suit a foreign culture. This process is labor-intensive and often requires significant efforts from development teams. There are tools that can simplify the localization process.
- lifecycle** The lifecycle of an object in object-oriented programming is the time between an object's creation (also known as instantiation or construction) until the object is no longer used, and is destructed or freed.

M

- metadata** A set of data that is used to describe a different set of data.
- microformat** Microformats are markup that allow expression of semantics in an HTML (or XHTML) web page. Programs can extract meaning from a standard web page that is marked up with microformats. Existing XHTML (and HTML) standards allow for semantics to be embedded and encoded within them, i.e. *class*, *rel*, *rev*.

N

- namespace** A namespace is a scope that uses the enclosing nature of the scope to group logically related identifiers under a single identifier.

O

- organization** A commercial, governmental, or non-profit entity which sponsors users of the Cisco Collaboration Cloud Platform.

P

- parameter** A variable that takes in information passed to it via a subroutine.
- presence** An indication of a user's availability, for example in an Jabber IM client.
- privilege** The ability to perform an action, such as create, modify or delete an object.
- publish** Publishers are loosely coupled to subscribers, and needn't even know of their existence. With the topic being the focus, publishers and subscribers are allowed to remain ignorant of system topology. Each can continue to operate normally regardless of the other. In the traditional tightly-coupled client-server paradigm, the client cannot post messages to the server while the server process is not running, nor can the server receive messages unless the client is running.

R

remote desktop control	Remote Desktop Control displays the screen of another computer (via Internet or network) on your own screen. The program allows you to use your mouse and keyboard to control the other computer remotely. It means that you can work on a remote computer, as if you were sitting in front of it.
role	A collection of privileges.

S

SAML	Security Assertion Markup Language. An XML standard used to transfer security information over networks.
scriptScope	A scope can: contain declarations or definitions of identifiers; contain statements and/or expressions which define an executable algorithm or part thereof; next or be nested.
SOAP	A protocol used to transfer XML messages over computer networks.
subscribe	<i>See</i> publish.

T

termElement	An abbreviation for the term “terminal element”.
--------------------	--

U

URL	Uniform Resource Locator.
user	A specific individual that is able to manipulate the data in some fashion.
UTC	Coordinated Universal Time. A high precision time standard.

W

WebDAV	World Wide Web Distributed Authoring and Versioning. It is the Internet Engineering Task Force (IETF) standard for collaborative authoring on the Web: a set of extensions to the Hypertext Transfer Protocol (HTTP) that facilitates collaborative editing and file management between users located remotely from each other on the Internet.
WSDL	Web Services Description Language.

X

XML Extensible Markup Language.



INDEX

A

- API Request Format [2-2](#)
 - REST API Requests [2-3](#)
 - SOAP API Requests [2-3](#)
- API Responses [2-3](#)
- API Response Stanzas
 - Results [2-4](#)
 - Security Context [2-3](#)
 - Status [2-4](#)

C

- CCCPS Pre-Configured Actions [2-21](#)
- CCCPS Pre-Configured Privileges [2-21](#)
- Client-Server Model [2-1](#)
- Commands [2-7](#)
 - add [2-13](#)
 - commit [2-18](#)
 - create [2-9](#)
 - delete [2-10](#)
 - enum [2-16](#)
 - execute [2-20](#)
 - get [2-11](#)
 - login [2-8](#)
 - logout [2-8](#)
 - remove [2-14](#)
 - rollback [2-19](#)
 - set [2-12](#)
 - shape [2-15](#)
 - validate [2-17](#)

D

- Document Management [2-2](#)

I

- Inheritance [5-4](#)

O

- Objects [4-1](#)
 - Extensions [4-2](#)
 - Recurring Elements [4-3](#)
 - Reference Elements [4-5](#)
 - Templates [4-4](#)
 - Types [4-1](#)
 - Validation [4-3](#)
- Object Types
 - Document [4-14](#)
 - Namespace [4-6](#)
 - Organization [4-9](#)
 - Policy [4-13](#)
 - Privilege [4-13](#)
 - Role [4-12](#)
 - Tags [4-14](#)
 - Thing [4-16](#)
 - User [4-7](#)

P

- Policy Definition [5-2](#)
- Policy Evaluation [5-1](#)
- Privilege Evaluation [5-3](#)

Privileges [5-3](#)

R

Results Stanza [2-4](#)

Roles [5-3](#)

S

Security Context Stanza [2-3](#)

Selection Method

 Field Selector [2-31](#)

 Where Parameter [2-24](#)

Selection Methods [2-24](#)

SOAP API Requests [2-3](#)

Status Stanza [2-4](#)