[MS-XCCOSIP]:
Extensible Chat Control Over Session Initiation Protocol (SIP)

Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation for protocols, file formats, languages, standards as well as overviews of the interaction among each of these technologies.
- **Copyrights.** This documentation is covered by Microsoft copyrights. Regardless of any other terms that are contained in the terms of use for the Microsoft website that hosts this documentation, you may make copies of it in order to develop implementations of the technologies described in the Open Specifications and may distribute portions of it in your implementations using these technologies or your documentation as necessary to properly document the implementation. You may also distribute in your implementation, with or without modification, any schema, IDL's, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications.
- **No Trade Secrets.** Microsoft does not claim any trade secret rights in this documentation.
- **Patents.** Microsoft has patents that may cover your implementations of the technologies described in the Open Specifications. Neither this notice nor Microsoft's delivery of the documentation grants any licenses under those or any other Microsoft patents. However, a given Open Specification may be covered by Microsoft Open Specification Promise or the Community Promise. If you would prefer a written license, or if the technologies described in the Open Specifications are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting iplg@microsoft.com.
- **Trademarks.** The names of companies and products contained in this documentation may be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights. For a list of Microsoft trademarks, visit www.microsoft.com/trademarks.
- **Fictitious Names.** The example companies, organizations, products, domain names, email addresses, logos, people, places, and events depicted in this documentation are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

Reservation of Rights. All other rights are reserved, and this notice does not grant any rights other than specifically described above, whether by implication, estoppel, or otherwise.

Tools. The Open Specifications do not require the use of Microsoft programming tools or programming environments in order for you to develop an implementation. If you have access to Microsoft programming tools and environments you are free to take advantage of them. Certain Open Specifications are intended for use in conjunction with publicly available standard specifications and network programming art, and assumes that the reader either is familiar with the aforementioned material or has immediate access to it.

Revision Summary

Date	Revision History	Revision Class	Comments
01/20/2012	0.1	New	Released new document.
04/11/2012	0.1	No change	No changes to the meaning, language, or formatting of the technical content.
07/16/2012	0.1	No change	No changes to the meaning, language, or formatting of the technical content.
10/08/2012	1.0	Major	Significantly changed the technical content.
02/11/2013	2.0	Major	Significantly changed the technical content.
07/30/2013	2.0	No change	No changes to the meaning, language, or formatting of the technical content.
11/18/2013	2.0	No change	No changes to the meaning, language, or formatting of the technical content.
02/10/2014	2.0	No change	No changes to the meaning, language, or formatting of the technical content.

Table of Contents

1 Introduction

The Extensible Chat Control Over Session Initiation Protocol provides messaging and control mechanism between users and the server in a persistent multiparty channel communication system.

Sections 1.8, 2, and 3 of this specification are normative and can contain the terms MAY, SHOULD, MUST, MUST NOT, and SHOULD NOT as defined in RFC 2119. Sections 1.5 and 1.9 are also normative but cannot contain those terms. All other sections and examples in this specification are informative.

1.1 Glossary

The following terms are defined in [MS-GLOS]:

access control entry (ACE)
access control list (ACL)
Active Directory
base64
GUID
Hypertext Transfer Protocol (HTTP)
XML
XML namespace

The following terms are defined in [MS-OFCGLOS]:

Boolean
endpoint
hash
node
organizational unit
Session Initiation Protocol (SIP)
Setting
Transport Layer Security (TLS)
Uniform Resource Identifier (URI)
Uniform Resource Locator (URL)
XML document
XML element
XML schema

The following terms are specific to this document:

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as described in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2 References

References to Microsoft Open Specifications documentation do not include a publishing year because links are to the latest version of the documents, which are updated frequently. References to other documents include a publishing year when one is available.

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information.

[ISO-8601] International Organization for Standardization, "Data Elements and Interchange Formats - Information Interchange - Representation of Dates and Times", ISO/IEC 8601:2004, December 2004, http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=40874&ICS1=1&ICS2=140&ICS3=30

Note There is a charge to download the specification.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, http://www.rfc-editor.org/rfc/rfc2119.txt

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and Schooler, E., "SIP: Session Initiation Protocol", RFC 3261, June 2002, http://www.ietf.org/rfc/rfc3261.txt

[XMLNS] Bray, T., Hollander, D., Layman, A., et al., Eds., "Namespaces in XML 1.0 (Third Edition)", W3C Recommendation, December 2009, http://www.w3.org/TR/2009/REC-xml-names-20091208/

[XMLSCHEMA1] Thompson, H.S., Beech, D., Maloney, M., and Mendelsohn, N., Eds., "XML Schema Part 1: Structures", W3C Recommendation, May 2001, http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/

[XMLSCHEMA2] Biron, P.V., and Malhotra, A., Eds., "XML Schema Part 2: Datatypes", W3C Recommendation, May 2001, http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/

1.2.2 Informative References

[MS-GLOS] Microsoft Corporation, "Windows Protocols Master Glossary".

[MS-OFCGLOS] Microsoft Corporation, "Microsoft Office Master Glossary".

[MS-PRES] Microsoft Corporation, "Presence Protocol".

[MS-SIP] Microsoft Corporation, "Session Initiation Protocol Extensions".

[MS-SIPRE] Microsoft Corporation, "Session Initiation Protocol (SIP) Routing Extensions".

[MS-SIPREGE] Microsoft Corporation, "Session Initiation Protocol (SIP) Registration Extensions".

[RFC6086] Holmberg, C., Burger, E., Kaplan, H.,, "Session Initiation Protocol (SIP) INFO Method and Package Framework", January 2011, http://tools.ietf.org/html/rfc6086

1.3 Overview

This document describes the Extensible Chat Control Over SIP (XCCOS) protocol. The primary scenario for XCCOS is to provide messaging and control mechanisms between users and the server in a persistent multiparty channel communication system, where the system implements access control, content persistency, and message distribution functions.

The following figure shows one sample implementation of such system and the relation between each component.

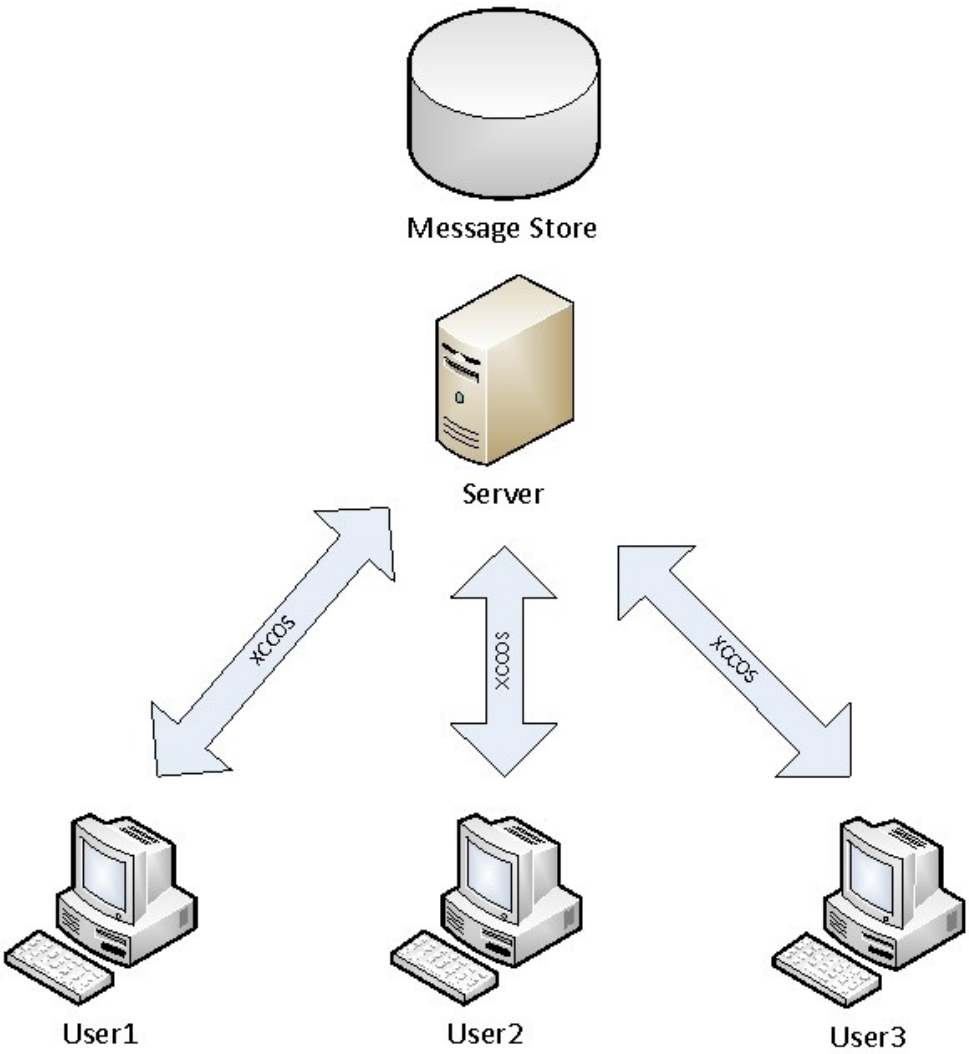

Figure 1: Sample implementation

XCCOS is inherently asynchronous, but stipulates that any client-initiated message is a request/response transaction. While channel-based messaging requires a client to accept messages it did not explicitly request, requiring a response to every client-initiated action allows XCCOS client authors to provide definitive feedback to users rather than attempting to infer the success or failure of a request based on the absence of errors.

This protocol does not provide access or channel management mechanism and assumes channels are provisioned and managed by a different protocol and interface.

1.4 Relationship to Other Protocols

XCCOS provides persistent channel communications capabilities by building on top of the SIP INFO as described in [RFC6086], which is itself based on **Session Initiation Protocol (SIP)** as described in [RFC3261].

XCCOS uses SIP INFO as a delivery mechanism for control messages between XCCOS clients and XCCOS servers. The use of SIP INFO follows the regular SIP session establishment semantics. Within the context of XCCOS, the SIP INFO request carries the XCCOS payload, and SIP INFO response carries the delivery status for the payload to the recipient. The XCCOS payload itself can be an XCCOS request or an XCCOS response. The SIP dialog is established using SIP INVITE.

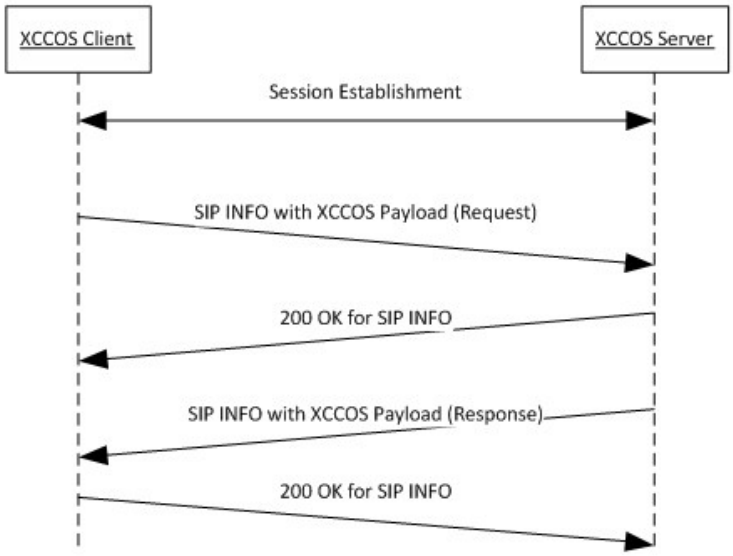

Figure 2: XCCOS uses XML to encode its payload

1.5 Prerequisites/Preconditions

This protocol assumes that both the clients and the server support SIP, and that they implement the extensions described in the following extension specifications as needed:

- Session Initiation Protocol Extensions ([MS-SIP]).
- Session Initiation Protocol Routing Extensions ([MS-SIPRE]).
- Session Initiation Protocol Registration Extensions ([MS-SIPREGE]).

1.6 Applicability Statement

This protocol is applicable when clients and the server support SIP and intend to use one or more features described in this protocol specification.

1.7 Versioning and Capability Negotiation

None.

1.8 Vendor-Extensible Fields

The XCCOS protocol uses **XML** to encode its payload. Extensions are allowed to the extent specified by the **XML schema**.

1.9 Standards Assignments

None.

2 Messages

2.1 Transport

This specification does not introduce a new transport to exchange messages. Messages are exchanged using SIP, as specified in [RFC3261]. SIP messages are transported over **Transport Layer Security (TLS)**.

2.2 Message Syntax

The following subsections define the message syntax for XCCOS messages, provisioning data, and roaming preferences.

2.2.1 Namespaces

This specification defines and references various **XML namespaces** using the mechanisms specified in [XMLNS]. Although this specification associates a specific XML namespace prefix for each XML namespace that is used, the choice of any particular XML namespace prefix is implementation-specific and not significant for interoperability.

The following table lists these namespaces, their prefixes, and the reference in which they are specified.

Prefix	Namespace URI	Reference
	urn:parlano:xml:ns:xccos	
xs	http://www.w3.org/2001/XMLSchema	[XMLSCHEMA1] [XMLSCHEMA2]

XCCOS **XML elements** are grouped under the "urn:parlano:xml:ns:xccos" namespace. There is currently no hierarchy defined beneath the root of this space, so all elements are defined at the top level.

2.2.2 XCCOS syntax

This section specifes in detail the syntax for XCCOS. The XCCOS protocol can be subdivided into two subsections: XCCOS data elements and XCCOS control elements. An XCCOS data element describes the state of the system, or used as a parameter to an XCCOS control element; whereas an XCCOS control element describes the action to be performed.

In this section, all elements and attributes are optional unless otherwise specified. Complete XML schema can be found in Appendix A: Full XML Schema (section 6).

2.2.2.1 XCCOS data elements

XCCOS data elements are used as parameters in XCCOS commands and represent the payload of XCCOS replies and notifications.

2.2.2.1.1 AuditDataBlock

The **AuditDataBlock** contains data about when a user/channel was created/updated and the display name of who performed the create/update. This data block is not used by the client. It has the following attributes:

Updatedby (string): Name of the user that performed the update operation.

Updatedon (string): UTC time when the update was performed. It is a string representation of the time format specified in [ISO-8601].

Createdby (string): Name of the user that created the user/category/channel.

Createdon (string): UTC time when the user/category/channel was created. It is a string representation of the time format specified in [ISO-8601].

Example

```
<audit updatedby="dummy"
    updatedon="2011-04-06T19:23:28.8842419Z"
    createdby="dummy"
    createdon="2011-04-06T19:23:28.8292419Z"
/>
```

2.2.2.1.2 InfoField

The **InfoField** element is a generic element used to describe a piece of information about its parent. It has the following attribute:

Id (string): This is the name of the information to be conveyed. This attribute is required.

In addition to the attributes, **InfoField** can have a text value.

Example

```
<info id="urn:parlano:ma:info:visibility">SCOPED</info>
```

2.2.2.1.3 PropertyField

The **PropertyField** element is a generic element used to describe a property about its parent. It has the following attribute:

Id (string): This is the name of the information to be conveyed.

In addition to the attribute, the **PropertyField** can have a boolean value.

Example

```
<prop id="urn:parlano:ma:prop:invite">True</prop>
```

2.2.2.1.4 UserInformationDataBlock

The **UserInformationDataBlock** elements are used to define user data. The use of this element depends on the context in which they are contained. For example, if it appears inside a **ChannelInformationDataBlock** (section 2.2.2.1.7), it defines the association between the user and the channel. The values of the attributes are retrieved from the **Active Directory**.

The element name is **uib**. It contains the following attributes:

uri (string): The SIP **URI** of the user. This attribute is required.

guid (string): A unique identifier in the form of a **GUID** string presentation. This attribute is required.

uname (string): The full name of the user; can be different from the SIP URI.

type (positive integer): User type that is always the value of 5. This attribute is required.

email (string): The email address of the user.

disabled (boolean): Specifies whether the user is disabled or not. This attribute is required.

dispname (string): The display name of the user.

company (string): The company the user belongs to.

chperms (integer): The permissions that the user has on the channel when **uib** is contained within a **ChannelInformationDataBlock**. This value is a bitmap of permissions. The bitmap is defined in the following table:

Bit Position	Permission
2	User can manage the channel
7	User can join the channel
8	User can chat on the channel
10	User can read the chat history of the channel
11	User can view the channel
12	User can chat in an auditorium channel

path (string): The distinguished name of the user.

id (integer): Optional index used in conjunction with the **ChannelInformationDataBlock** element (section 2.2.2.1.7) to convey the active participants in a room.

The **UserInformationDataBlock** also has the following optional elements:

audit: An **AuditDataBlock** (section 2.2.2.1.1) than defines the audit data for the user.

perms: A **UserPermissionDataBlock** that is hard-coded as shown in the following example.

aib: A structured representation of type **ActiveInformationDataBlock** (section 2.2.2.1.22) that represents the active channels and roles for a user. It is used in some notification messages.

from: A structured representation of type **From** (section 2.2.2.1.6) that is used when the **UserInformationDataBlock** refers to a role, and specifies where in the **node** hierarchy the role was defined. This is not useful information anymore because all the roles are now defined locally (in other words, they do not inherit).

Example

```
<uib uri="sip:user1@example.com"
    guid="93109AFC-D91D-45A1-96F4-6DCBBB31B640"
```

```
    type="5"
    uname="User1"
    disabled="false"
    dispname="User1">  <aib key="11652" value="93489432-b6be-4c67-932f-09e39a162072"
    domain="example.com" />  <perms inherited="1" inheriting="true" />
</uib>
```

2.2.2.1.5 GroupInformationDataBlock

The **GroupInformationDataBlock** elements are used to define group data (group has a broad meaning here because it could represent a domain, an Active Directory container or **organizational unit**, or Active Directory distribution and security groups). The use of this element depends on the context in which they are contained. For example, if it appears inside a **ChannelInformationDataBlock** (section 2.2.2.1.7), it defines the association between the group and the channel. The values of the attributes are retrieved from Active Directory.

The element name is **gib**. It contains the following attributes:

guid (string): A unique identifier in the form of a GUID string presentation. This attribute is required.

name (string): The full name of the group.

type (positive integer): Group type. This attribute is required. Possible values are:

- 8: Domain
- 9: Distribution or security group
- 10: Container or organizational unit

path (string): The distinguished name of the group.

The **GroupInformationDataBlock** also has the following optional elements:

audit: An **AuditDataBlock** (section 2.2.2.1.1) than defines the audit data for the group.

perms: A **UserPermissionDataBlock** that is not used anymore; content is empty, as shown in the following example.

from: A structured representation of type **from** (section 2.2.2.1.6) that is used when the **GroupInformationDataBlock** refers to a role, and specifies where in the node hierarchy the role was defined. This is not useful information anymore because all the roles are now defined locally (that is, they do not inherit).

Example

```
<gib guid="3025FA98-AFF7-49B7-AF8F-EA956244F173"
    type="9"
    name="The Team"
    path="CN=The Team,OU=Groups,DC=main,DC=example,DC=com">    <audit updatedby="systemuser"
        updatedon="2011-10-13T02:48:30.4188742Z"
        createdby="systemuser"
        createdon="2011-10-02T21:47:23.9953675Z" />
    <from name="Test Room">ma-chan://example.com/66b00dd5-6f18-4b6c-b51f-f2c7aada05cf</from>
<perms /></gib>
```

2.2.2.1.6 From

This element is present in data blocks that contain the description of where the object described in the data block is defined through the inheritance hierarchy. For scoped users and node creators, this element contains the category where the object is defined. For member and manager lists, the element contains the channel where the object is defined. For principal objects, the element contains the principal group with which the principal is affiliated.

The element name is **from** and it contains the following attribute:

Name (string): A node or principal name. This attribute is required.

The element value contains the node or principal **URL**.

2.2.2.1.7 ChannelInformationDataBlock

The **ChannelInformationDataBlock** element is a structured representation of information related to a channel. It does not specify a channel object in that it does not require all information about a channel to be present. It is a container which holds the relevant pieces of information required for any operation.

The element name is **chanib**.

2.2.2.1.7.1 Channel Attributes

Immutable attributes are server-assigned and controlled identifiers which the client can see, but can never change directly. This does not mean that the data does not change. The distinction is whether the client can change the value or not.

The immutable attributes pertaining to a channel are:

Uri (string): URI of the room.

Filerepository (string): Not used.

Core attributes are channel **Setting**s which are key identifiers for the channel (such as the list of immutable attributes), or affect channel permissions. Because a change to any of the core attributes can radically modify the **access control list (ACL)s** of a channel, they must be modified through specific commands rather than modifying them as channel meta-data changes.

The following attributes are considered core:

Parent (string): The URI of the parent category.

Behavior (string): Indicates the type of room and also affects the channel permissions including possible user lists, presenter behavior, and display Settings. It MUST have one of the following values:

Value	Meaning
UNSET	Value is not known
NORMAL	Regular room (all members can post and read chats)
AUDITORIUM	Auditorium room (presenters can post, all members can read)

Name (string): The channel name is a user reference to the channel for interacting with the channel before a user has the channel URI. The channel name MUST be unique for the entire domain in which it was created.

Disabled (boolean): Specifies whether the chat room is disabled or not. Core attributes MAY also be required during channel creation. A channel must always have a specified behavior at the time of channel creation, and unless the behavior is one which includes an automatic category path, the parent category must also be specified.

Some attributes are for informational purpose. The following attributes are considered information:

Description (string): Long textual description of the channel.

Keywords (string): Not used.

Topic (string): Not used.

Siopname (string): Name of the Standard Input Output Panel (SIOP) associated with the channel.

Siopurl (anyURI): URI of the SIOP.

Siopid (string): String representation of a GUID that uniquely identifies the SIOP.

OverrideMembers(boolean): Not used.

PartListOff (boolean): Specifies whether participant list updates are enabled or not.

2.2.2.1.7.2 Channel Elements

The ChannelInformationDataBlock (section 2.2.2.1.7) has the following children elements:

Aib: An **ActiveInformationDataBlock** (section 2.2.2.1.22) that represents the mapping between roles and users for this chat room.

Audit: An **AuditDataBlock** that represents the audit details of this particular chat room.

Info: An **InfoField** data element (section 2.2.2.1.2). Multiple instances of these elements describe information about the chat room.

Prop: A **PropertyField** data element (section 2.2.2.1.3)**.** Multiple instances of these elements describe the properties of the chat room.

Ace: Not used.

Uset: Not used.

Msg: A **GroupChatDataBlock** control element (section 2.2.2.2.9). Multiple instances of this element are returned when the chat history is retrieved from the chat room.

members: A **RoleList** element (section 2.2.2.1.14) that describes the member access control list (ACL) of the chat room.

managers: A **RoleList** element (section 2.2.2.1.14) that describes the manager **ACL** of the chat room.

presenters: A **RoleList** element (section 2.2.2.1.14) that describes the presenter ACL of an auditorium chat room.

2.2.2.1.7.2.1 Audit

This is an **AuditDataBlock** that represents the audit details of this particular channel.

2.2.2.1.7.2.2 Info

Multiple instances of the **Info** channel element describe information about a channel. The **info** element has a single attribute:

id (string): A string identifier of the channel "meta-data" type. The meta-data itself is represented by the **info** element value. The following ids are allowed:

urn:parlano:ma:info:path (string): The hierarchical path of the chat room, starting from the root of the tree. It can be used for display purposes.

urn:parlano:info:filestoreuri(string): A value that represents the URL of the web service to be used for uploading/downloading files.

urn:parlano:ma:info:visibility(string): A value that describes the channel visibility. The value MUST be one of a three-value enumeration of the string literals 'PRIVATE', 'SCOPED', and 'OPEN'. It defines who can see the chat room and chat room information during queries. It does not affect who can join the chat room.

urn:parlano:ma:info:manager(string): A value that describes a manager (represented as its SIP URI). This is used in some notifications (such as **usermodify**) to flag which user/participant is a manager in the context of a particular chat room.

urn:parlano:ma:info:ucnt(positive integer): A value that represents the number of current users joined to this chat room.

2.2.2.1.7.2.3 Prop

Multiple instances of the **Prop** channel element describe properties of the channel. The **prop** element has a single attribute:

id (string): A string identifier of the channel property. The property is represented by the **prop** element value.The following ids are allowed:

urn:parlano:ma:prop:logged (boolean): If true, the content of the channel will be logged for historical retrieval of channel participants. This does not mean that conversations are unlogged as all conversations must be logged for compliance.

urn:parlano:ma:prop:filepost (boolean): If true, any users of the channel will be allowed to post files to the chat room.

urn:parlano:ma:prop:invite (boolean): If true, users will receive invite notices when they register with the channel server.

2.2.2.1.7.2.4 msg

The **msg** element describes a chat message in the channel and is very similar to **GroupChatDataBlock** (section 2.2.2.2.9).It has the following attributes:

id (string): The value MUST be "grpchat". This attribute is required.

chanUri (string): The value is the URI of the chat room. This attribute is required.

author (string): The value is the SIP URI of the author. This attribute is required.

authdisp (string): The value is the display name of the author. This attribute is required.

alert (boolean): The value tells whether this message is a high priority message. This attribute is required.

chatId (long): This is the message identifier of this particular message. It is unique per channel. This attribute is required.

ts (string): This is the timestamp of the message as perceived by the server. It is a string representation of the time format as specified in [ISO-8601]. This attribute is required.

The **msg** element has two child elements:

chat (string): The value contains the plain text message content. This element is required.

rtf (string): The value is the Rich Text Format (RTF) representation of the chat element with formatting. This element is optional.

2.2.2.1.7.3 Examples

The following is an example of a **ChannelInformationDataBlock** (section 2.2.2.1.5) with embedded **msg** elements (section 2.2.2.1.7.2.4), which are usually obtained through chat history retrieval, content searches, etc.

```
<chanib uri="ma-chan://example.com/2a1a367c-5c14-4215-b5ae-d04eacb3b203"
    overridemembers="false"
    behavior="UNSET"
    keywords=""
    topic=""
    filerepository=""
    disabled="false">    <msg id="grpchat"
        chanUri="ma-chan://example.com/2a1a367c-5c14-4215-b5ae-d04eacb3b203"
        author="sip:user1@example.com"
        authdisp="User 1"
        alert="false"
        chatId="77"
        ts="2011-10-21T21:52:47.233Z">              <chat>Test</chat>   </msg>   <msg
id="grpchat"
        chanUri="ma-chan://example.com/2a1a367c-5c14-4215-b5ae-d04eacb3b203"
        author="sip:user2@example.com"
        authdisp="User 2"
        alert="false"
        chatId="78"
        ts="2011-10-22T00:21:06.993Z">              <chat>Test?</chat>

<rtf>{\urtf1\fbidis\ansi\ansicpg1252\deff0\nouicompat\deflang1033{\fonttbl{\f0\fnil\fcharset0
Segoe UI;}{\f1\fnil Segoe UI;}}{\colortbl ;\red51\green51\blue51;}{\*\generator Riched20
15.0.3419 (Debug)}{\*\mmathPr\mwrapIndent1440 }\viewkind4\uc1\pard\cf1\f0\fs18
Test?\f1\par}</rtf>   </msg>
</chanib>
```

Example of a **ChannelInformationDataBlock** obtained when joining a channel. It lacks the chat history, but metadata is richer.

```
<chanib name="GC Testing"
```

```
    description="A test room"
    parent="ma-cat://example.com/2642ebba-f56a-4891-9b92-3991eb865c92"
    uri="ma-chan://example.com/2a1a367c-5c14-4215-b5ae-d04eacb3b203"
    overridemembers="false"
    behavior="NORMAL"
    keywords=""
    topic=""
    filerepository=""
    disabled="false">    <aib key="3456" value="0,2,1,3,4,5,6,7" />    <aib key="11652"
value="1" />    <audit updatedby="User 1"
        updatedon="2011-10-05T22:10:39.9414558Z"
        createdby="User 2"
        createdon="2011-10-05T22:10:39.9154532Z" />    <info
id="urn:parlano:ma:info:filestoreuri">https://webserver.example.com/mgcwebservice/mgcwebservi
ce.asmx </info>
    <info id="urn:parlano:ma:info:ucnt">8</info>    <info
id="urn:parlano:ma:info:visibility">SCOPED</info>    <prop
id="urn:parlano:ma:prop:logged">True</prop>    <prop
id="urn:parlano:ma:prop:invite">True</prop>    <prop
id="urn:parlano:ma:prop:filepost">True</prop></chanib>
```

2.2.2.1.8 CategoryInformationDataBlock

The **CategoryInformationDataBlock** element is a structured representation of information related to a category. It does not specify a category object in that it does not require all information about a category to be present. It is a container which holds the relevant pieces of information required for any operation.

The element name is **catib**.

2.2.2.1.8.1 CategoryAttributes

Immutable attributes are server-assigned and controlled identifiers which the client can see, but can never change directly. This does not mean that the data does not change. The distinction is whether the client can change the value or not.

The immutable attributes pertaining to a category are:

uri (string): URI of the category

Core attributes are category Settings which are key identifiers for the category (such as the list of immutable attributes), or affect category permissions. Because a change to any of the core attributes can radically modify the scope of a category, they must be modified through specific commands rather than modifying them as category meta-data changes.

The following attributes are considered core:

parent (string): The URI of the parent category.

name (string): The category name is a user reference to the category for interacting with the category. The category name MUST be unique for the entire domain it was created in.

Some attributes are for informational purpose. The following attribute is considered informational:

description (string): Long textual description of the category

2.2.2.1.8.2 Category Elements

info: An **InfoField** data element (section 2.2.2.1.2).

2.2.2.1.8.2.1 info

There is only one **info** category element:

urn:parlano:ma:info:path: The hierarchical path of the category, starting from the root of the tree. It can be used for display purposes.

2.2.2.1.8.3 Examples

The following is an example of a **CategoryInformationDataBlock** (section 2.2.2.1.8)

```
<catib uri="ma-cat://example.com/2a1a367c-5c14-4215-b5ae-d04eacb3b203"
    name="Category One"
    parent="ma-cat://example.com/49b91e57-b2c9-4f7d-8eb0-0901c7c38f5d"
    description="Category description">   <info id="urn:parlano:ma:info:path">Category
One:Root</info>
</catib>
```

2.2.2.1.9 ChannelIdsInformationDataBlock

The **ChannelIdsInformationDataBlock** is used in the **join** or **bjoin** commands (specified in section 2.2.2.2.2) to specify which channel the client requests to join. It has the following attributes:

Key (string): Currently not used. The value SHOULD be 0. This attribute is required.

Value (string): The string representation of the GUID, which uniquely identifies the channel in a particular domain. This attribute is required.

Domain (string): The domain of the server. This attribute is required.

Example

```
<chanid key="0" value="944dc66c-f77f-435c-ae2c-b6b5a8ae7f33" domain="example.com" />
```

2.2.2.1.10 ServerInformationDataBlock

The **ServerInformationDataBlock** contains the information about the server to which the client is connected. It is used in the **getserverinfo** command to request server information, or in the **getserverinfo** reply to return the information. It has the following attributes:

domain (string) Domain to which the server belongs.

infoType (long): A bitmap indicating one of the following values, corresponding to the specified bits:

- None = 0,
- serverTime = 1,
- searchLimit = 2,

- pingInterval = 4,
- dbVersion = 8,
- rootUri = 16,
- messageSizeLimit = 32,
- storySizeLimit = 64,
- serverVersion = 128,
- retryDelay = 256,
- displayName = 512,
- roomManagementUrl = 1024

When a bit is set, it indicates that the corresponding information is requested (in the case of **getserverinfo** command) or is retrieved (in the case of **getserverinfo** reply). This attribute is required.

rootUri: The root node URI.

serverTime (string): Current time of the server in the format specified in [ISO-8601].

searchLimit (int): Maximum number of successful results retrieved for a search command.

pingInterval (string): Not currently used.

PoolId: The database pool ID.

RootCategoryUri: The URI of the root node.

messageSizeLimit (int): Maximum size for **grpchat** chat content the server would allow.

storySizeLimit (int): Maximum size for **grpchat** story the server would allow.

clientVersion (string): Client version string.

serverVersion (string): Server version string.

displayName (string): A human readable string of the server name.

roomManagementUrl (string): URL of a web application used to perform room management.

2.2.2.1.11 FilterInformationDataBlock

The **FilterInformationDataBlock** is used only as a parameter in the **cmd:chansrch** command as a search filter. The element name is **filtib.**

2.2.2.1.11.1 Filter Attributes

criteria (string): If specified, this attribute is used for a name search. Individual search terms are separated by a space character.

includeTopic (boolean): This attribute is optional and defaults to false. If set to true, the search includes the channel description in the search.

matchAll (boolean): This attribute is optional and defaults to true. If set to true, the channel name and optionally, the channel description (if **includeTopic** is set to true) MUST match all terms in the **criteria**. If set to false, any of the terms MUST match.

matchExactPhrase (boolean): This attribute is optional and defaults to true. If set to true, the entire **criteria** MUST match as an exact phrase.

catUri (Uri): If specified, the channel MUST also have the specified category as its **parent**.

addinGuid (string): If specified, the channel MUST also have the specified addin.

disabled (boolean): This attribute is optional. If set to true, the channel MUST be disabled. If missing or set to false, the channel MUST NOT be disabled.

vis (int): If specified, the channel visibility Setting MUST match this attribute. This attribute is an integer value that represents the visibility of a channel according to the following table:

Value	Privacy
2	Private
3	Scoped
6	OPEN

type (int): If specified, the channel behavior MUST match this attribute. The attribute is an integer value that represents the behavior of a channel according to the following table:

Value	Behavior
4	Normal
5	Auditorium

searchInvites (boolean): This attribute is optional and defaults to false. If set to true, the channel search will consider the value of the attribute **invites.**

invites (string): This attribute is optional and defaults to "inherit". If set to "inherit", the channel MUST inherit its invite Setting from its parent category. If set to false, the channel MUST explicitly disable invitations.

maxResults (int): If specified, this attribute defines the maximum number of channels to return in the search results.

2.2.2.1.11.2 Filter Elements

member: This element is either a single **UserInformationDataBlock** or **GroupInformationDataBlock**. If specified, the channel MUST have the specified user or group as a member.

manager: This element is either a single **UserInformationDataBlock** or **GroupInformationDataBlock**. If specified, the channel MUST have the specified user or group as a manager.

2.2.2.1.12 AceVerbEnum

This simple type is a string enumeration that is used when modifying the access control list (ACL) of channels in the **UpdateNode** message. The values are:

A: The associated principal is added to the specified ACL.

R: The associated principal is removed from the specified ACL.

X: The associated principal is a complete replacement of the specified ACL.

See section 2.2.2.1.14 for examples.

2.2.2.1.13 Ace

This element is used to describe individual **access control entries (ACEs)** in an access control list (ACL) of a channel.

2.2.2.1.13.1 Ace Attributes

The **Ace** element has a single required attribute:

verb (AceVerbEnum): Specifies the action to take with regard to the principal specified in the **uib** or **gib** element.

2.2.2.1.13.2 Ace Elements

uib (UserInformationDataBlock): This element describes a user principal.

gib (GroupInformationDataBlock): This element describes a group principal.

See section 2.2.2.1.14 for examples.

2.2.2.1.14 RoleList

This element is used to update the access control list (ACL) of a channel or for the server to return an ACL to the client.

2.2.2.1.14.1 RoleList Elements

prins (Ace): This element is used by a client to modify an access control list (ACL) of a channel. This element can occur multiple times within a **RoleList.**

uib (UserInformationDataBlock): This element is used by the server to return a user access control entry (ACE) to the client and can occur multiple times within a **RoleList**.

gib (GroupInformationDataBlock): This element is used by the server to return a group **ACE** to the client and can occur multiple times within a **RoleList**.

2.2.2.1.14.2 RoleList Examples

This is an example of a **RoleList** used by the client to add a user and a group to an access control list (ACL):

```
<members>
   <prins verb="A">
```

```
      <uib uri="sip:userone@example.com" type="5" disabled="false">
         <perms inherited="1" inheriting="true" />
      </uib>
   </prins>
   <prins verb="A">
      <gib type="3" path="DC=example,DC=com">
         <perms inherited="1" inheriting="true" />
      </gib>
   </prins>
</members>
```

The following is an example of the server returning an ACL to the client:

```
<members>
   <uib uri="sip:userone@example.com" guid="06C87F9C-56EA-4280-B5DF-9C4E835022BC" type="5"
uname="User One" disabled="false" dispname="User One" path="CN=User One,DC=example,DC=com">
      <audit updatedby="systemuser" updatedon="2012-05-29T22:41:12.0126719Z"
createdby="systemuser" createdon="2012-05-29T22:41:12.0126719Z" />
      <from name="User One">sip:userone@example.com</from>
      <perms inherited="1" inheriting="true" />
   </uib>
   <gib guid="792D66BA-3DE3-4D66-A3BC-89E501884237" type="8" name="example"
path="DC=example,DC=com">
      <audit updatedby="systemuser" updatedon="2012-05-29T22:41:12.0906804Z"
createdby="systemuser" createdon="2012-05-29T22:41:12.0906804Z" />
      <from name="vdomain">ma-grp:792D66BA-3DE3-4D66-A3BC-89E501884237@u.g</from>
      <perms inherited="1" inheriting="true" />
   </gib>
</members>
```

2.2.2.1.15 InviteDataBlock

The **InviteDataBlock** is used only as a parameter in the **getinv** command to retrieve channel invitations. The element name is **inv** and it has the following attributes:

inviteId (unsigned long): This is the sequence number that the client retrieves. If absent, the default value is zero.

register (boolean): If set to true, any channel returned from the **getinv** reply (specified in section 2.2.2.2.4) is considered acknowledged (registered). Only unregistered invitations will be returned in subsequent **getinv** commands. If absent, this attribute takes a default value of true.

domain (string): Domain of the server.

2.2.2.1.16 QueryInformationDataBlock

The **QueryInformationDataBlock** is used as a parameter to search related commands (such as **chansrch** and **getscoped**). The element name is **qib** and it has the following attributes:

qtype (string): This is the query type. It MUST take a value of BYNAME for name search.

Keywords (string): Space separated words used for keywords search (not used).

criteria (string): String used for name search

Recurse (boolean): Not used currently.

extended (boolean): If set to true, the search operation includes extended fields such as description for channel search.

MatchAll (boolean): If set to true, the results include entries that match all components of the criteria. If set to false, the results include entries that match at least one component of the criteria. This parameter applies when the **MatchExactPhrase** parameter is false.

MatchExactPhrase (boolean): If set to true, the matching is done on the criteria string interpreted as a whole. If set to false, the criteria string is tokenized, and the search is done on individual components. Results are returned for matches of all or any of the components, based on the **MatchAll** parameter.

Purpose (int): Not used currently.

Example

```
<qib qtype="BYNAME" criteria="A" extended="false" />
```

2.2.2.1.17 BcQueryDataBlock

The **BcQueryDataBlock** is used in retrieval of a contiguous block of messages in the chat history of a channel using the **bccontext** command (section 2.2.2.2.2) for a particular channel. It has a choice of the following two elements for specifying history range: **last** and **msgid**.

In addition to the choice of two elements, **BcQueryDataBlock** has one optional attribute:

get(boolean): This optional flag turns on or off message retrieval. Setting **get** to false means the response will only include a count of messages found that match the query. The messages themselves will not be retrieved. The default is true.

2.2.2.1.17.1 last

last is a simple element with a single attribute for the retrieval of a specified number of the most recent messages in the channel.

Cnt (unsigned int): This attribute specifies the number of the most recent messages to be retrieved. This attribute is required.

2.2.2.1.17.2 msgid

The **msgid** element retrieves a messages from a channel's history using the message identifier. It has the following attributes:

id (unsigned int): This attribute is the message identifier to start a chat history retrieval. This attribute is required.

cnt (unsigned int): This attribute is the number of the chat history to retrieve. This attribute is required.

pre (unsigned int): This attribute specifies the number of messages to return prior to the specified message identifier. If specified, the sum of **pre** and **post** MUST NOT be greater than **cnt**. This is currently not used.

post (unsigned int): This attribute specifies the number of messages to return after the specified message identifier. If specified, the sum of **pre** and **post** MUST NOT be greater than **cnt**.This is currently not used.

jump (boolean): This attribute is currently not used. This value MUST be set to false.

2.2.2.1.17.3 Example

```
<bcq>    <last cnt="100" /></bcq>
```

2.2.2.1.18 BcSearchDataBlock

The **BcSearchDataBlock** element is used to search the chat history with the **bcbydate** or **bcbymsg** command (section 2.2.2.2.2).

The element name is **bcs** and it contains one attribute:

Cmp (string): This attribute MUST be either "AND" or "OR" and specifies how the matching is done for the text criteria.

BcSearchDataBlock contains the following elements: **limit, text, msgId, matchcase, searchbkwds, sortbkwds, date, uib, cib**.

2.2.2.1.18.1 limit

limit is a simple element with a single attribute to specify the maximum number of results to be returned.

Cnt (unsigned int): This attribute specifies the maximum number of results to be returned. This attribute is required.

2.2.2.1.18.2 text

The **text** element specifies the search type in its attribute. Multiple instances of this element are allowed.

Mt (string): This attribute is required; MUST take the value of "PP" (Phrase-Partial), see section 2.2.2.1.18.10.

Phrase-Partial match. The phrase-partial match means the search text must be treated as a single unit, but can match only partial words, rather than matching the phrase exactly.

Examples would be:

- Searching for "may" would find "maybe", "may", and "mayflower."
- Searching for "ing can" would find "bringing canned goods."

2.2.2.1.18.3 msgId

If present, the value of the **msgId** element MUST be a normalized string specifying the message identifier for which the search is to be performed.

2.2.2.1.18.4 matchcase

The value of the **matchcase** element MUST be a boolean specifying whether the search is to be case sensitive.

2.2.2.1.18.5 searchbkwds

The value of the **searchbkwds** element MUST be a boolean specifying whether the search is to be performed from the range specified.

2.2.2.1.18.6 sortbkwds

The value of the **sortbkwds** element MUST be a **Boolean** specifying whether the search result is to be sorted in reverse chronological order, that is, most recent first.

2.2.2.1.18.7 date

If present, the **date** element specifies the date range from which the messages in chat history are to be searched.

From (string): The representation of the starting date range for chat history retrieval as specified in [ISO-8601]. This attribute is required.

To (string): The representation of the ending date range for chat history retrieval as specified in [ISO-8601]. This attribute is required.

2.2.2.1.18.8 uib

This is a **UserInformationDataBlock** element (section 2.2.2.1.4). Multiple instances are allowed in the **BcSearchDataBlock** (section 2.2.2.1.18). If present, only messages authored by the specified users are returned in the search result.

2.2.2.1.18.9 cib

This is a **ChannelInformationDataBlock** element. Multiple instances are allowed in the **BcSearchDataBlock** (section 2.2.2.1.18). If present, only messages in the specified channels are returned in the search result.

2.2.2.1.18.10 Example

```
<bcs cmp="OR">    <limit cnt="50" />    <text mt="PP">mayflower</text>
<matchcase>true</matchcase>    <searchbkwds>true</searchbkwds>    <sortbkwds>true</sortbkwds>
<date from="2011-10-20T07:00:00Z" to="2011-10-28T06:59:59.9Z" />    <cib uri="ma-
chan://example.com/66b00dd5-6f18-4b6c-b51f-f2c7aada05cf"
        overridemembers="false"
        behavior="UNSET"
        keywords=""
        filerepository=""
        disabled="false" /></bcs>
```

2.2.2.1.19 ResultCountDataBlock

The **ResultCountDataBlock** is used to retrieve a count of items in a result and whether there are more results available. The element name is **cnt**.

It contains the following attributes:

Value (Positive Integer): Count of items.

Over (boolean): True if there are more available items.

Example

```
<cnt value="1" over="false" />
```

2.2.2.1.20 AssociationDataBlock

The **AssociationDataBlock** is used in the **getassociations** command and the **getassociations** reply. It is used for retrievals of channels associated (either as a member of, or as a manager of) with the current user.

The element name is **association,** and it contains the following attributes:

Hash (unsigned long): Currently not used.

Domain (string): Domain of the server. (string)

Type (string): The value MUST be either MEMBER for retrieving channels the user is a member of, or MANAGER for retrieving channels the user is a manager of. This attribute is required.

maxResult (unsigned int): This returns the maximum number of results to be returned. If this attribute is absent, it takes a default value of 100.

When used in the **getassociations** reply, multiple instances of the **chanib** elements are returned inside the **AssociationDataBlock** (section 2.2.2.1.20). Each **chanib** element is a **ChannelInformationDataBlock** (section 2.2.2.1.7).

2.2.2.1.21 HashInformationDataBlock

The **HashInformationDataBlock** is used to build a generic representation of a hash table. It can be used alone or extended to add additional attributes.

The element name is **hash,** and it contains the following attributes:

Key (string): The key.

Value (string): The value.

2.2.2.1.22 ActiveInformationDataBlock

The **ActiveInformationDataBlock** represents an extension of a **HashInformationDataBlock** (section 2.2.2.1.21) that is used in relation to the active users of a chat room.

The element name is **aib,** and it contains the following attributes:

Key (string): The key (a number identifying the user's role).

Value (string): This is either a comma-separated set of user indexes in the **bjoin** and **join** replies, or a set of chat room **GUIDs** in the notifications..

Domain (string): The domain of the server.

2.2.2.1.23 FailureInformationDataBlock

The **FailureInformationDataBlock** represents an extension of a **HashInformationDataBlock** (section 2.2.2.1.21) that is used to convey the error information in a **bjoin** reply (section 2.2.2.2.4). The element name is **fib,** and it contains the following attributes:

key (string): Error description

value (string): A comma-separated set of chat room GUIDs.

domain (string): The domain of the server.

Example

```
<fib key="OPERATION_FAILED" value="2bdc091e-d8be-45ef-8c24-e28ee1b93a65,df042ddb-550a-4b1b-
bfd2-bfd8298ff892" domain="example.com" />
```

2.2.2.1.24 FileTokenDataBlock

The **FileTokenDataBlock** is used to send a request for a file transfer token to be used later for the actual file transfer. The element name is **ftdb**. It contains the following attributes:

channelUri (string): The chat room URI.

fileUrl (string): The URL to the server storage of the posted file.

Example

```
<ftdb channelUri="ma-chan://example.com/a3f66cba-e7f3-4549-ba96-e971cd65f756"
  fileUrl="ma-filelink://example.com/a3f66cba-e7f3-4549-ba96-e971cd65f756/4634bc9d-1cb5-4966-
96fa-41b67168c53f.js/somefile.txt/" />
```

2.2.2.1.25 TokenDataBlock

The **TokenDataBlock** is used to return a token to be used later in the actual file transfer. The element name is **token**. It contains the following attributes:

token (string): The token. This attribute is required.

serveruri (string): The URI of the web server that will honor the token.

Example

```
<token token="1391b791-d106-478b-8096-bca951f49b0f"
  serveruri="https://webserver.example.com/MGCWebService/MGCWebService.asmx" />
<ftdb channelUri="ma-chan://example.com/a3f66cba-e7f3-4549-ba96-e971cd65f756"
  fileUrl="ma-filelink://example.com/a3f66cba-e7f3-4549-ba96-e971cd65f756/4634bc9d-1cb5-4966-
96fa-41b67168c53f.js/somefile.txt/" />
```

2.2.2.1.26 PreferenceDataBlock

The **PreferenceDataBlock** is used to specify preference Settings. The element name is **pref**.

It contains the following attributes:

label (string): A label used to distinguish a particular Setting item in a set. This attribute is required.

seqid (positive integer): Sequence number that helps in the detection of version conflicts, for example,when a client tries to update a Setting that has already updated by a different client. This attribute is required.

createdefault (boolean): This attribute is required but is not used.

content (string): The content. It is a **base64** encoded zip archive of an **XML document** that enumerates user Settings.

Example

```
<pref label="kedzie.UserOptions"
 seqid="543"
 createdefault="false"
 content="H4sIAAAAAAAAEAO29B2AcSZYlJi9tynt/SvVlonglongstringH4XyMqyCAAA=" />
```

2.2.2.1.26.1 Content Format

Legacy user preferences are enumerated in an XML document as shown in the following example:

```
<parlanoxml ver="1">
  <object asm="CL" type="P.Domain.Channel.GroupChannelPreferencesListManager">
    <member name="_key_" type="S.String" value="GroupChannels" />
    <member name="items" asm="CL" type="PV.GroupChannelPreferencesVO" count="1">
      <item name="0" asm="CL" type="PV.GroupChannelPreferencesVO">
        <member name="_key_" type="S.String"
          value="ma-chan://example.com/b37fd924-8a1f-42e1-9fd9-868d6f811385" />
        <member name="name" type="S.String" value="TestRoom" />
        <member name="chatNotification" asm="CL" type="PV.ChatNotificationPVO">
          <member name="_key_" type="S.String" value="defaultGroupNotification" />
          <member name="floatOnActivate" type="S.Boolean" value="False" />
          <member name="alertFloatOnActivate" type="S.Boolean" value="False" />
          <member name="displayToast" type="S.Boolean" value="True" />
          <member name="alertDisplayToast" type="S.Boolean" value="True" />
          <member name="playSound" type="S.Boolean" value="False" />
          <member name="alertPlaySound" type="S.Boolean" value="True" />
        </member>
      </item>
    </member>
  </object>
</parlanoxml>
```

The root element is always **parlanoxml** and it has only one child element **object** whose attribute **type** MUST be "P.Domain.Channel.GroupChannelPreferencesListManager".The **object** element MUST have a child **member** element identifying this object as a collection of channel preferences – its **name** attribute is "_key_" and the **value** attribute is "GroupChannels", for example:

```
    <member name="_key_" type="S.String" value="GroupChannels" />
```

The **object** element MUST have a child **member** element that represents a collection of channel preferences. Its **name** attribute is "items" and its **count** attribute is equal to the number of child elements describing individual channel preferences, for example:

```
<member name="items" asm="CL" type="PV.GroupChannelPreferencesVO" count="4">
```

The items collection is represented by a number of **item** elements; the **name** attribute of such an item is equal to the item index in the items sequence, for example:

```
<item name="0" asm="CL" type="PV.GroupChannelPreferencesVO">
```

The **item** element MUST have a child **member** element whose **name** attribute is "_key_" and whose **value** attribute is the channel Uniform Resource Identifier (URI).

The **item** element MUST have a child **member** element whose **name** attribute is "name" and whose **value** attribute is the channel name.

The **item** element MUST have a child **member** element whose **name** attribute is "chatNotification" and whose **type** attribute is "PV.ChatNotificationPVO". This element serves as a container for various channel notification Settings, for example:

```
<member name="chatNotification" asm="CL" type="PV.ChatNotificationPVO">
```

This element MUST have a child **member** element whose **name** attribute is "_key_" and whose **value** attribute is "defaultGroupNotification". The item element MAY also have one or more child **member** elements describing individual notifications. The following Boolean notification Settings are used:

Boolean notification	Description
floatOnActivate	Open a separate channel window on new message arrival.
alertFloatOnActivate	Open a separate channel window on new high importance message arrival.
displayToast	Display a toast on new message arrival.
alertDisplayToast	Display a toast on new high importance message arrival.
playSound	Play sound on message arrival.
alertPlaySound	Play sound on high importance message arrival.

2.2.2.1.27 ResponseBlock

The **ResponseBlock** is a container for error response messages and codes. The element name is **resp**.

It contains the following attribute:

Code (Positive integer): An error code which must conform to a range similar to **Hypertext Transfer Protocol (HTTP)** with classes divided by the 100's. Possible values are between 100 and 699, and include informational, success, redirection, client error, server error, and transport error bands. This attribute is required.

2.2.2.1.28 XccosCommandDataBlock

The **XccosCommandDataBlock** is the container for all data elements necessary in commands. The contents are very free-form and validation is left up to the user based on the type of protocol message that is being transmitted and what is required for a valid message.

The element name is **data**.

The elements that can be contained are:

Type	Name	Count
ChannelInformationDataBlock	chanib	max 1
UserInformationDataBlock	uib	max 1
GroupInformationDataBlock	gib	max 1
BcQueryDataBlock	bcq	max 1
BcSearchDataBlock	bcs	max 1
QueryInformationDataBlock	qib	max 1
PreferenceDataBlock	pref	max 1
FileTokenDataBlock	ftdb	max 1
ChannelIdsInformationDataBlock	chanid	many
ServerInformationDataBlock	sib	max 1
InviteDataBlock	inv	max 1
AssociationDataBlock	association	max 1
FilterInformationDataBlock	filtib	max 1

2.2.2.1.29 XccosReplyNoticeDataBlock

The **XccosReplyNoticeDataBlock** is the container for all data elements necessary in replies and notices. The contents are very free-form and validation is left up to the user based on the type of protocol message that is being transmitted and what is required for a valid message. The **XccosReplyNoticeDataBlock** must allow multiple copies of the top-level data elements to satisfy commands that span many top-level things, such as chat history searches across multiple messaging targets.

The element name is **data**.

The elements that can be contained are:

Type	Name	Count
ChannelInformationDataBlock	chanib	many
CategoryInformationDataBlock	catib	many
UserInformationDataBlock	uib	many

Type	Name	Count
GroupInformationDataBlock	gib	many
FailureInformationDataBlock	fib	many
HashInformationDataBlock	hash	many
ResultCountDataBlock	cnt	max 1
PreferenceDataBlock	pref	max 1
TokenDataBlock	token	max 1
xs:nonNegativeInteger	status	max 1
ServerInformationDataBlock	sib	max 1
GroupChatDataBlock	grpchat	max 1
AssociationDataBlock	association	max 1
SiopWhitelistInformationDataBlock	siops	max 1
String	tag	max 1

2.2.2.2 XCCOS Control Elements

The **XCCOS** XML document is built from five "primitive" element definitions. These primitive elements are the high-level actions that the client and server can perform, those being:

- Commands (requests from client to server),
- Replies (responses to Commands from server to client),
- Notices (asynchronous updates to the system state sent from server to client),
- Errors, and
- System Notifications.

Command, Reply, and Notice primitives are intended to carry data between the endpoints. Therefore, each defines a data block, which is an element containing any number of informational elements required by the primitive messages. These data blocks are split into client-side (Command) and server-side (Reply and Notice) blocks because of the nature of the interactions.

The client must always be specific in its request so that the server’s response is useful and unambiguous to the client, while the server must be free to include all necessary data for the message.

2.2.2.2.1 XccosControlPrimitive

The **XccosControlPrimitive** element is the top level document element, which contains header information for tying independent messages together into a logical session. It carries commands, replies, notices, and errors between client and server through a message-oriented protocol (such as SIP/SIMPLE).

The element name is **xccos**, and it contains the following attributes:

ver (positive integer): Version of the protocol. Currently it is "1".

envid (positive integer): A monotonically increasing number identifying an **xccos** document. The embedded primitives are uniquely identified by the <Envid, SeqId of the primitive> tuple.

The following elements can be contained, in any number:

Type	Name	Description
XccosCommandPrimitive	cmd	Command (from client to server)
XccosReplyPrimitive	rpl	Reply (from server to client, usually in response to a command)
XccosNoticePrimitive	ntc	Notice (from server to client, unsolicited)
XccosErrorPrimitive	err	Error (from server to client, in response to a command)
XccosSystemPrimitive	sys	System notification from server, unsolicited
GroupChatDataBlock	grpchat	A chat message. For historical reasons this is not modeled by a primitive. This message flows in both directions (client to server and vice-versa).

Example

```
<xccos ver="1" envid="6698699123101735680" xmlns="urn:parlano:xml:ns:xccos">  <cmd
id="cmd:bjoin" seqid="1">
    <data>
      <chanid key="100"
  value="93489432-b6be-4c67-932f-09e39a162072,2a1a367c-5c14-4215-b5ae-d04eacb3b203"
             domain="example.com" />
    </data>
  </cmd>
  <cmd id="cmd:getpref" seqid="2">
    <data>
      <pref label="kedzie.ShowFeature" seqid="3" createdefault="true" />
    </data>
  </cmd>
</xccos>
```

2.2.2.2.2 XccosCommandPrimitive

The **XccosCommandPrimitive** element conveys a command from client to server with a request for an operation to be performed.

The name of the element is **cmd**, and it contains the following attributes:

id (string): The name of the command. This attribute is required. The following is a list of allowed values:

id	Purpose
cmd:bjoin	Batch chat room joining
cmd:bccontext	Get chat history
cmd:bcbydate	Search chat content by date

id	Purpose
cmd:bcbymsg	Search chat content by message
cmd:cateffquery	Get category effective details
cmd:chancreate	Creates a chat room
cmd:chaneffquery	Get chat room effective details
cmd:chansrch	Search for chat rooms
cmd:chanmodify	Modifies explicit chat room Settings
cmd:usrsrch	User search in Active Directory (by first, last name and email)
cmd:qeulmem	Get membership
cmd:getassociations	Get rooms I am a member of or I am a manager of
cmd:nodespermcreatechild	Get list of categories where the user is a creator
cmd:qeulmgr	Get managership
cmd:getpl	Get list of participants
cmd:getpref	Get user preferences
cmd:getscoped	Gets list of eligible principals (other than presenters) for roles
cmd:getscopedvoice	Gets list of eligible presenters for roles
cmd:getdetails	Get user/principal details
cmd:getuserchannels	Get channels a user is joined to
cmd:getfutok	Get file upload token
cmd:getfdtok	Get file download token
cmd:qeulvoiced	Get presenters
cmd:join	Join single chat room
cmd:chaninfo	Gets explicit channel details
cmd:sacemgr	Modify manager list
cmd:sacemem	Modify member list
cmd:sacevoiced	Modify presenter list
cmd:part	Leave chat room
cmd:requri	Get Channel Server Uniform Resource Identifier (URI)
cmd:setpref	Set user preferences
cmd:getserverinfo	Get global server information
cmd:getinv	Get invites

id	Purpose
cmd:updatenode	Update node
cmd:getroomperms	Get room permissions

Seqid (positive integer): A monotonically increasing number identifying a primitive in the context of the embedding **xccos** element (section 2.2.2.2.1).

XccosCommandPrimitive element can contain any amount of data within the data element of the type **XccosCommandDataBlock** (section 2.2.2.1.28), which is required to complete the command request.

Example

```
<cmd id="cmd:bjoin" seqid="1">
  <data>
    <chanid key="100"
value="93489432-b6be-4c67-932f-09e39a162072,2a1a367c-5c14-4215-b5ae-d04eacb3b203"
            domain="example.com" />
  </data>
</cmd>
```

2.2.2.2.3 XccosMessageIdentifier

The **XccosMessageIdentifier** element is used to uniquely identify a message. The element name is either **commandid** (when contained in **XccosReplyPrimitive** (section 2.2.2.2.4) and **XccosErrorPrimitive** (section 2.2.2.2.6)) or **originatingMessageId** (when used in a **GroupChatDataBlock** (section 2.2.2.2.9)).

It contains the following attributes:

Seqid (positive integer): The **sequence identification** of the message inside its embedding **xccos** document. This attribute is required.

Envid (positive integer): The **envelope identifier** of the embedding **xccos** document. This attribute is required.

2.2.2.2.4 XccosReplyPrimitive

The **XccosReplyPrimitive** element, or Reply primitive, is a server response to a client Command request. It must contain the sequence identifier and request identifier of the command it references, and it must contain a response statement.

If the reply must transmit data, a data block containing information is allowed, but is not required.

The name of the element is **rpl**, and it contains the following attributes:

id (string): The name of the reply. This attribute is required. The following is a list of allowed values.

id	Purpose
rpl:bjoin	Batch chat room joining

id	Purpose
rpl:bccontext	Get chat history
rpl:bc	Search chat content by date or message
rpl:cateffquery	Get category effective details
rpl:chancreate	Creates a chat room
rpl:chaneffquery	Get chat room effective details
rpl:chansrch	Search for chat rooms
rpl:chanmodify	Modifies explicit chat room Settings
rpl:usrsrch	User search in Active Directory (by first, last name and email)
rpl:qeulmem	Get membership
rpl:getassociations	Get rooms I am a member of or I am a manager of
rpl:nodespermcreatechild	Get list of categories where the user is a creator
rpl:qeulmgr	Get manager-ship
rpl:getpl	Get list of participants
rpl:getpref	Get user preferences
rpl:getscoped	Gets list of eligible principals (other than presenters) for roles
rpl:getscopedvoice	Gets list of eligible presenters for roles
rpl:getdetails	Get user/principal details
rpl:getuserchannels	Get channels a user is joined to
rpl:getftok	Get token for file upload or download
rpl:qeulvoiced	Get presenters
rpl:join	Join single chat room
rpl:chaninfo	Gets explicit channel details
rpl:sacemgr	Modify manager list
rpl:sacemem	Modify member list
rpl:sacevoiced	Modify presenter list
rpl:part	Leave chat room
rpl:setpref	Set user preferences
rpl:getserverinfo	Get global server information
rpl:getinv	Get invites
rpl:grpchat	Reply for a local chat post

id	Purpose
rpl:updatenode	Reply for update node
rpl:getroomperms	Reply for get room permissions

seqid (positive integer): A monotonically increasing number identifying a primitive in the context of the embedding **xccos** element (section 2.2.2.2.1).

The following are elements that MAY be contained within the **XccosReplyPrimitive** element:

commandid: This is an element of type **XccosMessageIdentifier** (section 2.2.2.2.3) identifying the command that is at the origin of the current reply.

resp: This is an element of type **ResponseBlock** (section 2.2.2.1.27) containing an error code

data: This is an element of type **XccosReplyNoticeDataBlock** (section 2.2.2.1.29) that contains the reply info.

Example

```
<rpl id="rpl:chaneffquery" seqid="1">
  <commandid seqid="1" envid="6698699123101735684" />
  <resp code="200">SUCCESS_OK</resp>
  <data>
    <chanib name="test"
            description=""
            parent="ma-cat://example.com/2642ebba-f56a-4891-9b92-3991eb865c92"
            uri="ma-chan://example.com/93489432-b6be-4c67-932f-09e39a162072"
            overridemembers="false"
            behavior="NORMAL"
            keywords=""
            topic=""
            filerepository=""
            disabled="false">
      <audit updatedby="Joe 1"
             updatedon="2011-10-24T21:11:22.3429958Z"
             createdby="Joe 2"
             createdon="2011-10-24T21:11:22.1489764Z" />
      <info id="urn:parlano:ma:info:filestoreuri">
          https://webserver.example.com/mgcwebservice/mgcwebservice.asmx
      </info>
      <info id="urn:parlano:ma:info:ucnt">2</info>
      <info id="urn:parlano:ma:info:visibility">SCOPED</info>
      <prop id="urn:parlano:ma:prop:logged">True</prop>
      <prop id="urn:parlano:ma:prop:invite">True</prop>
      <prop id="urn:parlano:ma:prop:filepost">True</prop>
    </chanib>
  </data>
</rpl>
```

2.2.2.2.5 XccosNoticePrimitive

The **XccosNoticePrimitive** element is an asynchronous update from a server to a client. These elements happen only when the server wants to transmit information to the client, so a data block MUST be present as well.

The name of the element is **ntc**, and it contains the following attributes:

id (string): name of the notice. This attribute is required. The following is a list of allowed values:

id	Purpose
ntc:bjoin	Notification that a user has joined one or more chat rooms.
ntc:chanmodify	Notification of info/prop attribute changes on the chat room.
ntc:usermodify	Alert for permissions changes of other chat room participants.
ntc:invite	Notification for a chat room invitation.
ntc:join	Notification that a user has joined a chat room.
ntc:kick	Notification that the user has been removed from the chat room.
ntc:part	Notification that some other user has left the chat room.
ntc:pl	Notification that participant updates have been turned on.
ntc:ploff	Notification that participant updates have been turned off.
ntc:quit	Notification that some other user disconnected from Group Chat. It is sent to any user that shares a chat room with the quitting one.
ntc:chatmodify	Notification that the chat was modified in a room.

seqid (positive integer): a monotonically increasing number identifying a primitive in the context of the embedding **xccos** element (section 2.2.2.2.1).

XccosNoticePrimitive MUST contain any amount of data within the data element of the type **XccosReplyNoticeDataBlock** (section 2.2.2.1.29), which is required to convey the notification information to the client.

Example

```
<ntc id="ntc:join" seqid="1">
  <data>
    <uib uri="sip:joe1@example.com"
         guid="E0F5F93A-1496-43E5-BCDB-011E6FA3189D"
         type="5"
         uname="Joe 1"
         disabled="false"
         dispname="Joe 1">
      <aib key="3456"
           value="2a1a367c-5c14-4215-b5ae-d04eacb3b203"
           domain="example.com" />
      <perms defined="1" inherited="1" inheriting="true" />
    </uib>  </data>
</ntc>
```

2.2.2.2.6 XccosErrorPrimitive

The **XccosErrorPrimitive** element, or Error primitive, is an acknowledgement from server to client when a more specific message is not possible. This includes channel messages that are rejected,

commands that are not understood, and control blocks that contain no data or bad data. In all cases, a more specific reply to command is desired when possible.

The Error MUST contain a sequence identifier that it refers to. If it is responding to a command block, it MAY also contain a request identifier. Errors that do not reference a specific message are not allowed as the client would not understand the purpose.

The name of the element is **err**, and it contains the following attributes:

Id (string): The name of the error message. This attribute is required and MUST be "error".

Seqid (positive integer): A monotonically increasing number identifying a primitive in the context of the embedding **xccos** element (section 2.2.2.2.1).

The following are elements that MAY be contained within the **XccosErrorPrimitive** element:

Commandid: This is an element of type **XccocsMessageIdentifier** (section 2.2.2.2.3) identifying the command that is at the origin of the current reply.

Resp: This is an element of type **ResponseBlock** (section 2.2.2.1.27) containing an error code.

2.2.2.2.7 XccosSystemStatusDataBlock

The **XccosSystemStatusDataBlock** is a simple element that conveys the busy/available states of the sender to the receiver.

The name of the element is **status**, and it contains the following attribute:

Busy (boolean): True, if the sender is busy and it cannot process messages; false otherwise.

2.2.2.2.8 XccosSystemPrimitive

XccosSystemPrimitive element is used for carrying system messages to the client. The name of the element is **sys,** and it contains the following attributes:

Id (string): The name of the system message. This attribute is required, and MUST be "sys:status".

Seqid (positive integer): A monotonically increasing number identifying a primitive in the context of the embedding **xccos** element (section 2.2.2.2.1).

The following element could be contained within the **XccosSystemPrimitive** element:

Status: This element is of type **XccosSystemStatusDataBlock** (section 2.2.2.2.7), and contains the free/busy status of the sender.

2.2.2.2.9 GroupChatDataBlock

The **GroupChatDataBlock** element is used to send and receive chats. It differs slightly from the other primitive elements by not following the command/reply/notice pattern.

The name of the element is **grpchat**, and it contains the following attributes:

id (string): The name of the message. This attribute is required, and MUST be "grpchat".

seqid (positive integer): A monotonically increasing number identifying a primitive in the context of the embedding **xccos** element (section 2.2.2.2.1).

chanUri (string): The chat room URI. This attribute is required.

author (string): The chat author's SIP URI. This attribute is required.

authdisp (string): The author's name of the chat. This attribute is required, but could be empty when sent by the client.

alert (boolean): True, if the chat is an alert. This attribute is required.

chatId (Long integer): The chat identifier uniquely identifying the chat in the chat room. This attribute is required, but it could be 0 when sent by client because the value is generated by the server.

ts (ISO8601 time string): The timestamp of the chat message as computed by the client (when sent by the client) or as generated by the server (when sent by the server), as specified in [ISO-8601]. This attribute is required.

The following elements can be contained within the **GroupChatDataBlock** element:

OriginatingMessageId: This is an element of type **XccosMessageIdentifier** (section 2.2.2.2.3) that identifies the original client to server **grpchat** message.

Chat: The value of this element is the plain text representation of the chat content.

Rtf (optional): The value of this optional element is the Rich Text Format (RTF) representation of the chat content.

Example

```
<grpchat id="grpchat"
         seqid="1"
         chanUri="ma-chan://example.com/93489432-b6be-4c67-932f-09e39a162072"
         author="sip:joe1@example.com"
         authdisp="Joe 1"
         alert="false"
         chatId="25"
         ts="2011-10-27T21:09:50.247Z">
  <originatingMessageId seqid="1" envid="6698699123101735682" />
  <chat>Hello World!</chat>
</grpchat>
```

3 Protocol Details

3.1 Client Details

The client details are broken down into specific functionality. Each subsection defines the functionality in detail. The client also maintains some state for each channel. The state is specified in the common channel state section. Each specific functionality also defines the abstract data model specific to that functionality as needed.

3.1.1 Common Channel State

The client SHOULD maintain the following state for each channel. Their data structure and use is as follows:

ParticipantList

- A list of strings where each entry is a SIP URI that represents a current participant of that channel.

LastReceivedMessageId

- Message identifier of the most recent chat message that the client received.

3.1.2 Sending XccosCommandPrimitives

With the exception of **GroupChatDataBlock**, all XccosControlPrimitives sent by the client are XccosCommandPrimitives. This section describes the behavior for sending XccosCommandPrimitives, as well as the specific behavior for specific commands.

3.1.2.1 XccosCommandPrimitive transaction handling

In XCCOS, commands are transactional in nature. If the client sends a specific command, a corresponding reply is expected to be returned. This section describes the behavior for matching the reply to its associated command.

3.1.2.1.1 Abstract Data Model

A data structure called **CmdId** is used to uniquely identify a command and its corresponding reply. It consists of 2 fields:

EnvId (64bit unsigned integer)

SeqId (64bit unsigned integer).

3.1.2.1.2 Timers

The client defines an **XccosTransactionTimer** to track the completion for each command. This timer is started when the command is sent. It has a value of 10 seconds.

3.1.2.1.3 Initialization

When the client sends a command, it MUST construct a **CmdId** and set the **EnvId** with the value of *envid* in the parent **XccosControlPrimitive**. It MUST also assign a value in the *seqid* for the **XccosCommandPrimitive**. The corresponding **CmdId** MUST take the same value in the **SeqId**.

When multiple XccosCommandPrimitives are present in the same **XccosControlPrimitive** (and therefore are using the same **EnvId** in the **CmdId**), the client MUST assign unique *seqids* between the different XccosCommandPrimitives within the same **XccosControlPrimitive** (and therefore must use unique **SeqId** across the different **CmdId**, even though the **EnvId** are the same). A sent command that has not received a reply, or a timeout, or other error condition for termination is considered pending.

3.1.2.1.4 Higher-Layer Triggered Events

The user can initiate cancelation of the XCCOS command. In such event, the **XccosTransactionTimer** is cancelled. The command is considered terminated and no corresponding reply for the command is processed.

3.1.2.1.5 Message Processing Events and Sequencing Rules

When the client receives an **XccosControlPrimitive**, the client first checks the **XccosControlPrimitive** for schema compliance. If it is not compliant with the schema, the **XccosControlPrimitive** is dropped. If a valid **XccosControlPrimitive** that contains XccosReplyPrimitives is received, for each **XccosReplyPrimitive**, the client MUST perform the following matching algorithm:

1. Inspect the *commandid* **XccosMessageIdentifier**. If none is present, the **XccosReplyPrimitive** is dropped.

2. Retrieve the value of *seqid* from the *commandid*. If none is present, the **XccosReplyPrimitive** is dropped.

3. Retrieve the value of *envid* from the *commandid*. If none is present, the **XccosReplyPrimitive** is dropped.

4. Find pending commands with **CmdId** having **EnvId** equal to the *envid* retrieved from the **XccosReplyPrimitive**, and **SeqId** equal to the *seqid* retrieved from the **XccosReplyPrimitive**. If there is no match, the **XccosReplyPrimitive** is dropped.

5. Retrieve the *code* attribute value from the **ResponseBlock** of the **XccosReplyPrimitive**. If none is present, the **XccosReplyPrimitive** is dropped.

6. Retrieve the value of the **id** attribute from the **XccosReplyPrimitive**. If the value is not a reply **id** corresponding to the original **commandid**, the **XccosReplyPrimitive** is dropped. That is, if the original **commandid** is "cmd:join", the client would drop a reply with **id** "rpl:bjoin", but not drop a reply with **id** "rpl:join".

If the **XccosReplyPrimitive** is not dropped, the **XccosReplyPrimitive** is deemed a reply to the original command, and the command is considered terminated. The corresponding **XccosTransactionTimer** will be cancelled. The value retrieved from the **ResponseBlock** in step 5, along with the **XccosReplyPrimitive** is passed on to the corresponding command handler for further processing.

The value of the code attribute retrieved from the **ResponseBlock** in step 5 is interpreted as follows: if the value equals 200, the client MUST treat the request as successful; for any other value, the client MUST treat the request as a failure.

3.1.2.1.6 Timer Events

If the **XccosTransactionTimer** is fired, the corresponding command is deemed to have terminated. This fact is passed to the corresponding command handler for further error handling and cleanup where deemed necessary.

3.1.2.1.7 Other Local Events

If the underlying SIP INFO carrying the **XccosCommandPrimitive** has failed, the command is considered terminated. This fact is passed to the corresponding command handler for further error handling and cleanup where deemed necessary.

3.1.3 Requesting Channel Server URI

After the client establishes a SIP dialog with the server, the client can send a command to request the channel server URI to handle further XCCOS requests.

3.1.3.1 Abstract Data Model

None.

3.1.3.2 Timers

None.

3.1.3.3 Initialization

The client MUST construct the **XccosCommandPrimitive** with an **id** value of *cmd:requri*. The client MUST send a **XccosCommandDataBlock** without any value.

3.1.3.4 Higher-Layer Triggered Event

None.

3.1.3.5 Message Processing Events and Sequencing Rules

If the client receives a failure reply (section 3.1.2.1.5), the client MUST NOT perform further processing on the **XccosReplyPrimitive**. If the client receives a success reply, the client MUST look for a **UserInformationDataBlock** inside the **XccosReplyNoticeDataBlock**. If the **UserInformationDataBlock** is present, the client MUST retrieve the value from the **uri** attribute inside the **UserInformationDataBlock**. The client MUST then terminate the existing dialog with the server by sending a SIP BYE request for both successful and failed cases.

If the **uri** attribute inside the **UserInformationDataBlock** is present, the client SHOULD establish a new dialog with this URI.

3.1.3.6 Timer Events

None.

3.1.3.7 Other Local Events

None.

3.1.4 Retrieving Server Information

The following section describes the logic for retrieving server information from the channel server URI retrieved (section 3.1.3).

3.1.4.1 Abstract Data Model

None.

3.1.4.2 Timers

None.

3.1.4.3 Initialization

The client MUST construct the **XccosCommandPrimitive** with an **id** value of *cmd:getserverinfo*. The client MUST send the command with exactly one **XccosCommandDataBlock**. Within the **XccosCommandDataBlock**, the client MUST set a **ServerInformationDataBlock**. Inside the **ServerInformationDataBlock**, the client MUST set the *domain* attribute with the domain of the server. The client MUST also set an integer value representing a bitmap for the *infoType* attribute. The bitmap values are defined in section 2.2.2.1.10. The client SHOULD use the following bitmap values: serverTime, searchLimit, pingInterval, PoolId, RootCategoryUri, messageSizeLimit, storySizeLimit, serverVersion, retryDelay, and displayName. The client MUST also set its own version string value in the *clientVersion* attribute.

3.1.4.4 Higher-Layer Triggered Events

None.

3.1.4.5 Message Processing Events And Sequencing Rules

If the client receives a failure reply as specified in section 3.1.2.1.5,, the client MUST tear down the SIP INVITE dialog. The client uses the values of the following attributes of the **ServerInformationDataBlock**:

displayName: The friendly display name of the server pool.

roomManagementUrl: The URL of a web application used to perform room management.

3.1.4.6 Timer Events

None.

3.1.4.7 Other Local Events

None.

3.1.5 Joining A Channel

The client can retrieve the channel URI from searches (section 3.1.9), invitations (section 3.1.10), or associated channel retrieval (section 3.1.11). The following sections specify how the client joins one particular channel.

3.1.5.1 Abstract Data Model

The client SHOULD use a map structure, **UserInfoMap**, where the key is an integer and the value is a string, for transient processing. The client also uses the **ParticipantList** defined in section 3.1.1.

3.1.5.2 Timers

None.

3.1.5.3 Initialization

The client MUST construct the **XccosCommandPrimitive** with an **id** value of *cmd:join*. The client MUST send the command with exactly one **XccosCommandDataBlock**. Within the **XccosCommandDataBlock**, the client MUST set exactly one **ChannelIdsInformationDataBlock**. The **value** attribute MUST be set to the GUID string of the channel URL. The **domain** attribute MUST be set to the server domain.

3.1.5.4 Higher-Layer Triggered Events

None.

3.1.5.5 Message Processing Events And Sequencing Rules

If the client receives a failure reply as described in section 3.1.2.1.5, the client MUST NOT perform further processing on the **XccosReplyPrimitive**. If the client receives a success reply, the client MUST look for a **ChannelInformationDataBlock** inside the **XccosReplyNoticeDataBlock**. For each **UserInformationDataBlock** inside the **ChannelInformationDataBlock**, the client MUST inspect the **id** attribute, and the **uri** attribute. If a **UserInfoMap** is defined, the client SHOULD add an entry into the map with the **id** attribute as the key and the **uri** attribute as the value. Once processing of all UserInformationDataBlocks is done, the client SHOULD inspect the **ActiveInformationDataBlock** within the **ChannelInformationDataBlock**. If an **ActiveInformationDataBlock** is present, the client takes the value of the **value** attribute. This value is a comma-separated string of the keys for the **UserInfoMap**. For each key in the comma-separated string, the client SHOULD retrieve the associated value from the **UserInfoMap**, and add an entry to the **ParticipantList**. After processing of the comma-separated key string, the resultant **ParticipantList** represents the current participants in the channel. The client SHOULD also retrieve the value of the **name** attribute to be used as the display name and the **description** attribute to be used as the description in the UI.

3.1.5.6 Timer Events

None.

3.1.5.7 Other Local Events

None.

3.1.6 Joining Multiple Channels

The client also supports joining multiple channels simultaneously. This section describes the process to join multiple channels.

3.1.6.1 Abstract Data Model

The client SHOULD maintain a **ParticipantList** for each channel it wants to join, and a **UserInfoMap** for transient processing. **ParticipantList** is defined in section 3.1.1, and UserInfoMap is defined in section 3.1.5.1.

3.1.6.2 Timers

None.

3.1.6.3 Initialization

The client MUST construct the **XccosCommandPrimitive** with an **id** value of *cmd:bjoin*. The client MUST send the command with exactly one **XccosCommandDataBlock**. Within the **XccosCommandDataBlock**, the client MUST set exactly one **ChannelIdsInformationDataBlock**. The **value** attribute MUST be set to a string of comma-separated GUID strings extracted from the channel URIs. The **domain** attribute MUST be set to the server domain.

3.1.6.4 Higher-Layer Triggered Events

None.

3.1.6.5 Message Processing Events And Sequencing Rules

If the client receives a failure reply as described in section 3.1.2.1.5, the client MUST NOT perform further processing on the **XccosReplyPrimitive**. If the client receives a success reply, the client MUST look for a **ChannelInformationDataBlock** inside the **XccosReplyNoticeDataBlock**. For each **UserInformationDataBlock** inside the **ChannelInformationDataBlock**, the client MUST inspect the **id** attribute, and the **uri** attribute. If a **UserInfoMap** is defined, the client SHOULD add an entry into the map with the **id** attribute as the key and the **uri** attribute as the value. For each **ChannelInformationDataBlock** present in the **XccosReplyNoticeDataBlock**, the client SHOULD inspect the **ActiveInformationDataBlock** within the **ChannelInformationDataBlock**. If an **ActiveInformationDataBlock** is present, the client takes the value of the **value** attribute. This value is a comma-separated string of the keys for the **UserInfoMap**. For each key in the comma-separated string, the client SHOULD retrieve the associated value from the **UserInfoMap**, and add an entry in the corresponding **ParticipantList**. After processing of the comma-separated key string, the resultant **ParticipantList** represents the current participants in this particular channel. For each **ChannelInformationBlock**, The client SHOULD also retrieve the value of the **name** attribute to be used as display name and the **description** attribute to be used as the description in the UI.

3.1.6.6 Timer Events

None.

3.1.6.7 Other Local Events

None.

3.1.7 Retrieving Most Recent Chat History From A Channel

After joining a channel, the client SHOULD request the most recent chat history from the channel.

3.1.7.1 Abstract Data Model

None.

3.1.7.2 Timers

None.

3.1.7.3 Initialization

The client MUST construct the **XccosCommandPrimitive** with an **id** value of *cmd:bccontext*. The client MUST send the command with exactly one **XccosCommandDataBlock**. Within the **XccosCommandDataBlock**, the client MUST set exactly one **ChannelInformationDataBlock**. The **uri** attribute in the **ChannelInformationDataBlock** MUST be set to the value of the channel URI. The client MUST also set a **BcQueryDataBlock** inside the **XccosCommandDataBlock**. Within the **BcQueryDataBlock**, the client MUST set the **last** element with the **cnt** attribute. This **cnt** attribute specifies the number of messages the client requests to retrieve. If the user wants to actively participate in the channel, that is if the user opens a conversation window in the UI, the client SHOULD set the **cnt** value to 25. If the user is not actively participating in the channel, the client SHOULD set the **cnt** value to 1.

3.1.7.4 Higher-Layer Triggered Events

Retrieving most recent chat history from a channel is triggered by either the user joining a channel, or if the user opens a UI element (such as the conversation window) to actively participate in the channel.

3.1.7.5 Message Processing And Sequencing Rules

If the client receives a failure reply as described in section 3.1.2.1.5, the client MUST NOT perform further processing on the **XccosReplyPrimitive**. If the client receives a success reply, the client MUST look for a **ChannelInformationDataBlock** inside the **XccosReplyNoticeDataBlock**. If the **uri** attribute in the **ChannelInformationDataBlock** does not match the URI from **ChannelInformationDataBlock** in the original command, the results are dropped. For each **GroupChatDataBlock** inside the **ChannelInformationDataBlock**, the client MAY inspect the **author**, **authdisp**, and **ts** attributes, and the chat content inside the **chat** element or the **rtf** element of the **GroupChatDataBlock**. If the user is actively participating in the channel, these message data is displayed to the user. The client SHOULD inspect the **chatId** attribute for each **GroupChatDataBlock**. If the received **chatId** is greater than the **LastReceivedMessageId** of channel, the **LastReceivedMessageId** is updated with the **chatId** of this **GroupChatDataBlock**.

3.1.7.6 Timer Events

None.

3.1.7.7 Other Local Events

None.

3.1.8 Searching Chat History

XCCOS provides the capability to search chat history. This section describes the client behavior for searching a specific channel. If the user searches multiple channels, the client MUST repeat the same operation for each channel.

3.1.8.1 Abstract Data Model

None.

3.1.8.2 Timers

None.

3.1.8.3 Initialization

The client MUST construct the **XccosCommandPrimitive** with an **id** value of *cmd:bcbydate*. The client MUST send the command with exactly one **XccosCommandDataBlock**. Within the **XccosCommandDataBlock**, the client MUST set exactly one **BcSearchDataBlock**. The client MUST set the **cmp** attribute with the value "OR" if the user wants to get results if any of the search criteria matched, or set the **cmp** attribute with the value "AND" if the user wants to get results only if all of the search criteria matched. Within the **BcSearchDataBlock**, the client MUST set the **limit** element with the **cnt** attribute specifying the maximum number of hits to be returned. For each word the user wants to search, the client MUST set a **text** element with the **mt** attribute equal to "PP". The client MUST NOT set the **msgid** element. If the user specifies authors the user wants to search for, for each author specified, the client MUST set a **UserInformationDataBlock** with only the **uri** attribute. The **uri** attribute of the **UserInformationDataBlock** MUST be set to the value of the SIP URI of the specified author. The client MUST set a **date** element in the **BcSearchDataBlock**. If a date range is specified by the user, the **from** and **to** attributes in the **date** element MUST be set to the representation of the start and end of the date range respectively, as specified in [ISO-8601]. If the user did not specify a date range, the client MUST set the **from** attribute with the value of "1899-12-30T00:00:00.000Z", and the **to** attribute with the value of the current time. The **matchcase**, **searchbkwds**, and **sortbkwds** elements MUST be set to false. If the user specifies a channel to search from, the client MUST set a **ChannelInformationDataBlock**. The **uri** attribute in the **ChannelInformationDataBlock** MUST be set to the value of the channel URI.

3.1.8.4 Higher-Layer Triggered Events

Chat history search is always initiated by user action. Search parameters are taken from user input.

3.1.8.5 Message Processing And Sequencing Events

If the client receives a failure reply as described in section 3.1.2.1.5, the client MUST NOT perform further processing on the **XccosReplyPrimitive**. If the client receives a success reply, the client MUST look for a **ChannelInformationDataBlock** inside the **XccosReplyNoticeDataBlock**. If the **uri** attribute in the **ChannelInformationDataBlock** does not match the URI from **ChannelInformationDataBlock** in the original command, the results are dropped. For each **GroupChatDataBlock** inside the **ChannelInformationDataBlock**, the client MAY inspect the **author**, **authdisp**, and **ts** attributes, and the chat content inside the **chat** element or the **rtf** element of the **GroupChatDataBlock**. This message data is then displayed to the user.

3.1.8.6 Timer Events

None.

3.1.8.7 Other Local Events

None.

3.1.9 Searching For Channels

Channel search is one of the ways the user can discover the channels to participate in. This section describes the client behavior for channel search.

3.1.9.1 Abstract Data Model

None.

3.1.9.2 Timers

None.

3.1.9.3 Initialization

The client MUST construct the **XccosCommandPrimitive** with an **id** value of *cmd:chansrch*. The client MUST send the command with exactly one **XccosCommandDataBlock**. Within the **XccosCommandDataBlock**, the client MUST set exactly one **QueryInformationDataBlock**. In the **QueryInformationDataBlock**, the client MUST set the **qtype** as "BYNAME". The client MUST set the **keywords** as empty. The client SHOULD set the value for the **criteria** attribute as the search input string from the user. If the user wants to search for channel name as well as for description, the **extended** attribute SHOULD be set to true; otherwise, if the user wants to search only for the channel name, the **extended** attribute SHOULD be set to false.

3.1.9.4 Higher-Layer Triggered Events

Channel search is always initiated by the user. Search parameters are taken from user input.

3.1.9.5 Message Processing And Sequencing Rules

If the client receives a failure reply as described in section 3.1.2.1.5, the client MUST NOT perform further processing on the **XccosReplyPrimitive**. If the client receives a success reply, for each **ChannelInformationDataBlock** inside the **XccosReplyPrimitive**, the client SHOULD inspect the **name**, **description**, and **uri** attributes. The results are then displayed to the user.

3.1.9.6 Timer Events

None.

3.1.9.7 Other Local Events

None.

3.1.10 Retrieving Invitations

Another way of discovering channels to participate in is retrieving channel invitations. Channel invitations are generated when the channel owner/manager, through an out-of-band mechanism, sets the user as a member of a channel where invitation is enabled. This section describes how a client retrieves the channel invitations. The client SHOULD retrieve the invitation on behalf of the user after it connects to the channel server.

3.1.10.1 Abstract Data Model

None.

3.1.10.2 Timers

None.

3.1.10.3 Initialization

The client MUST construct the **XccosCommandPrimitive** with an **id** value of *cmd:getinv*. The client MUST send the command with exactly one XccosCommandDataBlock. Within the **XccosCommandDataBlock**, the client MUST set exactly one **InviteDataBlock**. In the **InviteDataBlock**, the client MUST set the **inviteid** attribute to "0". The client MUST set the **domain** attribute as server domain.

3.1.10.4 Higher-Level Triggered Events

Retrieving invitations SHOULD be triggered after connection to the channel server succeeds.

3.1.10.5 Message Processing And Sequencing Rules

If the client receives a failure reply as described in section 3.1.2.1.5, the client MUST NOT perform further processing on the **XccosReplyPrimitive**. If the client receives a success reply, for each **ChannelInformationDataBlock** inside the **XccosReplyPrimitive**, the client SHOULD inspect the **name**, **description**, and **uri** attributes. Furthermore, the client SHOULD inspect the **HashInformationDataBlock** inside the **XccosReplyPrimitive**. The **key** attribute of that block will be set to the "inviteIds" string. The **value** attribute will be set to a comma-separated list of invitation id-timestamp pairs formatted as

<id>@<timestamp>,

where <id> is a numerical invitation identifier and <timestamp> is the Coordinated Universal Time, that is, UTC date/time when the invitation was issued by the server. For example:

<hash key="inviteIds" value="5@2012-04-23T23:12:39.29Z, 12@2012-04-24T23:55:45.793Z" />

The client SHOULD match invitation timestamps with corresponding channels using the fact that the invitation timestamps list is ordered exactly as the list of ChannelInformationDataBlocks. The results are then displayed to the user.

3.1.10.6 Timer Events

None.

3.1.10.7 Other Local Events

None.

3.1.11 Retrieving Associated Channels

Another way of discovering channels is to ask the server for any channels associated with the user. This section describes the client behavior for retrieving associated channels.

3.1.11.1 Abstract Data Model

None.

3.1.11.2 Timers

None.

3.1.11.3 Initialization

The client SHOULD construct two **XccosCommandPrimitives** for retrieving channels the user is a member of, and user is a manager of, respectively.

For retrieving each set of associated channels, the client MUST construct an **XccosCommandPrimitive** with an **id** value of *cmd:getassociations*. The client MUST send the command with exactly one **XccosCommandDataBlock**. Within the **XccosCommandDataBlock**, the client MUST set exactly one **AssociationDataBlock**. In the **AssociationDataBlock**, the client MUST set the **domain** attribute as server domain. The client MUST set the **type** attribute as "MEMBER" if it wants to retrieve channels the user is a member of, or as "MANAGER" if it wants to retrieve channels the user is a manager of. The client MUST set the **maxResult** attribute to request the number of results to be retrieved.

3.1.11.4 Higher-Layer Triggered Events

Retrieving associated channels SHOULD be triggered after connection to the channel server succeeds.

3.1.11.5 Message Processing And Sequencing Rules

If the client receives a failure reply as described in section 3.1.2.1.5, the client MUST NOT perform further processing on the **XccosReplyPrimitive**. If the client receives a success reply, for each **ChannelInformationDataBlock** inside the **XccosReplyPrimitive**, the client SHOULD inspect the **name**, **description**, and **uri** attributes. The results are then displayed to the user.

3.1.11.6 Timer Events

None.

3.1.11.7 Other Local Events

None.

3.1.12 Retrieving Channel Details

Xccos provides the functionality to retrieve additional details about a channel given a channel URI.

The client uses this functionality to determine which channel server the channel belongs to. When the client receives the channel URI by an out-of-band mechanism (such as email) it needs to find which channel server this channel belongs to. The client sends a command to all connected channel servers asking to retrieve additional details about a channel. The channel server returning a successful response would be the server that the channel belongs to.

This section describes the mechanism for retrieving channel details.

3.1.12.1 Abstract Data Model

None.

3.1.12.2 Timers

None.

3.1.12.3 Initialization

The client MUST construct a **XccosCommandPrimitive** with an **id** value of *cmd:chaneffquery*. The client MUST send the command with exactly one **XccosCommandDataBlock**. Within the **XccosCommandDataBlock**, the client MUST set exactly one **ChannelInformationDataBlock**. In the **ChannelInformationDataBlock**, the client MUST set the **uri** attribute with the channel URI.

3.1.12.4 Higher-Layer Triggered Events

None.

3.1.12.5 Message Sequencing And Processing Rules

If the client receives a failure reply as described in section 3.1.2.1.5, the client MUST NOT perform further processing on the **XccosReplyPrimitive**. If the client receives a success reply, it indicates the channel belongs to this particular channel server. The client does not perform any additional processing on the data returned from the response.

3.1.12.6 Timer Events

None.

3.1.12.7 Other Local Events

None.

3.1.13 Sending A Chat Message

The **GroupChatDataBlock** is used to deposit content at the server for a specific channel, and the client expects the **GroupChatDataBlock** to be propagated to all endpoints connected to the particular channel.

3.1.13.1 Abstract Data Model

A data structure called **MsgId** is used to uniquely identify a command and its corresponding reply. It consists of 2 fields:

EnvId (64bit unsigned integer);

SeqId (64bit unsigned integer).

3.1.13.2 Timers

The client defines a **GroupChatSendTimer** to track the completion of the **GroupChatDataBlock** it sends. The timer is started when the **GroupChatDataBlock** is sent. It has a value of 10 seconds.

3.1.13.3 Initialization

When the client sends a command, it MUST construct a **MsgId** and set the **EnvId** with the value of *envid* in the parent **XccosControlPrimitive**. It MUST also assign a value in the *seqid* for the **GroupChatDataBlock**. The corresponding **CmdId** MUST take the same value in the **SeqId**.

3.1.13.4 Higher-Layer Triggered Events

The user can initiate cancelation of the **GroupChatDataBlock**. In such event, the **GroupChatSendTimer** is cancelled.

3.1.13.5 Message Processing Events and Sequencing Rules

When the client sends a GroupChatDataBlock, the *chat* element MUST be present. The client MAY send an optional *rtf* element.

3.1.13.6 Timer Events

If the GroupChatSendTimer is fired, the corresponding GroupChatDataBlock is deemed to have failed. The client SHOULD display an error condition in the UI indication the failure occurred.

3.1.13.7 Other Local Events

Processing of received GroupChatDataBlock (section 3.1.14) can trigger a SentGroupChatReceived event. When this event is triggered, the GroupChatSendTimer is cancelled.

3.1.14 Receiving A Chat Message

An incoming chat message is represented by the GroupChatDataBlock.

3.1.14.1 Abstract Data Model

The client uses the same MsgId data structure when processing incoming GroupChatDataBlock.

3.1.14.2 Timers

None.

3.1.14.3 Initialization

None.

3.1.14.4 Higher-Layer Triggered Events

None.

3.1.14.5 Message Processing Events and Sequencing Rules

When the client receives an **XccosControlPrimitive** containing **GroupChatDataBlock**, for each **GroupChatDataBlock**, it MUST perform the following matching algorithm:

1. Inspect the **originatingMessageId** element of type **XccosMessageIdentifier**. If none is present, there is no match for any pending **GroupChatDataBlock** sent.
2. Retrieve the value of **seqid** from **originatingMessageId**. If none is present, there is no match for any pending **GroupChatDataBlock** sent.
3. Retrieve the value of **envid** from **originatingMessageId**. If none is present, there is no match for any pending **GroupChatDataBlock** sent.

4. Find pending **GroupChatDataBlock** with **MsgId** having **EnvId** equal to the **envid** retrieved from **originatingMessageId**, and **SeqId** equal to the **seqid** retrieved from **originatingMessageId**. If a match is found, raise a **SentGroupChatReceived** event on the matched **GroupChatDataBlock**.

Whether or not there is a match to a pending **GroupChatDataBlock**, the client MAY inspect the **author**, **authdisp**, and **ts** attributes of the **GroupChatDataBlock** and display them to the user. For each **GroupChatDataBlock**, the client SHOULD also inspect the **chatId** attribute. If the received **chatId** is greater than the **LastReceivedMessageId** of the channel, the **LastReceivedMessageId** is updated with the **chatId** of this **GroupChatDataBlock**.

3.1.14.6 Timer Events

None.

3.1.14.7 Other Local Events

None.

3.1.15 Receiving XccosNoticePrimitives

XccosNoticePrimitive is used by the server to notify the client of changes in the system, or to a specific channel. This section describes how the client handles receiving of XccosNoticePrimitives.

3.1.15.1 Abstract Data Model

None.

3.1.15.2 Timers

None.

3.1.15.3 Initialization

None.

3.1.15.4 Higher-Layer Triggered Events

None.

3.1.15.5 Message Processing And Sequencing Rules

When the client receives an **XccosControlPrimitive** with XccosNoticePrimitives, for each **XccosNoticePrimitive**, the client MUST perform the following operations:

The client SHOULD inspect the **id** attribute of the **XccosNoticePrimitive**. The client supports the following **id** values:

- *ntc:invite*
  - This is used by the server to notify the client of new invitations.
- *ntc:join*
  - This is used by the server to notify the client of a new participant in a particular channel.

- *ntc:bjoin*
  - Similar to *ntc:join*, this is used by the server to notify the client of a new participant to multiple channels.
- *ntc:part*
  - This is used by the server to notify the client a participant left a channel.
- *ntc:kick*
  - This is used by the server to notify the client a participant is forcibly removed from a channel.
- *ntc:quit*
  - This is used by the server to notify the client a participant has left all channels.
- *ntc:chatmodify*
  - This is used by the server to notify the client that the chat history of a channel has been modified.
- *ntc:chanmodify*
  - This is used by the server to notify the client that channel properties have been modified.

If the **id** value from the **XccosNoticePrimitive** is not any of the supported values, the **XccosNoticePrimitive** is dropped.

If the **id** value is *ntc:invite*, the client SHOULD inspect all ChannelInformationDataBlocks in the **XccosReplyNoticeDataBlock**. For each **ChannelInformationDataBlock**, the client SHOULD inspect the **name**, **description**, and **uri** attributes. The results are then displayed to the user.

If the **id** value is *ntc:join*, the client SHOULD inspect all UserInformationDataBlocks in the **XccosReplyNoticeDataBlock**. For each **UserInformationDataBlock**, the client SHOULD inspect the **uri** of the **UserInformationDataBlock**. If the **uri** is not present, this particular **UserInformationDataBlock** is dropped. The client SHOULD also inspect the **value** attribute inside the **ActiveInformationDataBlock**. If the **value** attribute is not present, this particular **UserInformationDataBlock** is dropped. Once the **value** is retrieved, the client SHOULD match the value with the GUID string portion of the channel URI for any channel the client is currently joined to. If no match is found, this particular **UserInformationDataBlock** is dropped. If a match is found, the client SHOULD add the URI retrieved from the **UserInformationDataBlock** into the channel's **ParticipantList** (defined in section 3.1.1) and consider the user presented by the URI as an active participant in the channel.

If the **id** value is *ntc:bjoin*, the handling is very similar to *ntc:join*. The client SHOULD inspect all UserInformationDataBlocks in the **XccosReplyNoticeDataBlock**. For each **UserInformationDataBlock**, the client SHOULD inspect the **uri** of the **UserInformationDataBlock**. If the **uri** is not present, this particular **UserInformationDataBlock** is dropped. The client SHOULD also inspect the **value** attribute inside the **ActiveInformationDataBlock**. If the **value** attribute is not present, this particular **UserInformationDataBlock** is dropped. The **value** is a comma-separated list of GUID strings. For each GUID string from the **value** attribute, the client SHOULD match the value with the GUID string portion of the channel URI for any channel the client is currently joined to. If no match is found, this particular **UserInformationDataBlock** is dropped. If a match is found, the client SHOULD add the URI retrieved from the **UserInformationDataBlock** into the channel's **ParticipantList** (defined in section 3.1.1) and consider the user presented by the URI as an active participant in the channel.

If the **id** value is *ntc:part*, the client SHOULD inspect all ChannelInformationDataBlocks in the **XccosReplyNoticeDataBlock**. For each **ChannelInformationDataBlock**, the client SHOULD inspect the **uri** of the **ChannelInformationDataBlock**. If the **uri** is not present, this particular **ChannelInformationDataBlock** is dropped. The client SHOULD look for a channel with a matching channel URI with any channel the client is currently joined to. If there is no match, this particular **ChannelInformationDataBlock** is dropped. The client SHOULD also inspect all UserInformationDataBlocks inside the **ChannelInformationDataBlock**. For each **UserInformationDataBlock**, the client SHOULD inspect the **uri** attribute. If the **uri** attribute is not present, this particular **UserInformationDataBlock** is dropped. Once the **uri** value is retrieved, the client SHOULD remove the URI from the channel's **ParticipantList** (defined in section 3.1.1) and consider that the user presented by the URI has left the channel.

If the **id** value is *ntc:kick*, the client SHOULD inspect all ChannelInformationDataBlocks in the **XccosReplyNoticeDataBlock**. For each **ChannelInformationDataBlock**, the client SHOULD inspect the **uri** of the **ChannelInformationDataBlock**. If the **uri** is not present, this particular **ChannelInformationDataBlock** is dropped. The client SHOULD look for a channel with a matching channel URI with any channel the client is currently joined to. If there is no match, this particular **ChannelInformationDataBlock** is dropped. The client SHOULD also inspect all UserInformationDataBlocks inside the **ChannelInformationDataBlock**. For each **UserInformationDataBlock**, the client SHOULD inspect the **uri** attribute. If the **uri** attribute is not present, this particular **UserInformationDataBlock** is dropped. Once a **uri** value is retrieved, the client SHOULD remove the URI from the channel's **ParticipantList** (defined in section 3.1.1) and consider that the user presented by the URI has left the channel. If the URI is the URI of the current user, the client is considered to have been forcibly removed from the channel. The client SHOULD clear the **ParticipantList** of this particular channel. If the user wishes to participate in this channel, the client MUST join the channel again using the logic described in section 3.1.5.

If the **id** value is *ntc:quit*, the client SHOULD inspect all ChannelInformationDataBlocks in the **XccosReplyNoticeDataBlock**. If there is no **ChannelInformationDataBlock** retrieved, the client MUST remove the **uri** retrieved from **UserInformationDataBlock** in the **XccosReplyNoticeDataBlock** from all the **ParticipantLists** the client is currently joined to. For each **ChannelInformationDataBlock**, the client SHOULD inspect the **uri** of the **ChannelInformationDataBlock**. If the **uri** is not present, this particular **ChannelInformationDataBlock** is dropped. The client SHOULD look for a channel with a matching channel URI with any channel the client is currently joined to. If there is no match, this particular **ChannelInformationDataBlock** is dropped. For each channel the **ChannelInformationDataBlock** specified, the client must remove the **uri** retrieved from **UserInformationDataBlock** in the **XccosReplyNoticeDataBlock** from its **ParticipantLists**.

If the **id** value is *ntc:chatmodify*, the client SHOULD inspect all ChannelInformationDataBlocks in the **XccosReplyNoticeDataBlock**. For each **ChannelInformationDataBlock**, the client SHOULD inspect the **uri** of the **ChannelInformationDataBlock**. If the **uri** is not present, this particular **ChannelInformationDataBlock** is dropped. The client SHOULD look for a channel with a matching channel URI with any channel the client is currently joined to. If there is no match, this particular **ChannelInformationDataBlock** is dropped. The client SHOULD request the most recent chat history for the channel (section 3.1.7).

If the **id** value is *ntc:chanmodify*, the client SHOULD inspect all ChannelInformationDataBlocks in the **XccosReplyNoticeDataBlock**. For each **ChannelInformationDataBlock**, the client SHOULD inspect the **uri** attribute of the **ChannelInformationDataBlock**. If the **uri** is not present, this particular **ChannelInformationDataBlock** is dropped. The client SHOULD look for a channel with a matching channel URI with any channel the client is currently joined to. If there is no match, this particular **ChannelInformationDataBlock** is dropped. The client SHOULD retrieve channel properties from the **ChannelInformationDataBlock** and update those changed channel properties that are used by the client

3.1.15.6 Other Local Events

None.

3.1.15.7 Timer Events

None.

3.1.16 Retrieving Channel Permissions

XCCOS provides the functionality to retrieve the permissions that a user has on a particular channel given the channel URI.

This section describes the mechanism for retrieving channel details.

3.1.16.1 Abstract Data Model

None.

3.1.16.2 Timers

None.

3.1.16.3 Initialization

The client MUST construct a **XccosCommandPrimitive** with an **id** value of cmd:getroomperms. The client MUST send the command with exactly one **XccosCommandDataBlock**. Within the **XccCommandDataBlock**, the client MUST set exactly one **ChannelInformationDataBlock**. In the **ChannelInformationDataBlock**, the client MUST set the **uri** attribute with the channel URI.

3.1.16.4 Higher-Layer Triggered Events

None.

3.1.16.5 Message Processing Events and Sequencing Rules

If the client receives a failure reply as described in section 3.1.2.1.5, the client MUST NOT perform further processing on the **XccosReplyPrimitive**. If the client receives a success reply, the client MUST look for a **ChannelInformationDataBlock**. If the **ChannelInformationDataBlock** is present, the client MUST look for a **UserInformationBlock** within the **ChannelInformationDataBlock**. If the **UserInformationBlock** is present, the client must inspect the **chperms** attribute of the **UserInformationDataBlock** to determine the user's permissions on the channel.

3.1.16.6 Timer Events

None.

3.1.16.7 Other Local Events

None.

3.1.17 Modifying a Channel

XCCOS provides the functionality to modify the attributes and settings of a channel.

3.1.17.1 Abstract Data Model

None.

3.1.17.2 Timers

None.

3.1.17.3 Initialization

The client MUST construct a **XccosCommandPrimitive** with an **id** value of *cmd:updatenode*. The client MUST send the command with exactly one **XccosCommandDataBlock**. Within the **XccosCommandDataBlock**, the client MUST set exactly one **ChannelInformationDataBlock**. In the **ChannelInformationDataBlock**, the client MUST set the **uri** attribute, the **name** attribute, the **parent** attribute, the **description** attribute, the **behavior** attribute, the **siopid** attribute, and the **disabled** attribute. Only these attributes are supported. The client MUST also set one **AuditDataBlock** within the **ChannelInformationDataBlock**. The **AuditDataBlock** MUST contain the current **AuditDataBlock** from the channel details in section 2.2.2.1.1. The client MUST also set the **urn:parlano:ma:info:visibility** Info element and the **urn:parlano:ma:prop:invite** element. The client MAY optionally define the **member**, **manager**, or **presenter RoleList** elements to modify the corresponding access control list (ACL)s.

3.1.17.4 Higher-Layer Triggered Events

None.

3.1.17.5 Message Processing Events and Sequencing Rules

If the client receives a failure reply as described section 3.1.2.1.5, the client MUST NOT perform further processing on the **XccosReplyPrimitive**. If the client receives a success reply, the client MUST look for a **ChannelInformationDataBlock**. If the **ChannelInformationDataBlock** is present, all attributes, elements, and **RoleList** elements for the modified channel are present.

3.1.17.6 Timer Events

None.

3.1.17.7 Other Local Events

None.

3.1.18 Retrieving Legacy User Preferences

Legacy implementations of XCCOS-based persistent messaging systems stored user preferences on an XCCOS server. Messaging systems built on the version of the XCCOS protocol documented in this specification store user preferences using a different mechanism as described in ([MS-PRES]). This section describes the client functionality that enables migration of user preferences from legacy systems to the current architecture.

3.1.18.1 Abstract Data Model

None.

3.1.18.2 Timers

None.

3.1.18.3 Initialization

The client MUST construct the **XccosCommandPrimitive** with an **id** value of *cmd:getpref*. The client MUST send the command with exactly one **XccosCommandDataBlock**. Within the **XccosCommandDataBlock**, the client MUST set exactly one **PreferenceDataBlock**. The **label** attribute in the **PreferenceDataBlock** MUST be set to the value of "kedzie.GroupChannels" to retrieve the list of channels that the user is following. The **seqid** attribute MUST be set to "0" and the **createdefault** attribute must be present, but its value (true or false) is irrelevant.

3.1.18.4 Higher-Layer Triggered Events

None.

3.1.18.5 Message Processing Events and Sequencing Rules

If the client receives a failure reply as described in section 3.1.2.1.5, the client MUST NOT perform further processing on the **XccosReplyPrimitive**. If the client receives a success reply, the client MUST look for a **PreferenceDataBlock** inside the **XccosReplyNoticeDataBlock**. If the value of **label** attribute in the **PreferenceDataBlock** is not "kedzie.GroupChannels", the reply is discarded. Otherwise, the client SHOULD inspect the **content** attribute of the **PreferenceDataBlock**. The client SHOULD:

- decode the base64 encoded string value of the **content** attribute into a binary stream.
- unzip that binary stream into a string containing the XML document, formatted as specified in section 2.2.2.1.26.1.
- parse the XML document and extract per-channel user preferences.
- publish user preferences following the protocol as described in [MS-PRES].

3.1.18.6 Timer Events

None.

3.1.18.7 Other Local Events

None.

3.2 Server Details

This section describes protocol details specific to the server.

3.2.1 Receiving XccosCommandPrimitive messages

XccosCommandPrimitive is used by the client to send requests to the server, any others than for chat messages. This section describes how the server handles receiving of XccosCommandPrimitives.

3.2.1.1 Abstract Data Model

A data structure called **CmdId** is used to uniquely identify a command and its corresponding reply. It consists of 2 fields:

EnvId (64bit unsigned integer).

SeqId (64bit unsigned integer).

3.2.1.2 Timers

A generic timer for timeout purposes related to server processing time exists. The default value is 25 seconds.

3.2.1.3 Initialization

None.

3.2.1.4 Higher-Layer Triggered Event

None.

3.2.1.5 Message Processing Events and Sequencing Rules

When the server receives an **XccosCommandPrimitive**, it first checks the **XccosCommandPrimitive** for schema compliance. If the **XccosCommandPrimitive** is not compliant to the schema, a reply of type **XccosErrorPrimitive** MUST be sent back. Its **commandid** MUST be set to the **CmdId** of the incoming message.

If the **XccosCommandPrimitive** is compliant to the schema, the **id** attribute MUST be checked against the list of supported/implemented commands. If not in the list, a reply of type **XccosErrorPrimitive** MUST be sent back. Its **commandid** MUST be set to the **CmdId** of the incoming message.

If the **id** is in the list, specific processing is done for the command, as described in the following sections.

As a result of processing, replies of type **XccosReplyPrimitive** or errors of type **XccosErrorPrimitive** MAY be sent back to the client. All these MUST be stamped with the **CmdId** of the associated incoming message in their **commandid** element.

3.2.1.6 Timer Events

A timeout event MUST trigger a SIP 503 reply to be sent for the pending INFO request that carried the command(s).

3.2.1.7 Other Local Events

Additional server related global state (such as load) MAY be taken into account when deciding on processing vs. dropping incoming messages..

3.2.2 Retrieving Server Information

A client can issue an XCCOS command to retrieve server information.

3.2.2.1 Abstract Data Model

None.

3.2.2.2 Timers

None.

3.2.2.3 Initialization

The **id** of the **XccosCommandPrimitive** is *cmd:getserverinfo*.

3.2.2.4 Higher-Layer Triggered Event

None.

3.2.2.5 Message Processing Events and Sequencing Rules

Server MUST reply with an **XccosReplyPrimitive** with **id** *rpl:getserverinfo*.

The containing **ServerInformationDataBlock** MUST be set based on the **infoType** field from the incoming **ServerInformationDataBlock**, as specified in section 2.2.2.1.10.

3.2.2.6 Timer Events

None.

3.2.2.7 Other Local Events

None.

3.2.3 Joining Multiple Channels

This describes a mechanism for fast joining (compared with joining chat rooms separately) of multiple chat rooms.

3.2.3.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

Server MUST maintain a database of chat rooms.

Server MUST maintain a database of users.

Server MUST maintain lists of connected **endpoints (4)** for each user.

Server MUST maintain state for each <endpoint, chat room> pair:

- **Not Joined**: Endpoint is not participating in the chat room.
- **Joined**: Endpoint is participating in the chat room.

A computed/derived state that corresponds to the user to chat room pair is defined as:

- **Not Joined**: All <Endpoint, Chat Room> tuples are Not Joined.
- **Joined**: At least one <Endpoint, Chat Room> tuple is Joined.

A database of roles (tuples of <user, chat room, role type>) MUST be maintained by the server.

Server MUST maintain a database of chats (backchat) for each chat room.

3.2.3.2 Timers

None.

3.2.3.3 Initialization

The **id** of the **XccosCommandPrimitive** is *cmd:bjoin* as specified in section 2.2.2.2.2.

3.2.3.4 Higher-Layer Triggered Event

None.

3.2.3.5 Message Processing events and Sequencing Rules

1. The server MUST check that the **XccosCommandPrimitive** contains a **data** element as specified in section 2.2.2.2.2. If it is missing, the server MUST reply with an **XccosErrorPrimitive** message.
2. Then the server MUST check the presence of **chanid** elements as specified in section 2.2.2.1.28 and section 2.2.2.1.9.
3. These elements are a representation of a **hash** table that the server must re-construct as follows:
4. The **key** attribute of a **chanid** element is a key in the hash table.
5. The **value** attribute of a **chanid** element is tokenized with "," used as separator. Each resulting token is a value in the hash table.
6. The server interprets the resulting hash table as follows:
7. The key is the number of backchat messages to be sent to the client.
8. The value is the GUID of the chat room to be joined.
9. For each chat room to be joined:
10. The server MUST check if the user is allowed to join based on role. If not allowed, the chat room joining is considered rejected.
11. Otherwise the server MUST change the state of <endpoint, chat room> tuple to **Joined**.
12. The server MUST create an **XccosReplyPrimitive** with the **id** *rpl:bjoin* to send back to the client, as specified in section 2.2.2.2.4.
13. The server MUST set the **data** element as specified in section 2.2.2.1.29.

14. The server MUST create a list of unique users that are **Joined** to at least one of the chat rooms where joining succeeded. The list will be numbered with an **id** for the purpose of this processing.

15. The server MUST create a hash table (key=error code, value=chat room URI) for each chat room where join failed.

16. For each chat room where join succeeded:

17. The server MUST add a **chanib** element as specified in section 2.2.2.1.7.

18. The server MUST add an **aib** element as specified in section 2.2.2.1.22 for each role type if at least one user of that role type is **Joined** to that chat room.

19. The key attribute is the value of the role type, and the value attribute is a comma separated list obtained by concatenating the corresponding user indexes as generated in step 7.

20. The server MUST add **uib** elements for each item in the list created at step 7, as specified in section 2.2.2.1.4

21. If at least one chat room join failed (at step 4), the server MUST add **fib** elements as specified in section 2.2.2.1.23.

22. For each chat room where join succeeded:

    - The server MUST create an **XccosReplyPrimitive** with **id** *rpl:bccontext* as specified in section 2.2.2.2.4
    - The server MUST add a **data** element as specified in section 2.2.2.1.29.
    - The server MUST add backchat **msg** elements as specified in section 2.2.2.2.9.

23. The server MUST send all assembled messages to the client.

3.2.3.6 Timer Events

None.

3.2.3.7 Other Local Events

None.

3.2.4 Joining Single Channel

This describes the joining of a single chat room.

3.2.4.1 Abstract Data Model

This section is similar to section 3.2.3.1.

3.2.4.2 Timers

None.

3.2.4.3 Initialization

The **id** of the **XccosCommandPrimitive** is *cmd:join*, as specified in section 2.2.2.2.2.

3.2.4.4 Higher-Layer Triggered Event

None.

3.2.4.5 Message Processing Events and Sequencing Rules

1. The server MUST check that the **XccosCommandPrimitive** contains a **data** element as specified in section 2.2.2.2.2. If it is missing the server MUST reply with an **XccosErrorPrimitive** message.
2. Then the server MUST check the presence of **chanid** element as specified in section 2.2.2.1.28 and section 2.2.2.1.9. The server MUST interpret the **value** attribute as the GUID of the chat room to join.
3. The server MUST check that the user is allowed to join based on role. If not allowed, the chat room joining is considered rejected and the server must send back an **XccosErrorPrimitive** message.
4. Otherwise the server MUST change the state of the <endpoint, chat room> tuple to **Joined**.
5. The server MUST create an **XccosReplyPrimitive** with the **id** *rpl:join* to send back to the client, as specified in section 2.2.2.2.4.
6. The server MUST set the **data** element as specified in section 2.2.2.1.29.
7. The server MUST create a list of users that are **Joined** to the chat room. The list will be numbered with an **id** for the purpose of this processing.
8. The server MUST add a **chanid** element as specified in section 2.2.2.1.7.
9. Server MUST add an **aib** element as specified in section 2.2.2.1.22 for each role type if at least one user of that role type is **Joined** to that chat room. The **key** attribute is the value of the role type, and the **value** attribute is a comma separated list obtained buy concatenating the corresponding user indexes as generated in step 7.
10. The server MUST add **uib** elements for each item in the list created at step 7, as specified in section 2.2.2.1.4
11. The server MUST send the assembled message to the client.

3.2.4.6 Timer Events

None.

3.2.4.7 Other Local Events

None.

3.2.5 Retrieving Most Recent Chat History From A Channel

This is used to retrieve the latest chat messages for a chat room.

3.2.5.1 Abstract Data Model

This is similar to section 3.2.3.1

3.2.5.2 Timers

None.

3.2.5.3 Initialization

The **id** of the **XccosCommandPrimitive** is *cmd:bccontext*, as specified in section 2.2.2.2.2.

3.2.5.4 Higher-Layer Triggered Event

None.

3.2.5.5 Message Processing Events and Sequencing Rules

1. The server MUST check that the **XccosCommandPrimitive** contains a **data** element as specified in section 2.2.2.2.2. If it is missing the server MUST reply with an **XccosErrorPrimitive** message.
2. Then the server MUST check the presence of **chanib** elements as specified in section 2.2.2.1.28 and section 2.2.2.1.7.
3. The server MUST check the presence of a **bcq** element as specified in section 2.2.2.1.28 and section 2.2.2.1.17.
4. The server MUST check the presence of a **cnt** attribute in the last element, as specified in section 2.2.2.1.17.1.
5. The server MUST search in the chat database associated to the room for the latest messages, not more than what the last attribute specified.
6. The server MUST create an **XccosReplyPrimitive** with **id** *rpl:bccontext* to send back to the client, as specifibed in section 2.2.2.2.4.
7. The server MUST set the **data** element as specified in section 2.2.2.1.29.
8. The server MUST add backchat **msg** elements as specified in section 2.2.2.2.9
9. The server MUST send the assembled message to the client.

3.2.5.6 Timer Events

None.

3.2.5.7 Other Local Events

None.

3.2.6 Processing Chat Messages

This describes the receiving and then the fanning out of chat messages.

3.2.6.1 Abstract Data Model

Similar to section 3.2.3.1.

In addition, the server MUST maintain a "last chat id" number for each chat room.

3.2.6.2 Timers

None.

3.2.6.3 Initialization

The **id** of the **GroupChatDataBlock** element is *grpcht*, as specified in section 2.2.2.2.9.

3.2.6.4 Higher-Layer Triggered Event

None.

3.2.6.5 Message Processing Events and Sequencing Rules

1. The server MUST check that the **GroupChatDataBlock** contains the mandatory attributes as specified in section 2.2.2.2.9. If any is missing the server MUST reply with an **XccosErrorPrimitive** message.
2. The server MUST generate a new "last chat id" for the chat room identified in the message.
3. The server MUST set **chatId** to the new "last chat id".
4. The server MUST generate a chat timestamp and set it to **ts**.
5. The server SHOULD fill the **authdisp** field with a displayable name of the user identified by **author**, as specified in section 2.2.2.2.9.
6. The server MUST set the **OriginatingMessageId** as follows:
7. The **SeqId** must be set to the value held by the similar field in **GroupChatDataBlock**.
8. The **EnvId** must be set to the value held by the similar field in the embedding xccos document.
9. The server MAY decide to add the chat to the database of per chat room chats.
10. The server MUST send the updated message to all endpoints (4) joined to the chat room.

3.2.6.6 Timer Events

None.

3.2.6.7 Other Local Events

None.

3.2.7 Retrieving Channel Permissions

This provides a mechanism for the signed-in user to determine permissions on a channel.

3.2.7.1 Abstract Data Model

This section is similar to section 3.2.3.1

3.2.7.2 Timers

None.

3.2.7.3 Initialization

The **id** of the **XccosCommandPrimitive** is *cmd:getroomperms* as specified in section 2.2.2.2.2

3.2.7.4 Higher-Layer Triggered Events

None.

3.2.7.5 Message Processing Events and Sequencing Rules

1. The server MUST check that the **XccosCommandPrimitive** contains a **data** element as specified in section 2.2.2.2.2. If it is missing, the server MUST reply with an **XccosErrorPrimitive** message.
2. The server MUST check for the presence of a **chanib** element as specified in section 2.2.2.1.23 and section 2.2.2.1.7.
3. The server MUST check that the user is known by the server. If the user is unknown, the server MUST reply with an **XccosErrorPrimitive** message.
4. The server MUST check that the **uri** attribute value in the **chanib** is a valid channel URI. If the URI is not valid, the server MUST reply with an **XccosErrorPrimitive** message.
5. The server MUST check that the user has permission to know that the channel exists. If the user does not have the required permission, the server MUST reply with an **XccosErrorPrimitive** message.
6. The server MUST create an **XccosReplyPrimitive** with the **id** *rpl:getroomperms* to return to the client as specified in section 2.2.2.2.4.
7. The server MUST add a **data** element to the **XccosReplyPrimitive**.
8. The server MUST add a **chanib** element to the **data** element.
9. The server MUST add a **uib** element to the **chanib** element.
10. The server MUST add a **chperms** attribute to the **uib** element. The **chperms** value is the permission bitmap for the user and channel as specified in section 2.2.2.1.4.
11. The server MUST send the **XccosReplyPrimitive** to the client.

3.2.7.6 Timer Events

None.

3.2.7.7 Other Local Events

None.

3.2.8 Modifying a Channel

This describes a mechanism for modifying a channel.

3.2.8.1 Abstract Data Model

This section is similar to section 3.2.3.1

3.2.8.2 Timers

None.

3.2.8.3 Initialization

The **id** of the **XccosCommandPrimitive** is *cmd:updatenode* as specified in section 2.2.2.2.2

3.2.8.4 Higher-Layer Triggered Events

None.

3.2.8.5 Message Processing Events and Sequencing Rules

1. The server MUST check that the **XccosCommandPrimitive** contains a **data** element as specified in section 2.2.2.2.2. If it is missing, the server MUST reply with an **XccosErrorPrimitive** message.
2. The server MUST check for the presence of a **chanib** element as specified in section 2.2.2.1.7.
3. The server MUST check that the user is known by the server. If the user is unknown, the server MUST reply with an **XccosErrorPrimitive** message.
4. The server MUST check that the **uri** attribute value in the **chanib** is a valid channel URI. If the URI is not valid, the server MUST reply with an **XccosErrorPrimitive** message.
5. The server MUST check that the user has permissions to modify the channel. If the user does not have permission, the server MUST reply with an **XccosErrorPrimitive** message.
6. The server MUST check that all required and optional attributes and elements of the **chanib** element are present and complete. If any are not valid, the server MUST reply with an **XccosErrorPrimitive** message.
7. The server MUST update the channel according to the data in the **chanib** element.
8. The server MUST create an **XccosReplyPrimitive** with the **id** *rpl:updatenode* to return to the client as specified in section 2.2.2.2.4.
9. The server MUST add a **data** element to the **XccosReplyPrimitive**.
10. The server MUST add a **chanib** element to the **data** element.
11. The server MUST add the resultant channel attributes and elements to the **chanib** element.
12. The server MUST return this **XccosReplyPrimitive** to the client.

3.2.8.6 Timer Events

None.

3.2.8.7 Other Local Events

None.

4 Protocol Examples

4.1 Retrieving Server Information

Client requests server information:

```
<xccos ver="1" envid="6698699123101735678" xmlns="urn:parlano:xml:ns:xccos">
  <cmd id="cmd:getserverinfo" seqid="1">
    <data>      <sib domain="example.com" infoType="507" clientVersion="4.0.7577.253" />
    </data>
  </cmd>
</xccos>
```

Server returns requested information:

```
<xccos xmlns:xsd="http://www.w3.org/2001/XMLSchema"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       ver="1"
       envid="1472189363123229190"
       xmlns="urn:parlano:xml:ns:xccos">
  <rpl id="rpl:getserverinfo" seqid="1">
    <commandid seqid="1" envid="6698699123101735678" />
    <resp code="200">SUCCESS_OK</resp>
    <data>
      <sib infoType="251"
           serverTime="2011-10-27T21:05:48.4345771Z"
           searchLimit="999"
           messageSizeLimit="8000"
           storySizeLimit="16000"
           rootUri="ma-cat://example.com/cf724a9b-4595-4556-809d-b7846c4c4320"
           dbVersion="de64325e-d4ab-44d5-9ea8-55785b7c8a3b"
           serverVersion="5.0.7853.0" />
    </data>
  </rpl>
</xccos>
```

4.2 Batch joining

Client requests a batch joining:

```
<xccos ver="1" envid="6698699123101735680" xmlns="urn:parlano:xml:ns:xccos">
  <cmd id="cmd:bjoin" seqid="1">
    <data>
      <chanid key="100"
              value="93489432-b6be-4c67-932f-09e39a162072,
                     6e43547f-9152-46e3-ad76-82197694fdb9"
              domain="example.com" />
    </data>
  </cmd>
</xccos>
```

Server sends the chat room information, user information and user to chat room mapping in a reply.

Additional "rpl:bccontext" messages are sent with backchat, usually in the same envelope:

```
<xccos xmlns:xsd="http://www.w3.org/2001/XMLSchema"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       ver="1"
       envid="14721893631232291 93"
       xmlns="urn:parlano:xml:ns:xccos">
  <rpl id="rpl:bjoin" seqid="1">
    <commandid seqid="1" envid="6698699123101735680" />
    <resp code="200">SUCCESS_OK</resp>
    <data>
      <chanib name="Projects"
              description="Various public projects"
              parent="ma-cat://example.com/2642ebba-f56a-4891-9b92-3991eb865c92"
              uri="ma-chan://example.com/93489432-b6be-4c67-932f-09e39a162072"
              overridemembers="false" behavior="NORMAL" topic="" disabled="false">
        <aib key="3456" value="0,1,2" />
        <aib key="11652" value="1" />
        <audit updatedby="Jane Doe" updatedon="2011-10-24T21:11:22.3429958Z"
               createdby="Jane Doe" createdon="2011-10-24T21:11:22.1489764Z" />
        <info id="urn:parlano:ma:info:filestoreuri">
          https://webserver.example.com/mgcwebservice/mgcwebservice.asmx
        </info>        <info id="urn:parlano:ma:info:ucnt">1</info>
        <info id="urn:parlano:ma:info:visibility">SCOPED</info>
        <prop id="urn:parlano:ma:prop:logged">True</prop>
        <prop id="urn:parlano:ma:prop:invite">True</prop>
        <prop id="urn:parlano:ma:prop:filepost">True</prop>
      </chanib>
      <chanib name="Top Secret"
              description="Top Secret stuff"
              parent="ma-cat://example.com/62d7af6b-236a-45bc-88d8-74dcedd4854f"
              uri="ma-chan://example.com/6e43547f-9152-46e3-ad76-82197694fdb9"
              overridemembers="false" behavior="NORMAL" topic="" disabled="false"
              siopname="Top Secret Portal"
              siopurl="http://topsecretportal.webserver.example.com"
              siopid="a01632c4-20ae-44a7-8ccf-24dc81cf3b32">
        <aib key="3456" value="0,2" />
        <audit updatedby="sysuser" updatedon="2011-10-04T21:37:55.1235192Z"
               createdby="sysuser" createdon="2011-10-02T22:08:07.7617317Z" />
        <info id="urn:parlano:ma:info:filestoreuri">
          https://webserver.example.com/mgcwebservice/mgcwebservice.asmx
        </info>
        <info id="urn:parlano:ma:info:ucnt">2</info>
        <info id="urn:parlano:ma:info:visibility">PRIVATE</info>
        <prop id="urn:parlano:ma:prop:logged">True</prop>
        <prop id="urn:parlano:ma:prop:invite">False</prop>
        <prop id="urn:parlano:ma:prop:filepost">True</prop>
      </chanib>      <uib uri="sip:johnsm@example.com"
           guid="2106938B-BEA5-45D6-A74E-16A4BB2FC710" type="5"
           uname="John Smith" disabled="false" dispname="John Smith" id="0">
        <perms defined="1" inherited="1" inheriting="true" />
      </uib>
      <uib uri="sip:janedoe@example.com"
           guid="93109AFC-D91D-45A1-96F4-6DCBBB31B640" type="5"
           uname="Jane Doe" disabled="false" dispname="Jane Doe" id="1">
        <perms defined="1" inherited="1" inheriting="true" />
      </uib>
      <uib uri="sip:johndoe@example.com"
           guid="E5934BA3-B487-48A6-9D46-3436D6325B6D" type="5"
           uname="John Doe" disabled="false" dispname="John Doe" id="2">
        <perms defined="1" inherited="1" inheriting="true" />
```

```
      </uib>
    </data>
  </rpl>
  <rpl id="rpl:bccontext" seqid="2">
    <commandid seqid="1" envid="6698699123101735680" />
    <resp code="200">SUCCESS_OK</resp>
    <data>
      <chanib uri="ma-chan://example.com/93489432-b6be-4c67-932f-09e39a162072"
              overridemembers="false" behavior="UNSET" topic="" disabled="false">
        <msg id="grpchat"
             chanUri="ma-chan://example.com/93489432-b6be-4c67-932f-09e39a162072"
             author="sip:janedoe@example.com" authdisp="Jane Doe"
             alert="false" chatId="1" ts="2011-10-24T21:23:33.887Z">
          <chat>Test</chat>
        </msg>
      </chanib>
      <cnt value="1" over="false" />
      <status>1</status>
    </data>
  </rpl>

</xccos>
```

4.3 Retrieve Most Recent Chat History

Client sends request for backchat context:

```
<xccos ver="1" envid="6698699123101735704" xmlns="urn:parlano:xml:ns:xccos">
  <cmd id="cmd:bccontext" seqid="1">
    <data>
      <chanib uri="ma-chan://example.com/2a1a367c-5c14-4215-b5ae-d04eacb3b203"
              overridemembers="false" behavior="UNSET" disabled="false" />
      <bcq>
        <last cnt="100" />
      </bcq>
    </data>
  </cmd>
</xccos>
```

Server returns the backchat:

```
<xccos xmlns:xsd="http://www.w3.org/2001/XMLSchema"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       ver="1"
       envid="1472189363123229193"
       xmlns="urn:parlano:xml:ns:xccos">
  <rpl id="rpl:bccontext" seqid="1">
    <commandid seqid="1" envid="6698699123101735704" />
    <resp code="200">SUCCESS_OK</resp>
    <data>
      <chanib uri="ma-chan://example.com/2a1a367c-5c14-4215-b5ae-d04eacb3b203"
              overridemembers="false" behavior="UNSET" topic="" disabled="false">
        <msg id="grpchat"
             chanUri="ma-chan://example.com/2a1a367c-5c14-4215-b5ae-d04eacb3b203"
             author="sip:johndoe@example.com" authdisp="John Doe"
             alert="true" chatId="78" ts="2011-10-21T21:52:47.233Z">
```

```
          <chat>Let's meet!</chat>
        </msg>
        <msg id="grpchat"
             chanUri="ma-chan://example.com/2a1a367c-5c14-4215-b5ae-d04eacb3b203"
             author="sip:janedoe@example.com" authdisp="Jane Doe"
             alert="false" chatId="79" ts="2011-10-22T00:21:06.993Z">
          <chat>Where?</chat>
          <rtf>
{\urtf1\fbidis\ansi\ansicpg1252\deff0\nouicompat\deflang1033{\fonttbl{\f0\fnil\fcharset0
Segoe UI;}{\f1\fnil Segoe UI;}}{\colortbl ;\red51\green51\blue51;}{\*\generator Riched20
15.0.3419 (Debug)}{\*\mmathPr\mwrapIndent1440 }\viewkind4\uc1\pard\cf1\f0\fs18 Where?\f1\par}
          </rtf>
        </msg>
      </chanib>
      <cnt value="2" over="false" />
      <status>2</status>
    </data>
  </rpl>
</xccos>
```

4.4 Chat Room Search

Client requests a chat room search:

```
<xccos ver="1" envid="6698699123101735688" xmlns="urn:parlano:xml:ns:xccos">
  <cmd id="cmd:chansrch" seqid="1">
    <data>
      <qib qtype="BYNAME" criteria="Proj" extended="false" />
    </data>
  </cmd>
</xccos>
```

Server returns the results:

```
<xccos xmlns:xsd="http://www.w3.org/2001/XMLSchema"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       ver="1" envid="14721893631232292 57"
       xmlns="urn:parlano:xml:ns:xccos">
  <rpl id="rpl:chansrch" seqid="1">
    <commandid seqid="1" envid="6698699123101735688" />
    <resp code="200">SUCCESS_OK</resp>
    <data>
      <chanib name="Projects"
              description="Internal projects"
              parent="ma-cat://example.com/2642ebba-f56a-4891-9b92-3991eb865c92"
              uri="ma-chan://example.com/e145b4be-d76a-4854-bac9-6cd101a96650"
              overridemembers="false" behavior="NORMAL" topic="" disabled="false">
        <audit updatedby="Admin" updatedon="2011-10-03T21:21:52.9538233Z"
               createdby="Admin" createdon="2011-10-03T21:21:52.9538233Z" />
        <info id="urn:parlano:ma:info:ucnt">1</info>
        <info id="urn:parlano:ma:info:visibility">SCOPED</info>
        <prop id="urn:parlano:ma:prop:invite">True</prop>
      </chanib>
      <cnt value="1" over="false" />
    </data>
  </rpl>
```

```
</xccos>
```

4.5 Chat Room Content Search by Date

Client sends the search request:

```
<xccos ver="1" envid="6698699123101735689" xmlns="urn:parlano:xml:ns:xccos">
  <cmd id="cmd:bcbydate" seqid="1">
    <data>
      <bcs cmp="OR">
        <limit cnt="50" />
        <text mt="PP">Joe</text>
        <matchcase>false</matchcase>
        <searchbkwds>true</searchbkwds>
        <sortbkwds>true</sortbkwds>
        <date from="2004-12-15T22:02:50Z" to="2011-10-28T21:11:57.1247226Z" />
        <cib uri="ma-chan://example.com/66b00dd5-6f18-4b6c-b51f-f2c7aada05cf"
             overridemembers="false" behavior="UNSET" disabled="false" />
      </bcs>
    </data>
  </cmd>
</xccos>
```

Server returns the matching chats:

```
<xccos xmlns:xsd="http://www.w3.org/2001/XMLSchema"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       ver="1" envid="1472189363123229258"
       xmlns="urn:parlano:xml:ns:xccos">
  <rpl id="rpl:bc" seqid="1">
    <commandid seqid="1" envid="6698699123101735689" />
    <resp code="200">SUCCESS_OK</resp>
    <data>
      <chanib uri="ma-chan://example.com/66b00dd5-6f18-4b6c-b51f-f2c7aada05cf">
        <msg id="grpchat"
             chanUri="ma-chan://example.com/66b00dd5-6f18-4b6c-b51f-f2c7aada05cf"
             author="sip:janedoe@example.com" authdisp="Jane Doe"
             alert="false" chatId="20" ts="2011-10-26T23:06:20.99Z">
          <chat>@Joe: no, I'm on a W14 one, it seems.</chat>
        </msg>
        <msg id="grpchat"
             chanUri="ma-chan://example.com/66b00dd5-6f18-4b6c-b51f-f2c7aada05cf"
             author="sip:johnsm@example.com" authdisp="John Smith"
             alert="false" chatId="3" ts="2011-10-07T17:45:59.873Z">
          <chat>Who is Joe?</chat>
        </msg>
      </chanib>
      <cnt value="2" over="false" />
    </data>
  </rpl>
</xccos>
```

4.6 Sending Chats

Client sends a chat:

```
<xccos ver="1" envid="6698699123101735682" xmlns="urn:parlano:xml:ns:xccos">
  <grpchat id="grpchat" seqid="1"
           chanUri="ma-chan://example.com/93489432-b6be-4c67-932f-09e39a162072"
           author="sip:johns@example.com" authdisp=""
           alert="false" chatId="0" ts="2011-10-27T21:09:48.3368091Z">
    <originatingMessageId seqid="1" envid="6698699123101735682" />
    <chat>Hello, World!</chat>
  </grpchat>
</xccos>
```

Server replies to the client and sends a similar message to all the other participants:

```
<xccos xmlns:xsd="http://www.w3.org/2001/XMLSchema"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       ver="1" envid="14721893631232292226"
       xmlns="urn:parlano:xml:ns:xccos">
  <grpchat id="grpchat" seqid="1"
           chanUri="ma-chan://example.com/93489432-b6be-4c67-932f-09e39a162072"
           author="sip:johnsm@example.com" authdisp="John Smith"
           alert="false" chatId="227" ts="2011-10-27T21:09:50.247Z">
    <originatingMessageId seqid="1" envid="6698699123101735682" />
    <chat>Hello, World!</chat>
  </grpchat>
</xccos>
```

5 Security

5.1 Security Considerations for Implementers

None.

5.2 Index of Security Parameters

None.

6 Appendix A: Full XML Schema

6.1 XCCOS Schema

The namespace is identified by the URN:

urn:parlano:xml:ns:xccos

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema version="1" xmlns="urn:parlano:xml:ns:xccos"
xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="urn:parlano:xml:ns:xccos"
elementFormDefault="qualified" attributeFormDefault="unqualified">
    <!-- XCCOS Control Document Definition +++++++++++++++++++++++++++++++ -->
    <xs:element name="xccos" type="XccosControlPrimitive">
    </xs:element>
    <xs:complexType name="XccosControlPrimitive">
        <xs:sequence>
            <xs:choice minOccurs="1" maxOccurs="unbounded">
                <xs:element name="cmd" type="XccosCommandPrimitive" minOccurs="0"
maxOccurs="unbounded" />
                <xs:element name="rpl" type="XccosReplyPrimitive" minOccurs="0"
maxOccurs="unbounded" />
                <xs:element name="ntc" type="XccosNoticePrimitive" minOccurs="0"
maxOccurs="unbounded" />
                <xs:element name="err" type="XccosErrorPrimitive" minOccurs="0"
maxOccurs="unbounded" />
                <xs:element name="sys" type="XccosSystemPrimitive" minOccurs="0"
maxOccurs="unbounded" />
                <xs:element name="grpchat" type="GroupChatDataBlock" minOccurs="0"
maxOccurs="unbounded" />
            </xs:choice>
        </xs:sequence>
        <xs:attribute name="ver" type="xs:nonNegativeInteger" use="required" />
        <xs:attribute name="envid" type="xs:positiveInteger" use="required" />
    </xs:complexType>
    <!-- XCCOS Payload Definitions ++++++++++++++++++++++++++++++++++++++++ -->
    <!-- XCCOS Command Definition -->
    <xs:complexType name="XccosPrimitive">
        <xs:attribute name="id" type="xs:anyURI" use="required" />
        <xs:attribute name="seqid" type="xs:nonNegativeInteger" use="required" />
    </xs:complexType>
    <xs:complexType name="XccosResponsePrimitive">
        <xs:complexContent>
            <xs:extension base="XccosPrimitive">
                <xs:sequence>
                    <xs:element name="commandid" type="XccosMessageIdentifier" minOccurs="0"
/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <xs:complexType name="XccosCommandPrimitive">
        <xs:complexContent>
            <xs:extension base="XccosPrimitive">
                <xs:sequence>
                    <xs:element name="data" type="XccosCommandDataBlock" nillable="false" />
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
```

```
    </xs:complexType>
    <!-- XCCOS Reply Definition -->
    <xs:complexType name="XccosReplyPrimitive">
        <xs:complexContent>
            <xs:extension base="XccosResponsePrimitive">
                <xs:sequence>
                    <xs:element name="resp" type="ResponseBlock" nillable="true" />
                    <xs:element name="data" type="XccosReplyNoticeDataBlock" minOccurs="0" />
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <!-- Message identifier used to identify a single unique message. -->
    <xs:complexType name="XccosMessageIdentifier">
        <xs:attribute name="seqid" type="xs:nonNegativeInteger" use="required" />
        <xs:attribute name="envid" type="xs:nonNegativeInteger" use="required" />
    </xs:complexType>
    <!-- XCCOS Notice Definition -->
    <xs:complexType name="XccosNoticePrimitive">
        <xs:complexContent>
            <xs:extension base="XccosPrimitive">
                <xs:sequence>
                    <xs:element name="data" type="XccosReplyNoticeDataBlock" nillable="false"
/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <!-- XCCOS Error Definition -->
    <xs:complexType name="XccosErrorPrimitive">
        <xs:complexContent>
            <xs:extension base="XccosResponsePrimitive">
                <xs:sequence>
                    <xs:element name="resp" type="ResponseBlock" nillable="false" />
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <!-- XCCOS Status Definition -->
    <xs:complexType name="XccosSystemPrimitive">
        <xs:complexContent>
            <xs:extension base="XccosPrimitive">
                <xs:sequence>
                    <xs:element name="status" type="XccosSystemStatusDataBlock" minOccurs="0"
maxOccurs="1" />
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <!-- XCCOS Status Definition -->
    <xs:complexType name="XccosSystemStatusDataBlock">
        <xs:attribute name="busy" type="xs:boolean" use="optional" default="false" />
    </xs:complexType>
    <!-- XCCOS Data Block Definitions
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->
    <xs:complexType name="XccosCommandDataBlock">
        <xs:sequence>
            <xs:element name="chanib" type="ChannelInformationDataBlock" minOccurs="0" />
            <xs:element name="catib" type="CategoryInformationDataBlock" minOccurs="0" />
```

```
            <xs:element name="uib" type="UserInformationDataBlock" minOccurs="0" />
            <xs:element name="gib" type="GroupInformationDataBlock" minOccurs="0" />
            <xs:element name="bcq" type="BcQueryDataBlock" minOccurs="0" />
            <xs:element name="bcs" type="BcSearchDataBlock" minOccurs="0" />
            <xs:element name="qib" type="QueryInformationDataBlock" minOccurs="0" />
            <xs:element name="pref" type="PreferenceDataBlock" minOccurs="0" />
            <xs:element name="chanid" type="ChannelIdsInformationDataBlock" minOccurs="0"
maxOccurs="unbounded" />
            <xs:element name="sib" type="ServerInformationDataBlock" minOccurs="0" />
            <xs:element name="inv" type="InviteDataBlock" minOccurs="0" />
            <xs:element name="association" type="AssociationDataBlock" minOccurs="0" />
            <xs:element name="siops" type="SiopWhitelistDataBlock" minOccurs="0" />
            <xs:element name="scope" type="ScopeInformationDataBlock" minOccurs="0" />
            <xs:element name="filtib" type="FilterInformationDataBlock" minOccurs="0" />
            <xs:element name="delchat" type="DeleteChatDataBlock" minOccurs="0" />
            <xs:element name="purgeb" type="PurgeChannelDataBlock" minOccurs="0" />
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="XccosReplyNoticeDataBlock">
        <xs:sequence>
            <xs:element name="chanib" type="ChannelInformationDataBlock" minOccurs="0"
maxOccurs="unbounded" />
            <xs:element name="catib" type="CategoryInformationDataBlock" minOccurs="0"
maxOccurs="unbounded" />
            <xs:element name="uib" type="UserInformationDataBlock" minOccurs="0"
maxOccurs="unbounded" />
            <xs:element name="gib" type="GroupInformationDataBlock" minOccurs="0"
maxOccurs="unbounded" />
            <xs:element name="fib" type="FailureInformationDataBlock" minOccurs="0"
maxOccurs="unbounded" />
            <xs:element name="hash" type="HashInformationDataBlock" minOccurs="0"
maxOccurs="unbounded" />
            <xs:element name="cnt" type="ResultCountDataBlock" minOccurs="0" />
            <xs:element name="status" type="xs:nonNegativeInteger" minOccurs="0"
maxOccurs="1" />
            <xs:element name="pref" type="PreferenceDataBlock" minOccurs="0" />
            <xs:element name="tag" type="xs:string" minOccurs="0" />
            <xs:element name="sib" type="ServerInformationDataBlock" minOccurs="0" />
            <xs:element name="grpchat" type="GroupChatDataBlock" minOccurs="0" />
            <xs:element name="siops" type="SiopWhitelistDataBlock" minOccurs="0" />
            <xs:element name="association" type="AssociationDataBlock" minOccurs="0" />
            <xs:element name="scope" type="ScopeInformationDataBlock" minOccurs="0" />
            <xs:element name="purgesizes" type="PurgeChannelSizesDataBlock" minOccurs="0" />
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="SiopWhitelistDataBlock">
        <xs:sequence>
            <xs:element name="siop" type="SiopDataBlock" minOccurs="1" maxOccurs="unbounded"
/>
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="SiopDataBlock">
        <xs:attribute name="guid" type="xs:string" use="optional" />
        <xs:attribute name="name" type="xs:normalizedString" use="optional" />
        <xs:attribute name="uri" type="xs:normalizedString" use="optional" />
        <xs:attribute name="action" type="SiopVerbEnum" use="optional" />
    </xs:complexType>

    <xs:simpleType name="SiopVerbEnum">
```

```
        <xs:restriction base="xs:string">
            <xs:enumeration value="A" />
            <xs:enumeration value="R" />
            <xs:enumeration value="M" />
        </xs:restriction>
    </xs:simpleType>

    <xs:complexType name="QueryInformationDataBlock">
        <xs:attribute name="qtype" type="xs:anyURI" use="optional" />
        <xs:attribute name="criteria" type="xs:normalizedString" use="optional" />
        <xs:attribute name="recurse" type="xs:boolean" use="optional" />
        <xs:attribute name="extended" type="xs:boolean" use="optional" />
        <xs:attribute name="matchAll" type="xs:boolean" use="optional" default="true"/>
        <xs:attribute name="matchExactPhrase" type="xs:boolean" use="optional"
default="true"/>
        <xs:attribute name="purpose" type="xs:int" />
        <xs:attribute name="keywords" type="xs:normalizedString" use="optional" />
        <xs:attribute name="catUri" type="xs:anyURI" use="optional" />
        <xs:attribute name="maxResults" type="xs:int" use="optional" />
    </xs:complexType>
    <xs:complexType name="FilterInformationDataBlock">
        <xs:sequence>
            <xs:element name="member" type="Ace" minOccurs="0" maxOccurs="1" />
            <xs:element name="manager" type="Ace" minOccurs="0" maxOccurs="1" />
        </xs:sequence>
        <xs:attribute name="criteria" type="xs:normalizedString" use="optional" />
        <xs:attribute name="includeTopic" type="xs:boolean" default="false" />
        <xs:attribute name="matchAll" type="xs:boolean" use="optional" default="true"/>
        <xs:attribute name="matchExactPhrase" type="xs:boolean" use="optional"
default="true"/>
        <xs:attribute name="catUri" type="xs:anyURI" use="optional" />
        <xs:attribute name="addinGuid" type="xs:string" use="optional" />
        <xs:attribute name="disabled" type="xs:boolean" use="optional" />
        <xs:attribute name="vis" type="xs:int" use="optional" />
        <xs:attribute name="type" type="xs:int" use="optional" />
        <xs:attribute name="invites" type="xs:string" use="optional" />
        <xs:attribute name="searchInvites" type="xs:boolean" default="false" />
        <xs:attribute name="exceedsMB" type="xs:int" use="optional" />
        <xs:attribute name="maxResults" type="xs:int" use="optional" />
    </xs:complexType>
    <xs:complexType name="InformationDataBlock" abstract="true">
        <xs:sequence>
            <xs:element name="uib" type="UserInformationDataBlock" minOccurs="0"
maxOccurs="unbounded" />
            <xs:element name="gib" type="GroupInformationDataBlock" minOccurs="0"
maxOccurs="unbounded" />
            <xs:element name="aib" type="ActiveInformationDataBlock" minOccurs="0"
maxOccurs="unbounded" />
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="AuditableInformationDataBlock" abstract="true">
        <xs:complexContent>
            <xs:extension base="InformationDataBlock">
                <xs:sequence>
                    <xs:element name="audit" type="AuditDataBlock" minOccurs="0"
maxOccurs="1" />
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
```

```
    </xs:complexType>
    <xs:complexType name="NodeInformationDataBlock" abstract="true">
        <xs:complexContent>
            <xs:extension base="AuditableInformationDataBlock">
                <xs:sequence>
                    <xs:element name="info" type="InfoField" minOccurs="0"
maxOccurs="unbounded" />
                    <xs:element name="prop" type="PropertyField" minOccurs="0"
maxOccurs="unbounded" />
                    <xs:element name="ace" type="Ace" minOccurs="0" maxOccurs="unbounded" />
                </xs:sequence>
                <xs:attribute name="name" type="xs:normalizedString" use="optional" />
                <xs:attribute name="description" type="xs:normalizedString" use="optional" />
                <xs:attribute name="parent" type="xs:anyURI" use="optional" />
                <xs:attribute name="uri" type="xs:anyURI" use="optional" />
                <xs:attribute name="overridemembers" type="xs:boolean" use="optional" />
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <xs:complexType name="ChannelInformationDataBlock">
        <xs:complexContent>
            <xs:extension base="NodeInformationDataBlock">
                <xs:sequence>
                    <xs:element name="uset" type="UserSettingField" minOccurs="0"
maxOccurs="unbounded" />
                    <xs:element name="msg" type="GroupChatDataBlock" minOccurs="0"
maxOccurs="unbounded" />
                    <xs:element name="members" type="RoleList" minOccurs="0" maxOccurs="1" />
                    <xs:element name="managers" type="RoleList" minOccurs="0" maxOccurs="1"
/>
                    <xs:element name="presenters" type="RoleList" minOccurs="0" maxOccurs="1"
/>
                </xs:sequence>
                <xs:attribute name="behavior" type="xs:anyURI" use="optional" />
                <xs:attribute name="topic" type="xs:normalizedString" use="optional" />
                <xs:attribute name="disabled" type="xs:boolean" use="optional" />
                <xs:attribute name="partListOff" type="xs:boolean" use="optional" />
                <xs:attribute name="siopname" type="xs:normalizedString" use="optional" />
                <xs:attribute name="siopurl" type="xs:anyURI" use="optional" />
                <xs:attribute name="siopid" type="xs:string" use="optional" />
                <xs:attribute name="keywords" type="xs:normalizedString" use="optional" />
                <xs:attribute name="filerepository" type="xs:anyURI" use="optional" />
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <xs:complexType name="CategoryInformationDataBlock">
        <xs:complexContent>
            <xs:extension base="NodeInformationDataBlock">
                <xs:sequence>
                    <xs:element name="creators" type="RoleList" minOccurs="0" maxOccurs="1"
/>
                </xs:sequence>
                <xs:attribute name="childinherits" type="xs:boolean" use="optional" />
                <xs:attribute name="allowscopechange" type="xs:boolean" use="optional" />
                <xs:attribute name="numChatRooms" type="xs:nonNegativeInteger" use="optional"
/>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <xs:complexType name="HashInformationDataBlock">
```

```
            <xs:complexContent>
                <xs:extension base="InformationDataBlock">
                    <xs:attribute name="key" type="xs:string" use="required" />
                    <xs:attribute name="value" type="xs:string" use="required" />
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
        <xs:complexType name="ChannelIdsInformationDataBlock">
            <xs:complexContent>
                <xs:extension base="HashInformationDataBlock">
                    <xs:attribute name="domain" type="xs:string" use="required" />
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
        <xs:complexType name="FailureInformationDataBlock">
            <xs:complexContent>
                <xs:extension base="HashInformationDataBlock">
                    <xs:attribute name="domain" type="xs:normalizedString" use="required" />
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
        <xs:complexType name="ActiveInformationDataBlock">
            <xs:complexContent>
                <xs:extension base="HashInformationDataBlock">
                    <xs:attribute name="domain" type="xs:normalizedString" use="optional" />
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
        <xs:complexType name="ServerInformationDataBlock">
            <xs:complexContent>
                <xs:extension base="InformationDataBlock">
                    <xs:attribute name="domain" type="xs:anyURI" use="required" />
                    <xs:attribute name="infoType" type="xs:long" use="required" />
                    <xs:attribute name="serverTime" type="Iso8601TimeString" use="optional" />
                    <xs:attribute name="searchLimit" type="xs:int" use="optional" />
                    <xs:attribute name="messageSizeLimit" type="xs:int" use="optional" />
                    <xs:attribute name="storySizeLimit" type="xs:int" use="optional" />
                    <xs:attribute name="rootUri" type="xs:anyURI" use="optional"/>
                    <xs:attribute name="dbVersion" type="xs:string" use="optional"/>
                    <xs:attribute name="clientVersion" type="xs:string" use="optional"/>
                    <xs:attribute name="serverVersion" type="xs:string" use="optional"/>
                    <xs:attribute name="displayName" type = "xs:string" use="optional"/>
                    <xs:attribute name="roomManagementUrl" type = "xs:anyURI" use="optional"/>
                </xs:extension>
            </xs:complexContent>
        </xs:complexType>
        <xs:complexType name="InviteDataBlock">
            <xs:attribute name="register" type="xs:boolean" use="optional" default="true" />
            <xs:attribute name="inviteId" type="xs:unsignedLong" use="optional" default="0" />
            <xs:attribute name="domain" type="xs:string" use="required" />
        </xs:complexType>
        <xs:complexType name="AssociationDataBlock">
            <xs:sequence>
                <xs:element name="chanib" type="ChannelInformationDataBlock" minOccurs="0"
  maxOccurs="unbounded" />
            </xs:sequence>
            <xs:attribute name="domain" type="xs:string" use="required" />
            <xs:attribute name="type" type="AssociationTypeEnum" use="required" />
            <xs:attribute name="maxResults" type="xs:unsignedInt" use="optional" default="100"/>
```

```
        <xs:attribute name="hash" type="xs:unsignedLong" use="optional" />
    </xs:complexType>
    <xs:simpleType name="AssociationTypeEnum">
        <xs:restriction base="xs:string">
            <xs:enumeration value="MEMBER" />
            <xs:enumeration value="MANAGER" />
        </xs:restriction>
    </xs:simpleType>
    <xs:complexType name="ScopeInformationDataBlock">
        <xs:sequence>
            <xs:element name="entry" type="ScopeDefinition" minOccurs="0"
maxOccurs="unbounded" />
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="ScopeDefinition">
        <xs:attribute name="uri" type="xs:anyURI" use="optional"/>
        <xs:attribute name="path" type="xs:string" use="optional"/>
        <xs:attribute name="denied" type="xs:boolean" use="required"/>
        <xs:attribute name="name" type="xs:string" use="optional"/>
        <xs:attribute name="guid" type="xs:string" use="optional"/>
        <xs:attribute name="type" type="xs:string" use="optional"/>
    </xs:complexType>
    <!-- Audit fields:  All the audit fields that should be sent over the wire -->
    <xs:complexType name="AuditDataBlock">
        <xs:attribute name="updatedby" type="Username" use="optional" />
        <xs:attribute name="updatedon" type="Iso8601TimeString" use="optional" />
        <xs:attribute name="createdby" type="Username" use="optional" />
        <xs:attribute name="createdon" type="Iso8601TimeString" use="optional" />
    </xs:complexType>
    <!-- Permission fields for principals:  All the permission fields that should be sent
over the wire -->
    <xs:complexType name="UserPermissionDataBlock">
        <xs:attribute name="defined" type="xs:nonNegativeInteger" use="optional" />
        <xs:attribute name="inherited" type="xs:nonNegativeInteger" use="optional" />
        <xs:attribute name="inheriting" type="xs:boolean" use="optional" />
    </xs:complexType>
    <!-- The User Information Block -->
    <xs:complexType name="UserInformationDataBlock">
        <xs:complexContent>
            <xs:extension base="AuditableInformationDataBlock">
                <xs:sequence>
                    <xs:element name="from" type="From" minOccurs="0" maxOccurs="1" />
                    <!--When UserInfo objects specify their affiliations, this should be set
to minOccurs=1 -->
                    <xs:element name="affiliation" type="GroupInformationDataBlock"
minOccurs="0" maxOccurs="unbounded" />
                    <xs:element name="perms" type="UserPermissionDataBlock" minOccurs="0"
maxOccurs="1" />
                </xs:sequence>
                <xs:attribute name="uri" type="Username" use="required" />
                <xs:attribute name="guid" type="xs:string" use="required" />
                <xs:attribute name="type" type="xs:nonNegativeInteger" use="required" />
                <xs:attribute name="uname" type="xs:string" use="optional" />
                <xs:attribute name="email" type="xs:string" use="optional" />
                <xs:attribute name="disabled" type="xs:boolean" use="required" />
                <xs:attribute name="dispname" type="xs:string" use="optional" />
                <xs:attribute name="company" type="xs:string" use="optional" />
                <xs:attribute name="path" type="xs:string" use="optional" />
                <xs:attribute name="chperms" type="xs:integer" use="optional" />
```

```
                <xs:attribute name="id" type="xs:integer" use="optional" />
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <!-- Group References:  The Group Stub and the Group Information Block -->
    <xs:complexType name="GroupInformationDataBlock">
        <xs:complexContent>
            <xs:extension base="AuditableInformationDataBlock">
                <xs:sequence>
                    <xs:element name="from" type="From" minOccurs="0" maxOccurs="1" />
                    <xs:element name="perms" type="UserPermissionDataBlock" minOccurs="0"
maxOccurs="1" />
                </xs:sequence>
                <xs:attribute name="guid" type="xs:string" use="required" />
                <xs:attribute name="type" type="xs:nonNegativeInteger" use="optional" />
                <xs:attribute name="name" type="xs:string" use="optional" />
                <xs:attribute name="path" type="xs:string" use="optional" />
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <!-- Used in GIB and UIB for indicating where the particular user or group was specified
in the node hierarchy. -->
    <xs:complexType name="From">
        <xs:simpleContent>
            <xs:extension base="xs:anyURI">
                <xs:attribute name="name" type="xs:string" use="required" />
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
    <!-- Preference References:  The Preference Data Block -->
    <xs:complexType name="PreferenceDataBlock">
        <xs:attribute name="label" type="xs:string" use="required" />
        <xs:attribute name="seqid" type="xs:positiveInteger" use="required" />
        <xs:attribute name="createdefault" type="xs:boolean" use="required" />
        <xs:attribute name="content" type="xs:string" use="optional" />
    </xs:complexType>
    <!-- Backchat References:  Queries, Searches, and Returns -->
    <xs:complexType name="BcQueryDataBlock">
        <xs:choice>
            <xs:element name="last" type="CountSpecifier" minOccurs="0" maxOccurs="1" />
            <xs:element name="msgid" type="BcQueryMsgID" minOccurs="0" maxOccurs="1" />
        </xs:choice>
    </xs:complexType>
    <xs:complexType name="BcSearchDataBlock">
        <xs:sequence>
            <xs:element name="limit" type="CountSpecifier" minOccurs="0" maxOccurs="1" />
            <xs:element name="text" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:simpleContent>
                        <xs:extension base="xs:string">
                            <xs:attribute name="mt" type="SearchMatchTypeEnum" use="required"
/>
                        </xs:extension>
                    </xs:simpleContent>
                </xs:complexType>
            </xs:element>
            <xs:element name="msgid" type="xs:normalizedString" minOccurs="0" maxOccurs="1"
/>
```

```
            <xs:element name="matchcase" type="xs:boolean" nillable="false" minOccurs="1"
maxOccurs="1" />
            <xs:element name="searchbkwds" type="xs:boolean" nillable="false" minOccurs="1"
maxOccurs="1" />
            <xs:element name="sortbkwds" type="xs:boolean" nillable="false" minOccurs="1"
maxOccurs="1" />
            <xs:element name="date" type="DateRangeSpecifier" minOccurs="0" maxOccurs="1" />
            <xs:element name="uib" type="UserInformationDataBlock" minOccurs="0"
maxOccurs="unbounded" />
            <xs:element name="cib" type="ChannelInformationDataBlock" minOccurs="0"
maxOccurs="unbounded" />
        </xs:sequence>
        <xs:attribute name="cmp" type="LogicOperatorEnum" use="required" />
    </xs:complexType>
    <xs:complexType name="ResultCountDataBlock">
        <xs:simpleContent>
            <xs:extension base="EmptyEnumeration">
                <xs:attribute name="value" type="xs:positiveInteger" use="required" />
                <xs:attribute name="over" type="xs:boolean" use="required" />
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
    <xs:complexType name="DeleteChatDataBlock">
      <xs:sequence>
        <xs:element name="bcs" type="BcSearchDataBlock" minOccurs="1"/>
        <xs:element name="chat" type="xs:string" minOccurs="0"/>
        <xs:element name="rtf" type="xs:string" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  <xs:complexType name="PurgeChannelDataBlock">
    <xs:sequence>
      <xs:element name="cib" type="ChannelInformationDataBlock" minOccurs="1" />
    </xs:sequence>
    <xs:attribute name="uptodate" type="Iso8601TimeString" use="required" />
    <xs:attribute name="nocommit" type="xs:boolean" use="required" />
  </xs:complexType>
  <xs:complexType name="PurgeChannelSizesDataBlock">
    <xs:sequence>
      <xs:element name="purged" type="CountAndSizeSpecifier" minOccurs="1" />
      <xs:element name="left" type="CountAndSizeSpecifier" minOccurs="1" />
    </xs:sequence>
  </xs:complexType>
  <!-- ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->
    <xs:complexType name="GroupChatDataBlock">
        <xs:complexContent>
            <xs:extension base="XccosPrimitive">
                <xs:sequence>
                    <xs:element name="originatingMessageId" type="XccosMessageIdentifier"
minOccurs="0" />
                    <xs:element name="chat" type="xs:string" />
                    <xs:element name="rtf" type="xs:string" minOccurs="0"/>
                </xs:sequence>
                <xs:attribute name="chanUri" type="xs:anyURI" use="required" />
                <xs:attribute name="author" type="Username" use="required" />
                <xs:attribute name="authdisp" type="xs:string" use="required" />
                <xs:attribute name="alert" type="xs:boolean" use="required" />
                <xs:attribute name="chatId" type="xs:long" use="required" />
                <xs:attribute name="ts" type="Iso8601TimeString" use="required" />
            </xs:extension>
```

```
        </xs:complexContent>
    </xs:complexType>
    <xs:complexType name="BcQueryMsgID">
        <xs:simpleContent>
            <xs:extension base="EmptyEnumeration">
                <xs:attribute name="id" type="xs:positiveInteger" use="required" />
                <xs:attribute name="cnt" type="xs:positiveInteger" use="required" />
                <xs:attribute name="pre" type="xs:positiveInteger" use="optional" />
                <xs:attribute name="post" type="xs:positiveInteger" use="optional" />
                <xs:attribute name="jump" type="xs:boolean" use="optional" />
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
    <xs:simpleType name="AceVerbEnum">
        <xs:restriction base="xs:string">
            <xs:enumeration value="A" />
            <xs:enumeration value="R" />
            <xs:enumeration value="X" />
        </xs:restriction>
    </xs:simpleType>
    <xs:complexType name="Ace">
        <xs:sequence>
            <xs:element name="uib" type="UserInformationDataBlock" minOccurs="0"
maxOccurs="1" />
            <xs:element name="gib" type="GroupInformationDataBlock" minOccurs="0"
maxOccurs="1" />
        </xs:sequence>
        <xs:attribute name="verb" type="AceVerbEnum" use="required" />
    </xs:complexType>
    <xs:complexType name="RoleList">
        <xs:sequence>
            <xs:element name="prins" type="Ace" minOccurs="0" maxOccurs="unbounded" />
            <xs:element name="uib" type="UserInformationDataBlock" minOccurs="0"
maxOccurs="unbounded" />
            <xs:element name="gib" type="GroupInformationDataBlock" minOccurs="0"
maxOccurs="unbounded" />
        </xs:sequence>
    </xs:complexType>
    <!-- Base Type Definitions
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ -->
    <xs:complexType name="CountSpecifier">
        <xs:simpleContent>
            <xs:extension base="EmptyEnumeration">
                <xs:attribute name="cnt" type="xs:positiveInteger" use="required" />
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
    <xs:complexType name="DateRangeSpecifier">
        <xs:simpleContent>
            <xs:extension base="EmptyEnumeration">
                <xs:attribute name="from" type="Iso8601TimeString" use="required" />
                <xs:attribute name="to" type="Iso8601TimeString" use="optional" />
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
    <xs:complexType name="CountAndSizeSpecifier">
        <xs:simpleContent>
            <xs:extension base="EmptyEnumeration">
                <xs:attribute name="cnt" type="xs:nonNegativeInteger" use="required" />
```

```
            <xs:attribute name="sizemb" type="xs:nonNegativeInteger" use="required" />
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
<xs:simpleType name="EmptyEnumeration">
    <xs:restriction base="xs:string">
        <xs:enumeration value="" />
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name="ErrorCode">
    <xs:restriction base="xs:positiveInteger">
        <xs:whiteSpace value="collapse" />
        <xs:minInclusive value="100" />
        <xs:maxExclusive value="700" />
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name="Iso8601TimeString">
    <xs:restriction base="xs:string">
        <xs:whiteSpace value="collapse" />
        <xs:pattern value="[0-9]{8}T[0-9]{6}Z?" />
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name="LogicOperatorEnum">
    <xs:restriction base="xs:string">
        <xs:enumeration value="AND" />
        <xs:enumeration value="OR" />
    </xs:restriction>
</xs:simpleType>
<xs:complexType name="ResponseBlock">
    <xs:simpleContent>
        <xs:extension base="xs:normalizedString">
            <xs:attribute name="code" type="ErrorCode" use="required" />
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
<xs:simpleType name="SearchMatchTypeEnum">
    <xs:restriction base="xs:string">
        <xs:enumeration value="PP" />
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name="Username">
    <xs:restriction base="xs:normalizedString">
        <xs:whiteSpace value="collapse" />
    </xs:restriction>
</xs:simpleType>
<xs:complexType name="Field">
    <xs:simpleContent>
        <xs:extension base="xs:normalizedString">
            <xs:attribute name="id" type="xs:anyURI" use="required" />
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
<xs:complexType name="InfoField">
    <xs:complexContent>
        <xs:extension base="Field">
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
<xs:complexType name="PropertyField">
```

```
        <xs:complexContent>
            <xs:extension base="Field">
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
</xs:schema>
```

7 Appendix B: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include released service packs:

- Microsoft Lync 2013

Exceptions, if any, are noted below. If a service pack or Quick Fix Engineering (QFE) number appears with the product version, behavior changed in that service pack or QFE. The new behavior also applies to subsequent service packs of the product unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

Unless otherwise specified, any statement of optional behavior in this specification that is prescribed using the terms SHOULD or SHOULD NOT implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that the product does not follow the prescription.

8 Change Tracking

No table of changes is available. The document is either new or has had no changes since its last release.

9 Index