Client-Server Protocol Session and Transactions

Approved Version 1.3 – 23 Jan 2007

Open Mobile Alliance

OMA-TS-IMPS_CSP-V1_3-20070123-A

Contents

Figures

Tables

1. Scope

The present specification introduces the definition of the client/server protocol (CSP) for the IMPS 1.3 enabler.

CSP is a session-oriented protocol, as explained in Section 5. An abstract level description of all the CSP Transactions is provided form Sections 6 thru 10. The description starts from general primitives and continues through the feature-specific ones. The IMPS enabler consists of four primary functional features:

- Presence
- Instant Messaging
- Groups
- Shared Content

The above features are described in [Arch].

Section 11 describes the Status or Error Codes that are used in response primitive of every IMPS transaction.

Appendix B lists the Static Conformance Requirements for CSP, which specifies the Minimum Interoperability requirements as well as the optional features that allow enhancing the IMPS service.

2. References

2.1 Normative References

[CSP DataType]	"Client-Server Protocol Data Types", Version 1.3, Open Mobile Alliance™, OMA-TS-IMPS_CSP_Data_Types-V1_3, URL: http://www.openmobilealliance.org
[CSP Trans]	"Client-Server Protocol Transport Bindings", Version 1.3, Open Mobile Alliance™, OMA-TS-IMPS_CSP_Transport-V1_3, URL: http://www.openmobilealliance.org
[E.164]	ITU-T Recommendation E.164 (02/05) The international public telecommunication numbering plan. URL: http://www.itu.int/search/searchredirect.asp?recommendation.asp?type=items&lang=E&parent=T-REC-E.164-200502-I
[FIPS180-2]	"Secure hash standard", August 1 2002. URL: http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf
[IANA]	Character sets registered at IANA (MIBenum assignments), URL: http://www.iana.org/assignments/character-sets
[IOPPROC]	"OMA Interoperability Policy and Process", Version 1.1, Open Mobile Alliance™, OMA-IOP-Process-V1_1, ´ URL: http://www.openmobilealliance.org
[ISO3166-1]	ISO 3166-1: Codes for the Representation of Names of Countries and their Subdivisions – Part 1: Country Codes, 1997. URL: http://www.iso.ch/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=24591
[PA]	"Presence Attributes", Version 1.3, Open Mobile Alliance™, OMA-TS-IMPS_PA-V1_3, URL: http://www.openmobilealliance.org
[PRIVACY]	"Privacy requirements for mobile services", Version 1.0, Open Mobile Alliance™, OMA-RD_Privacy-V_1_0, URL: http://www.openmobilealliance.org
[RFC1320]	"The MD4 Message-Digest Algorithm", April 1992. URL: http://www.ietf.org/rfc/rfc1320.txt?number=1320
[RFC1321]	"The MD5 Message-Digest Algorithm", April 1992. URL: http://www.ietf.org/rfc/rfc1321.txt?number=1321
[RFC2045]	"Multipurpose Internet Mail Extensions (MIME) Part one: Format of Internet Message Bodies". Section 6.8 "Base64 Content-Transfer-Encoding". URL: http://www.ietf.org/rfc/rfc2045.txt?number=2045
[RFC2046]	"MIME (Multipurpose Internet Mail Extensions) Part Two: Media Types". Borenstein N., and N. Freed. November 1996. URL: http://www.ietf.org/rfc/rfc2046.txt?number=2046
[RFC2119]	"Keywords for using RFCs to Indicate Requirements levels", Bradner, S. URL: http://www.ietf.org/rfc/rfc2119.txt?number=2119
[RFC2234]	Augmented BNF for Syntax Specifications: ABNF. URL: http://www.ietf.org/rfc/rfc2234.txt?number=2234
[RFC2396]	Uniform Resource Identifiers (URI): Generic Syntax. URL: http://www.ietf.org/rfc/rfc2396.txt?number=2396
[RFC2822]	"Internet Message Format", April 2001. URL: http://www.ietf.org/rfc/rfc2822.txt?number=2822
[TS23040]	"3rd Generation Partnership Project; Technical Specification Group Terminals; Technical Realization of the Short Message Service (SMS)", (Release 5), 3GPP TS 23.040 v5.4.0, June 2002, URL: ftp://ftp.3gpp.org/Specs/archive/23_series/23.040/23040-540.zip
[TS23140]	"3rd Generation Partnership Project; Technical Specification Group Terminals; Multimedia Messaging Service (MMS); Functional Description; Stage 2", (Release 5), 3GPP TS 23.140 v5.4.0", September 2002, URL: ftp://ftp.3gpp.org/specs/archive/23_series/23.140/23140-540.zip
[VCAL10]	"vCalendar - The Electronic Calendaring and Scheduling Format", version 1.0, The Internet Mail Consortium (IMC), September 18, 1996, URL: http://www.imc.org/pdi/vcal-10.doc

[VCARD21]	"vCard - The Electronic Business Card", version 2.1,The Internet Mail Consortium (IMC), September 18, 1996, URL: http://www.imc.org/pdi/vcard-21.doc
[WAPMMS]	"Wireless Application Protocol - MMS Encapsulation Protocol, Version 05-Jan-2002", Open Mobile Alliance™, WAP-209-MMSEncapsulation, URL: http://www.openmobilealliance.org

2.2 Informative References

[Arch]	"IMPS Architecture", Version 1.3, Open Mobile Alliance™, OMA-AD-IMPS-V1_3, URL: http://www.openmobilealliance.org
[CSP PTS]	"Client-Server Protocol Plain Text Syntax", Version 1.3, Open Mobile Alliance™, OMA-TS-IMPS_CSP_PTS-V1_3, URL: http://www.openmobilealliance.org
[CSP WBXML]	"Client-Server Protocol Binary XML Definition and Examples", Version 1.3, Open Mobile Alliance™, OMA-TS-IMPS_CSP_WBXML-V1_3, URL: http://www.openmobilealliance.org
[CSP XMLS]	"Client-Server Protocol XML Syntax", Version 1.3, Open Mobile Alliance™, OMA-TS-IMPS_CSP_XMLS, URL: http://www.openmobilealliance.org
[PA XMLS]	"Presence Attribute XML Syntax", Version 1.3, Open Mobile Alliance™, OMA-TS-IMPS_PA_XMLS-V1_3, URL: http://www.openmobilealliance.org
[SSP]	"Server-Server Protocol Semantics", Version 1.3, Open Mobile Alliance™, OMA-TS-IMPS_SSP-V1_3, URL: http://www.openmobilealliance.org
[SSP Syntax]	"Server-Server Protocol XML Syntax", Version 1.3, Open Mobile Alliance™, OMA-TS-IMPS_SSP_XMLS-V1_3, URL: http://www.openmobilealliance.org
[SSP Trans]	"Server-Server Protocol Transport Binding", Version 1.3, Open Mobile Alliance™, OMA-TS-IMPS_SSP_Transport-V1_3, URL: http://www.openmobilealliance.org

3. Terminology and Conventions

3.1 Conventions

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in [RFC2119].

All sections and appendixes, except “Scope” and “Introduction”, are normative, unless they are explicitly indicated to be informative.

3.2 Definitions

Alias	The name a user suggests others to use as NickName. Part of the User Presence – available only those who are authorized to retrieve it.
Application	An implementation of a related set of functions, often enabling one or more services. It may consist of software and/or hardware elements.
Auto login	A user setting within the device that allows the IMPS client to log into the IMPS service automatically without user action (e.g. selecting a "login" function or starting an *application*). The login sequence may be triggered at the client's discretion in a variety of ways (e.g. at power-on, when the service becomes available after an outage due to coverage, phone calls, use of other services etc.)
Client	A device, user agent, or other entity that acts as the receiver of a service. The IMPS protocol designates the IMPS client to be an entity that is establishing and maintaining the IMPS session.
Client-originated content delivery transaction	The client-originated content delivery transactions include all of those transactions that are explicitly initiated from client side disregarding the fact that content is delivered from the client to the server, or from the server to the client, such as: sending an instant message, retrieving an instant message that has been notified to the client earlier (also referred to as “Notify/Get”), requesting map of a user, getting presence information of a user, updating presence information, etc.
Contact	The representation of a single user in the IMPS technology – consist of a User-ID and an optional NickName.
ContactList	A collection of Contacts that resides on the server, and essential part of presence authoring functions. It is possible to address IMs using the ContactListID as well.
Device	Equipment – including the hardware and the operating system software that is necessary for the operation of the hardware – that provides the functions necessary for the operation of the access protocols by the user. Device types may include (but are not limited to): mobile phones (GSM, CDMA, 3GSM, etc.), data-only terminals, PDAs, laptop computers, desktop computers, PCMCIA cards for data communication, unattended data-only Devices (e.g., vending machines), and smart cards.
End-to-end message	An instant message or an invitation that is sent from one client to another client via the server(s).
Friendly Name	A name that the user suggests for the general public to use as a nickname and is available to all users at all times.
Inbox	A local repository on the client device where the incoming messages are stored.
NickName	A name that is used internally in a client device to hide the User-ID of contacts from the end-user. It is not possible to address other users using the NickName.
Inband transaction	Transaction in the context of the session. The transaction carries an active Session-ID.
Offline Message Notification	A notification used to inform a logged-out recipient about an Instant Message sent to his User-ID and received on the IMPS server from his home domain.
Outband transaction	Transaction outside of a session context. The transaction does not carry an active Session-ID.
Private Group Conversation	A temporary chat group that is the result of extending a one-to-one messaging to many-to-many messaging.
Phonebook	A local database in the terminal containing phone numbers, names, etc.
Private profile	A collection of information, which is available for the service provider only, and there is no way to retrieve it directly by any user. Users can perform searches based on private profile if the service provider allows it. The information is typically collected when the user buys his/her IMPS subscription.
Public Profile	A user’s profile information that is available to the public – it is maintained by the owner user and is searchable with some restrictions.

Publisher	The user that owns the presence information.
Registration	The action or process by which the IMPS service is activated for an individual, who generally is a subscriber of the mobile network operator. The Registration is logically subsequent to the user provisioning onto the IMPS system, which does not exclude that provisioning and registration are chronologically adjacent. If the Registration process is triggered seamlessly – i.e. without User intervention – when the Client makes the first login attempt onto the Server, then it is called Automatic Registration.
Registered User	A user for whom the IMPS service has been activated via Registration.
ScreenName	A combination of a name a user chooses in a group session, and the Group-ID itself. The user MAY have different ScreenNames for different occasions as well as on different groups. The ScreenName is always connected to a group.
Server-originated content delivery transaction	The server-originated content delivery transactions include all of those transactions that are initiated from server side without explicit request from the client and deliver content from the server to the client, such as: delivering an instant message directly (also referred to as “Push”), sending presence notification of subscribed presence attributes, etc.
Session	The IMPS session is a framework in which the IMPS services are provided to the IMPS client. The IMPS session is established using the ‘Login’ transaction and it is terminated with a ‘Logout’ or ‘Disconnect’ transaction.
Subscriber	A subscriber is a user who requests access to another user’s (a publisher) presence information.
System Message	A special type of message sent by the IMPS system for different purposes (e.g. advice of charge, service notifications, advertisements, instructions, etc). System Messages MAY contain a list of possible options and require actions or response from the user.
Watcher	A watcher is user who is either a current or former subscriber of another user’s presence information, or who did not subscribe but used the Get Presence transaction to obtain another user’s presence information.
Watcher history	The list of watchers and their status that has been collected by the server over some time period.

3.3 Abbreviations

ABNF	Augmented BNF, see [RFC2234].
AoC	Advise of charge
CSP	Client-Server Protocol
GRSE	Group Service Element
IM	Instant Message
IMPS	Instant Messaging and Presence Service
IMSE	Instant Messaging Service Element
ISDN	Integrated Service Digital Network
MDN	Mobile Directory Number
MMS	Multimedia Message Service
MSISDN	Mobile Station International ISDN Number
OMA	Open Mobile Alliance
PRSE	Presence Service Element
SAP	Service Access Points
SMS	Short Message Service
SSP	Server to Server Protocol
T&C	Terms and conditions
WV	Wireless Village

4. Introduction

This document starts with a description of the fundamental concept of Session in IMPS. Then, it continues describing the transactions and the information elements that permit CSP to provide IMPS services and to support interoperability between different client and server implementations.

5. IMPS Session

5.1 Session Management

The Instant Messaging and Presence Service session is a framework in which the IMPS services are provided to the IMPS client. The IMPS session is transport-independent. If the transport connection is broken, the client MAY reconnect the transport connection and MUST be able to continue the session (see [CSP Trans]). The client device MAY be even turned on and off and the session MAY still be continued.

In order to establish a session, the client MUST log into a SAP using one of the available CSP protocol versions, transport mechanisms (see [CSP Trans]) and a CSP syntax definition ([CSP XMLS], [CSP WBXML] or [CSP PTS]) that is supported over the selected transport. The client and the server MUST maintain the initial protocol version, and syntax definition that was used during the login transaction throughout the whole session – until logout or disconnect. The server MAY deny access if the protocol version, the transport, or the syntax definition is not supported.

The IMPS session is established when the client logs in and terminated when either the client logs out or the SAP decides to disconnect the session. The session is identified by a *Session-ID*. In addition, the IMPS client MUST provide at login phase a client-unique *session cookie* for the session, which is used by the server to trigger communications in some cases.

The client and the server MUST use the same version of the CSP protocol that was used in the login primitive in every transaction throughout the whole session. The versions supported on the server side MAY be retrieved any time – it is independent from IMPS sessions. Please refer to chapter 5.2 Version Management and 6.3 Version Discovery Transaction for further information of this mechanism.

If the server for some reason does not accept the IMPS version that the client used in LoginRequest, the client MAY initiate version discovery transaction to retrieve the list of supported versions from the server, and select an IMPS version (to be used in the login phase again) that is supported on both client and server side.

The authentication of the user is done at the login phase. The authentication is, in general, considered to be valid throughout the session. However, the server MAY, at any time disconnect the session and request the client to re-login. In this case, the client provides the old Session-ID. After valid authentication, the IMPS server MAY accept to *re-establish* the old session.

During the IMPS session, both the client and SAP MUST maintain *session context*. The session context contains dynamic information of the services the client is currently using. The IMPS specifications do not explicitly define what the actual state of services the session context contains, but it assumes that the negotiated services, capabilities and subscriptions are valid throughout the session.

The session context in the IMPS server and the IMPS client are tied together by the services the user is currently using (subscribed presence attributes, joined groups, authorized presence attributes, etc). The IMPS server and IMPS client MAY assume that the link between the contexts is valid throughout the IMPS session as well as in a re-established session.

Associated with each session is an auto logout timer value. The client MAY request a particular value or request an infinite time-to-live time. The server can set any timer value and the client MUST obey that. Server implementers SHOULD NOT however select a lower time-to-live value then the one requested by the client.

Normally all transactions are performed within an established session. The ‘Type’ of all transactions that are sent within a session MUST be ‘Inband’. There might be special cases when a client desires to perform a transaction without having an established session. The ‘Type’ of all transactions that are sent outside of a session MUST be ‘Outband’. Inband transactions MUST include a Session-ID. Outband transactions MUST NOT include Session-ID. To all transaction requests that include Session-ID the response MUST include the identical Session-ID.

5.1.1 Session context

The session context contains dynamic information of a session of the user. During the session, the session context MUST be maintained on both client and server. The session context MUST contain the following session information:

- Negotiated services: the services negotiated during the Service Negotiation transaction (see chapter 6.9).
- Negotiated client capabilities: client capabilities negotiated during the Client Capability Negotiation transaction (see chapter 6.9).

- Presence subscriptions: the current presence subscription(s) (see chapter 8.3.2).
- General notifications subscriptions: the current subscription(s) to general notifications (see chapter 7.2).

Each session MUST have 1 and only 1 session context.

The server MAY store the session context for a period of time after the session is terminated. The time period a session is stored on the server is a server-wide setting and can vary from 0 to infinite:

- 0: the session context is not stored
- n: integer number indicating the time period a session is stored in seconds after termination of the session
- Missing value (Infinite): the session context is stored until the user sets up another session with the same client.

It is RECOMMENDED not to store session for unlimited period of time because clients MAY change Client-ID and users MAY not use the same client ever again. The IMPS server stores the session context after termination to allow the users to restore the session (see chapter 5.1.2).

5.1.2 Re-establishing sessions

Clients MAY request re-establishment of a disconnected session and the associated *session context*. The reason code of the disconnection is not significant – the client MAY request re-establishment of sessions that have been terminated successfully or with some error.

A successfully restored session carries over all credentials from the disconnected session and the associated *session context*:

Session re-establishment is described in detail in [CSP Trans].

5.1.3 Support for multiple sessions

Starting from IMPS v1.3 protocol each server MUST support multiple sessions. It means that any user MAY establish at least two sessions, or more if the server implementation allows it. The server implementation MAY limit the maximum number of parallel sessions to an implementation-specific value. The server MUST inform the client of such limitations during client capability negotiation, see 6.9 Service and Capability Negotiation.

The parallel sessions MAY be established by the same device/client, however the Client-ID that is used to establish the new sessions MUST be unique in the scope of the user as described in 5.3.7 Client Addressing.

The *session context* of the parallel sessions MUST be completely independent of each other on both client and server sides and there MUST NOT be interference between the sessions. It is RECOMMENDED that clients subscribe all supported notification types of the General Notification Mechanism (see 7.2 General Notification Transactions) to avoid the client getting out of context.

When a user employs multiple sessions from different clients that are using presence feature, the server MUST merge all presence information of the user/client into a single presence document as described in Client Status and User Status in [PA].

5.1.4 Direct application-to-application communication

The IMPS protocol MAY be utilized for direct application-to-application communication. It means that *applications* MAY establish direct communication using invitations, while they MAY communicate directly with each other using instant messages to exchange information. The direct communication allows the use of proprietary extensions/mechanisms and instant message content between the relevant *application* instances.

In order to guarantee that the *application* does not interfere with other communications, the direct communication SHOULD be established.

When the desired *application* is online:

- Clients MAY discover the remote *applications* using the Application-ID field in the ClientInfo presence attribute, and find out the corresponding Client-ID. The client SHOULD send an invitation to the desired User-ID/Client-ID pair. The invitation type to be used should be 'AP', see 'AP' invitation type in 7.5 Invitations. The textual description in the invitation SHOULD include information for the user about the relevant *application*, however it MAY be missing since the online *application* MAY handle the invitation without user interaction. The direct communication is considered established when the invitation has been accepted, and the inviting client has received the invitation response.

When the desired *application* is offline:

- When the ClientInfo presence attribute is not available or the desired *application* was not found, clients MAY send an invitation to the desired User-ID/Application-ID pair with invitation type 'AP'. The textual description in the invitation SHOULD include proper instructions for the user about obtaining/installing the relevant *application*. The Invite-Content element in the invitation MAY contain the URI, which MUST refer to a location where the *application* is available for download. When the user accepts the invitation on the user interface, the client SHOULD NOT send the invitation response itself. Instead, it SHOULD download, install and start the relevant *application* (if necessary) and provide the invitation credentials to the started *application*. The started *application* MUST establish its own session, and it SHOULD respond to the invitation itself based on the credentials received from the client that started it. The direct communication is considered established when the invitation has been accepted, and the inviting client has received the invitation response.

When the direct communication has been established, all application-to-application communication SHOULD continue based on Client-ID of the relevant *application*, meaning that all invitations and instant messages SHOULD be sent using User-ID with Client-ID. *Applications* that involve more than two users SHOULD create and join their own groups instead of sending every instant message to all users.

When the direct communication is not established using the mechanism described above, the sending client MAY send instant messages to the recipients using User-ID with Application-ID, however this is NOT RECOMMENDED since the instant messages with proprietary content might interfere with the user's normal communications.

5.2 Version Management

Each set of the IMPS specifications defines a version of the IMPS Client-Server protocol. XML, and WBXML based bindings use specific namespaces to identify which version of the protocol is being used for each of the following specific purposes:

- 5.1 Session Management;
- 5.4 Transaction Management; and
- Presence Attributes as defined in [PA]

Sessions are established using matched sets of namespaces defined in a specific version of the IMPS specifications.

The Plain Text Syntax [CSP PTS] does not use namespaces to identify different versions of the IMPS specifications: it uses only two digits in the preamble. Thus the version discovery is very much simplified in Plain Text Syntax.

The client and the server MUST maintain the same IMPS version (and hence namespaces) that was used in the login transaction throughout the whole duration of the session.

The Version Discovery transaction provides a mechanism by which a client can discover the protocol versions implemented by a specific server. The mechanism is described in details in chapter 6.3 Version Discovery Transaction.

5.3 Addressing

5.3.1 Addressing introduction

The IMPS protocol uses addresses based on the URI format [RFC2396]. The addressable entities are:

- User
- Contact list (private and public)
- User group (private and public)
- Content (private and public)

The addressing model introduces a unique IMPS address space, as defined in 5.3.2 Generic address format, which MUST be supported by the IMPS system. Use of other address spaces e.g. “sip:”, MAY be used to interoperate with other systems, but their use and definition is out of the scope of IMPS specifications.

In addition to the user, the IMPS client the user is using MAY be addressed as well.

The sending client MUST NOT provide a Friendly Name in any transaction with the User-ID.

Every time the IMPS server sends the User-ID in a transaction and the Friendly Name is available, the IMPS server MUST include the Friendly Name with the User-ID. If the Friendly Name does not exist, the IMPS server MUST NOT include the Friendly Name.

5.3.2 Generic address format

The generic address syntax is based on URI [RFC2396]. The generic syntax for a WV-URI is defined using ABNF [RFC2234] as follows:

```
WV-URI = "wv:" ( User-part / Resource-part) [ Domain-part]
User-part = Mobile-Identity / Internet-Identity
Domain-part = "@" sub-domain *( "." sub-domain )
Mobile-Identity = [ "+" ] 1*DIGIT
Internet-Identity = ALPHA 1*extalpha
Resource-part = [ User-part ] "/" 1*extalpha
extalpha = alphanum / "!" / DQUOTE / "#" / "$" / "%" / "&" / "'" / "(" / ")" / "," / "-" / "." /
   ";" / ":" / "<" / "=" / ">" / "[" / "\" / "]" / "^" / "_" / "`" / "{" / "|" / "}" / "~"
alphanum = ALPHA / DIGIT
```

Where User-part refers to the identification of the IMPS user, Domain-part identifies the IMPS server domain, and Resource-part further identifies the referred public or private resource within the domain. The sub-domain is defined in [RFC2822]. ALPHA, DIGIT, and DQUOTE are defined in [RFC2234].

A WV-URI specifies either a user or a resource. A resource address MUST have the Resource-part but a user address MUST NOT. A resource is either private or public and that is specified by the address. The Resource-part of a private resource address MUST contain the User-part but a public resource address MUST NOT.

The Domain part is OPTIONAL. When it is not present, the address refers implicitly to the home domain, which is the domain that the client specified at login. If the user does not specify a domain at login, the server default domain will be used. Both the client and the server MUST support local and fully qualified addressing. In order to improve interoperability, the server MUST maintain the same addressing format within the scope of a transaction: if client requested using local address, the server MUST reply using local address. The server-initiated transactions MUST always contain fully qualified addresses.

A WV-URI MUST include the schema “wv:”.

The “*” and “?” characters MUST NOT be used in any addresses for addressing purposes. These characters are reserved for wildcard characters to be used in access control rules, see 9.3 Access Control Transactions.

The lengths of the schema and the optional domain parts MUST be included in the total address length. The total length MUST NOT exceed the maximum address length as specified in [CSP DataType].

The addresses are case insensitive.

5.3.3 Address encoding

As per URI [RFC2396], certain reserved characters MUST be escaped if they occur within the User-ID, Resource, or Domain portions of an IMPS address. This includes the characters “<”, “>”, “;”, “?”, “:”, “&”, “=”, “+”, “$” and “,”. For example, a valid IMPS address for the user “$mith” in the “server.com” domain is:

```
wv:%24mith@server.com
```

Certain characters are not allowed in the User-ID portion of IMPS addresses (see chapter 5.3.4 User addressing). This includes the characters “/”, “@”, “:”, “ “ and TAB. This restriction is independent of the encoding of a User-ID within an IMPS address. For example, this IMPS address is not permissible:

```
wv:john%40aol.com@server.com
```

This address is not permissible because after URI-decoding, the User-ID portion contains a forbidden character (“@”). If a server’s internal representation of a username permits the occurrence of forbidden characters, such characters MUST be double-escaped when they occur in a IMPS address, such that they do not occur un-escaped in the User-ID portion after URI-decoding, or they MUST be escaped via some other scheme that does not employ forbidden characters.

When calculating the length of an address, an escape sequence MUST be considered as the character represented.

5.3.4 User addressing

The IMPS protocol uses User-IDs to uniquely identify any IMPS User. The User-ID is syntactically equivalent to an e-mail address, and as such is subject to the same restrictions for character set, as described in “Standard for the Format of ARPA Internet Text Messages” [RFC2822]. The User-ID MUST be either a local User-ID or a fully qualified User-ID. The local User-ID MUST NOT include the Domain-part and this User-ID MUST refer to a user in the Home Domain. A fully qualified User-ID MUST include the Domain-part and reference a user in the home, or in another domain.

The User-ID MUST either refer to an Internet-type address or to a mobile number of the user. If the User-ID refers to the mobile number of the user, the User-Part MUST always start either with a digit or with a '+' sign. A User-Part referring to an Internet-type address MUST NOT start with a '+' sign or digit.

The syntax of the User-ID is defined as follows:

```
User-ID = "wv:" User-part [ "@" Domain-part ]
```

When the User-ID refers to the mobile number address, the User-ID preceded with '+' sign MUST refer to the international numbering in The International Public Telecommunication Numbering Plan [E.164]. Without '+' sign, it MUST refer to the national numbering in the [E.164].

Examples:

```
Local User-ID:  wv:yuriyt
                   wv:+1234567890
                   wv:4567890
Fully qualified User-ID: wv:Jon.Smith@imps.com
                      wv:+1234567890@imps.com
                      wv:4567890@imps.com
```

5.3.5 Contact List Addressing

The IMPS protocol uses ContactList-IDs to uniquely identify any contact list of any user. The ContactList-ID is based on the generic address syntax. The contact list MAY be a public contact list or a private contact list. The syntax is defined as follows:

```
ContactList-ID = "wv:" Resource-part [ "@" Domain-part ]
```

Examples for the ContactList-IDs are:

```
wv:john/colleagues@imps.com
wv:/managers
wv:john/friends
wv:/managers@imps.com
```

Only the owner of a private contact list MAY access, use and manage the private contact list. When a server encounters a request that refers to a contact list address that is private and it does not belong to the requesting user, the server MUST reply with an error to the request.

5.3.6 User Group

The IMPS protocol uses a Group-ID to uniquely identify a group. The Group-ID is based on the generic address syntax. The user group MUST be public user group or private user group. The syntax of the Group-ID is defined as follows:

```
Group-ID = "wv:" Resource-part [ "@" Domain-part ]
```

Examples of the Group-IDs are:

```
wv:john/mygroup@imps.com
ww:john/mygroup
wv:/technicalforum
wv:/technical_forum@imps.com
```

5.3.7 Client Addressing

An IMPS client MAY be a device or an *application*, depending on the context within the device itself. The IMPS protocol designates the *IMPS client* to be an entity that is establishing and maintaining the IMPS session. The client MUST identify itself using Client-ID during the login phase or whenever an outband transaction is performed.

The Client-ID is a unique identifier of the IMPS client that the user is currently using. The client generates the Client-ID. The server MUST verify the Client-ID and MUST NOT accept it when it is not unique in scope of the user. The Client-ID identifies the IMPS client that accesses the IMPS services. The Client-ID is intended to allow multiple accesses from the same user over the same or other devices.

The Client-ID MUST be an URI (as defined in [RFC2396]). Both the client and the server MUST support client addressing.

All clients are REQUIRED to establish separate sessions, and identify themselves with a unique Client-ID, thus third party *applications* MUST be addressed using unique Client-ID as well. However, in order to provide a framework in which the clients and third party *applications* are able identify each other, clients MAY additionally identify themselves using Application-ID during login.

5.3.8 Application-ID

The Application-ID identifies *applications*, which helps similar *applications* to locate each other, and allow direct application-to-application communication between them. The Application-ID MUST be a free-form string as described in [CSP DataType], however it is strongly RECOMMENDED that it includes a sub string that allows unique identification of the *application*. To identify an *application*, the Application-ID MUST be provided during login, and the Application-ID provided during login MAY be used in invitations and instant messages. The operator MAY block certain Application-IDs upon login. The user cannot manage the operator-specific access control rules. When Application-ID was provided during login and presence is supported, the server MUST publish the Application-ID in the ClientInfo presence attribute for the corresponding client; see [PA].

The Application-ID is related to the Client-ID: every Client-ID MAY have one and only one Application-ID assigned, however the same Application-ID MAY be assigned to several different Client-IDs if desired.

Examples of the Application-IDs are:

```
MyChessGamev1.1b123
MyMessagingApplicationv2.0
```

5.4 Transaction Management

An IMPS transaction is a basic communication mechanism between an IMPS client and an IMPS SAP. A transaction usually consists of a request and a response primitive. (Exceptions: disconnect, 4-way login.) The purpose of the transaction is to exchange data between the entities or request an operation: usually both within the same transaction. The transactions MAY originate from either IMPS client or IMPS SAP.

Initiator of the transaction MUST expect the Status primitive as the result of transaction even if it is not specified explicitly in the description of transaction. This behavior is used to notify the initiator about error(s) caused by the request. The only exception is the Disconnect request from server. Some response primitives carry a Result element that allows error reporting within the normal response primitive. The error management – including the exact usage of Status primitive and Result element – is described in 6.1 Status Primitive.

Due to the asynchronous nature of the SSP protocol the server MAY return informational status codes. The client MUST be prepared to accept one or more 1xx status codes prior to a regular response even if the client does not expect a 100 “Continue” status code. A user agent MUST ignore all unexpected 1xx status codes. None of the status codes in the 1xx category close transactions.

The initiating entity, the IMPS client or the SAP, allocates a transaction identifier that the responding entity returns in the response message. This links together the requesting message and response message, thus the transaction identifier MUST be identical in the request and the response. The originator of the transaction MUST maintain the uniqueness of the transaction identifiers in a particular direction (C->S or S->C) within a session.

The response to a request message SHOULD be received within 20 seconds from the initiation of the transaction. After that period, the requesting entity MAY *resend* the request message using the same transaction identifier. The responding entity SHOULD guarantee that the requested operation or data is carried out only once, even if multiple request messages with the same transaction identifier are received.

The transactions MUST be sequential until the capability negotiation is completed. Sequential means that one transaction MUST be complete (closed) before a next one is started (open). A transaction MUST be considered to be closed when a the final response primitive has been received, or a time-out waiting for a response primitive has occurred, or the underlying transport has been detected as "broken." After completing the capability negotiation the client MAY overcome the limitation of having sequential transactions by negotiating multiple open transactions with the SAP. See 'MultiTrans' and 'MultiTransPerMessage' capability in client capability negotiation in 6.9. After a transaction has been closed, the transaction identifier MUST be invalidated on both the client and server side. Invalidated transaction identifiers MUST NOT be re-used within the same session-context.

Multiple transactions mean that two or more parallel transactions are open.

All mandatory information elements MUST be present in the primitives. All conditional information elements MUST be either present or absent according to the relevant requirement. All optional information elements MUST be either present or absent according to the relevant usage context.

5.5 Routing

5.5.1 Client-server communication routing

The client and the server MUST perform all inband communications within the scope of an established session context, as described in 5.4 Transaction Management.

The client and the server MUST perform all óutband communications outside of the scope of any established session context, as described in 5.4 Transaction Management.

5.5.2 End-to-end message routing

This section lists the requirements for routing end-to-end messages that MUST be fulfilled to guarantee interoperability of various client and server implementations. Both the client and the server MUST support the end-to-end message routing requirements when invitation or instant messaging support has been agreed during service negotiation.

The Recipient element MUST be used to describe how the requests MUST be routed, while the Sender element MUST be used to describe where the responses SHOULD be sent. The invitation-related primitives MUST include the Recipient and Sender information elements directly, while instant messaging-related primitives MUST include Recipient and Sender elements in the Message-Info structure. This section refers to the Recipient and Sender elements only disregarding the fact that for instant messaging these elements are part of the Message-Info structure.

For privacy reasons, whenever an end-to-end message is requested to be sent to multiple recipients, the server MUST deliver the end-to-end message to every recipient only once without delivering the actual list of recipients. (E.g. a requested end-to-end message MUST NOT be received twice, and the recipients MUST NOT be revealed to each other.)

Before delivering the end-to-end message to the requested recipient(s), the server MUST verify the access control rules of the recipient versus the Recipient and Sender elements as described in chapter 9.3 Access Control Transactions.

When Client-ID or Application-ID was not specified in an end-to-end message with the User-ID, the server MUST NOT add any of these to the Sender/Recipient elements.

The client MAY request a specific validity period for each end-to-end message separately when desired. To request a specific validity period, the client MUST include the Validity element in the requested end-to-end message. The server MUST follow the requirements for validity of all end-to-end messages as described in 5.6.4.2 Validity.

5.5.2.1 The 'Recipient' element

The Recipient element MUST include those recipient(s) to whom the end-to-end message delivery is desired. Whenever an end-to-end message is sent, the client MUST NOT include Friendly Name with the User-ID(s) in the Recipient element, while the server SHOULD add Friendly Name to the User-ID(s) in the Recipient element as described in 5.3.1 Addressing introduction. The Recipient element MAY include:

- ContactList-ID,
- User-ID,

- User-ID with Client-ID,
- User-ID with Application-ID,
- Group-ID,
- ScreenName - not applicable to invitations,

however the exact semantics are transaction-specific: invitation-specific primitives MUST NOT identify recipients using ScreenName. The transaction-specific semantics are described in the sections related to end-to-end transactions.

When the recipient(s) are identified using ContactList-ID(s), the server MUST translate the ContactList-ID(s) to a list of User-IDs that are currently included in the requested Contact List(s). After this translation the server MUST replace the Recipient field in the generated end-to-end messages to User-ID(s) while making sure that no User-ID is included more than once. The server MUST route the generated end-to-end messages as if the original Recipient element was specified using User-ID(s).

When the Recipient element includes only User-ID without identifying a specific client/application, the server MUST perform routing according to the user's OnlineETEMHandling setting.

When the Recipient element includes User-ID and Client-ID and the requested client of user is online, the server MUST route the requested invitation exactly to the requested client(s) only. When the recipient client(s) are offline, the server MUST perform routing according to the offline clients' OfflineETEMHandling setting.

When the Recipient element includes User-ID and Application-ID, and

- the user has suitable client(s) online, server MUST route the end-to-end message to all suitable and online clients that registered the specific Application-ID during their login.
- the user does not have any suitable/online clients, and
    - the server offers offline storage, the server MUST store the end-to-end message for later deliver. The server MUST attempt to deliver the end-to-end message when there is an online client for the user that supports the transactions that are essential to deliver the end-to-end message.
    - the server does not offer offline storage, the server MUST reject the requested transaction.

When the Recipient element includes Group-ID in the Message-Info structure of an instant messaging-related primitive, the server SHOULD route the requested instant message to the group indicated with Group-ID. The server MUST reject the instant message when:

- the screen name in the Sender element is not available within the related group,
- the screen name in the Sender element does not belong to the requesting user within the related group,
- the requesting user is not joined to the group when the instant message is sent.

When the Recipient element includes Group-ID in an invitation-related primitive, the server SHOULD route the requested invitation transaction to all administrator/moderators of the group indicated with Group-ID. When some administrators or moderators of the group are offline, the server MUST perform routing according to the offline administrators' and moderators' client(s) OfflineETEMHandling setting. The server MUST reject the requested invitation transaction when the requesting user is included in the reject list of the requested group.

When the Recipient element includes ScreenName, the server SHOULD route the requested IM to the indicated screen name in the related group only. The server MUST reject the IM when:

- the screen name in the Sender element is not available within the related group,
- the screen name in the Sender element does not belong to the requesting user within the related group,
- the requesting user is not joined to the group when the instant message is sent,
- private messaging is disabled within the related group,
- the recipient has disabled private messaging for himself/herself within the related group.

5.5.2.2 The 'Sender' element

The Sender element is not used for routing; it is used to indicate where the response to an end-to-end message SHOULD be sent upon reply. Whenever an end-to-end message is sent, the client MUST NOT include Friendly Name with the

User-ID in the Sender element, while the server SHOULD add Friendly Name to the User-ID(s) in the Sender element as described in 5.3.1 Addressing introduction. When the user replies to an end-to-end message, the responding client MUST copy the Sender element from the received end-to-end message to the Recipient element in the response end-to-end message, while the responding client SHOULD copy the Recipient element from the received end-to-end message to the Sender element in the response end-to-end message however the Sender element MAY be altered by adding Client-ID. After the client performed the required adjustments to the Recipient and Sender elements upon reply, the server MUST route the end-to-end message based on the Recipient element as described in 5.5.2 End-to-end message routing. The Sender element MUST include either one of the followings:

- User-ID,
- User-ID with Client-ID,
- User-ID with Application-ID,
- Group-ID,
- ScreenName - not applicable to invitations.

For invitations, see 7.5 Invitations and 7.6 Canceling Invitations.

For instant messages, see message context in 9.2 Message-Info Structure.

5.6 Content for Presence/Instant Messaging

5.6.1 Content type support

The IMPS protocol supports delivering any content. This section lists the requirements for all client and server implementations that support instant messaging and/or presence that MUST be fulfilled to guarantee interoperability of various client and server implementations. These are mandatory requirements for any client and server implementations and thus not negotiated.

All content MUST have a proper content type indication in the corresponding information element, with the exception of “text/plain”. “text/plain; charset=UTF-8” content type MUST be supported by all client and server implementations, and it is the default content type. Default content type means that when the content type indication is missing, the content MUST be handled assuming “text/plain” content type.

The suggested content types are:

- vCard 2.1 as defined in [VCARD21],
- vCalendar 1.0 as defined in [VCAL10],
- MMS, “application/vnd.wap.mms-message” as defined in [WAPMMS].
- EMS, “application/vnd.3gpp.sms” as defined in [TS23040] and “application/x-sms” as defined in [TS23140]

The suggested content types, while not mandatory, are content types that are highly RECOMMENDED to further maximize the interoperability of the various client and server implementations.

Note that the MMS standardization is an ongoing effort in 3GPP and OMA forums. The RECOMMENDED MMS content-type “application/vnd.wap.mms-message” MUST be consistent with the standardization effort, and support the standard.

The message MAY carry other types of content. In this case, the “Content-Type” MUST be consistent with the MIME types that are standardized in IETF [RFC2045], [RFC2046] or Open Mobile Naming Authority (OMNA).

While some content types are RECOMMENDED to facilitate interoperability, the server MUST recognize the capabilities of clients through the client capability negotiation to ensure the interoperability of different types of content between clients, and deliver the content as described in 5.6.4 Content delivery. The server MAY also offer content transcoding as described in 5.6.2 Content transcoding.

The IMPS protocol supports Plain Text [CSP PTS], XML [CSP XMLS], and Binary XML [CSP WBXML] syntax. The Plain Text Syntax supports only “text/plain”, while the XML and Binary XML syntaxes support any media type. XML documents MUST NOT contain rich content directly; all rich content MUST be transfer-encoded using any of the transfer-encoding methods that have been agreed during client capability negotiation in the AcceptedTransferEncoding setting. See AcceptedTransferEncoding setting in 6.9 Service and Capability Negotiation for the agreement and the available list of transfer-encoding methods.

Instant messages MAY refer to a content using URI instead of including the content directly, however the content management is not in the scope of the IMPS specifications. The client SHOULD provide a mechanism to retrieve the content from the URI whenever feasible, but it is up to the client implementation to support the necessary protocol(s) and retrieval mechanism(s).

All clients SHOULD follow the content-related security guidelines described in 5.6.1.1 Security guidelines.

5.6.1.1 Security guidelines

The security guidelines do not apply explicitly to any specific features/transactions; it is a generic guideline to all features/transactions.

The client SHOULD perform implementation-specific tests on the directly included or URI-referred content before activation (digital rights, viruses, mal-ware, spy-ware, etc). Since these tests are implementation-specific, these are not in the scope of the IMPS specification.

Whenever a client identifies content as potential security threat, the client MUST notify the user and take a protective action, such as: discarding/isolating the content. The client MAY also notify the server/service provider using a proprietary solution to prevent the possible future use of the hazardous content.

5.6.2 Content transcoding

The IMPS server MAY offer content transcoding when the content type to be delivered to a client is not supported by the client for some reason (content type, content size, content policy limitation). Content transcoding is an OPTIONAL feature for the servers. The content transcoding algorithms and the supported content types are subject to the server implementation and as such not in the scope of the IMPS specifications. The IMPS specification provides the means to support content transcoding by conveying the supported content types (either truly supported or provided using content transcoding) from the server to the client in the AcceptedContentType element during client capability negotiation – as if the server was supporting the content type requested by the client.

When the server performs content transcoding, the server

- MUST transcode the original content to another content type based on the following requirements:
  - the content type – including character set when applicable – to which the content will be transcoded MUST be supported by the client, and
  - the transcoded content MUST fit into the size limitation of the client without triggering any kind of policy limitations related to the selected content type.
- SHOULD guarantee that the quality of the transcoded content is comparable with the original content.

5.6.3 Content handling policies

The IMPS protocol supports different content handling policies. The policies are agreed on the basis of rich content types during client capability negotiation as described in 6.9 Service and Capability Negotiation. The description in this section assumes that the content type for which the policy applies has been agreed during the client capability negotiation. This section lists the requirements for all client and server implementations that support instant messaging and/or presence that MUST be fulfilled to guarantee interoperability of various client and server implementations. These are mandatory requirements for any client and server implementations the support instant messaging and/or presence, and thus not negotiated.

The IMPS specification defines the following content handling policies:

- No policy ('N') – the content can be delivered from/to the server without limitation on the content itself. Note that even though the content policy is 'N', the AcceptedPushLength and AcceptedPullLength capability settings might limit the content anyway. The server MUST deliver the particular rich content according to the requirements described in 5.6.4 Content delivery.
- Extra cost policy ('C') – the content can be delivered from/to the server within the limitation implied by the policy, however some extra charge will be added for the user when the content is delivered. When the rich content is oversized in a way that it falls into the 'C' policy, the server SHOULD NOT deliver the particular

content initially. It is RECOMMENDED that the service provider confirms the extra charges with the user using a System Message. When System Message has been sent to confirm the extra charges, and the user

- did not accept the extra cost for content delivery, and the rich content is part of a
    - presence attribute, the particular presence attribute MUST NOT be delivered to the client, even though the publisher has authorized the subscriber properly. A System Message MAY inform the user of such event, which MAY provide instructions for the user how to overcome the problem.
    - an instant message, a System Message MAY inform the user of such event. The user MAY reject or forward the instant message to another client based on the System Message.
- accepted the extra cost for content delivery, the server MUST deliver the particular rich content according to the requirements described in 5.6.4 Content delivery.

- Reject policy ('R') – when the content is larger than the limitation implied by the policy, the rich content will be automatically rejected. The server MAY send an informational System Message to the user describing the event, however such behavior is NOT RECOMMENDED as it might lead to a bad user experience. In any case, if the rich content is part of
    - an instant message, MUST reject the instant message automatically.
    - a presence attribute, the server MUST NOT deliver the particular presence attribute(s) that contain the rich content with 'R' policy.

To further enhance the user experience it is RECOMMENDED not to bother the user with System Messages every time the content policy restricts rich content delivery – the server should remember the user's decision based on the policy setting, content type and size, however remembering the decision might also be part of the System Message itself – these recommendations are all implementation-specific issues that are not in the scope of the IMPS specifications. See 7.1 System Message for possible System Message utilization scenarios.

5.6.4 Content delivery

The client MUST agree with the server on the supported content types and limitations when the session context is set up, as described in 6.9 Service and Capability Negotiation. This section describes the client/server behavior for content delivery. Content delivery is applicable only while the client is online. The requirements that are common to instant messaging, presence, and other features including the proprietary extension framework are listed below, while the additional sub-section lists the additional requirements that are either instant messaging specific, or describe the server behavior for validity of end-to-end messages.

Before any rich content is delivered to the client, the server MUST verify the capabilities of the client versus the credentials of the rich content by:

- evaluating the MIME type of the content, and
- evaluating the general size limitation that originates from the AcceptedPushLength (for server-originated content delivery transactions) or AcceptedPullLength (for client-originated content delivery transactions) capability settings, as described above in this section, and
- evaluating the rich content according to the content handling policy of the rich content when applicable as described in 5.6.3 Content handling policies.

When the server finds that the content size fits into the AcceptedPushLength and AcceptedPullLength capability settings (whichever is applicable in the context of the transaction), the server MUST continue with further evaluations, including the content handling policy. The server MUST NOT continue with the evaluation when the content size does not fit into the AcceptedPushLength or AcceptedPullLength capability settings (whichever is applicable in the context of the transaction), unless the client supports delivering the instant message using the other delivery method as described in 5.6.4.1 Instant messaging-specific delivery methods. When the server concludes that content delivery can be performed, the server MUST deliver the content without changes according the semantics defined in the relevant transaction. When the server concludes that content delivery cannot be performed, the server SHOULD queue up the instant message for later delivery, however the server MAY attempt a workaround as well:

- The server MAY send a System Message to the user informing him/her about the limitations and include instructions how the user MAY attempt to overcome the limitations. However, such behavior is NOT RECOMMENDED since the user might not have control over the capabilities/services offered by the client, and it might lead to a bad user experience.

- The server MAY attempt to transcode the content, and deliver the transcoded content instead, as described in 5.6.2 Content transcoding.

5.6.4.1 Instant messaging-specific delivery methods

This section lists the requirements for all client and server implementations that supporting instant messaging that MUST be fulfilled to guarantee interoperability of various client and server implementations. These are mandatory requirements for any client and server implementations supporting instant messaging, and thus not negotiated.

A client that supports instant messaging MUST support instant message delivery using either the “Push” or the “Notify/Get”, or both of these delivery methods, as described in 9 Instant Messaging Feature. The client MUST agree on the initial delivery method during client capability negotiation using the InitialDeliveryMethod setting, as described in 6.9 Service and Capability Negotiation. When both delivery methods are supported, the client MAY change the delivery method during the session using the Set Delivery Method Transaction, as described in 9.1.2 Set Delivery Method transaction. In any case, the delivery method will be either “Push” or “Notify/Get”.

The “Push” delivery method delivers an instant message from the server to the client without explicit request from the client, as described in 9.1.5 New Message Transactions. When the server is attempting to deliver an instant message and the currently selected delivery method is “Push”, and:

- the content type is “application/vnd.wap.mms-message” or the client does not support the rich content for some reason (content type, size or policy limitation as described in the general section), the server MUST attempt to use “Notify/Get” instead of “Push”. The server will be unable to deliver the instant message when the client did not negotiate support for “Notify/Get”, or when the size limitation for “Notify/Get” is not sufficient. When the delivery using “Notify/Get” cannot be performed, the server SHOULD queue up the instant message for later delivery. When the attempt to deliver the instant message using “Notify/Get” instead fails, the server MAY attempt a workaround:
    - The MAY attempt to transcode the content, as described in 5.6.2 Content transcoding, and attempt to deliver it using “Push”, or when “Push” fails, attempt to deliver it using “Notify/Get”.
    - If the client did not negotiate support for “Notify/Get”, the server MAY send a System Message to the user informing him/her about this, however such behavior is NOT RECOMMENDED since the user might not have control over the services offered by the client, and it might lead to a bad user experience.
- When the client supports the rich content without limitations, the server MUST deliver the instant message using the “Push” delivery method.

The “Notify/Get” method involves a notification from the server to the client, where the server informs the client about the arrival of a new instant message. The notification does not contain the message content itself, however it contains all other information about the instant message, as described in 9.2 Message-Info Structure. When the server is attempting to deliver an instant message and the currently selected delivery method is “Notify/Get”, the server MUST NOT send the content to the client directly, it MUST notify the client about the instant message instead as described in 9.1.6 Message Notification Transactions. The client MAY also retrieve information about the undelivered and stored instant messages on the server as described in 9.1.3 Get List of Messages Transactions. The client MUST retrieve, reject, or forward the instant message, as described in 9.1.8 Get Message Transactions, 9.1.4 Reject Message Transactions or 9.1.10 Forward message transaction. The client MUST NOT retrieve the instant message when it does not have the capabilities required to handle the instant message. When the client is not capable of handling the instant message, the behavior is up to the client, however it is RECOMMENDED to take the following course of actions:

- When the client is unable to retrieve the instant message with newly negotiated capabilities, it MAY ask the user to forward the instant message to another client that it capable of handling the content – this behavior MAY also be automated based on the ClientInfo presence attribute of the other clients.
    - When the instant message was not forwarded, it is RECOMMENDED that the client informs the user and either rejects the instant message or leaves it on the server until the instant message expires or it is retrieved from another client.

5.6.4.2 Validity

The client MAY request a specific validity period for each end-to-end message separately when desired. To request a specific validity period, the client MUST include the Validity element in the requested end-to-end message.

When the client does not include the Validity element in the requested end-to-end message, the client requests the maximum allowed time by the server. The maximum time is a server implementation specific issue, and as such not in the scope of the IMPS specifications. A server receiving an end-to-end message without the Validity element MAY add the Validity element to the end-to-end message specifying the maximum allowed time by the server that adds the Validity element. When the Validity element is missing from a received end-to-end message, the recipient's server MUST drop the end-to-end message after the implementation specific validity period has expired.

When there is a Validity element included in an end-to-end message, and the validity period of the end-to-end message:

- has expired, the end-to-end message delivery is not required, and the server(s) MUST drop the end-to-end message without notification.
- did not expire yet and the recipient's server
  - offers offline storage, the recipient's server MUST store the end-to-end message until the specified time period expires, even when the recipient goes offline without retrieving the end-to-end message.
  - does not offer offline storage and the recipient is
    - online, the server MUST attempt to deliver the end-to-end messages, however when delivery is not possible, the server MUST store the end-to-end message until the recipient goes offline or the specified time period expires.
    - offline, the server MUST drop the end-to-end message.

The recipient's server MAY additionally inform an offline recipient about various events taking place due to the recipient being offline, as described in 9.1.7 Offline Message Notification.

5.6.5 Emoticons

5.6.5.1 List of supported emoticons

The IMPS enabler allows a client to take advantage of emoticons. A client taking advantage of such feature will replace specific character sequences known as emoticons when an instant message is received and is displayed to the user, or when the user composes an instant message the emoticons are replaced with the specific character sequence(s) before the instant message is sent. The exact behaviour for clients is described in 5.6.5.2. This feature is an implementation choice, and as such is OPTIONAL for the client. The list of character sequences and emoticons are listed in Table 1. The table captures only a subset of emoticons that are widely used - client implementations MAY extend the list with additional character sequences. The first column in the table identifies the emoticons; the second column defines one or more character sequences for each emoticon so that different client implementations can associate the same meaning to a character sequence and ensure interoperability between them. The third column provides a description of possible graphical renditions. Note that the third column provides examples only and client implementations are not constrained to follow the examples. The exact images representing the individual emoticons are not in the scope of this enabler – it is a client implementation choice.

Emoticons	Character sequences	Examples describing graphical renditions
Happy, smile	:-) or :)	A happy or smiling face
Sad	:-(or :(	A sad face
Wink	;-) or ;) or ;o) or ;O)	A winking face
Big grin	:-D or :D or :oD or :-d or :d or :od or :Od or :OD	A big grin face
Confused	:-/ or :-\	A confused face
Blushing, embarrassed	:"-) or :") or :"> or :- or $ or :$	A blushing, embarrassed face
Stick-out tongue	:-P or :P or :oP or :-p or :p or :op or :OP or :Op	A stick-out tongue face
Kiss, red lips	**:-*** or **:***	A kissing face or red lips
Shocked, surprised	:-O or :-o or :o or :O	A shocked, Surprised face
Angry	:-@ or :@ or X-(or X(or x-(or x(or xo(or XO(	An angry face

Emoticons	Character sequences	Examples describing graphical renditions
Cool, sunglasses	B) or B-) or (H) or (h) or Bo) or BO)	A face with sunglasses
Worried	:-S or :S or :-s or :s or :oS	A worried face
Devilish	>:-) or >:) or >:o) or >:O)	A devilish face
Crying	:,-(or :,(or :’-(or :’(or :,o(or :’o(or :,O(or :’O(	A crying face
Laughing	:-)) or :)) or :o)) or :O))	A laughing face
Straight face, disappointed	:-| or :| or :o| or :O|	A straight face
Angel, innocent	O:-) or O:) or o:-) or o:)	An innocent face
Nerd	:-B or :B	A nerdish face
Sleepy	|-O or |O or |-o or |o	A sleepy face
Rolling eyes	8-) or 8) or 8o) or 8O)	A rolling eyes face
Sick, berk	:-& or :& or ;o& or :O&	A sick/ill face
Shhh! No speak, lips sealed	:-SS or :SS or :ss or :-ss	A face with sealed lips
Thinking, pensive	:-? or :?	A pensive face
Raised eyebrow, sarcastic look	/:-) or /:) or /:o) or /:O)	A raised eyebrow face or a face with a sarcastic look
Rose, flower	@):-	A rose
Cup of coffee	~o)	A cup of coffee
Drink, cocktail	)-|	A cocktail glass
Idea (light bulb)	*-:-) or *-:)	A light bulb
Love struck, heart	(L) or <3	A heart

Table 1: Character sequences and emoticons

5.6.5.2 Emoticon processing by the clients

The client will process the character sequences associated with each emoticon, converting the character sequences from/to their corresponding image representation when an instant message is received/displayed or composed/sent.

5.6.5.2.1 Emoticons in composed/sent instant messages

Whenever an instant message is composed by the user, the client MAY replace the recognized character sequences with the corresponding emoticon as the user types them into the instant message. Depending on client implementation the client MAY also offer the possibility to insert the emoticons using shortcuts. When the client uses shortcuts to insert emoticons into the instant message, it SHOULD choose the first character sequence from Table 1 corresponding to the inserted emoticon. In any case, when the client sends the instant message, the recognized emoticons MUST be converted to their corresponding character sequence before the instant message is submitted to the server.

5.6.5.2.2 Emoticons in received/displayed instant messages

Whenever an instant message is received or displayed by/for the user, the client MAY replace the recognized character sequences with the corresponding emoticon.

6. Fundamental Primitives and Transactions

The fundamental primitives and transactions provide the basic functionalities that are not related to any particular service – instant messaging, presence or group chat.

In order to achieve minimum level of interoperability both the client and the server MUST support the following transactions:

- 6.1 Status Primitive
- 6.2 PollingRequest Primitive – except over SMS transport, see [CSP Trans].
- 6.4 Logging In
- 6.6 Logging Out and Disconnecting
- 6.7 Keep Alive
- 6.9 Service and Capability Negotiation

The rest of the fundamental transactions are all OPTIONAL. The individual client or server implementations MAY decide whether support for a particular transaction is implemented or not.

6.1 Status Primitive

The Status primitive is used as a generic response primitive to all primitives that do not have a specific response primitive, and it is used as a generic response to all primitives to manage error situations as well.

The processing party MUST always respond with Status primitive to those requests that do not have their own response primitives.

When there is a specific response primitive defined for a request,

- in case of success, the specific response primitive MUST be sent.
- in case of failure
  - when the processing party is unable to understand the request because it is malformed and cannot be processed, the Status primitive MUST be sent including error code 400 (Bad Request) instead of the expected response primitive.
  - when the processing party is able to understand the request but there was an error during processing, and
    - the specific response primitive has no placeholder for the Result element, a Status primitive MUST be sent instead of the expected response primitive.
    - the specific response primitive has placeholder for the Result element, the specific response primitive MUST be sent.

The Result structure MUST contain one of the status codes specified in this document, see error codes in chapter 11 Status Codes and Descriptions. It MAY also contain a Description string and if necessary, a detailed description explaining the error in the DetailedResult element. The DetailedResult MAY be used with every error code but has its main use when the error concerns a specific Message-ID(s), User-ID(s), GroupID(s), Contact-List-ID(s), ScreenName(s) or domain(s). *An example is when sending a message to several recipients and one User-ID is invalid. The Result could then have status code 201 (Partially successful) and the DetailedResult element identifying the bad User-ID together with status code 531 (Unknown user).*

When the transaction is completely successful, the Result element MUST NOT include detailed results.

The Result structure MAY also contain SystemMessage notifications from the server. See section 7.1 System Message.

The status code 201 (Partially successful) MUST be used when only part of the request was successfully processed. In this case the DetailedResult element MUST include all part(s) that resulted in error(s) but the result(s) of successful part(s) of the transaction MAY be omitted. *An example is the creation of a contact list with initial contacts. If the server creates the contact list, but fails to add one or more of the contacts, the 201 (Partially successful) with DetailedResult for the bad contacts will be returned. If the server cannot create the contact list at all there was no partial success and the Result code will be for example 701 (Contact list already exists).*

When no part of the transaction was successfully processed but the error cannot be described by only one status code, the code 900 (Multiple errors) is used. In this case DetailedResult element MUST include all errors. By definition there can be no successful parts and that is the difference from status code 201 (Partially successful). *An example is the addition of users to a contact list, when a client requests addition of several users, but the server does not add any of those due to various reasons.*

The server MAY return status code 501 (Not Implemented) to any request to indicate that a particular feature has not been implemented or to indicate that an unrecognized request has been received.

The server MAY return status codes 902 (Not enough credit) or 903 (Operation requires a higher class) to indicate that a charging related error has occurred. Interface to a charging system is however outside of the scope of this specification and is implementation specific.

The client and the server MUST support the Status Primitive.

Information Element	Req	Type	Description
Message-Type	M	Status	Message identifier.
Transaction-ID	M	String	Identifies the transaction.
Session-ID	O	String	Identifier for the session.
Client-ID	O	Structure	Identifies the requesting IM client. Unique (for this user) identifier.
Result	M	Structure	The result of the transaction.

Table 2: Information elements in Status primitive

6.2 PollingRequest Primitive

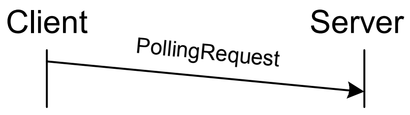

Figure 1: Polling request

The polling request is used in the context of the server pushing data to the Client. When a client has received Communication Initiation Request (CIR) from the CIR channel, or the 'Poll' element in a response primitive from the server indicates 'T' (true), the client MUST send the PollingRequest to the server to enable the server to insert the server-initiated transaction in the reply. The exact cases for the use of poll messages are elaborated in [CSP Trans]

The client MUST NOT send PollingRequest primitive to the server more often than the ServerPollMin value agreed during the latest client capability negotiation.

When the server receives PollingRequest from the client, the server MUST send the next unsent primitive, however it MUST NOT resend primitives that have been sent already to the client utilizing a previous PollingRequest but did not time out yet. The timeout for primitives is described in 5.4 Transaction Management.

The client and the server MUST NOT support the PollingRequest over SMS transport, but both MUST support the PollingRequest over any other transport [CSP Trans].

The PollingRequest primitive follows the basic primitive structure, but it MUST NOT carry Transaction-ID.

Information Element	Req	Type	Description
Message-Type	M	PollingRequest	Message identifier.
Session-ID	M	String	Identifier for the session.

Table 3: Information elements in PollingRequest primitive

6.3 Version Discovery Transaction

6.3.1 Transactions

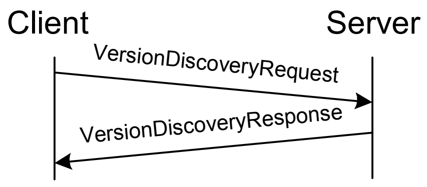

Figure 2: Version Discovery

From time to time, it MAY be necessary for a client to evaluate the best protocol version to use in sessions with a particular server. This is achieved using the Version Discovery transaction, which determines the protocol versions that are mutually acceptable to both the client and server. The IMPS client MAY and server MUST support the version discovery transaction, however its support is not negotiated, as this is an outband transaction.

The client initiates the Version Discovery transaction by sending a VersionDiscoveryRequest primitive to the intended server.

When the server receives the VersionDiscoveryRequest, it examines the client's proposed namespaces (if any).

- If the client proposes different namespaces, the server MUST select the namespace that it supports. It includes this namespace in the VersionDiscoveryResponse.
- If the server supports none of the namespaces that the client proposed in a particular set, then the server MUST return an empty result.
- If the client proposes no namespaces, then the server MUST include all of the supported namespaces in the response.

If the server knows of other servers that better support the namespaces proposed by the client, then the server MAY return the URLs (or MSISDNs/MDNs over SMS transport) of those servers to the client.

Servers that only implement version 1.1 of the IMPS specifications return an error indication when they received a VersionDiscoveryRequest. Clients that receive such an error response MAY choose to assume that the server only supports version 1.1 protocols.

The Version Discovery transaction does not follow the basic message structure; it is a dedicated transaction that is meant to be consistent with all future versions of the IMPS protocol. All servers MUST implement this transaction, while it is OPTIONAL for the clients.

6.3.2 Primitives and information elements

Primitive	Direction
VersionDiscoveryRequest	Client → Server
VersionDiscoveryResponse	Client ← Server

Table 4: Primitive directions for Version Discovery

Information Element	Req	Type	Description
Version-List	O	Structure	List of versions supported by the client.

Table 5: Information elements in VersionDiscoveryRequest primitive

Information Element	Req	Type	Description
Version-List	C	Structure	List of versions implemented by the server. If the Version-List was present in the request, only those versions are included in this list that are also implemented by the server.
Other-Servers	O	Structure	List of access points of servers that better support the namespaces proposed by the client.

Table 6: Information elements in VersionDiscoveryResponse primitive

6.4 Logging In

In order to use the IMPS services the user MUST log in into a Service Access Point. This login MAY happen automatically, meaning that the IMPS client performs the required login actions, without any user interaction. This is called "*Auto Login*". A user MUST be able to turn on/off "*Auto Login*" using a client setting.

The IMPS server MAY require Registration. When the IMPS server requires Registration and the user is not yet a Registered User of the service, the IMPS server MUST discontinue the login by sending an error message or by performing Auto-Registration (see 6.5 Auto-Registration)

After a successful login a client capability negotiation MAY be, and a service negotiation MUST be performed. If the server requests client capability negotiation – the 'Client-Capability-Request' element indicates 'T' (true) – then the client MUST perform it, otherwise the client MAY perform it (see 6.9 Service and Capability Negotiation).

In order to minimize latency, the client MAY request login, service negotiation, and client capability in a single transaction by including the appropriate information elements in the LoginRequest primitive, however the client MUST NOT utilize this feature when session re-establishment is requested. Note that the information elements are not suitable to discover the services: this feature supports only negotiation. Clients that desire to discover the available capabilities/services SHOULD utilize the negotiations separately. To perform service and/or client capability negotiation during the login transaction, the client MAY include Requested-Functions and/or Requested-Capabilities in the LoginRequest primitive. The included information elements MUST conform to the related semantics described in 6.9 Service and Capability Negotiation. In order to save bandwidth, the client SHOULD NOT include these information elements in the first LoginRequest primitive when the client intends to perform a four-way login. The server MUST process the information elements according to the related semantics described in 6.9 Service and Capability Negotiation, and include its response – when applicable – in the LoginResponse primitive. When the client capability negotiation has been performed successfully, the server MUST include the Client-Capability-Request element with value 'F'.

The Client-ID MUST be unique in the scope of the user and the server MUST NOT allow establishing multiple sessions for the same user with the same Client-ID. If the user logs in using a Client-ID already in use, the server MUST respond to the login with error code 608. The client MAY try to login again with a different Client-ID.

The server MAY send System Message notifications to the client during the Login transaction using the Result element in the Status primitive. See 7.1 System Message and 6.1 Status Primitive. The server MAY require a response to the System Message(s) before allowing the user to access the IMPS service - for example to acknowledge an advice of charge message. The server MUST NOT send System Message notifications before the User-ID is confirmed.

When the server requires response to System Message(s), it MUST send a LoginResponse primitive, which MUST include error code 436 in the Result element. When the client receives a response with error code 436, it MUST NOT attempt to continue the login sequence and it MUST prompt the end-user with the content of the System Message(s). When the end-user has responded to the System Message(s), the client MUST send a new LoginRequest primitive containing all information elements from the previous LoginRequest primitive and the System-Message-Response-List element. The System-Message-Response-List element MUST contain the end-user's response(s) to the System Message(s). If the server finds that the end-user's response is non-conformant to a System Message, it MAY re-send the System Message in the LoginResponse primitive any number of times, or it MAY send other System Message(s) in the LoginResponse primitive, or it MAY refuse the client – using error code 607 – to perform login attempts for an implementation-specific period of time.

The server MAY keep track of outstanding System Message notifications requiring response for an implementation specific period and resend those notifications to the end-user in future transaction if no response is received from the end-user. The server MAY also generate a new System Message notification (identified by a new SystemMessage-ID) if no response is received within an implementation specific period to the original notification.

When the response from the server does not contain error code 436 – the server does not require a response to the System Message(s) – the client MUST prompt the end-user with the System Message(s), and it MAY continue with the login sequence normally, and MAY send the user's response(s) to the System Message(s) later.

The client MUST support either 2-way login or 4-way login transaction and MAY support both.

The IMPS server MUST support the 2-way Login transaction and the 4-way Login transactions and MAY support network-based authentication.

The client MUST choose either a 2-way access control (6.4.1 2-way access control) or a 4-way access control (6.4.2 4-way access control). The server MAY choose - in any case - to authenticate the user based on authentic network information instead (6.4.3 Authentication based on authentic network information).

When the login transaction was successful, and the client included Application-ID along with the Client-ID, then the server MUST publish the Application-ID along with the Client-ID in the client's ClientInfo presence attribute.

When the login was successful (code 200) and if the client used a Local User-ID in the LoginRequest primitive, the server MUST return a detailed result with code 200 and the Domain-part of the User-ID in the (first) LoginResponse primitive.

When the login transaction was successful, after the successful service negotiation the server MUST verify the Auto-Join settings of the user, and perform the operations described in 10.4 Join Group Feature.

6.4.1 2-way access control

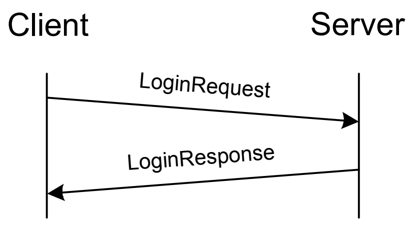

Figure 3: 2-way access control

If the client chooses the 2-way access control, the IMPS server MUST authenticate the IMPS user either by verifying the UserID/password combination or it MUST authenticate the IMPS user based on MSISDN/MDN or other authentic network information (see 6.4.3 Authentication based on authentic network information). The IMPS server MUST respond with success, failure, or further authorization. In the latter case, the server MUST send a LoginResponse primitive containing error code 401. It means that the server requires 4-way access control and the client MUST use the 4-way access control instead in order to login (see 6.4.2 4-way access control).

The following information elements

- MUST be present in the LoginRequest primitive:
  - User-ID
  - Client-ID
  - Password-String
  - Session-Cookie
- MAY be present in the LoginRequest primitive:
  - Time-To-Live
  - Application-ID
  - Requested-Functions
  - Requested-Capabilities

- System-Message-Response-List

If Time-to-Live is omitted, it MUST be considered to be infinite.

The server MUST verify that the 'Password-String' is valid for the user, store the Session-Cookie, and process the OPTIONAL information elements according to their semantics.

After the IMPS server processed the request, it MUST send a LoginResponse primitive to the client; which will contain the details of the login result:

- When the login operation into the Service Access Point is successful, the LoginResponse primitive
  - MUST contain:
    - Client-ID
    - Session-ID
    - Result (containing status code 200)
    - Keep-Alive-Time
    - Client-Capability-Request
  - MAY contain:
    - Not-Available-Functions
    - Agreed-Capabilities
- When the login operation into the Service Access Point is not successful, the LoginResponse primitive
  - MUST contain
    - Client-ID
    - Result (the result MUST indicate the reason of failure)
  - MUST NOT contain
    - Session-ID

6.4.2 4-way access control

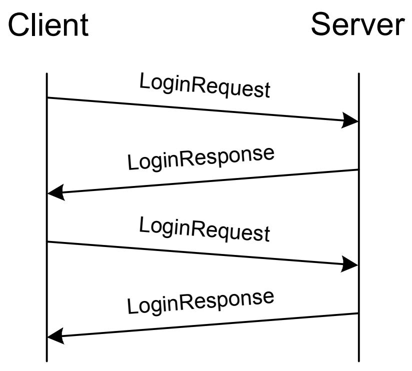

Figure 4: 4-way access control

If the client chooses the 4-way access control, the IMPS server MUST authenticate the IMPS user either by verifying the UserID/password combination or it MUST authenticate the IMPS user based on MSISDN/MDN or other authentic network information (see 6.4.3 Authentication based on authentic network information). The 4-way access control consists of 2 sets of LoginRequest/LoginResponse primitives. In the 4-way access control, the Client concatenates the 'Nonce' received from the server with the password, and encrypts the resulting string using an agreed digest schema before sending this BASE64 encoded string to the server. The server MUST use the same digest schema to decode the string and authenticate the user.

The server SHOULD support the following Digest Schemas:

- MD5: see [RFC1321]
- SHA-1: see [FIPS 180-1]

The server MAY support the MD4 Digest schema: see [RFC1320].

Other standard digest schemas MAY be introduced. Introducing other standard digest schemas is not in scope of this specification, however these schemas MUST NOT interfere with the IMPS schemas.

The Transaction-ID for the 1st set and the 2nd set of the 4-way login MUST be the same.

6.4.2.1 1st set of 4-way login

The 1st set is used to negotiate the digest schema and to deliver the 'Nonce' to the client. The following information elements

- MUST be present in the LoginRequest primitive:
  - User-ID
  - Client-ID
  - Supported-Digest-Schema
  - Session-Cookie
- MAY be present in the LoginRequest primitive:
  - Application-ID

The server MUST examine the Supported-Digest-Schema element, and generate a 'Nonce' based on a digest schema that is supported by both the client and the server.

After the IMPS server processed this request, it MUST send a LoginResponse primitive to the client; which will contain the details of the login result:

- When the 1st login request is successful, the LoginResponse primitive
  - MUST contain:
    - Client-ID
    - Nonce
    - Digest-Schema
    - Result (containing status code 200)
  - MUST NOT contain:
    - Session-ID
- When the login operation into the Service Access Point is not successful, the LoginResponse primitive
  - MUST contain
    - Client-ID
    - Result (the result MUST indicate the reason of failure)
  - MUST NOT contain:
    - Session-ID

An exception to this LoginResponse is when the server authenticates the user based on MSISDN/MDN or other authentic network information as described in 6.4.3 Authentication based on authentic network information.

If the server does not support any of the Digest-Schema's in the Supported-Digest-Schema information element, the server MUST respond with error code 543 or authenticates the user based on MSISDN/MDN or other authentic network information.

6.4.2.2 2nd set of 4-way login

In the 2nd set, the client uses the Nonce and the Digest-Schema from the 1st set LoginResponse to create a BASE64 encoded string, the Digest-Bytes:

Digest-Bytes = Digest-Schema (Nonce + Password-String)

The client MUST use the digest schema returned by the server in the 1st set LoginResponse.

The following information elements

- MUST be present in the 2nd LoginRequest primitive:
  - User-ID
  - Client-ID
  - Digest-Bytes
  - Session-Cookie
- MAY be present in the LoginRequest primitive:
  - Time-To-Live
  - Application-ID
  - Requested-Functions
  - Requested-Capabilities
  - System-Message-Response-List

If Time-to-Live is omitted, it MUST be considered to be infinite.

The server MUST verify that the 'Digest-Bytes' is valid for the user, store the Session-Cookie, and process the OPTIONAL information elements according to their semantics.

After the IMPS server processed this request, it MUST send a LoginResponse primitive to the client; which will contain the details of the login result:

- When the login operation into the Service Access Point is successful, the LoginResponse primitive
  - MUST contain:
    - Client-ID
    - Session-ID
    - Result (containing status code 200)
    - Keep-Alive-Time
    - Client-Capability-Request
  - MAY contain:
    - Not-Available-Functions
    - Agreed-Capabilities
- When the login operation into the Service Access Point is not successful, the LoginResponse primitive
  - MUST contain
    - Client-ID
    - Result (the result MUST indicate the reason of failure)
  - MUST NOT contain:
    - Session-ID

6.4.3 Authentication based on authentic network information

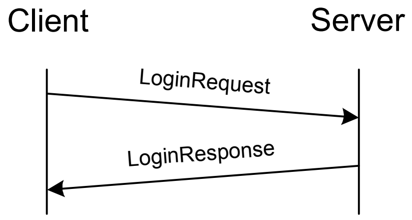

Figure 5: Authentication based on authentic network information

The server MAY choose to authenticate the user based on MSISDN or other authentic network information. It is out of scope of this specification to specify how the server authenticates the user. The 1st LoginRequest MUST be responded by a final LoginResponse (also in case the user initiated 4-way access control). Therefore the client MUST be prepared to accept a final LoginResponse primitive although the 4-way access control was initiated.

The LoginRequest MUST be a 2-way access control LoginRequest (as described in 6.4.1 2-way access control) or the 1st set LoginRequest of the 4-way access control (as described in 6.4.2.1 1st set of 4-way login).

The IMPS specifications do not define what kind of information the servers MAY use to authenticate the user, however the server MUST use some user-specific information that is authentic.

After the IMPS server authenticated the user, it MUST send a LoginResponse primitive to the client; which will contain the details of the login result:

- When the login operation into the Service Access Point is successful, the LoginResponse MUST contain:
  - Client-ID
  - Session-ID
  - Result (containing status code 200)
  - Keep-Alive-Time
  - Client-Capability-Request
- When the login operation into the Service Access Point is successful, the LoginResponse MAY contain:
  - Not-Available-Functions
  - Agreed-Capabilities
- When the login operation into the Service Access Point is not successful the LoginResponse MUST contain
  - Client-ID
  - Result (the result MUST indicate the reason of failure)

6.4.4 Session recovery

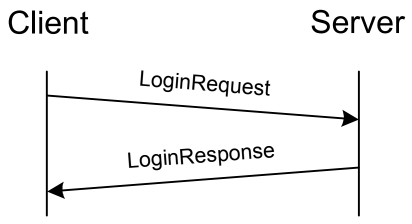

Figure 6: Session recovery

The client SHOULD store the Session-IDs – when a new session has been established – in a storage area that is capable of keeping the Session-ID safe when an unexpected disconnection occurs (e.g. power loss, crash, etc), and the client

MUST remove the safely stored Session-ID upon logout/disconnect. When the client has such Session-ID stored, and the client

- remembers the session context, it SHOULD NOT attempt to log in – the client SHOULD attempt to re-establish the session first, and when session re-establishment fails, the client MAY attempt to log in normally.
- does not remember the session context, it SHOULD NOT attempt to log in – it SHOULD attempt to re-establish the session first. If the session re-establishment was successful, the client MUST log out from the re-established session and log in again – this will reset the session context. When session re-establishment fails, the client MAY attempt to log in normally.

When the client does not have such Session-ID stored, the client SHOULD attempt to log in. When the login attempt fails due to non-unique Client-ID (error code 608), the client MAY attempt to log in again with a different Client-ID.

The client MAY attempt to recover a session as described in 5.1.2 Re-establishing sessions. To attempt recovering of a session, the client MUST perform the normal login transaction (either using 2-way access control or 4-way access control) using the Session-ID of the desired session context.

The IMPS server MAY recover the session. When the server recovers the session, the session context of the user MUST be re-established into the same state as it was at the time the session was terminated (see 5.1.1 Session context), and the server MUST perform the required actions to bring the session into this state. E.g. if the session context contains subscriptions to presence information, the IMPS server MUST re-subscribe to this presence information during the session recovery.

When the IMPS server does not recover the session, it MUST respond the 1st LoginRequest with a LoginResponse indicating that the server is not able to recover the session (error code 502). The user MUST NOT be logged in when the recovery attempt fails. To log in, the client MUST perform normal login transaction (either using 2-way access control or 4-way access control) without the Session-ID information element.

6.4.5 Error conditions

Generic error conditions:

- Service unavailable. (503)
- Version not supported. (505)
- Too many non-conformant System Message replies (607)

LoginRequest error conditions:

- Further authorization needed to use the server. (401)
- Invalid password. (409)
- The particular user is not allowed to use the server. (403)
- Session-ID, User-ID and Client-ID not matching. (422)
- System Message Response required (436)
- The server could not recover the session. (502)
- *Application* is forbidden by operator. (515)
- Unknown user. (531)
- No matching digest scheme supported (543)
- Some services are not available (606)
- Client-ID is not unique (608)
- User session limitation reached (610)
- MSISDN error (920)
- Registration confirmation. (921)
- Service provider agreement missing (907)

6.4.6 Primitives and information elements

Primitive	Direction
LoginRequest	Client → Server
LoginResponse	Client ← Server

Table 7: Primitive directions for Logging in

Information Element	Req	Type	Description
Message-Type	M	LoginRequest	Message identifier.
Transaction-ID	M	String	Identifies the transaction, set by client.
User-ID	M	String	Identifies the requesting User.
Client-ID	M	Structure	Identifies the requesting IMPS client. Unique (for this user) identifier.
Application-ID	O	String	Identifies the client application to be associated with the established session.
Password-String	C	String	The password digest corresponding to the User-ID
Digest-Bytes	C	String	The digest is BASE64 encoded.
Supported-Digest-Schema	C	String	A list of supported digest schema (PWD, SHA, MD4, MD5)
Session-ID	C	String	Identifier for the session when session reestablishment is requested.
Time-To-Live	O	Integer	Time requested between client to server messages before client is considered disconnect. If information element is not present client is requesting an infinite time-to-live time. Indicated in seconds.
Session-Cookie	M	String	The session cookie used by WV SAP to initiate communications within the session.
Requested-Functions	O	Structure	Identifies the service elements and functions the client requests when the client desires to perform service negotiation with login in a single transaction.
Requested-Capabilities	O	Structure	Identifies the capabilities requested by the client when the client desires to perform client capability negotiation with login in a single transaction.
System-Message-Response-List	O	Structure	The list of System Message response(s) from the end-user.

Table 8: Information elements in LoginRequest primitive

Information Element	Req	Type	Description
Message-Type	M	LoginResponse	Message identifier.
Transaction-ID	M	String	Identifies the request transaction.
User-ID	C	String	Identifies the requesting user.
Client-ID	M	Structure	Identifies the requesting WV client. Unique (for this user) identifier.
Password-String	O	String	The password corresponding to the User-ID.
Result	M	Structure	Result of the login request.

Nonce	C	String	Random string generated by server for password digest. The string is not BASE64 encoded.
Digest-Schema	C	String	Type of digest schema to use.
Session-ID	C	String	Identifier for the session. String generated by the server to identify this session. Session-ID MUST be supplied with all following inband requests to the server. Present only if login was successful.
Keep-Alive-Time	C	Integer	Auto logout timer value in seconds. The server can set any timer value and the client MUST obey that. Each message transaction resets the Keep-Alive-Time timer. Present only if login was successful.
Client-Capability-Request	C	Boolean	Informs the Client that it needs to perform a Client Capability Request transaction. Present only if login was successful.
Not-Available-Functions	C	Structure	Identifies the delta of services the client requested and what is available for that user when the client requested to perform service negotiation with login in a single transaction.
Agreed-Capabilities	C	Structure	Identifies the capabilities agreed by the server when the client requested to perform client capability negotiation with login in a single transaction.

Table 9: Information elements in LoginResponse primitive

6.5 Auto-Registration

In order to use the IMPS services the service provider MAY require that the user MUST be registered for the service.

When the service provider requires the user to be registered, the IMPS server MUST be able to determine upon login whether this user is registered for the service. If the user is not registered to the service, the IMPS server MAY perform Auto-Registration. The IMPS server automatically registers the user for the service. A User-ID MUST be created and sent to the client in the LoginResponse. A password MAY be created and also sent to the client in the LoginResponse. The client MUST update the User-ID and MAY update the password (if present) with the values received in the LoginResponse.

If extra registration information from the user is required before completing the registration (e.g. AoC, T&C), the result of the LoginResponse MUST contain status code 921 and the user MUST NOT be able to use the service before the IMPS server received this input. System message (see 7.1 System Message) MUST be used to exchange this extra registration information. To inform the client that the registration is complete and the client MAY continue with login, the server MUST send LoginResponse primitive to the client

If a new User-ID was created during registration, the server MUST include the newly created User-ID in the LoginResponse primitive. When the server included User-ID in the LoginResponse primitive, the client MUST take the User-ID into use immediately, however the client is not required to restart the login procedure with the new User-ID as the server from this point forward MUST use the User-ID returned in the LoginResponse primitive.

If the LoginResponse contains 200 OK as result, the user can start using the server immediately – unless other negotiations MUST be performed first.

6.6 Logging Out and Disconnecting

6.6.1 Transactions

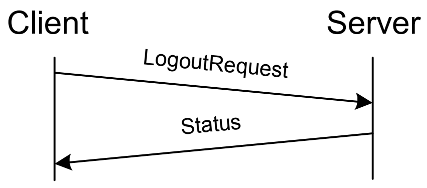

Figure 7: Logging Out

The user MAY log out from the IMPS by using the LogoutRequest message. The server MUST respond with a Status primitive.

If the user is joined to one or more discussion groups when the logout request is issued, the server MUST automatically remove (leave group) the user from the discussion group.

The client and the server MUST support the Logout transaction.

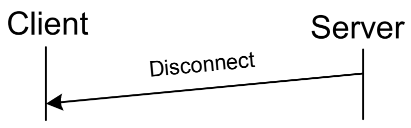

Figure 8: Server Initiated Disconnection

Whenever the server disconnects a client, it MUST send a Disconnect primitive to the client. The client MUST NOT send any response to the Disconnect primitive. The server SHOULD disconnect a client if the session Keep-Alive-Time timer has been exceeded. The server MAY also disconnect the client for some other reason, however it MUST NOT disconnect a client to allow another client to log in. The server sends the Disconnect message to the client containing the Session-ID, and the Result containing code and descriptive text. The Transaction-ID MUST be present in the primitive for compatibility reason only (with other primitives) – the client MUST ignore its content.

The client and the server MUST support the server-initiated disconnection.

The Disconnect primitive follows the basic primitive structure, but it MUST NOT carry Transaction-ID.

If the user has logged-out or is disconnected from the server, the session context and the related information MUST be dropped, unless the server offers session recovery and it keeps the session context for a later recovery. For more information see session context in chapter 5.1.1 Session context.

6.6.2 Error conditions

Generic error conditions:

- Service unavailable. (503)
- Not logged in. (604)

LogoutRequest error conditions:

- None except the generic error conditions.

Disconnect error conditions:

- Forced logout. (601)
- Session expired. (600)

6.6.3 Primitives and information elements

Primitive	Direction
LogoutRequest	Client → Server
Status	Client ← Server
Disconnect	Client ← Server

Table 10: Primitive directions for Logging Out

Information Element	Req	Type	Description
Message-Type	M	LogoutRequest	Message identifier
Transaction-ID	M	String	Identifies the transaction.
Session-ID	M	String	Identifier for the session.

Table 11: Information elements in LogoutRequest

Information Element	Req	Type	Description
Message-Type	M	Disconnect	Message identifier.
Session-ID	M	String	Identifier for the session.
Result	M	Structure	Indicates the code and description why the disconnection happened.

Table 12: Information elements in Disconnect primitive

6.7 Keep Alive

6.7.1 Transactions

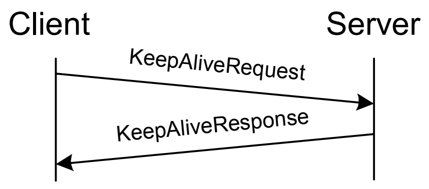

Figure 9: Keep alive transaction

The client MUST send KeepAliveRequest to reset the keep-alive timer if no other transaction has occurred during the KeepAliveTime interval. The server MUST reset the Keep-Alive timer not only when the client sends that KeepAliveRequest primitive, but also when any other transaction occurs over the data channel. The CIR channel activities are completely separate and MUST NOT reset the Keep-Alive timer.

The client and the server MUST support the Keep-Alive transaction.

The server MUST NOT disconnect a client before the keep-alive timer expires due to timer expiration – however the server MAY disconnect a client before timer expiration for some other reason.

If the client requested a new timeout value by including Time-To-Live in the request, the server MUST include the accepted Keep-Alive-Time in the response. The Keep-Alive-Time that the server accepts and sends back in the response MAY be different from the Time-To-Live value that the client requested. If the server does not accept the new timeout value requested by the client, the response MUST either include the new timeout value proposed by the server, or the status code indicating that the old value is in use. The client MUST obey the new Keep-Alive-Time value back by the server.

The server MAY specify different Keep-Alive-Time values for different sessions (as shown in Figure 10) and over time the server MAY occasionally return a modified Keep-Alive-Time to the requesting client.

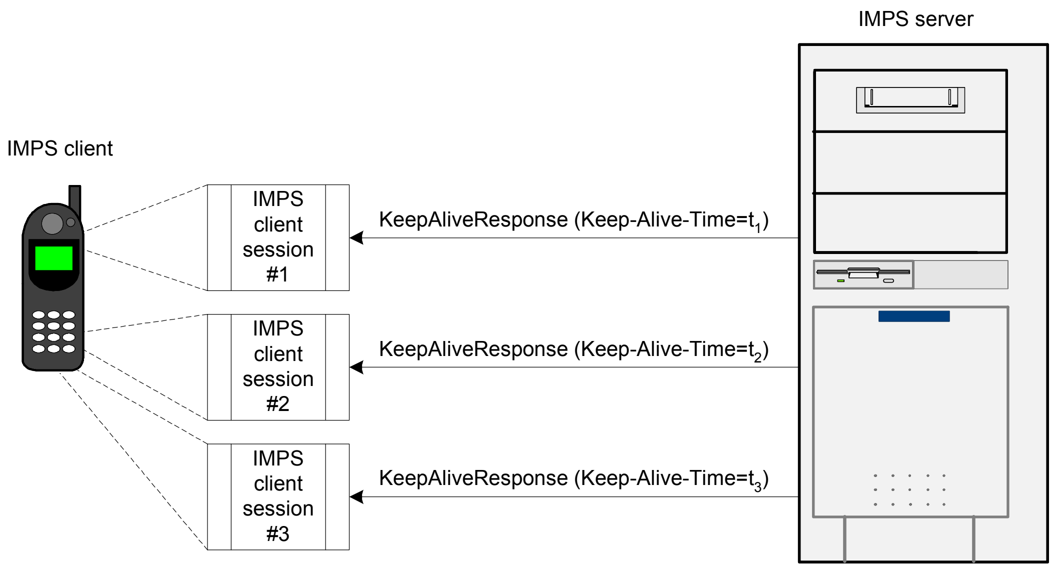
IMPS server
IMPS client
IMPS client session #1
IMPS client session #2
IMPS client session #3
KeepAliveResponse (Keep-Alive-Time=t_1)
KeepAliveResponse (Keep-Alive-Time=t_2)
KeepAliveResponse (Keep-Alive-Time=t_3)

Figure 10: Keep-Alive-Times t1, t2 and t3 can be adapted by server over time across different sessions

Example:

The IMPS client in Figure 10 supports three concurrent sessions: one session for IM and presence services (#1); another one for group messaging services (#2); and a third one for application-to-application communications (#3).

The timeouts t1, t2 and t3 returned by the server to the requesting client is not necessarily equal or even constant over time. The server can adapt those values in order to optimize the traffic load over the air interface.

6.7.2 Error conditions

Generic error conditions:

- Service unavailable. (503)
- Not logged in. (604)

KeepAliveRequest error conditions:

- New timeout value not accepted – old value is in use. (605)

6.7.3 Primitives and information elements

Primitive	Direction
KeepAliveRequest	Client → Server
KeepAliveResponse	Client ← Server

Table 13: Primitive directions for keep alive transaction

Information Element	Req	Type	Description
Message-Type	M	KeepAliveRequest	Message identifier.
Transaction-ID	M	String	Identifies the transaction.
Session-ID	M	String	Identifier for the session.

Time-To-Live	O	Integer	Indicates the new time-to-live of the session in seconds.

Table 14: Information elements in KeepAliveRequest primitive

Information Element	Req	Type	Description
Message-Type	M	KeepAliveResponse	Message identifier.
Transaction-ID	M	String	Identifies the transaction.
Session-ID	M	String	Identifier for the session.
Result	M	Structure	The result of the request.
Keep-Alive-Time	C	Integer	Indicates the new time-to-live of the session in seconds.

Table 15: Information elements in KeepAliveResponse primitive

6.8 Get Service Provider Info

6.8.1 Transactions

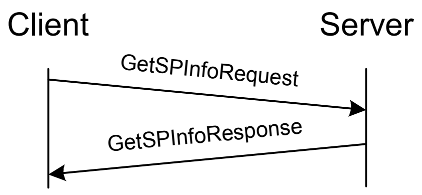

Figure 11: Get Service Provider Info transaction

The Get Service Provider information retrieves information about the Service Provider. The name of the provider as well as a multimedia message MAY be used as a splash screen or "about information", or link to a web/wap page that might contain more useful information. This transaction MAY be done without login in to the server.

The IMPS client and server MAY support for the Get Service Provider Info transaction. The service tree leaf that allows negotiation of this transaction is 'GETSPI'. The transaction MAY be performed without having an established session – if the feature is not supported on the server as such, the server MUST respond with a Status primitive indicating this.

The Session-ID MUST be present in an 'Inband' request, and MUST NOT be present in an 'Outband' request. The server MUST respond to the request accordingly – 'Inband' response to and 'Inband' request, and 'Outband' response to an 'Outband' request.

6.8.2 Error conditions

Generic error conditions:

- Service not supported. (405)
- Service unavailable. (503)
- Service not agreed. (506)
- Not logged in. (604)

GetSPInfoRequest error conditions:

- Client-ID not matching this user. (422)
- Service provider agreement missing (907)

6.8.3 Primitives and information elements

Primitive	Direction
GetSPInfoRequest	Client → Server
GetSPInfoResponse	Client ← Server

Table 16: Primitive directions for Get Service Provider Info

Information Element	Req	Type	Description
Message-Type	M	GetSPInfoRequest	Message identifier.
Transaction-ID	M	String	Identifies the transaction.
Session-ID	C	String	Identifier for the session.
Client-ID	C	Structure	Identifies the requesting client.

Table 17: Information elements in Get Service Provider Info Request primitive

Information Element	Req	Type	Description
Message-Type	M	GetSPInfoResponse	Message identifier.
Transaction-ID	M	String	Identifies the transaction.
Session-ID	C	String	Identifier for the session.
Client-ID	C	Structure	Identifies the requesting client.
Name	M	String	Name of the service provider.
Logo	O	Structure	Service-provider specific image. (*e.g.*, logo)
Text	O	String	Descriptive text.
URL	O	String	Link to a web page.

Table 18: Information elements in Get Service Provider Info Response primitive

6.9 Service and Capability Negotiation

6.9.1 Transactions

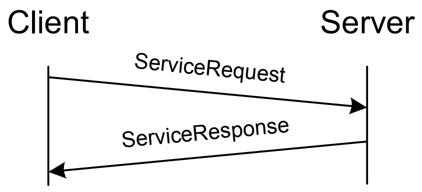

Figure 12: Service negotiation

After successful login and client capability negotiation (if applicable) the client and server MUST set up the context of the session. Service negotiation MUST be done either during the login transaction, or after the successful login and client capability negotiation transaction (if applicable), and MAY be repeated during the session at any time. To perform service negotiation during login transaction, see LoginRequest/LoginResponse primitives in chapter 6.4 Logging In.

The goals of the service negotiation are:

- To discover the available services.

  To discover which services are available on the server, the client MAY request the list of available services using the All-Function-Request information element. If All-Functions-Request element indicates 'T' in the ServiceRequest primitive, the server MUST send the list of all services that the server supports in the response. If All-Functions-Request element indicates 'F' in the ServiceRequest primitive, the Requested-

Functions element MUST be present in the request. If requested, the ServiceResponse primitive will contain the service tree, which gives the information about the available services. Having the tree available the client can select which services it would like to use; it is ready to make an agreement.

- To agree what services will be used.

  The client and the server MUST agree which features and functions will be used during the session. The client MUST send its service tree to the server in the Requested-Services element of the ServiceRequest primitive describing what kind of services it would like to use. The server MUST process the tree, and MUST send back the ServiceResponse primitive containing the inverted tree (meaning that the tree MUST contain the features and functions the client has requested but not allowed to use). The client throughout the whole session MUST NOT use any of the features and functions that the server did not agree to provide: all non-negotiated features and functions MUST fail. Non-negotiated features and functions are the ones that were either not requested by the client, or not agreed by the server for some reason.

The service tree is a set of features and functions structured as a tree, WVCSPFeat being the root (which indicates all available features and functions). The tree MAY have multiple sub-trees (with multiple roots). There is no need to indicate the whole sub-tree when all features or functions are requested/denied under a specific sub-tree: only the sub-root element MUST be present in the tree. However, it is necessary to indicate the sub-tree when certain features (that are part of a sub-root element) are not requested.

When only the mandatory functions of a particular feature are requested, only the corresponding mandatory (MF, MP, MM, MG) element of the feature MUST be present in the service tree. If any other function is requested under a certain feature than the mandatory functions, the MF, MP, MM, MG element MUST NOT be indicated in the tree. When a sub-root element is requested without any elements under it, it means that all features and functions are requested under the particular sub-tree that are either MANDATORY or OPTIONAL.

The client and the server MUST support service negotiation transaction. The server MUST support repeating the service negotiation during the session at any time.

The server MAY provide support for Fundamental Features, Instant Messaging, Presence, Group and Content services, however at least one of these services MUST be negotiated.

Note that the service tree in Plain Text Syntax [CSP PTS] is handled differently.

The abbreviated names of the service tree elements are described in the sections related to the transactions.

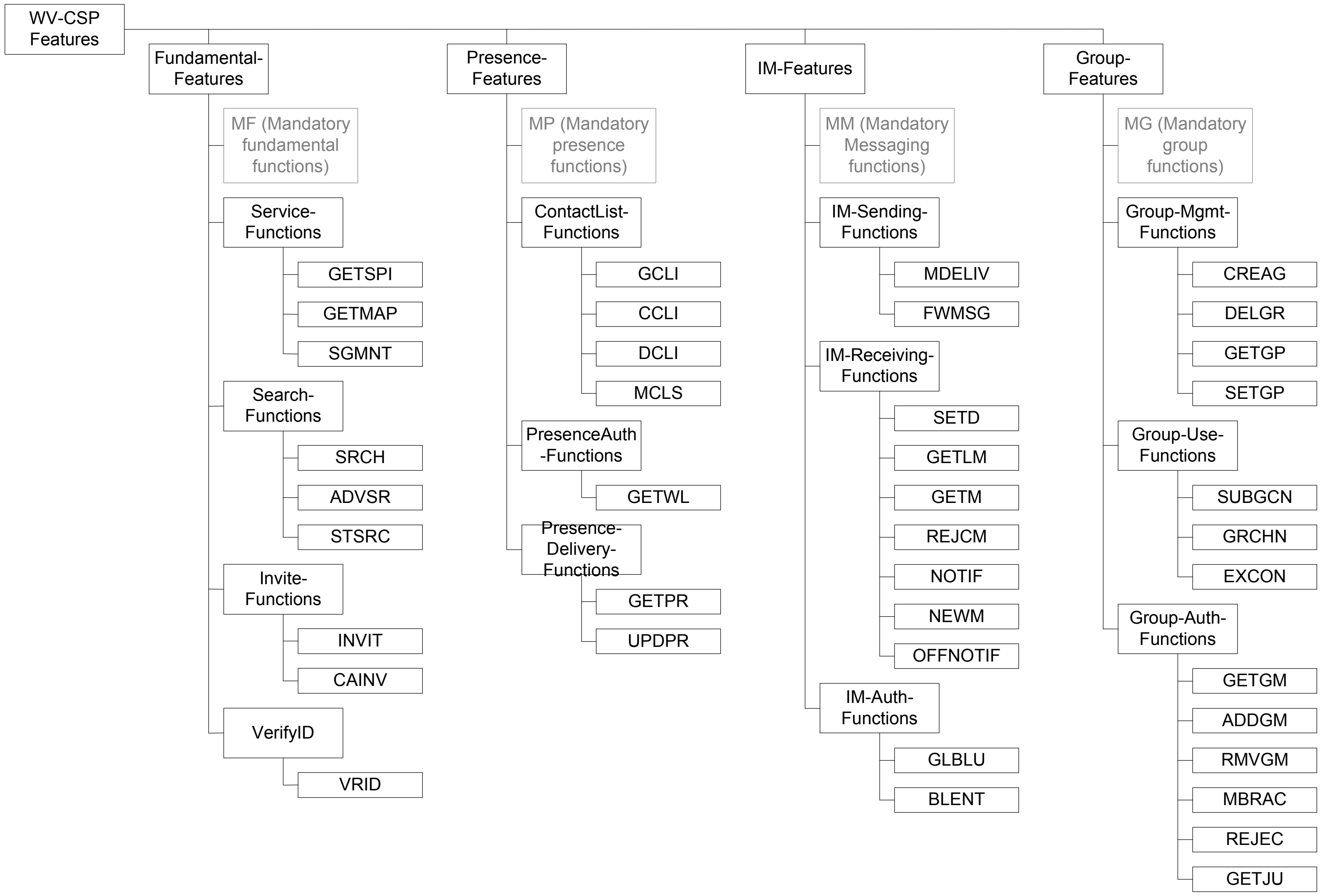

Figure 13: The service tree

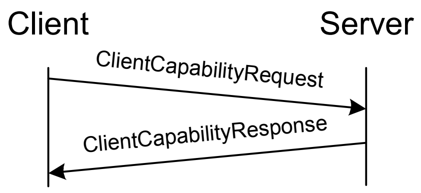

Figure 14: Client Capability Request

Client capability negotiation MAY be performed during the login transaction or after the successful login transaction. To perform client capability negotiation during login transaction, see LoginRequest/LoginResponse primitives in chapter 6.4 Logging In. When the 'Client-Capability-Request' element in the LoginResponse indicates 'T' (true), the client MUST perform client capability negotiation after the login transaction has been completed successfully – before service negotiation.

The server MUST maintain the client capability information during the session, and it MAY cache these capabilities between sessions. If the usage of offline message notification (OFFNOTIF) is agreed during service negotiation, the supported offline message notification bearer(s) (the SupportedOfflineBearer setting) MUST be cached between sessions. The server MUST store the OfflineETEMHandling and the related Client-ID for the user even after the session has been disconnected. If the client capability negotiation is needed after login, the server MUST indicate it in the login response. The client capability negotiation MAY also be repeated any time during a session.

The client capability request sets up the communication preferences for the session, for example the initial IM delivery method.

The ClientCapabilityRequest primitive MUST contain the Requested-Capabilities element that conveys the client capability information to the server. The client capability information includes:

- AcceptedContentType – Structure(s) containing the media types and the related credentials of those content types that the client supports. The client MUST NOT include the text/plain content type – it is negotiated separately using PlainTextCharset and AcceptedTextContentLength. Applicable only when AnyContent is missing. The structure contains the followings:
  - ContentType – the MIME type of the media object.
  - AcceptedRichContentLength – An integer number in character count that indicates the length of this particular media type that the client supports without special conditions. See ContentPolicy for special conditions.
  - ContentPolicy – An enumerated value that indicates whether if there is a special handling policy active regarding this media type. It MUST be one of the following values:
    - N – No active policy meaning that the client does not wish to limit this particular content type in any way. The ContentPolicyLimit MUST NOT be present. The server MAY change this value to 'R' or 'C' in the ClientCapabilityResponse.
    - R – the client will not retrieve those instant messages where the size of the media content is higher than the size defined in ContentPolicyLimit. The ContentPolicyLimit MUST be present. The server MUST accept and MUST NOT change the 'R' value.
  - ContentPolicyLimit – An integer value that indicates the special handling policy limitation regarding this particular media type. The client MUST make sure that when ContentPolicyLimit specified, the ContentPolicy is 'R' and the ContentPolicyLimit value is higher than the AcceptedRichContentLength value.
- AcceptedPullLength – An integer number in character count that indicates the maximum length of content (either text/plain or multimedia object) that the client accepts for client-originated content delivery transactions.
- AcceptedPushLength – An integer number in character count that indicates the maximum length of content (either text/plain or multimedia object) that the client accepts for server-originated content delivery transactions.
- AcceptedTextContentLength – An integer number in character count that indicates the length of the text/plain messages that the client supports. It MUST be negotiated separately from the rich content types. The value MUST NOT be higher than AcceptedPullLength and AcceptedPushLength.

- AcceptedTransferEncoding – The list of supported transfer-encoding methods in the client device, such as “BASE64”. The client MUST NOT request any transfer-encoding method when the encoding syntax in use is Binary XML - [CSP WBXML]. The IMPS standard recommends the BASE64 transfer-encoding method, since BASE64 is the most commonly used transfer-encoding method. However client and server implementations are not restricted to BASE64, various client and server implementations MAY use other transfer-encoding methods as well, such as uuencode.
- AnyContent – A Boolean value indicating that the client accepts any rich content types, or does not accept any rich content types. When the value is ‘T’, the client MUST NOT include any AcceptedContentType. When AnyContent is ‘T’, AcceptedPullLength and AcceptedPushLength applies and maximum content length is not negotiated. When the client does not support any rich content types, AnyContent MUST be included with the value is ‘F’, and AcceptedContentType MUST NOT be included. In all other cases, the AnyContent MUST NOT be included.
- ClientType – the type of the client. The server MUST accept the value and MAY update this value into the ClientInfo/ClientType presence attribute after the negotiation has been successfully completed. See the related table in [PA].
- DefaultLanguage – The current language setting in the client. The server MUST accept the value. The language code is specifying that the client prefers to receive text information in the indicated language from the server. The information is OPTIONAL – it is used to override the user profile/presence info language preference.
- InitialDeliveryMethod – the initial IM delivery method that the recipient client prefers in the set of “PUSH” and “Notify/Get”. The server MUST accept the value.
- MultiTrans – The maximum number of open transactions from both client and server side at any given time. The value MUST be greater than 0.
- MultiTransPerMessage – Integer value indicating the maximum number of primitives that the client can handle within the same transport message at any given time. The value MUST be greater than 0.
- OfflineETEMHandling – enumerated value indicating how the client expects the server to handle end-to-end messages that have been addressed particularly to this client after the client logged out or was disconnected. The client MAY request one of the following values:
  - PRIORITYREJECT – the server MUST send all end-to-end messages that have been addressed to this client to another client with the highest priority that supports the content of the end-to-end message, but if no such client exists, the server MUST reject the end-to-end message.
  - PRIORITYSTORE – the server MUST send all end-to-end messages that have been addressed to this client to another client with the highest priority that supports the content of the end-to-end message, but if no such client exists, the server MUST store the end-to-end message for later delivery.
  - REJECT – the server MUST reject all end-to-end messages that have been address to this client.
  - SENDREJECT – the server MUST send all end-to-end messages that have been addressed to this client to all other online clients that supports the content of the end-to-end message, but if no such client exists, the server MUST reject the end-to-end message.
  - SENDSTORE – the server MUST send all end-to-end messages that have been addressed to this client to all other online clients that supports the content of the end-to-end message, but if no such client exists, the server MUST store the end-to-end message for later delivery.
- OnlineETEMHandling – enumerated value indicating how the user expects the server to handle end-to-end messages that have not been addressed to any specific client/application of the user while the user is online. The value MUST NOT be a per-client setting, it MUST be an overall setting for the user. During login sequence the clients SHOULD retrieve the current value instead of setting a value explicitly. Clients SHOULD change the value during the session upon explicit request from the user. The server MUST either accept the requested value, or override and return the new value in the ClientCapabilityResponse. All servers MUST support ‘DETECT’, and either ‘SERVERLOGIC”, ‘FORKALL’, or both methods. Before the first login of an IMPS user the server MUST set a default value, which MUST be ‘SERVERLOGIC’ or ‘FORKALL’. In some cases, the server might not find a suitable/online client to deliver the end-to-end message. When the server is unable to find suitable/online clients, the server MUST either store the end-to-end message for later delivery (when offline storage is supported), or reject the requested transaction. When the end-to-end message has been stored for later

delivery, the server MUST attempt to deliver the end-to-end message when there is an online client for the user that supports the transactions that are essential to deliver the end-to-end message. The client MAY request one of the following values for OnlineETEMHandling:

  - DETECT – the client does not request any particular value, it simply asks the server to return the current user setting.
  - SERVERLOGIC – the server MUST route all end-to-end messages to a single client of the recipient user that supports the transactions that are essential to deliver the end-to-end message. The server MUST select the appropriate client using an intelligent decision based on the available information, such as: SessionPriority, presence status, Application-ID, content type and size limitations, etc.
  - FORKALL – the server MUST send all end-to-end messages that have been addressed to this user to all of his/her online clients that support the transactions that are essential to deliver the end-to-end message.

- ParserSize – the maximum character (byte) count of XML (WBXML, SMS - depending on the actual encoding) primitive size that the client-side parser can handle. The server SHOULD accept this value without changes unless the server finds the value unreasonably high. When the value is too high, the server MUST include ParserSize in the ClientCapabilityResponse primitive indicating a value that was accepted. The originator MUST NOT send primitives that are larger than ParserSize. When a primitive is too large to fit into this limit, the originator MUST handle the primitive according to the segmentation mechanism as described in 6.10 Segmentation Mechanism.
- PlainTextCharset – the list of supported character sets for text/plain documents in the client device. Integer number assigned by IANA (see MIBenum numbers in [IANA]).
- ServerPollMin – integer value indicating the minimum time interval (in seconds) that MUST pass before two subsequent PollingRequest transactions.
- SessionPriority – an integer number from 0 (lowest) to 10 (highest) indicating the priority of the session for end-to-end messages – see OfflineETEMHandling. The user MAY give the same session priority to two clients. When more than one clients have identical SessionPriority value, it is up to the server implementation how to handle the situation.
- SupportedBearer – the list of supported bearers (HTTP(S), WSP, SMS)
- SupportedCIRMethod – the list of supported CIR methods that are supported by the client. The list MUST be in the order that the client prefers for selection of CIR method(s). When more than one CIR method has been agreed between the client and the server, the server MUST select, among the methods that are available in that moment, the one with highest priority. The client SHOULD at least support one SMS-bearer based CIR method in order to recover from situations such as loss of PDP context.
- SupportedOfflineBearer – the list of bearers for offline message notifications (WAP PUSH, SMS) that are supported by the client. The list MUST be in the order that the client prefers for selection of offline bearer(s). When more than one offline bearer has been agreed between the client and the server, the server MUST select, among the bearers that are available in that moment, the one with highest priority.
- UDPPort – the client MAY indicate that it requests other than the default port for the standalone UDP/IP CIR method. If the client indicates in the request that SUDP is supported, it MUST provide this value in the request as well. It is a decimal integer number. If the client indicates a value of 0 (zero), the server should deduce the IP address and the port number from the HELO message on the UDP/IP CIR channel. See [CSP Trans] for details.

The ClientCapabilityResponse primitive MUST contain the Agreed-Capabilities element that conveys the agreed capability information back to the client. The agreed capability information includes the following derived elements:

- AcceptedContentType – Structure(s) containing the media types and the related credentials of those content types that the SAP accepted to support. When the SAP accepts without any changes all of the AcceptedContentType credentials that have been specified by the client in the ClientCapabilityRequest, the server MUST NOT include this element in the ClientCapabilityResponse primitive – however if any changes are applied to any of the AcceptedContentType credentials, the server MUST include the entire list of the accepted content types, whereas the list MUST be empty when none have been accepted. The server MUST NOT change the ContentType, but it MAY change the AcceptedRichContentLength, ContentPolicy and ContentPolicyLimit, however not all changes are permitted. The

AcceptedRichContentLength in ClientCapabilityResponse MUST NOT be higher than the AcceptedRichContentLength in ClientCapabilityRequest. The permitted changes to the ContentPolicy are: ‘N’→’C’, ‘N’→’R’ and the server MUST add ContentPolicyLimit in both cases. When the ContentPolicy in the ClientCapabilityRequest is ‘R’, the ContentPolicyLimit in the ClientCapabilityResponse MUST NOT be higher than the ContentPolicyLimit in the ClientCapabilityRequest. The server MUST NOT include the text/plain content type – it is negotiated separately using PlainTextCharset and AcceptedTextContentLength. The server MUST include AcceptedContentType when the client requested Any-Content as ‘T’, but the server did not accept it – see description of Any-Content later on. If there is support for presence delivery on the server side, the SAP MUST publish all of the agreed content types along with the related credentials in the ClientInfo/ClientContentLimit/ AcceptedContentType presence attribute after the service negotiation has been completed. The structure contains the followings:

- ContentType – the MIME type of the media object.
- AcceptedRichContentLength – An integer number in character count that indicates the length of this particular media type that the SAP accepted to support without special conditions. See ContentPolicy for special conditions.
- ContentPolicy – An enumerated value that indicates whether if there is a special handling policy active regarding this media type. It MUST be one of the following values:
  - C – the SAP will add extra cost to the customer based on the extra size of the media content when the size of the media content is higher than the size defined in ContentPolicyLimit. The ContentPolicyLimit MUST be present.
  - N – No active policy meaning that the SAP does not limit this particular content type in any way. The ContentPolicyLimit MUST NOT be present.
  - R – the SAP will reject those instant messages and not deliver those presence attributes where the size of the media content is higher than the size defined in ContentPolicyLimit. The ContentPolicyLimit MUST be present.
- ContentPolicyLimit – An integer value that indicates the special handling policy limitation regarding this particular media type. The SAP MUST make sure that when ContentPolicyLimit specified, the ContentPolicy is either ‘C’ or ‘R’, and the ContentPolicyLimit value is higher than the AcceptedRichContentLength value.

- AcceptedPullLength – An integer number in character count that indicates the maximum length of content (either text/plain or multimedia object) that the SAP accepts to support for client-originated content delivery transactions. The SAP MUST accept or override the value. If the SAP accepts the value without changes, it MUST NOT include AcceptedPullLength in the ClientCapabilityResponse primitive. If the SAP overrides the value, the AcceptedPullLength in the ClientCapabilityResponse primitive MUST NOT be higher than the AcceptedPullLength in the ClientCapabilityRequest primitive. If there is support for presence delivery on the server side, the SAP MUST publish this value in the ClientInfo/ClientContentLimit/MaxPullLength presence attribute after the service negotiation has been completed.
- AcceptedPushLength – An integer number in character count that indicates the maximum length of content (either text/plain or multimedia object) that the SAP accepts to support for server-originated content delivery transactions. The SAP MUST accept or override the value. If the SAP accepts the value without changes, it MUST NOT include AcceptedPushLength in the ClientCapabilityResponse primitive. If the SAP overrides the value, the AcceptedPushLength in the ClientCapabilityResponse primitive MUST NOT be higher than the AcceptedPushLength in the ClientCapabilityRequest primitive. If there is support for presence delivery on the server side, the SAP MUST publish this value in the ClientInfo/ClientContentLimit/MaxPushLength presence attribute after the service negotiation has been completed.
- AcceptedTextContentLength – An integer number in character count that indicates the length of the text messages that the SAP accepts to support. It MUST be negotiated separately from the rich content types. The value MUST NOT be higher than AcceptedPullLength and AcceptedPushLength. The SAP MUST accept or override the value. If the SAP accepts the value without changes, it MUST NOT include AcceptedTextContentLength in the ClientCapabilityResponse primitive. If the SAP overrides the value, the AcceptedTextContentLength in the ClientCapabilityResponse primitive MUST NOT be higher than the AcceptedTextContentLength in the ClientCapabilityRequest primitive. If there is support for presence delivery on the server side, the SAP MUST publish this value in the

ClientInfo/ClientContentLimit/AcceptedTextContentLength presence attribute after the service negotiation has been completed.

- AcceptedTransferEncoding – The list of supported transfer-encoding methods – such as “BASE64” – that the SAP accepts to support. The SAP MUST either agree to support all of the requested transfer-encoding methods, or MUST agree to support only a common subset of transfer-encoding methods that are supported/allowed by the SAP.When the SAP agrees to support all of the requested transfer-encoding methods, the server MUST NOT include this element in the ClientCapabilityResponse primitive. When the SAP agrees to support only a common subset of transfer-encoding methods that are supported/allowed by the SAP, the server MUST include this element in the ClientCapabilityResponse primitive, whereas the list MUST be empty when there is no common subset of transfer-encoding methods. If there is support for presence delivery on the server side, the SAP MUST publish the AcceptedTransferEncoding list in the ClientInfo/ClientContentLimit/AcceptedTransferEncoding presence attribute after the service negotiation has been completed. When there is no common subset of transfer-encoding methods, rich content cannot be delivered using XML syntax, thus the server MUST NOT agree to support any rich content type (if applicable) – if such case occurs and rich content delivery is desired by the client, the client SHOULD disconnect and establish a new session using WBXML syntax instead.
- AnyContent – A Boolean value. If there is support for presence delivery on the server side, the SAP MUST publish the agreed value in the ClientInfo/ClientContentLimit/AnyContent presence attribute after the service negotiation has been completed.
    - When the client requested AnyContent as ‘T’, the SAP MUST agree to support all content types, or MUST agree to support only those content types that are supported/allowed by the server.
        - When the SAP agrees to support all content types, the server MUST NOT include AnyContent or AcceptedContentType in the response.
        - When the SAP agrees to support only the content types that are allowed/supported by the SAP, the SAP MUST include AnyContent with value ‘F’ and the list of supported content type credentials in AcceptedContentType in the response.
    - When the client requested AnyContent as ‘F’, the SAP MUST agree not to support any rich content types, and MUST NOT include AnyContent or AcceptedContentType in the response.
- CIRHTTPAddress – A URL used for standalone HTTP binding of CIR channel. See [CSP Trans] specification for description on how to use this binding.
- CIRSMSAddress – An MSISDN/MDN used for standalone SMS binding of CIR channel. See [CSP Trans] for description on how to use this binding.
- MultiTrans – The maximum number of open transactions from both client and server side at any given time. The value MUST be greater than 0, however it MUST NOT be greater than the value requested by the client in ClientCapabilityRequest.
- MultiTransPerMessage - Integer value indicating the maximum number of primitives that the client can handle within the same transport message, as well as the maximum number of open transactions from both client and server side at any given time. The value MUST be greater than 0, however it MUST NOT be greater than the value requested by the client in ClientCapabilityRequest.
- OfflineETEMHandling – enumerated value indicating how the server will handle end-to-end messages that have been addressed particularly to the requesting client after the client logged out or was disconnected. The SAP MUST either accept the OfflineETEMHandling requested by the client or change it to a more appropriate value. When the SAP accepts the requested OfflineETEMHandling setting, the server MUST NOT include this element in the ClientCapabilityResponse primitive. When the SAP does not accept the requested OfflineETEMHandling setting, the server MUST include this element in the ClientCapabilityResponse primitive, however not all changes are permitted. The permitted changes to the OfflineETEMHandling are:
        - ‘PRIORITYREJECT’→’REJECT’,
        - ‘PRIORITYSTORE’→’PRIORITYREJECT’,
        - ‘PRIORITYSTORE’→’REJECT’,
        - ‘SENDREJECT’→’REJECT’,

  - ‘SENDSTORE’→’SENDREJECT’,
  - ‘SENDSTORE’→’REJECT’.

- OnlineETEMHandling – enumerated value indicating how the server agrees to handle end-to-end messages that have not been addressed to any specific client of the user while the user is online. OnlineETEMHandling MUST NOT be included in the ClientCapabilityResponse primitive when the server agrees to provide the setting that was requested by the client in the ClientCapabilityRequest primitive, unless the “DETECT” value has been used when the server MUST return the current setting.
- ParserSize – the maximum character (byte) count of XML (WBXML, SMS - depending on the actual encoding) primitive size that the server-side parser can handle. The value MUST be smaller than the requested ParserSize value in the ClientCapabilityRequest primitive.
- PlainTextCharset – The list of character sets that the server accepts to support for text/plain documents in the client device. Integer number assigned by IANA (see MIBenum numbers in [IANA]). The SAP MAY accept all of the PlainTextCharset credentials not only when it supports it, but also when the SAP offers character set transcoding – however in this case the SAP MUST transcode the content to a character set that the client supports before the content is delivered to the client, as described in 5.6.2 Content transcoding. When the SAP accepts without any changes all of the PlainTextCharset credentials that have been specified by the client in the ClientCapabilityRequest, the server MUST NOT include this element in the ClientCapabilityResponse primitive – however if any changes are applied to any of the PlainTextCharset credentials, the server MUST include the entire list, whereas the list MUST be empty when none have been accepted. If there is support for presence delivery on the server side, the SAP MUST publish the AcceptedTransferEncoding list in the ClientInfo/ClientContentLimit/PlainTextCharset presence attribute after the service negotiation has been completed.
- ServerPollMin – integer value indicating the minimum time interval (in seconds) that MUST pass before two subsequent PollingRequest transactions. If the SAP accepts the value without changes, it MUST NOT include ServerPollMin in the ClientCapabilityResponse primitive.
- SupportedBearer – the list of supported bearers (HTTP(S), WSP, SMS) that both the client and server support.
- SupportedCIRMethod – the list of CIR methods that both the client and server support. The list MUST be in the prioritized order given by the client in the request. The server MUST at any time use the highest priority CIR method available, and the client SHOULD be able to receive CIR messages by means of all agreed methods.
- SupportedOfflineBearer – the list of bearers for offline message notifications (WAP PUSH, SMS), supported by both client and server that will be used in with this client. The list MUST be in the prioritized order given by the client in the request. The server MUST at any time use the highest priority offline bearer available, and the client SHOULD be able to receive offline notifications by means of all agreed methods.
- TCPAddress – If the client indicates that it supports STCP in the request and the server accepted it in SupportedCIRMethod, the server MUST provide an IP address for standalone TCP/IP CIR method described in [CSP Trans]. It is an IP address.
- TCPPort – If the client indicated that it supports STCP in the request and the server accepted it in SupportedCIRMethod the server MUST provide a port number if it is different from the default port for the standalone TCP/IP CIR method described in [CSP Trans]. Decimal integer number.
- UDPAddress – if the client indicates that it supports SUDP in the request and the server accepted it in SupportedCIRMethod, the server MUST provide an IP address for standalone UDP/IP CIR method described in [CSP Trans]. It is an IP address.
- UDPPort – If the client indicated that it supports SUDP in the request and the server accepted it in SupportedCIRMethod, the server MUST provide a port number if it is different from the default port for the standalone UDP/IP CIR method described in [CSP Trans]. It is a decimal integer number.
- UserSessionLimit – Integer number indicating the maximum number of total concurrent sessions for the user over all SAPs. The value MUST be equal or higher than the value in SAPSessionLimit. When there is no SAPSessionLimit specified, UserSessionLimit is not applicable, thus it MUST NOT be included. When SAPSessionLimit is specified however there is no such limitation on the number of overall sessions, UserSessionLimit MUST NOT be included in the response.

The client and server MUST support – with all capabilities related to it's supported features – client capability negotiation over any other transports than SMS. When SMS transport is utilized, the client and the server MUST support client capability negotiation, however the only negotiated capabilities for SMS transport are 'SessionPriority', 'ClientType', 'DefaultLanguage', 'MultiTrans', 'MultiTransPerMessage', 'OfflineETEMHandling', 'OnlineETEMHandling', 'ParserSize' and 'UserSessionLimit', as the rest of the capabilities are not applicable to SMS transport.

The client MAY and the server MUST support repeating the client capability negotiation during the session at any time.

The server MUST NOT provide capabilities that were not requested by the client. The server MUST NOT assume and use capabilities that are not supported by the client. The client MUST accept and store the agreed capabilities.

The ClientCapabilityResponse primitive MUST contain the list of capabilities that the server agrees to provide or have been updated by the server.

The server MUST use the agreed addresses and port numbers for CIR channel.

The following settings SHOULD be agreed between all clients/server:

- Generic settings:
  - ClientType,
  - DefaultLanguage,
  - MultiTrans,
  - MultiTransPerMessage,
  - OfflineETEMHandling,
  - OnlineETEMHandling,
  - ParserSize,
  - UserSessionLimit,
  - SessionPriority,
- Transport settings for CIR channel:
  - CIRHTTPAddress,
  - CIRSMSAddress,
  - SupportedCIRMethod,
  - TCPAddress,
  - TCPPort,
  - UDPAddress,
  - UDPPort,
- Transport settings for data channel:
  - ServerPollMin,
  - SupportedBearer,
  - SupportedOfflineBearer,
- IM/Presence related settings:
  - AcceptedContentType or AnyContent,
  - AcceptedPullLength,
  - AcceptedPushLength,
  - AcceptedTextContentLength,
  - AcceptedTransferEncoding,

  - InitialDeliveryMethod,
  - PlainTextCharset.

The generic settings are independent of other capability settings – except ParserSize, which MUST be higher than the content size limitations in AcceptedPullLength, AcceptedPushLength, AcceptedTextContentLength or any media type limitation in AcceptedContentType.

The transport settings for CIR channel are negotiated using the SupportedCIRMethod setting. When the server accepts STCP for CIR channel, it MUST provide an IP address to the client in TCPAddress, and if the port number is not the same as the default port number, then the server MUST provide a port number in TCPPort. When the server accepts SUDP for CIR channel, it MUST provide an IP address to the client in UDPAddress, and if the port number is not the same as the default port number, then the server MUST provide a port number in UDPPort. When the server accepts SHTTP for CIR channel, the server MUST provide an HTTP address in CIRHTTPAddress that is valid through the session. When the server accepts SSMS for CIR channel, the server MUST provide an MSISDN/MDN number in CIRSMSAddress that is valid through the session. After the agreements have been made, the client SHOULD establish the agreed connection-oriented CIR channels, and listen to CIR messages on the agreed connectionless and connection-oriented channels.

The transport settings for data channel are negotiated using the ServerPollMin, SupportedBearer, SupportedOfflineBearer settings. ServerPollMin MUST be agreed when the client intends to use the PollingRequest primitive. The SupportedBearer setting might seem informational since the session has been established already, but it is a valuable information for the client in case it needs to find another bearer when the currently used bearer becomes unavailable. SupportedOfflineBearer MUST be agreed to use offline notifications.

The IM/Presence related settings convey the content type limitations of the client/server that apply to both IM and presence – where text/plain or rich content can be added. Either AnyContent or AcceptedContentType(s) MUST be agreed, both settings MUST NOT be agreed. AcceptedPullLength MUST be agreed when the client desires to use client-originated content delivery transactions, as described in 5.6.4 Content delivery. AcceptedPushLength MUST be agreed when the client desires to use server-originated content delivery transactions, as described in 5.6.4 Content delivery. AcceptedTextContentLength and PlainTextCharset MUST be always agreed as text/plain is a mandatory content type. AcceptedTransferEncoding MUST be agreed when any other content type than text/plain was agreed, unless the encoding syntax in use is Binary XML - [CSP WBXML]. InitialDeliveryMethod MUST be agreed when any IM-Features are used.

6.9.2 Error conditions

Generic error conditions:

- Service unavailable. (503)
- Not logged in. (604)

ServiceRequest error conditions:

- Client-ID not matching this user. (422)
- Service provider agreement missing (907)

ClientCapabilityRequest error conditions:

- Client-ID not matching this user. (422)
- Service provider agreement missing (907)

6.9.3 Primitives and information elements

Primitive	Direction
ServiceRequest	Client → Server
ServiceResponse	Client ← Server
ClientCapabilityRequest	Client → Server
ClientCapabilityResponse	Client ← Server

Table 19: Primitive directions for service request and capability request

Information Element	Req	Type	Description
Message-Type	M	ServiceRequest	Message identifier.
Transaction-ID	M	String	Transaction identifier.
Session-ID	M	String	Identifier for the session.
Requested-Functions	C	Structure	Identifies the service elements and functions the client requests.
All-Functions-Request	M	Boolean	Request the server to send all services that it supports in the reply.

Table 20: Information elements in ServiceRequest primitive

Information Element	Req	Type	Description
Message-Type	M	ServiceResponse	Message identifier.
Transaction-ID	M	String	Transaction identifier.
Session-ID	M	String	Identifier for the session.
Not-Available-Functions	C	Structure	Identifies the delta of the client requested and what is available for that user.
All-Functions	C	Structure	Identifies all of the functions that the server supports.

Table 21: Information elements in ServiceResponse primitive

Information Element	Req	Type	Description
Message-Type	M	ClientCapabilityRequest	Message identifier.
Transaction-ID	M	String	Transaction identifier.
Session-ID	M	String	Identifier for the session.
Requested-Capabilities	M	Structure	Identifies the capabilities requested by the client.

Table 22: Information elements in ClientCapabilityRequest primitive

Information Element	Req	Type	Description
Message-Type	M	ClientCapabilityResponse	Message identifier.
Transaction-ID	M	String	Transaction identifier.
Session-ID	M	String	Identifier for the session.
Agreed-Capabilities	M	Structure	Identifies the capabilities agreed by the server.

Table 23: Information elements in ClientCapabilityResponse primitive

6.10 Segmentation Mechanism

The client MAY set a limit on the maximum primitive size using the ParserSize setting during client capability negotiation – see 6.9 Service and Capability Negotiation. There are numerous reasons why lists MAY grow long over the time, and eventually a list might not fit into the agreed limitations anymore. Similarly, the client is unable to determine the size of the requested presence information or public profiles before retrieval. The segmentation mechanism provides a workaround so that the excess information can be transferred as well.

The segmentation mechanism itself addresses only the lists and other information elements that are convenient to split. These information elements are:

- Public Profile,
- Contact lists,
- The list of instant messages on the server (Message-ID-List)
- Block/grant lists,
- Group-related lists:
    - Joined users' list(s),
    - Access list(s),
    - Members' list,
    - Reject list,
- Presence-related lists:
    - Authorization list(s),
    - Attribute list(s),
    - Presence information,
    - Watcher list.

Since splitting always happens at XML element boundaries, splitting a single terminating XML element is not possible. Thus for example, the content of an instant message cannot be split.

When the originator of a primitive encounters a primitive that is larger than ParserSize, it MUST either split the primitive or the response MUST indicate failure:

- When the segmentation mechanism was not agreed during service negotiation, the original transaction cannot be completed thus it MUST fail – see 11.3.22 432 – Response too large – or if it is a request, the sender MAY attempt to repeat the request using multiple smaller requests without segmentation.
- When the segmentation mechanism was agreed during service negotiation and
    - the primitive does not include any of the lists described above, the original transaction cannot be completed thus it MUST fail – see 11.3.22 432 – Response too large – or if it is a request, the sender MAY attempt to repeat the request using multiple smaller requests without segmentation.
    - the primitive includes one or more of the list types described above, the originator of the primitive MUST split the primitive according to the segmentation mechanism described below.

The segmentation MUST take place according to these steps:

- The originator MUST remove the excess of information – shorten the list(s) – from the original primitive so that it fits – including the Segment-Info element – into the agreed ParserSize limit. The originator MUST place the removed information into separate primitives – which are going to be the segments. A Segment-ID MUST be assigned to each segment. The Segment-ID MUST contain the Transaction-ID from the original primitive (the one that is being split up), and an integer number (SegmentReference) that identifies the individual segments – this integer number MUST be always zero for the first segment, and increase sequentially segment by segment. In order to guarantee that the terminating end is aware of the contents of the generated segments in advance, the originator MUST send at least one list element from each list that has been split.
- The originator MUST send the original primitive having the excess information removed normally including the Segment-Info element. The Segment-Info element MUST include the total number of segments and the Segment-ID of the first segment. This shortened original primitive MUST be treated as a segment – the SegmentReference MUST be zero, and it MUST be included in the total segment count.
- The terminating end notices that the primitive has been split on the originator side, thus it SHOULD retrieve the rest of the information using a series of Get Segment transactions. The server MUST always retrieve all segments from the client, and it MUST NOT perform the requested transaction until it has received all of the segments. The client SHOULD retrieve all segments as well. When then the client does not retrieve all segments, it MUST inform the server about this using the Drop Segment transaction.

The segments MUST NOT be persistent: all segments MUST be dropped upon logout or disconnect. All segments MUST be cached until the segment chain has been dropped using the DropSegmentRequest transaction or the client was disconnected for some reason.

The segmented data that is being transmitted MAY be changed while the transmission is still in progress. For example, a contact list might be updated from another client while it is being retrieved from the SAP. Such updates MUST NOT interfere with the segmentation mechanism – the sender MUST cache the segments with the state before the update until all segments have been transmitted or the segment chain has been dropped.

The following flowchart describes the entire decision tree for the originator:

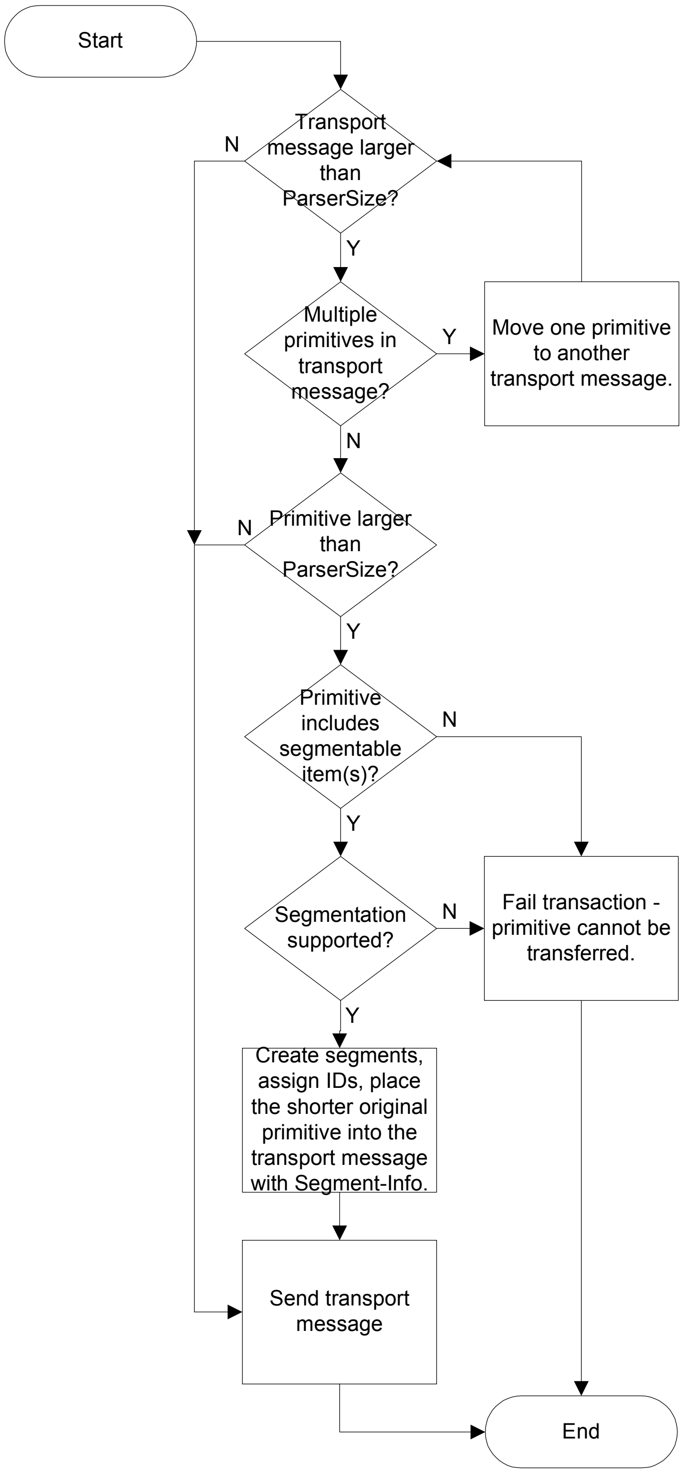
Start
Transport message larger than ParserSize?
N
Y
Multiple primitives in transport message?
Y
Move one primitive to another transport message.
N
Primitive larger than ParserSize?
N
Y
Primitive includes segmentable item(s)?
N
Y
Segmentation supported?
N
Fail transaction - primitive cannot be transferred.
Y
Create segments, assign IDs, place the shorter original primitive into the transport message with Segment-Info.
Send transport message
End

Figure 15: Decision tree for segmentation mechanism

The client and the server MAY support the segmentation mechanism including all related transactions – the related transactions MUST be negotiated as a whole. The service tree leaf that allows negotiation of the segmentation mechanism is 'SGMNT'.

6.10.1 Transactions

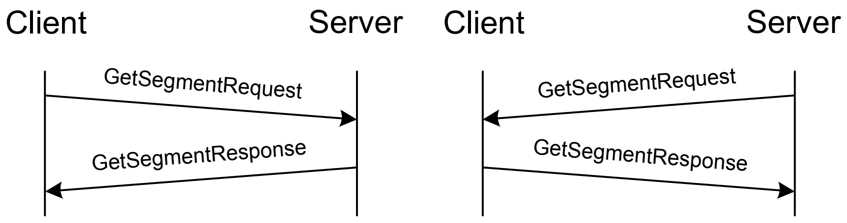

Figure 16: Get Segment transaction

The Get Segment transaction allows a client or a server to retrieve a segment from the other end. To retrieve a segment, the terminating end MUST send the GetSegmentRequest primitive to the originating end. The GetSegmentRequest primitive MUST include Segment-ID. The segments MUST be retrieved sequentially unless a segment has to be repeated for some reason.

When the transaction is successful, the GetSegmentResponse primitive MUST contain the Segment-ID and the Segment-Content. When the terminating end has retrieved all segments, the terminating end MUST perform Drop Segment transaction using a Segment-ID from the finished segment chain. Invalidating a segment means that it is marked as received by both the receiver and the sender and MUST NOT be requested again.

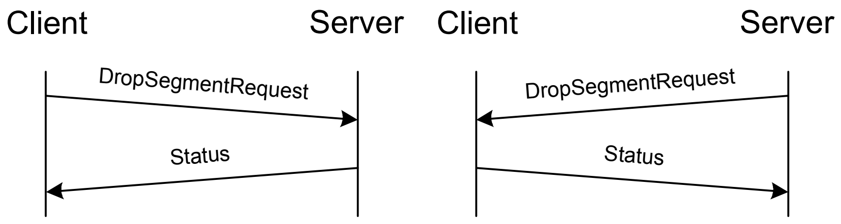

Figure 17: Drop Segment transaction

The Drop Segment transaction indicates to the originator that the terminating end does not need the segments anymore. The server MUST NOT perform the Drop Segment transaction until it has retrieved all segments from the client – the server MUST always retrieve all segments from the client to prevent possible data loss, unless the client indicates to the server that the transfer is not desired anymore. To indicate to the server that the transfer is not desired anymore, the client MUST respond with error code 580 or 581.

To inform the originator that the segments are not needed anymore, the terminating end MUST send the DropSegmentRequest primitive to the originator. The DropSegmentRequest primitive MUST include a Segment-ID – it is irrelevant whether the indicated segment has been retrieved already or not– the originatorMUST drop the whole segment chain related to the indicated Segment-ID and MUST respond with a Status primitive.

6.10.2 Error conditions

Generic error conditions:

- Service not agreed. (506)
- Not logged in. (604)

GetSegmentRequest error conditions:

- Unknown Segment reference (434)
- Segment index out of bounds (435)
- Segments dropped before delivery. (580)

- Segment dropped before delivery with rollback. (581)

DropSegmentRequest error conditions:

- Unknown Segment reference (434)

6.10.3 Primitives and information elements

Primitive	Direction
GetSegmentRequest	Client → Server
GetSegmentResponse	Client ← Server
DropSegmentRequest	Client → Server
Status	Client ← Server
GetSegmentRequest	Client ← Server
GetSegmentResponse	Client → Server
DropSegmentRequest	Client ← Server
Status	Client → Server

Table 24: Primitive directions for Segmentation mechanism

Information Element	Req	Type	Description
Message-Type	M	GetSegmentRequest	Message identifier.
Transaction-ID	M	String	Identifies the transaction.
Session-ID	M	String	Identifier for the session.
Segment-ID	M	Structure	Identifier for the segment to be retrieved and the segmented transaction.

Table 25: Information elements in Get Segment Request primitive

Information Element	Req	Type	Description
Message-Type	M	GetSegmentResponse	Message identifier.
Transaction-ID	M	String	Identifies the transaction.
Session-ID	M	String	Identifier for the session.
Segment-Content	M	Structure	The content of the requested segment.

Table 26: Information elements in Get Segment Response primitive

Information Element	Req	Type	Description
Message-Type	M	DropSegmentRequest	Message identifier.
Transaction-ID	M	String	Identifies the transaction.
Session-ID	M	String	Identifier for the session.
Segment-ID	M	Structure	Identifier for the segment to be dropped and the segmented transaction.

Table 27: Information elements in Drop Segment Request primitive

7. Common Features

The common features provide additional functionalities that are not closely related to any particular service – fundamental features, instant messaging, presence and group chat – but somewhat related to all of these.

In order to achieve minimum level of interoperability both the client and the server MUST support the following functionalities:

- 7.1 System Message
- 7.2 General Notification Transactions
- 7.3 Public Profile

The rest of the common features are all OPTIONAL. The individual client or server implementations MAY decide whether support for a particular transaction is implemented or not.

7.1 System Message

7.1.1 Transactions

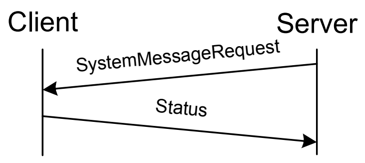

Figure 18: System Message Request transaction

The Server MAY send a System Message notification to the user using the SystemMessageRequest transaction. The SystemMessageRequest transaction supports sending a System Message notification to the client triggered by a server event. The server MAY also send a System Message notification in any response to a client-triggered event through the Result element in those transactions that contain a Result element 6.1 Status Primitive.

The System Message notification MUST contain a text element describing the System Message and it MAY contain a list of one or more answer option(s). The client MUST prompt the user with the text and give him/her the option of replying with the user-selected answer option. When there are multiple system messages in SystemMessageList, the client MUST process and prompt the user with the individual System Messages in the order they appear in the primitive. The server MAY sort the System Messages in SystemMessageList before sending according to its needs.

The server MAY require a response to the System Message(s) before allowing the user to continue accessing any IMPS services. When the server requires a conformant response to System Message(s), it MUST reject all requests from the client – except Keep Alive – until the user responds to the System Message. The server MUST reject the requests by sending a Status primitive that includes the Result element with error code 436, and the System Message(s) that MUST be responded in order to continue the session.

The Server MAY require verification from the client to make sure that the user has read and responded the System Message notification. The System Message structure contains a Verification-Mechanism element that defines the type of verification required in the System Message response. The following verification mechanisms are defined in this specification:

- InText – the text describes the information that the end-user MUST provide in the System Message response.
- URL - the client MUST fetch the content of the URL, which describes the information that the end-user MUST provide in the System Message response. The URL must refer to a content type supported by the client according the client capability negotiation.

Clients MUST support the InText method.

Clients MUST support the URL method, unless SMS transport is utilized.

The client MUST display the text or the content of the URL and ask the end-user to enter a corresponding verification text.

The server MAY utilize the verification text to request other input from the user as well - for example, to request the desired User-ID during the registration process.

The different types of System Message notifications are defined in the table below:

Requires response	Answer options	Used for	System Message Contents	User's response
Yes	None.	Key-based verification. The service provider wants the user to read, understand and give correct answer to something. (For example: 'Advise of charge', 'Terms and conditions', etc)	InText or URL, optional Verification-Mechanism and error code 436.	Optional Verification-Key.
	Exactly 1.	Confirmation. The service provider wants the user to read, understand, and confirm something using a single pre-defined value. (For example: 'Quota reached', etc.)	InText or URL, optional Verification-Mechanism, Answer-Options (with only one option) and error code 436.	Optional Verification-Key, and the only answer option included in the System Message.
	2 or more.	Confirmed selection. The service provider wants the user to read and understand something, make a choice and confirm the choice using a series of pre-defined values. (For example: 'New User-ID to be assigned.')	InText or URL, optional Verification-Mechanism, Answer-Options (two or more answer options) and error code 436.	Optional Verification-Key, and one of the answer options included in the System Message.
No	None.	Information. The service provider informs the user of some event, or gives advise to the user. (For example; Help, hints or advise based on usage patterns, etc.)	InText or URL, optional Verification-Mechanism.	Optional Verification-Key.
	Exactly 1.	Acknowledgement. The service provider sends some information for confirmation to the user, but the provider leaves the response up to the user's discretion. (For example: Generic service information such as expected upgrades, downtime, etc.)	InText or URL, optional Verification-Mechanism, Answer-Options (with only one option)	Optional Verification-Key, and the only answer option included in the System Message.
	2 or more.	Survey. The service provider is interested in the opinion of the user regarding some issues, but the provider leaves the response up to the user's discretion. (For example: Service improvement queries, etc.)	InText or URL, optional Verification-Mechanism, Answer-Options (two or more answer options)	Optional Verification-Key (free text entered by the user for verification), and one of the answer options included in the System Message.

Table 28: System Message types

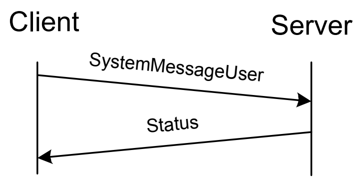

Figure 19: System Message User transaction

When the end-user has acted on a System Message notification, the client MUST send a SystemMessageUser primitive to the server containing the System Message response. The response MUST contain the end-user's response or the selected answer option. The client MAY also provide the System Message response in the LoginRequest to minimize latency when the server requires response before login to the service See Login transaction in 6.4 Logging In.

The SytemMessage-ID element of the response MUST match the SystemMessage-ID of the System Message notification.

The response MUST contain a Verification-Key element when the System Message notification included Verification-Mechanism. The Verification-Key element MUST contain the text string provided by the user in response to the notification.

If the server finds that the end-user's response is non-conformant to a System Message, it MAY re-send the System Message in the Status primitive any number of times, or it MAY send other System Message(s) in the Status primitive, or it MAY disconnect the client – using error code 607 – for a period of time using the TryAgainTimeout value in the status details.

The server MAY keep track of outstanding System Message notifications requiring response for an implementation specific period and resend those notifications to the end-user in future transaction if no response is received from the end-user. The server MAY also generate a new System Message notification (identified by a new SystemMessage-ID) if no response is received within an implementation specific period to the original notification.

The client and the server MUST support all transactions related to System Message, therefore the support for these transactions is not negotiated. All IMPS client implementations MUST be able to prompt to the end user with a System Message containing up to 128 characters. All IMPS client implementations SHOULD be able to prompt to the end user with a System Message containing up to 512 characters.

7.1.2 Error conditions

Generic error conditions:

- Not logged in. (604)

System Message Request error conditions:

- Verification Mechanism Not Supported. (439)

System Message User error conditions:

- Unknown System Message ID. (437)
- Incorrect Verification Key. (438)
- Too many non-conformant System Message replies (607)

7.1.3 Primitives and information elements

Primitive	Direction
SystemMessageRequest	Client ← Server
Status	Client → Server
SystemMessageUser	Client → Server
Status	Client ← Server

Table 29: Primitive directions for System Message transactions

Information Element	Req	Type	Description
Message-Type	M	SystemMessage Request	Message identifier.
Transaction-ID	M	String	Identifies the transaction.
Session-ID	M	String	Identifier for the session.
System-Message-List	M	Structure	System Message(s) from server

Table 30: Information elements in SystemMessageRequest primitive

Information Element	Req	Type	Description
Message-Type	M	SystemMessageUser	Message identifier.
Transaction-ID	M	String	Identifies the transaction.
Session-ID	M	String	Identifier for the session.
System-Message-Response-List	M	Structure	System Message Response(s) from user.

Table 31: Information elements in SystemMessageUser primitive

7.2 General Notification Transactions

7.2.1 Transactions

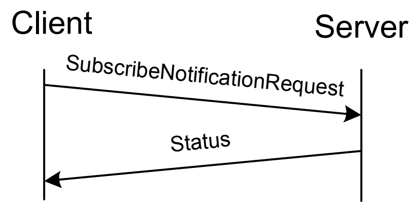

Figure 20: Subscribe Notification transaction

The client MAY be interested in receiving notifications about specific events on the server. To receive these notifications, the client MUST send a SubscribeNotificationRequest primitive to the server including the list of notification types that it would like to receive. If the list of types is not present, then all types are requested. The server MUST respond with a Status primitive.

The clients MUST NOT subscribe for services not agreed during service negotiation – e.g. if the usage of the invitation transaction where not agreed, the client MUST NOT subscribe to events related to invitation-related events.

After successful subscription, the server MUST send the requested notifications to the client as soon as events matching the subscribed notification types occur.

The subscription MUST NOT be persistent through different sessions – the server MUST remove all notification subscriptions for the particular client that logged out or was disconnected.

The server MUST differentiate notification subscription per client. The server MUST send exactly those notification types that the individual clients have subscribed to.

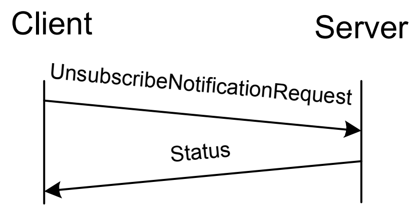

Figure 21: Unsubscribe Notification transaction

When the requesting client does not want to receive any more notifications, it MAY unsubscribe the notification by sending an UnsubscribeNotificationRequest primitive. Upon reception of such request the server MUST stop delivering notification for the notification types defined in the request. If no list of types is present, then all notification types MUST be un-subscribed. The server MUST respond with a Status primitive and stop sending the un-subscribed notifications.

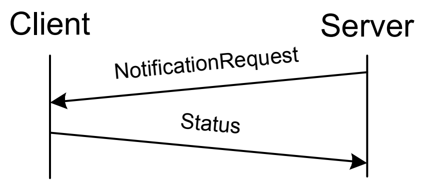

Figure 22: Notification transaction

If the client has subscribed to receiving general notifications from the server, the server MUST send NotificationRequest primitive to the client as soon as events matching the subscribed notification types occur. The server MUST only send out notifications of those types that the client has subscribed. The Client MUST respond with a Status primitive.

The different notification types are described in the table below:

Notification-Type	Requested elements	Description
Added-To-Contact-List	User-ID-List	Server MUST send notification whenever the user has been added to another users contact list and the DoNotNotify property of the Contact List to which the user has been added is false (‘F’).
Authorization-Changed	User-ID-List and/or Contact-List-ID-List and/or Default-List and Presence-Attribute-List	Server MUST send notification whenever another client (of the same user) makes changes to the authorizations.
Authorization-Needed-Contact-List	Contact-List-ID-List, User-ID-List and Presence-Attribute-List	Server MUST send notification whenever reactive authorization is necessary for a user on a Contact List that has attribute list assigned and the ContactList-Notify flag for the particular Contact-List is ‘T’. The ContactList-Notify flag MAY be managed using the CreateAttributeListRequest primitive, see 8.2.4.
Authorization-Needed-Default-List	Default-List, User-ID-List and Presence-Attribute-List	Server MUST send notification whenever reactive authorization is necessary for a user not proactively authorized in any way and the DefaultList-Notify flag for the Default-List is ‘T’. The DefaultList-Notify flag MAY be managed using the CreateAttributeListRequest primitive, see 8.2.4.
Authorization-Needed-User	User-ID-List and Presence-Attribute-List	Server MUST send notification whenever reactive authorization is necessary for a proactively authorized user and the User-Notify flag for the particular user is ‘T’ The User-Notify flag MAY be managed using the CreateAttributeListRequest primitive, see 8.2.4.
Block-List-Changed	None	Server MUST send notification whenever another client (of the same user) makes changes user’s BlockList. The notified client is responsible for retrieving the updated BlockList.

Notification-Type	Requested elements	Description
Block-List-UsageChange	Blocked-List-Inuse	Server MUST send notification whenever another client (of the same user) switches the usage flag of the BlockList. The notified client is responsible for updating its local cache (if applicable) with the change.
Contact-List-Created	ContactList-ID-List	Server MUST send notification whenever another client (of the same user) creates a contact list for the user. The notified client is responsible for retrieving the created contact list.
Contact-List-Changed	ContactList-ID-List	Server MUST send notification whenever another client (of the same user) makes changes to any of the user's contact list. The notified client is responsible for retrieving the updated contact list.
Contact-List-Deleted	ContactList-ID-List	Server MUST send notification whenever another client (of the same user) deletes any of the user's contact list(s). The notified client is responsible for updating its local cache (if applicable) with the change.
Grant-List-Changed	None	Server MUST send notification whenever another client (of the same user) makes changes to the user's GrantList. The notified client is responsible for retrieving the updated GrantList.
Grant-List-UsageChange	Granted-List-Inuse	Server MUST send notification whenever another client (of the same user) switches the usage flag of the GrantList. The notified client is responsible for updating its local cache (if applicable) with the change.
Group-Created	Group-ID	Server MUST send notification whenever another client (of the same user) creates a group for the user on the server.
Group-Deleted	Group-ID	Server MUST send notification whenever another client (of the same user) deletes a group of the user on the server.
Group-MemberAccess-Updated	Group-ID	Server MUST send notification to all Administrators and Moderators whenever an Administrator updates the access rights of any members in a group.
Group-Members-Updated	Group-ID	Server MUST send notification to all Administrators and Moderators whenever an Administrator or a Moderator updates the memberships in a group by adding/removing members.
Group-Membership-Granted	Group-ID	Server MUST send notification to all clients of a user who has been added to the members' list in a group.
Group-Membership-Revoked	Group-ID	Server MUST send notification to all clients of a user who has been removed from the members' list in a group.
Group-Removed	Group-ID	Server MUST send notification whenever a group of which the user is either a member of or joined to, have been deleted by the server.
Invitation-Accepted	Invite-ID	Server MUST send notification whenever another client (of the same user) accepts an invitation. The notified client is responsible for updating its local cache (if applicable) with the change.
Invitation-Cancelled	Invite-ID	Server MUST send notification whenever another client (of the same user) cancels an invitation. The notified client is responsible for updating its local cache (if applicable) with the change.
Invitation-Rejected	Invite-ID	Server MUST send notification whenever another client (of the same user) rejects an invitation. The notified client is responsible for updating its local cache (if applicable) with the change.

Notification-Type	Requested elements	Description
OnlineETEMHandling-Updated	OnlineETEMHandling	Server MUST send notification whenever another client (of the same user) updates the user's OnlineETEMHandling setting.
PublicProfile-Updated	None.	Server MUST send notification whenever another client (of the same user) updates the user's public profile. The notified client is responsible for retrieving the updated public profile.
Session-Priority-Adjusted	Session-Priority	Server MUST send notification to the client whenever it adjusts the session priority. The notified client is responsible for retrieving the new value from its ClientInfo presence attribute.
User-ID-Changed	User-ID-Pair	Whenever a user addresses another user in a transaction and the server does not recognize the requested User-ID, or there is an unrecognized User-ID in the user's private storage (contact list, block list, grant list, etc.), then the server SHOULD attempt to find out the new User-ID of the requested user when the server supports such feature. When the server has found the valid User-ID for the unrecognized User-ID, the server MUST send a notification to the client containing the unrecognized and the valid User-ID pair.

Table 32: General notification types

The client and the server MUST support all transactions related to the general notification, therefore the support for these transactions is not negotiated.

7.2.2 Error conditions

Generic error conditions:

- Not logged in. (604)
- Service not agreed. (506)
- Service not supported. (405)

SubscribeNotificationRequest error conditions:

- Invalid notification type. (433)
- Not allowed notification type. (440)

NotificationRequest error conditions:

- Client MAY ignore any error and respond with Successful. (200)

7.2.3 Primitives and information elements

Primitive	Direction
SubscribeNotificationRequest	Client → Server
Status	Client ← Server
UnsubscribeNotificationRequest	Client → Server
Status	Client ← Server
NotificationRequest	Client ← Server
Status	Client → Server

Table 33: Primitive directions for General Notification transactions

Information Element	Req	Type	Description
Message-Type	M	Subscribe NotificationRequest	Message identifier.
Transaction-ID	M	String	Identifies the subscription request transaction.
Session-ID	M	String	Identifier for the session.
Notification-Type-List	O	Structure	A list of notification types. An empty or missing list indicates all available types are desired.

Table 34: Information elements in SubscribeNotificationRequest primitive

Information Element	Req	Type	Description
Message-Type	M	Unsubscribe NotificationRequest	Message identifier.
Transaction-ID	M	String	Identifies the subscription request transaction.
Session-ID	M	String	Identifier for the session.
Notification-Type-List	O	Structure	A list of notification types. An empty or missing list indicates all available types are desired.

Table 35: Information elements in UnsubscribeNotificationRequest primitive

Information Element	Req	Type	Description
Message-Type	M	NotificationRequest	Message identifier
Transaction-ID	M	String	Identifies the transaction.
Session-ID	M	String	Identifies the session
Notification-Type	M	Enumerated string	Indicates the type of notification.
Blocked-List-Inuse	C	Boolean	Indicates if the list of blocked entities is currently in use (active).
Group-ID	C	String	Identifies the group: ▪ of which the user is either a member of or joined to, have been deleted by the server, or ▪ of the user that has been created/deleted by the user, or ▪ of the Administrator for which the member access rights have been updated, or ▪ of the Administrator or Moderator for which the memberships have been updated, or ▪ for which the user has been granted/revoked membership..
ContactList-ID-List	C	Structure	Identifies the contact list(s): ▪ which have been created, or ▪ which have been changed, or ▪ which have been deleted, or ▪ for which the authorizations

Information Element	Req	Type	Description
			have been updated, or ▪ for which additional reactive authorization is needed.
Default-List	C	Boolean	Indicates whether: ▪ the attributes included in Presence-Attribute-List have been applied to the Default-List, or ▪ additional reactive authorization is needed for the Default-List.
Granted-List-Inuse	C	Boolean	Indicates if the list of granted entities is currently in use (active).
Invite-ID	C	String	Identifies the invitation that has been: ▪ accepted, or ▪ rejected, or ▪ cancelled.
OnlineETEMHandling	C	String	The newly set OnlineETEMHandling setting for the user.
Presence-Attribute-List	C	Structure	The presence attribute list that have been assigned to the indicated user(s), contact list(s) and/or the Default-List.
Session-Priority	C	Integer	Indicates the newly assigned session priority value.
User-List	C	Structure	Identifies the user(s): ▪ who added the user to their contact list(s), or ▪ to whom the presence authorizations have been updated, or ▪ to whom additional reactive authorization is needed.
User-ID-Pair	C	Structure	Contains the unrecognized and the valid User-ID pair that the server has successfully discovered.

Table 36: Information elements in NotificationRequest primitive

7.3 Public Profile

The public profile is a user's profile information that MUST be available to the public. The public profile MUST be maintained by the owner user and MUST be searchable with some restrictions.

In order to achieve minimum level of interoperability both the client and server MUST support the following functionalities:

- Retrieve public profile – see the transaction later on in this chapter.
- Clear and Update public profile – see the transaction later on in this chapter.
- Delivery of Friendly Name with UserIDs – see 5.3.1 Addressing introduction.

Additionally to the above, the client and the server MAY support the following functionalities:

- Search based on public profile – see the 7.4 General Search Transactions

An empty value for a public profile field MUST be treated as if it was not filled in.

The XML Syntax DTD - see [CSP XMLS] - has been defined in a manner that allows custom public profile fields. The client and the server MAY support these custom fields, but MUST ignore (without generating an error) the properties that they are not able to process. For extending the public profile with custom fields, see 7.3.1 Extending the Public profile. The IMPS protocol defines the fields described in Table 37. An empty value for a public profile field in all related transactions MUST be treated as if it was not filled in.

The client and the server MUST support all of the predefined fields and the assigned keys defined in Table 37. The 'Req' column in the following table describes whether the specific field is REQUIRED ('M') or OPTIONAL ('O') for using the public profile.

Field	Req	Key	Searchable	Comment
Age	M	PP_AGE	Y	The year and month of birth. Format: YYYYMM, where YYYY – decimal number indicating the year. MM – decimal number indicating the month. Default value: N/A – it MUST be specified by the user.
Country	M	PP_COUNTRY	Y	The country of residency. While filling in the public profile, the client MAY automatically fill this field for the user based on the MSISDN. Format: Two-letter "Alpha-2" format as defined in [ISO3166-1]. Default value: N/A– it MUST be specified by the user.
Friendly Name	M	PP_FRIENDLY_NAME	Y	A name that the user suggests for the general public to use as a nickname and is available to all users at all times. Format: Free text up to 50 characters. Default value: N/A– it MUST be specified by the user.
City	O	PP_CITY	Y	The city of residency. Format: Free text up to 50 characters. Default value: Empty value.
Free text	O	PP_FREE_TEXT	N	Any kind of information that the user wishes to disclose to the public about himself/herself. Format: Free text up to 200 characters. Default value: Empty value.
Gender	O	PP_GENDER	Y	The gender of the user. Format: either one of the following enumerated values: F – Female M – Male U – Unspecified (the user did not disclose it) Default value: U
Intention	O	PP_INTENTION	Y	The intention of the user. Format: Free text up to 100 characters. Default value: Empty value.
Interests/hobbies	O	PP_INTERESTS	Y	Interests/hobbies of the user. Format: Free text up to 100 characters. Default value: Empty value.

Field	Req	Key	Searchable	Comment
Marital status	O	PP_MARITAL_STATUS	Y	The marital status of the user. Format: either one of the following enumerated values: C – Cohabitant D – Divorced E – Engaged M – Married S – Single U – Unspecified (the user did not disclose it) W – Widowed Default value: U

Table 37: Public Profile standard fields

7.3.1 Extending the Public profile

The extension creator MUST:

- Consider if the extension belongs to the public profile. There are several issues to consider, however it is RECOMMENDED to evaluate the followings:
  - Dynamics. The public profile is not updated to the clients automatically, thus it SHOULD include only fields that are more or less static. For frequently updated information the inventor MUST use presence extension mechanism instead. See [PA] for more information for extending presence attributes.
  - Privacy. The information included in the public profile is available to every IMPS client, thus the inventor MUST evaluate the sensitivity of the disclosed information. The created public profile field must conform to [PRIVACY].
  - Contents. The public profile fields allow only text/plain format. The value in a single public profile field MUST NOT exceed 200 characters – the server MUST cut the excess part out and include only the first 200 characters in the public profile field.
  - Support. All of the custom fields are OPTIONAL. Note that it is possible that an IMPS client or server implementation might not support the public profile field, and MAY ignore it – thus the field SHOULD NOT include information that MUST reach the end-user.
- Define a unique identifier. To guarantee uniqueness, the inventor MUST add a short prefix to the name of the field that reflects the inventor's credentials. The prefix MUST be terminated with a hash mark ('#') character. The client MUST NOT render the text before the first hash mark ('#') character on the display – nor the hash mark itself. The rest of the string MUST be rendered on the display – however it MAY be un-localized. For example, a company named 'Foo Industries' would like to create an extension that includes the user's education level: the field name could be:'FOOI#Education level' –and the receiving client will render it as 'Education level'.

7.3.2 Transactions

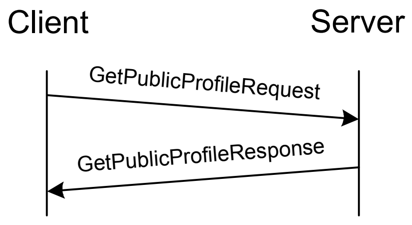

Figure 23: Retrieve public profile transaction

A user MAY retrieve the public profile of any user including his/her own. To retrieve public profile(s) of user(s), the client MUST send the GetPublicProfileRequest primitive to the server containing the UserIDs of those users whose public profile is desired. The server MUST reply with a GetPublicProfileResponse primitive that contains the public profiles of the requested users.

The service provider MAY restrict the retrieval of public profile by imposing the following rule: the requesting user MUST have the mandatory fields of his/her own public profile (see Table 37) filled. If the service provider uses this restriction and the requesting user's mandatory fields are empty, attempting to retrieve another user's public profile will result in failure. In that case the server MUST respond with a Status primitive except when the user is requesting to retrieve his/her own public profile. The Status primitive the MAY also contain a System Message that describes to the user that he/she SHOULD fill in the public profile as well as explaining the privacy issues related to filling in the Public Profile.

The server SHOULD include the requested public profiles of those users that have filled out the mandatory fields in their public profile. The server MUST NOT include the public profile of those users who did not fill in the mandatory fields of their public profile – but it MUST indicate these users with an error code in the Result element.

The server MUST allow the user to retrieve his/her own public profile even if it is empty.

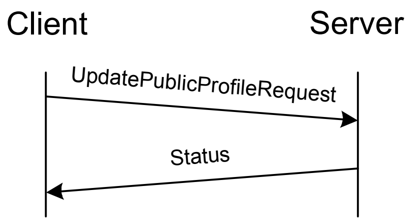

Figure 24: Clear/update public profile transaction

A user MAY clear and update his/her own public profile. To clear the public profile, the Clear-Public-Profile element can be utilized. To update the public profile, the fields that need update MUST be included in the Public-Profile element. The client MUST sent the UpdatePublicProfileRequest primitive to the server, and the server MUST reply with a Status primitive.

The transaction MUST fail when the requesting user does not have the mandatory fields of his/her own public profile filled and the mandatory fields are not included in the request primitive. The Status primitive MAY also contain System Message that describes to the user that he/she SHOULD fill in the public profile as well as explaining the privacy issues related to filling in the Public Profile.

The client MUST NOT send a request that includes Clear-Public-Profile element with 'F' (false) value and the whole Public-Profile element is missing.

If the Clear-Public-Profile element indicates 'T' (true) the server MUST – with the exception of the Friendly Name field – clear the entire public profile. The default values are to be restored. The default values are described in Table 37: Public Profile standard fields. If the Clear-Public-Profile element indicates 'F' (false), clear operation was not requested, thus the server MUST NOT clear the profile.

If both clearing and update operations have been requested, the server MUST perform the clear operation first – before the update. The server MUST update those public profile fields that are included in the Public-Profile element. If the whole Public-Profile element is missing, update operation was not requested, thus the server MUST NOT update any of the public profile fields.

Whenever the public profile of the user has been updated or cleared, the server MUST send notification(s) of the event to all other subscribed clients of the user using the General Notification Mechanism; see 7.2 General Notification Transactions.

7.3.3 Error conditions

Generic error conditions:

- Service unavailable. (503)
- Not logged in. (604)

GetPublicProfileRequest error conditions:

- Unknown user ID. (531)
- Missing mandatory profile field(s) of requesting user (904)
- Public profile of requested user is not available(905)
- Too many public profiles requested (201/906)
- Service provider agreement missing (907)

UpdatePublicProfileRequest error conditions:

- Missing mandatory field(s)of requesting user. (904)
- Number of characters exceeds the maximum number of characters. (441)
- Wrong value type. (442)

7.3.4 Primitives and information elements

Primitive	Direction
GetPublicProfileRequest	Client → Server
GetPublicProfileResponse	Client ← Server
UpdatePublicProfileRequest	Client → Server
Status	Client ← Server

Table 38: Primitive directions for public profie management transactions

Information Element	Req	Type	Description
Message-Type	M	GetPublicProfile Request	Message identifier.
Transaction-ID	M	String	Identifies the transaction.
Session-ID	M	String	Identifier for the session.
User-ID-List	M	Structure	Identifies the user(s) whose public profile is requested.

Table 39: Information elements in GetPublicProfileRequest primitive

Information Element	Req	Type	Description
Message-Type	M	GetPublicProfile Response	Message identifier.
Transaction-ID	M	String	Identifies the transaction.
Segment-Info	O	Structure	Contains segmentation information when the primitive is segmented.
Session-ID	M	String	Identifier for the session.
Result	M	Structure	The result of the request.
Public-Profile-List	M	Structure	The list of Public profiles per UserID.

Table 40: Information elements in GetPublicProfileResponse primitive

Information Element	Req	Type	Description
Message-Type	M	UpdatePublicProfile Request	Message identifier.

Transaction-ID	M	String	Identifies the transaction.
Session-ID	M	String	Identifier for the session.
Clear-Public-Profile	M	Boolean	Indicates whether the public profile is cleared or not.
Public-Profile	O	Structure	The public profile fields to be updated for the requesting user.

Table 41: Information elements in UpdatePublicProfileRequest primitive

7.4 General Search Transactions

7.4.1 Transactions

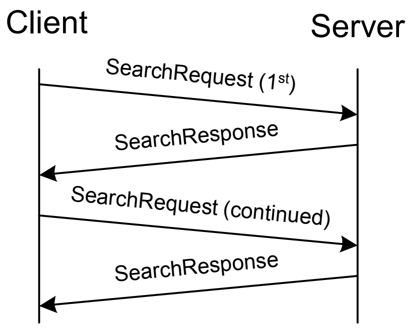

Figure 25: General search transactions

A user MAY search for users/groups based on various user/group properties. On protocol level there is no difference between searching for users/groups based on various user/group properties – the separation is solely logical based on the search criteria: if the requested search criteria cannot be processed, the server will simply deny the request. The user MAY limit the number of results to be retrieved at a time, and MAY continue the search and go through all the results.

The search is performed using a list of one or more Search-Pairs. The Search-Pair list MUST be included in the first request only, it MUST NOT be included in the subsequent requests.

A Search-Pair consists of a Search-Element and a Search-String. The Search-Element indicates which property of the user/group SHALL be searched for the Search-String. When more than one search pairs are specified in the primitive without using advanced search, logical AND operation MUST be employed between the different pairs. The advanced search feature allows defining a more flexible search criteria. The advanced search is described in details in 7.4.2 Advanced search mechanism. Every Search-Element MUST NOT be present more than once within the same search request, unless advanced search is used.

The service provider MAY restrict the searching based on public profile by imposing the following rule: the requesting user MUST have the mandatory fields of his/her own public profile (see Table 37) filled. If the service provider uses this restriction and the requesting user's mandatory fields are empty, the IMPS server MUST NOT allow searching based on Public Profile, and the transaction MUST fail. In this case the server MUST respond with a Status primitive. The Status primitive MAY also contain a System Message that describes to the user that he/she SHOULD fill out the public profile as well as explaining the privacy issues related to filling in the Public Profile.

The result of a user search MUST always be User-ID and Friendly Name. If the Friendly Name exists in the public profile of the user then IMPS server MUST include the Friendly Name from the public profile, otherwise the UserID MUST be included in the Friendly Name field instead. Similarly, the result of a group search MUST always be Group-ID. The server MUST guarantee that the search results are matching the search criteria.

When searching for users based on public profile fields, the server:

- MUST NOT include those users in the results who have not filled out the mandatory fields in their public profile.

- MUST verify the age field in the public profile of the users that have been found matching according the Search-Pair-List element, and:
  - if there is a local age limitation policy – to provide child protection – in the requesting user's home domain, the server MUST exclude those users from the search results who are under the age limitation according to their public profile. When a server extends the search to other servers, the remote servers MUST NOT return the protected users as results – see [SSP].
  - if there is no local age limitation policy in the requesting user's home domain, the server MUST include all matching users in the search results. When a server extends the search to other servers, the remote servers MUST NOT return the protected users as results – see [SSP].

While searching based on public profile, the local age limitation refers to the age limitation setting on the home domain of the user who is queried as a possible search result.

When searching for groups, the server MUST compare the age from the public profile of the requesting user versus the age limitation of the groups that have been found matching according the Search-Pair-List element, and:

- if the age is missing from the public profile, return all groups in the search results.
- if the age field is present in the public profile, return only those groups that do not have age limitation, or the age limitation is smaller or equal than the requested user's age.

It is possible to search based on combined private profile and public profile search elements. The server MUST apply the rules/restrictions from both private profile based search and public profile based search.

When searching for groups and a restricted group is found, the server MUST check that the searching user is a group member (user, moderator or administrator of the group) before this search result is regarded as valid and the Group-ID is delivered to the searching user.

When the searching user owns the matching group, the group property "Searchable" MUST be ignored and the group MUST be treated as searchable.

Search-Element	Search-String
USER_AGE_MIN	The *Search-String* is a decimal integer indicating a minimum age in years. The search results MUST be those users whose age in their public profile is equal or higher than the specified value – unless the server implements local age limitation policy. When both USER_AGE_MIN and USER_AGE_MAX are specified as search criteria, USER_AGE_MAX MUST be equal or higher than USER_AGE_MIN and the search results MUST be those users whose age in their public profile is within the range specified by the USER_AGE_MIN and USER_AGE_MAX values (e.g. USER_AGE_MIN<=user's age<= USER_AGE_MAX) – unless the server implements local age limitation policy.
USER_AGE_MAX	The *Search-String* is a decimal integer indicating a maximum age in years. The search results MUST be those users whose age in their public profile is equal or lower than the specified value – unless the server implements local age limitation policy. When both USER_AGE_MIN and USER_AGE_MAX are specified as search criteria, USER_AGE_MAX MUST be equal or higher than USER_AGE_MIN and the search results MUST be those users whose age in their public profile is within the range specified by the USER_AGE_MIN and USER_AGE_MAX values (e.g. USER_AGE_MIN<=user's age<= USER_AGE_MAX) – unless the server implements local age limitation policy.
USER_COUNTRY	The *Search-String* is used to search public profiles based on the 'Country' (PP_COUNTRY) definition in Table 37 .
USER_FRIENDLY_NAME	The *Search-String* is used to search public profiles based on the 'Friendly Name' (PP_FRIENDLY_NAME) definition in Table 37.

Search-Element	Search-String
USER_CITY	The *Search-String* is used to search public profiles based on the 'City' (PP_CITY) definition in Table 37.
USER_GENDER	The *Search-String* is used to search public profiles based on the 'Gender' (PP_GENDER) definition in Table 37. Using the 'U' (Unspecified) value is NOT RECOMMENDED – clients SHOULD exclude the criteria instead.
USER_INTENTION	The *Search-String* is used to search public profiles based on the 'Intention' (PP_INTENTION) definition in Table 37. It MAY be a sub-string.
USER_INTERESTS_HOBBIES	The *Search-String* is used to search public profiles based on the 'Interests/hobbies' (PP_INTERESTS) definition in Table 37. It MAY be a sub-string.
USER_MARITAL_STATUS	The *Search-String* is used to search public profiles based on the 'Marital status' (PP_MARITAL_STATUS) definition in Table 37. Using the 'U' (Unspecified) value is NOT RECOMMENDED – clients SHOULD exclude the criteria instead.

Table 42: Search elements for public profile-based user search

Search-Element	Search-String
USER_ALIAS	The *Search-String* is a substring of a user's *alias*.
USER_ONLINE_STATUS	The Search-String is the online status value.
USER_EMAIL_ADDRESS	The *Search-String* is a substring of a user's e-mail address.
USER_FIRST_NAME	The *Search-String* is a substring of a user's firstname.
USER_ID	The *Search-String* is a substring of a User-ID.
USER_LAST_NAME	The *Search-String* is a substring of a user's lastname.
USER_MOBILE_NUMBER	The *Search-String* is a mobile number. [E.164]

Table 43: Search elements for private profile-based user search

Search-Element	Search-String
GROUP_ID	The *Search-String* is a substring of a Group-ID.
GROUP_NAME	The *Search-String* is a substring of a group's name (part of group properties).
GROUP_TOPIC	The *Search-String* is a substring of a group's topic (part of group properties).
GROUP_USER_ID_JOINED	The *Search-String* MUST be the searching user's own User-ID. Search result MUST contain the list of groups that the requesting user is joined to
GROUP_USER_ID_OWNER	The *Search-String* MUST be the searching user's own User-ID. Search result MUST contain the list of groups owned by the requesting user.
GROUP_USER_ID_AUTOJOIN	The *Search-String* MUST be the searching user's own User-ID. Search result MUST contain the list of groups that have AutoJoin property set to "T" for the requesting user.

Table 44: Search elements for group search

The first SearchRequest message that the client sends to the server:

- MUST contain Search-Pair-List and the Search-Limit (maximum number of results at a time).
- if it contains more than one Search-Pair-List, the Search-Element MUST be different in each Search-Pair-List, however, all of the Search-Elements MUST be the of the same type i.e. for user or group.
- MUST NOT contain Search-ID and Search-Index.

The first SearchResponse that the server sends to the client MUST contain the Result of the search. If the search was successful, the SearchResponse message MUST also contain the Search-ID, the Search-Index (a continuation index to indicate where the search SHOULD be continued), the Search-Findings (the number of items found that match the criteria so far), and the Search-Results (the actual data).

When the server does not support all of the search-elements that the client included in the first SearchRequest primitive, the request MUST fail and the server MUST respond with a Status primitive indicating error code 560. To inform the client about the non-supported search elements, the server MUST return the entire list of non-supported search elements within the DetailedResult element where each non-supported search element MUST be indicated using code 562. The server MAY also inform the client about the supported search-elements by including the supported search elements within the DetailedResult element where each supported search element MUST be indicated using code 561.

When the search request includes presence elements, the server MUST verify the authorization rules of the requested presence attributes for the searching user. If the searching user is not authorized to see the requested presence values for a user, then the server MUST NOT trigger reactive authorization and it MUST NOT include those user in the search results. This behavior allows publishers to expose certain presence attributes via searching – using their default attribute list –, while protecting the publishers from possible undesired presence authorization requests.

The server MUST provide means to continue a successful search, so that the user MAY continue the search. In this case the SearchRequest message MUST include the Search-ID and the Search-Index – which were assigned by the server in the first SearchResponse –, and MUST NOT include Search-Pair-List or Search-Limit. The client MAY modify the Search-Index value, so that the search MAY be continued at a different place – this allows browsing through the results forth and back. The Search-Index is valid until a new search is performed or the session ends (a previous search MUST be invalidated when a new search is started) or the search has been stopped using the StopSearch transaction. The server MUST then respond with the SearchResponse similar to the one responded to the first SearchRequest but the subsequent SearchResponses MUST NOT contain Search-ID. Since the server MAY find more and more results while the client is retrieving different parts of the results, and the value of the Search-Findings MAY be increasing - however it MUST NOT be decreasing.

The client SHOULD send StopSearchRequest to the server when the search is not needed anymore.

If a search is not successful, the Search-Results element of the SearchResponse primitive MUST indicate failed search, and the primitive MUST NOT contain any of the conditional information elements.

The client and the server MAY support searching based on various user and group properties. The service tree leaf that allows negotiation of this transaction is 'SRCH'.

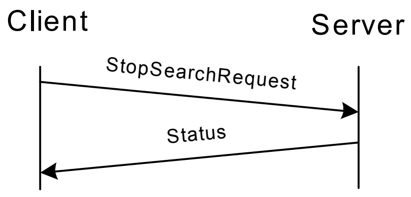

Figure 26: Stop search transaction

In order to indicate to the server that results are no further REQUIRED from a previously issued search request, the client MAY send StopSearchRequest primitive to the server containing the Search-ID. The server MUST invalidate the indicated search, and reply with a Status primitive. The invalidated Search-ID cannot be used after invalidation.

The client and the server MAY support stop search transaction, and it SHOULD be supported if the general search transaction is supported. The service tree leaf that allows negotiation of this transaction is 'STSRC'.

7.4.2 Advanced search mechanism

The advanced search mechanism allows a user to compose a free-form search criteria where not only logical AND, but also logical NOT and OR operations as well as nesting is allowed. The advanced search mechanism uses exactly the same transactions as the general search transaction, thus it inherits the same requirements – these requirements are not described here, please refer to 7.4.1 Transactions. The only difference between the advanced search and the general search mechanism is that when advanced search is performed, the 1st SearchRequest primitive MUST include additional information – the rest of the search mechanism is identical, including the StopSearch transaction.

In order to perform an advanced search, the 1st SearchRequest primitive MUST include:

- at least two search pairs in the Search-Pair-List information element, and each search pair in the Search-Pair-List MUST have a unique identifier in the scope of a single SearchRequest primitive. The unique identifier MUST be an integer number.
- the Advanced-Criteria information element.

The Advanced-Criteria information element MUST describe the logical relationship of the search pairs according to its syntax defined in [CSP DataType]. The 1st SearchRequest primitive MUST NOT include search pairs that are not used in the Advanced-Criteria.

An example

> User wants to find his friend whose last name he knows, however he is not sure about how he registered his first name. The search elements will be:
> USER_LAST_NAME = "Smith" with ID assigned to 0.
> USER_FIRST_NAME = "John" with ID assigned to 1.
> USER_FIRST_NAME = "Johnny" with ID assigned to 2.
>
> The logical expression will be:
> 0+[1|2]
>
> The server – upon receiving this request – will return all User-IDs and Friendly-Names that have "Smith" as their last name, and either "John" or "Johnny" as their first name.

The client and the server MAY support advanced search. The service tree leaf that allows negotiation of this transaction is 'ADVSR'. All clients and server that support advanced search MUST support the general search mechanism and the applicable requirements described in 7.4.1.

7.4.3 Error conditions

Generic error conditions:

- Service not supported. (405)
- Service unavailable. (503)
- Service not agreed. (506)
- Not logged in. (604)

SearchRequest error conditions:

- Unable to parse criteria (Invalid Search-Element). (402)
- Initial search request was not sent (Invalid Search-ID). (424)
- Invalid Search-Index (out of range). (425)
- Search timeout (in case of continued search the subsequent request primitive is late). (535)
- Too many hits (536)
- Too broad search criteria (537)
- Too many elements in advanced criteria (544)
- Too many levels of nesting in advanced criteria (545)
- Unsupported search-element was requested. (560)
- Supported search-element. (561)
- Unsupported search-element. (562)
- Missing mandatory elements of requesting user (904)
- Public profile of requested user is not available (905)
- Service provider agreement missing (907)

StopSearchRequest error conditions:

- Initial search request was not sent (Invalid Search-ID). (424)

7.4.4 Primitives and information elements

Primitive	Direction
SearchRequest	Client → Server
SearchResponse	Client ← Server
StopSearchRequest	Client → Server
Status	Client ← Server

Table 45: Primitive directions for searching

Information Element	Req	Type	Description
Message-Type	M	SearchRequest	Message identifier.
Transaction-ID	M	String	Identifies the transaction.
Session-ID	M	String	Identifier for the session.
Search-Pair-List	C	Structure	Searching criteria in terms of user or group properties. For advanced searches, each search pair MUST have its own unique identifier included. Present only in the 1st search request.
Advanced-Criteria	C	String	The advanced search criteria in form of a logical expression.
Search-Limit	C	Integer	Indicates the number of maximum search results can be received at a time. Present only in the 1st search request.
Search-ID	C	String	Uniquely identifies a search transaction. The server assigns this ID when the first search is performed, thus not present in the 1st search request.
Search-Index	C	Integer	Indicates that the results SHALL be sent starting from this particular index. Present only when the search is continued.

Table 46: Information elements in SearchRequest primitive

Information Element	Req	Type	Description
Message-Type	M	SearchResponse	Message identifier.
Transaction-ID	M	String	Identifies the transaction.
Session-ID	M	String	Identifier for the session.
Search-ID	C	String	Uniquely identifies a search transaction. The server assigns this ID when the 1st search is performed successfully.
Search-Findings	M	Integer	Indicates the number of the current findings.
Completion-Flag	M	Boolean	Indicates whether the client can expect new results. ‘F’ if server MAY provide new results (still searching), ‘T’ if new results will not be provided.
Search-Index	C	Integer	Indicates the particular index from which the next search can start. This provides the information to the user where to continue the

			search.
Search-Results	C	Structure	Search results.

Table 47: Information elements in SearchResponse primitive

Information Element	Req	Type	Description
Message-Type	M	StopSearchRequest	Message identifier.
Transaction-ID	M	String	Identifies the transaction.
Session-ID	M	String	Identifier for the session.
Search-ID	M	String	Identifies the search to be invalidated.

Table 48: Information elements in StopSearchRequest primitive

7.5 Invitations

7.5.1 Transactions

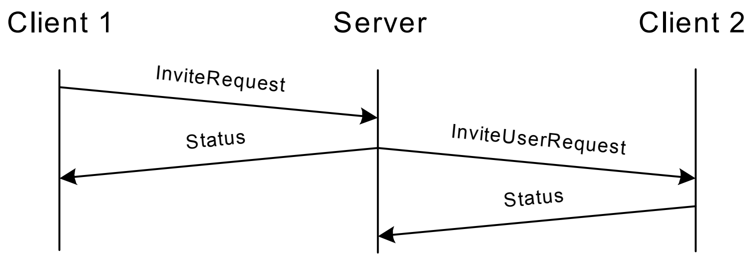

Figure 27: Invite user(s)

A user MAY invite other user(s). The subject of the invitation MAY be an end-to-end application, joining a group, exchanging instant messages, a shared content, sharing presence information, group membership request or *Private Group Conversation*. Note that the transaction is only informational: the invitation action is not taken; the user MAY do it manually separately.

To invite other users, the client MUST send the InviteRequest primitive to the server containing the ID of the invitation, the 'Type' of the invitation, the ID of the subject, Recipient and Sender elements. The InviteRequest primitive MAY include Invite-Reason containing a short text describing the reason for the invitation and Own-Screen-Name containing the inviting user's own screen-name. The Own-Screen-Name element MUST NOT be used for identifying purpose: it is purely informational. The ID of invitation (Invite-ID) MUST be assigned by the inviting client and MUST be unique in the scope of the client and the user.

The Recipient element in the InviteRequest primitive MUST identify the users to be invited using User-ID(s), User-ID(s) with Client-ID(s), User-ID(s) with Application-ID(s), ContactList-ID(s), or any combination of those, unless the Invite-Type is 'GM' (group membership).

The conditional elements MUST NOT be present in the InviteRequest primitive, unless it is indicated differently below:

- If the Invite-Type is 'AP' (end-to-end application), the Invite-Application element in the InviteRequest primitive MUST include the Application-ID of the application to which the users are to be invited. The Invite-Group, Invite-Presence, Invite-Content elements MAY be included when the information is necessary for the operation of the related application. If Invite-Content is included, there MUST be only one URI, which MUST refer to a location from where the application is available for download. The download and the installation of the relevant application are not in the scope of the IMPS specifications.
- If the Invite-Type is 'GM' (group membership), the Invite-Group and the Recipient elements in the InviteRequest primitive MUST include the Group-ID that identifies the group for which the membership is desired.
- If the Invite-Type is 'GR' (group), Invite-Group element in the InviteRequest primitive MUST include the Group-ID that identifies the group to which the recipient users are to be invited.

- If the Invite-Type is 'PR' (presence), Invite-Presence element in the InviteRequest primitive SHOULD include the list of presence attributes to which the recipient users are to be invited. When the Invite-Presence element is missing from the InviteRequest primitive, the invitees are invited to all presence attributes.
- If the Invite-Type is 'SC' (shared content), Invite-Content element in the InviteRequest primitive MUST include the list of URIs that refer to the shared content(s) to which the recipient users are to be invited.

The server MUST validate the User-ID of the sending user in the Sender element before the InviteUserRequest is sent. If applicable, the server MUST also validate the Client-ID or the Application-ID of the sending client in the Sender element before the InviteUserRequest is sent. The server MUST reject the invitation request when

- User-ID is included in the Sender element, but the User-ID does not match the User-ID of the requesting user.
- Client-ID is included in the Sender element, but the Client-ID does not match the Client-ID that the requesting client used during login.
- Application-ID is included in the Sender element, but the Application-ID does not match the Application-ID that has been registered for the requesting client during login.
- the Invite-Type is 'AP', 'GM' or 'GR', and the requesting or the recipient user has been rejected from the group indicated in Invite-Group.
- the Invite-Type is 'AP', 'GM' or 'GR', and the requesting or the recipient user is not a member of the 'Restricted' group indicated in Invite-Group.
- the Invite-Type is 'AP', 'GM' or 'GR', and the requesting or the recipient user does not meet the minimum age requirement for the group indicated in Invite-Group.
- the Invite-Type is 'EC' or 'GR', and the requesting user is not joined to the group when the invitation is sent.

The server MUST respond to InviteRequest primitive with a Status primitive. If no error occurs, the server MUST generate and send InviteUserRequest message to every user who has been invited by the inviting client according to the invitation routing requirements described in 5.5.2 End-to-end message routing. The generated InviteUserRequest primitive(s) MUST all contain the following information from the InviteRequest primitive unchanged:

- Invite-ID
- Invite-Type
- Recipient
- Sender
- Invite-Application (if applicable)
- Invite-Group (if applicable)
- Invite-Presence (if applicable)
- Invite-Content (if applicable)
- Invite-Reason (if applicable)
- Own-Screen-Name (if applicable)
- Validity (if applicable)

The generated InviteUserRequest primitive(s) MUST all contain only one recipient.

Each invited client MUST reply to the InviteUserRequest primitive with a Status primitive.

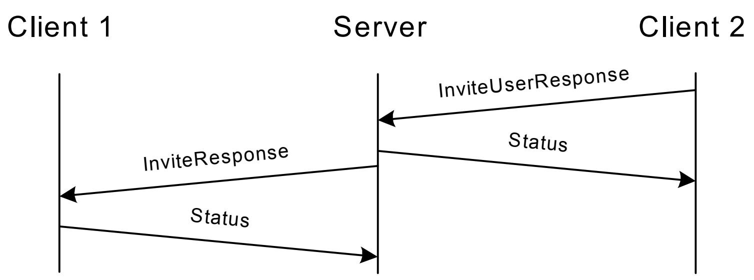

Figure 28: Invited users' response

The invitee MAY accept, reject or ignore the invitation. To accept or reject the invitation, the client MUST send an InviteUserResponse primitive to the server. The InviteUserResponse primitive MUST contain Invite-Acceptance, and the Invite-ID from the corresponding InviteUserRequest primitive without changes. The client MUST copy the Sender element from the corresponding InviteUserRequest primitive to the Recipient element in the InviteUserResponse primitive, while the responding client SHOULD copy the Recipient element from the corresponding InviteUserRequest primitive to the Sender element in the InviteUserResponse primitive however the Sender element MAY be altered by adding Client-ID. The client MAY also include Invite-Response containing a short response text and Own-Screen-Name containing the responding user's screen-name. The Own-Screen-Name element MUST NOT be used for identifying purpose: it is purely informational.

The server MUST respond with a Status primitive. If no error occurs, the server MUST generate and send InviteResponse to the inviting client according to the invitation routing requirements described in 5.5.2 End-to-end message routing.

The generated InviteResponse primitive MUST contain the following information from the InviteUserResponse primitive unchanged:

- Invite-ID
- Invite-Acceptance
- Recipient
- Sender
- Invite-Response (if applicable)
- Own-Screen-Name (if applicable)

The client MUST respond to the InviteResponse primitive with a Status primitive.

While it is OPTIONAL for an invited user to act according to the acceptance indicator of his/her response in the scope of this function, the invited user SHOULD act according to the response that has been sent in Invite-Acceptance to the user from whom the invitation originates.

The client and the server MAY support the invitation transactions, however when this feature is supported, all of the related transactions MUST be supported – meaning that creation, reception and response of invitations are all supported. The negotiated related services MUST be regarded: a client MUST NOT receive invitation to a group when group feature was not negotiated, and the server MUST NOT allow to send such invitation either. If the invitation is based on User-ID with Client-ID, and the invited client does not support the requested feature by the inviting client, the server MUST generate appropriate Invite-Response indicating the lack of support to the inviting client by automatically declining the invitation on behalf of the user/client. The Invitation ID (Invite-ID) MUST be the same InviteRequest, InviteUserRequest, InviteUserResponse, and InviteResponse primitives. The service tree leaf that allows negotiation of this transaction is 'INVIT'.

Whenever a user accepts/rejects and invitation, the server MUST send notification of the event to those other online clients of the user that are subscribed to such event using the General Notification Mechanism, see 7.2 General Notification Transactions. If any more responses to the invitation comes from any of the invited clients, then the server MUST ignore the requests. If any more invitations is queued towards clients of the recipient user, then they MUST be discarded by the server.

7.5.2 Error conditions

Generic error conditions:

- Service not supported. (405)
- Service unavailable. (503)
- Service not agreed. (506)
- Not logged in. (604)

InviteRequest error conditions:

- Invalid invitation type. (402)
- Delivery to recipient not available. (410)
- Invalid Invitation-ID. (423)
- Invalid User-ID. (427)
- Invalid Client-ID. (428)
- Missing Group-ID. (429)
- Missing Application-ID. (450)
- Invalid Application-ID. (451)
- Forbidden Application-ID. (452)
- Delivery to recipient domain not available. (516)
- Recipient unknown (User-ID or screen-name). (531)
- Recipient unknown (Contact list). (700)
- Invalid or unsupported presence value. (751)
- Group does not exist. (800)
- Group is not joined. (808)
- User has been rejected. (809)
- Not a group member. (810)
- Minimum age requirement not fulfilled. (818)
- Service provider agreement missing (907)

InviteResponse error conditions:

- Client MAY ignore any error and respond with Successful. (200)

InviteUserRequest error conditions:

- Client MAY ignore any error and respond with Successful. (200)

InviteUserResponse error conditions:

- Invalid acceptance type. (402)
- Invalid invite-ID. (423)

7.5.3 Primitives and information elements

Primitive	Direction
InviteRequest	Client 1 → Server
Status	Client 1 ← Server
InviteResponse	Client 1 ← Server

Status	Client 1 → Server
InviteUserRequest	Client 2 ← Server
Status	Client 2 → Server
InviteUserResponse	Client 2 → Server
Status	Client 2 ← Server

Table 49: Primitive directions in invitation transaction

Information Element	Req	Type	Description
Message-Type	M	InviteRequest	Message identifier.
Transaction-ID	M	String	Identifies the transaction requested.
Session-ID	M	String	Identifier for the session.
Invite-ID	M	String	Identifies the invitation.
Invite-Type	M	Enumerated string	Indicates the type of the invitation. (Group (GR), Messaging (IM), Presence (PR), ShareContent (SC), Group Membership (GM), ExtendConversation (EC), ExtendGroup (EG), End-To-End *Application* (AP))
Recipients	M	Structure	Identifies the user(s) to be invited. (User-ID, User-ID with Client-ID, User-ID with Application-ID, or ContactList-ID, or screen name). If Invite-Type is GM, identifies the group for which the group membership is requested.
Sender	M	Structure	Identifies the user from whom the invitation originates. (User-ID, User-ID with Client-ID or User-ID with Application-ID.)
Invite-Application	C	String	Identifies the related application. (Mandatory if Invite-Type is 'AP', otherwise not present.)
Invite-Group	C	String	Identifies the related group. (Mandatory if Invite-Type is 'GR', 'GM', 'EC' or 'EG', OPTIONAL if Invite-Type is 'AP', otherwise not present.)
Invite-Presence	C	Structure	Identifies the related presence attributes. (OPTIONAL if Invite-Type is 'PR' or 'AP', otherwise not present.)
Invite-Content	C	Structure	Identifies the related shared content as a list of URLs. (Mandatory if Invite-Type is 'SC', OPTIONAL if Invite-Type is 'AP', otherwise not present.)
Invite-Reason	O	String	Textual description of the invitation.
Own-Screen-Name	O	Structure	The user's screen name from whom the invitation originates; to inform the invited user about it when desired.
Validity	O	Integer	Indicates the time interval in which the invitation is valid. Indicated in seconds.

Table 50: Information elements in InviteRequest primitive

Information Element	Req	Type	Description
Message-Type	M	InviteResponse	Message identifier.
Transaction-ID	M	String	Identifies the transaction requested.
Session-ID	M	String	Identifier for the requesting user's session.
Invite-ID	M	String	Identifies the invitation.

Information Element	Req	Type	Description
Invite-Acceptance	M	Boolean	Indicates the user's choice.
Sender	M	Structure	Identifies the responding user. (User-ID, User-ID with Client-ID or User-ID with Application-ID and Friendly Name if available). If Invite-Type is GM, identifies the group for which the membership is requested. This value MUST be the same as the Sender element received in InviteUserResponse unless the server adds Friendly Name to the User-ID(s)as described in 5.3.1 Addressing introduction.
Recipient	M	Structure	Identifies the user from whom the invitation originates. (User-ID, User-ID with Client-ID or User-ID with Application-ID). This value MUST be the same as the Recipient element received in InviteUserResponse unless the element contained ContactList-ID(s) or the server adds Friendly Name to the User-ID(s) as described in 5.3.1 Addressing introduction.
Invite-Response	O	String	Textual description of the response. Present if it was present in InviteUserResponse.
Own-Screen-Name	O	Structure	The invited user's screen name to inform the user from whom the invitation originates about it when desired.

Table 51: Information elements in InviteResponse primitive

Information Element	Req	Type	Description
Message-Type	M	InviteUserRequest	Message identifier.
Transaction-ID	M	String	Identifies the transaction requested.
Session-ID	M	String	Identifier for the requested user's session.
Invite-ID	M	String	Identifies the invitation.
Invite-Type	M	Enumerated string	Indicates the type of the invitation. (Group (GR), Messaging (IM), Presence (PR), ShareContent (SC), Group Membership (GM), ExtendConversation (EC), ExtendGroup (EG), End-To-End *Application* (AP))
Sender	M	Structure	Identifies the user from whom the invitation originates. (User-ID, User-ID with Client-ID or User-ID with Application-ID and Friendly Name if available.). This value MUST be the same as the Sender element received in InviteRequest unless the server adds Friendly Name to the User-ID(s) as described in 5.3.1 Addressing introduction.
Recipient	M	Structure	Identifies a single user to be invited. (User-ID, User-ID with Client-ID or User-ID with Application-ID). If Invite-Type is GM, identifies the group for which the membership is requested. This value MUST be the same as the Recipient element received in InviteRequest unless the server adds Friendly Name to the User-ID(s) as described in 5.3.1 Addressing introduction.

Information Element	Req	Type	Description
Invite-Application	C	String	Identifies the related application. (Mandatory if Invite-Type is ‘AP’, otherwise not present.)
Invite-Group	C	String	Identifies the related group. (Mandatory if Invite-Type is ‘GR’, ‘GM, ‘EC’, or ‘EG’, OPTIONAL if Invite-Type is ‘AP’, otherwise not present)
Invite-Presence	C	Structure	Identifies the related presence attributes. (OPTIONAL if Invite-Type is ‘PR’ or ‘AP’, otherwise not present.)
Invite-Content	C	Structure	Identifies the related shared content as a list of URLs. (Mandatory if Invite-Type is ‘SC’, OPTIONAL if Invite-Type is ‘AP’, otherwise not present.)
Invite-Reason	O	String	Textual description of the invitation.
Own-Screen-Name	O	Structure	The user’s screen name from whom the invitation originates; to inform the invited user about it when desired.
Validity	O	Integer	Indicates the time interval in which the invitation is valid. Indicated in seconds.

Table 52: Information elements in InviteUserRequest primitive

Information Element	Req	Type	Description
Message-Type	M	InviteUserResponse	Message identifier.
Transaction-ID	M	String	Identifies the transaction requested.
Session-ID	M	String	Identifier for the requested user’s session.
Invite-ID	M	String	Identifies the invitation.
Invite-Acceptance	M	Boolean	Indicates the user’s choice.
Sender	M	Structure	Identifies the responding user. (User-ID or User-ID with Client-ID). If Invite-Type is GM, identifies the group for which the membership is requested. This value SHOULD be the same as the Recipient element received in InviteUserRequest.
Recipient	M	Structure	Identifies the user from whom the invitation originates. (User-ID, User-ID with Client-ID or User-ID with Application-ID). This value MUST be the same as the Sender element received in InviteUserRequest unless the server adds Friendly Name to the User-ID(s) as described in 5.3.1 Addressing introduction.
Invite-Response	O	String	Textual description of the response.
Own-Screen-Name	O	Structure	The invited user’s screen name to inform the user from whom the invitation originates about it when desired.

Table 53: Information elements in InviteUserResponse primitive

7.6 Canceling Invitations

7.6.1 Transactions

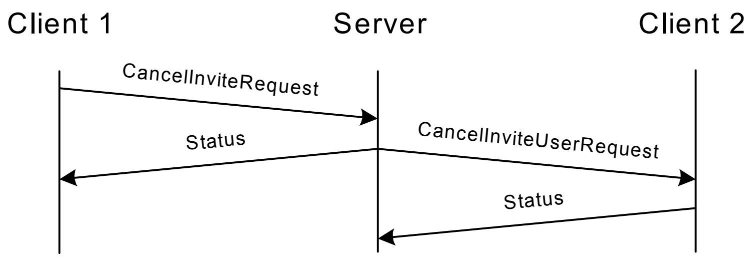

Figure 29: Canceling invitation

A user MAY cancel any invitation(s) of his/her own that have been previously requested. To cancel an invitation, the client MUST send the CancelInviteRequest primitive to the server. The CancelInviteRequest primitive MUST contain Invite-ID, the Recipient and Sender elements. The CancelInviteRequest primitive MAY include Cancel-Reason containing a short text describing the reason for the canceling the invitation and Own-Screen-Name containing his/her own screen-name. The Own-Screen-Name element MUST NOT be used for identifying purpose: it is purely informational. A valid Invite-ID MUST refer to a previously sent invitation.

The Recipient element in the CancelInviteRequest primitive MUST identify the users to be invited using User-ID(s), User-ID(s) with Client-ID(s), User-ID(s) with Application-ID(s), ContactList-ID(s), or any combination of those, unless the Invite-Type is 'GM' (group membership). The invitation MAY be cancelled for specific users only, thus the Recipient element in the CancelInviteRequest primitive MAY be different from the Recipient element in the corresponding InviteRequest primitive. The client MUST NOT attempt to cancel those invitations again that have been cancelled already for specific users.

The conditional elements MUST NOT be present in the InviteRequest primitive, unless it is indicated differently below:

- If the Invite-ID refers to an invitation that included Invite-Type as 'AP' (end-to-end application), the Invite-Application element in the InviteRequest primitive MUST include the Application-ID of the application to which the users have been invited previously. If Invite-Group, Invite-Presence, Invite-Content elements have been included in the referenced invitation, the same information elements MUST be included.
- If the Invite-ID refers to an invitation that included Invite-Type is 'GM' (group membership), the Invite-Group and the Recipient elements in the InviteRequest primitive MUST include the Group-ID that identifies the group for which membership has been requested previously.
- If the Invite-ID refers to an invitation that included Invite-Type is 'GR' (group), Invite-Group element in the InviteRequest primitive MUST include the Group-ID that identifies the group to which the recipient users have been invited previously.
- If the Invite-ID refers to an invitation that included Invite-Type is 'PR' (presence), If the Invite-Presence element have been included in the referenced invitation, the same information element MUST be included containing the list of presence attributes to which the recipient users have been invited previously. If the Invite-Presence element was missing from the referenced invitation, Invite-Presence element MUST NOT be included.
- If the Invite-ID refers to an invitation that included Invite-Type is 'SC' (shared content), Invite-Content element in the InviteRequest primitive MUST include the list of URIs that refer to the shared content(s) to which the recipient users have been invited previously.

The server MUST respond with a Status primitive to the client. If no error occurred, the server MUST generate and send out CancelInviteUserRequest primitives to all requested users according to the invitation routing requirements described in 5.5.2 End-to-end message routing. The generated CancelInviteUserRequest primitive(s) MUST all contain the following information from the CancelInviteRequest primitive unchanged:

- Invite-ID
- Recipient
- Sender

- Cancelled-Application (if applicable)
- Cancelled-Group (if applicable)
- Cancelled-Presence (if applicable)
- Cancelled-Content (if applicable)
- Cancel-Reason (if applicable)
- Own-Screen-Name (if applicable)

The generated InviteUserRequest primitive(s) MUST all contain only one recipient. The sender MAY disclose his/her current screen name by adding it to the Own-Screen-Name element in the InviteRequest primitive. The Own-Screen-Name element does not serve identifying purpose: it is purely informational.

All clients MUST respond to the server with a Status primitive to CancelInviteUserRequest.

If a client receives a CancelInviteUserRequest primitive with unknown Invite-ID, the client MUST reply with a successful Status primitive and MUST also ignore the request.

The client and the server MAY support cancel invitation transactions, however when this feature is supported, all of the related transactions and information elements MUST be supported – meaning that creation, reception and response of invitations are all supported and the negotiated related services are disregarded: a client MAY receive canceling invitation to a group even though group feature was not negotiated. The service tree leaf that allows negotiation of this transaction is ‘CAINV’.

Whenever an invitation has been cancelled for the user, the server MUST send notification of the event to those other online clients of the user that are subscribed to such event using the General Notification Mechanism see 7.2 General Notification Transactions.

7.6.2 Error conditions

Generic error conditions:

- Service not supported. (405)
- Service unavailable. (503)
- Service not agreed. (506)
- Not logged in. (604)

CancelInviteRequest error conditions:

- Invalid invite-ID. (423)
- Delivery to recipient not available. (410)
- Delivery to recipient domain not available. (516)
- Recipient unknown (User-ID or screen-name). (531)
- Recipient unknown (Contact list). (700)
- Service provider agreement missing (907)

CancelInviteUserRequest error conditions:

- Client MAY ignore any error and respond with Successful. (200)

7.6.3 Primitives and information elements

Primitive	Direction
CancelInviteRequest	Client 1 → Server
Status	Client 1 ← Server
CancelInviteUserRequest	Client 2 ← Server
Status	Client 2 → Server

Table 54: Primitive directions in cancel invitation transaction

Information Element	Req	Type	Description
Message-Type	M	CancelInviteRequest	Message identifier.
Transaction-ID	M	String	Identifies the transaction requested.
Session-ID	M	String	Identifier for the session.
Invite-ID	M	String	Identifies the invitation.
Sender	M	Structure	Identifies the user from whom the canceling request originates. (User-ID, User-ID with Client-ID or User-ID with Application-ID).
Recipient	M	Structure	Identifies the user(s) to be cancelled. (User-ID, User-ID with Client-ID, User-ID with Application-ID or ContactList-ID). If Invite-Type of the related invitation was 'GM', it identifies the group for which the membership request is cancelled.
Cancel-Reason	O	String	Textual description of the cancellation.
Cancelled-Application	C	String	Identifies the related application. (Mandatory if Invite-Type of the related invitation was 'AP', otherwise not present.)
Cancelled-Group	C	String	Identifies the related group. (Mandatory if Invite-Type of the related invitation was 'GR', 'GM, 'EC' or 'EG', OPTIONAL if Invite-Type of the related invitation was 'AP', otherwise not present.)
Cancelled-Presence	C	Structure	Identifies the related presence attributes. (OPTIONAL if Invite-Type of the related invitation was 'PR' or 'AP', otherwise not present.)
Cancelled-Content	C	String	Identifies content to be cancelled as a list of URLs. (Mandatory if Invite-Type of the related invitation was 'SC', OPTIONAL if Invite-Type of the related invitation was 'AP', otherwise not present.)
Own-Screen-Name	O	Structure	The user's screen name from whom the canceling request originates; to inform the canceled user about it when desired.

Table 55: Information elements in CancelInviteRequest primitive

Information Element	Req	Type	Description
Message-Type	M	CancelInvite UserRequest	Message identifier.
Transaction-ID	M	String	Identifies the transaction requested.
Session-ID	M	String	Identifier for the session.
Invite-ID	M	String	Identifies the invitation.
Sender	M	Structure	Identifies the user from whom the canceling request originates. (User-ID, User-ID with Client-ID or User-ID with Application-ID and Friendly Name if available.) This value MUST be the same as the Sender element received in CancelInviteRequest unless the server adds Friendly Name to the User-ID(s) as described in 5.3.1 Addressing introduction.
Recipient	M	Structure	Identifies a single user to be cancelled. (User-ID, User-ID with ClientID or User-ID with Application-ID and Friendly Name if available.). If Invite-Type of the related invitation was 'GM', identifies the group for which the membership request is canceled. This value MUST be the same as the Recipient element received in CancelInviteRequest unless the server adds Friendly Name to the User-ID(s) as described in 5.3.1 Addressing introduction.
Cancel-Reason	O	String	Textual description of the cancellation.
Cancelled-Application	C	String	Identifies the related application. (Mandatory if Invite-Type of the related invitation was 'AP', otherwise not present.)
Cancelled-Group	C	String	Identifies the related group. (Mandatory if Invite-Type of the related invitation was 'GR', 'GM, 'EC' or 'EG', OPTIONAL if Invite-Type of the related invitation was 'AP', otherwise not present.)
Cancelled-Presence	C	Structure	Identifies the related presence attributes. (OPTIONAL if Invite-Type of the related invitation was 'PR' or 'AP', otherwise not present.)
Cancelled-Content	C	String	Identifies content to be cancelled as a list of URLs. (Mandatory if Invite-Type of the related invitation was 'SC', OPTIONAL if Invite-Type of the related invitation was 'AP', otherwise not present.)
Own-Screen-Name	O	Structure	The user's screen name from whom the canceling request originates; to inform the canceled user about it when desired.

Table 56: Information elements in CancelInviteUserRequest primitive

7.7 Get Map

7.7.1 Transactions

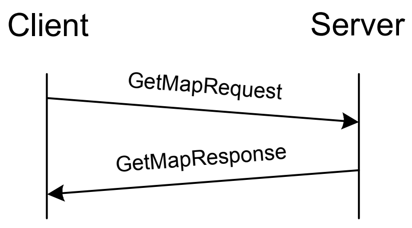

Figure 30: Get Map transaction

A user MAY request map information based on location information or User-ID. The client and the server MUST fulfill the applicable content delivery related requirements in 5.6 Content for Presence/Instant Messaging.

The IMPS client and server MAY support the Get Map transaction. The service tree leaf that allows negotiation of this transaction is ‘GETMAP’.

When the client requests map information based on location information, the GetMapRequest primitive MUST include the GeoLocation element. The GeoLocation is identical to the GeoLocation presence attribute and its semantics; see GeoLocation in [PA]. The GeoLocation MAY refer to any location.

When the client requests map information based on User-ID, the GetMapRequest primitive MUST include the User-ID.

The client MUST NOT include both GeoLocation and User-ID elements in the request.

When the request contains User-ID, the server MUST verify that the requested user has authorized the requesting user for the GeoLocation presence attribute, and the transaction MUST fail if the requesting user is not authorized – the server MUST NOT reveal the fact that the requested user did not authorize the requesting user the GeoLocation attribute, thus the server MUST reply with error code 517 in this case. In case no proactive authorization was done, reactive authorization MUST NOT be performed and the transaction MUST fail with error code 517.

Upon success the server MUST return either the map or a URL allowing the client to retrieve the requested map. If the server returns a map, it MUST select a content type and size – based on the content type and size limitations that have been agreed during client capability negotiation – that the client is able to handle. If the server is not able to deliver such content, it MUST respond with error 411. If the server is unable to generate a map or a URL for the requested location, it MUST reply with error code 517.

In order to allow location-aware devices to rotate the image, the image containing the map information SHOULD:

- Contain the requested location in the center of the map.
- Be oriented in a way that an arrow pointing from the geometric center of the map towards the center of the upper edge of the map is the ‘North’ direction.

7.7.2 Error Conditions

Generic error conditions:

- Service not supported. (405)
- Service unavailable. (503)
- Service not agreed. (506)
- Not logged in. (604)

GetMapRequest error conditions:

- Unable to find suitable content type (411)
- Response too large (432)
- Location Not Supported (517)

- Service provider agreement missing (907)

7.7.3 Primitives and information elements

Primitive	Direction
GetMapRequest	Client → Server
GetMapResponse	Client ← Server

Table 57: Primitive directions for Get Map transaction

Information Element	Req	Type	Description
Message-Type	M	GetMapRequest	Message identifier.
Transaction-ID	M	String	Identifies the transaction.
Session-ID	M	String	Identifier for the session.
PresenceAttributeList	C	Structure	MUST only contain a GeoLocation presence attribute with value. It is present only when User-ID is not present.
User-ID	C	String	Identifies the user whose map information is requested. It is present only when GeoLocation is not present.

Table 58: Information elements in GetMapRequest primitive

Information Element	Req	Type	Description
Message-Type	M	GetMapResponse	Message identifier.
Transaction-ID	M	String	Identifies the transaction.
Session-ID	C	String	Identifier for the session.
Map	C	Structure	An image that contains the map of the requested location or user.
URL	C	String	The URL allowing the client to retrieve the requested map.

Table 59: Information elements in GetMapResponse primitive

7.8 Verify ID

7.8.1 Transactions

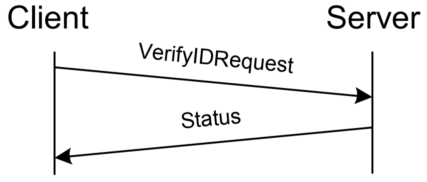

Figure 31: Verify ID

A client MAY send a list of IDs to the server to be verified for their validity; the server MUST respond with a Status primitive. The ID-List is a structure containing a combination of user name (UID), resource (CLID or GID) and domain name (Domain).

The VerifyIDRequest MUST support local and fully qualified addressing. If the local addressing is employed in VerifyIDRequest primitive, the Status primitive contains fully qualified User-ID, ContactList-ID, or Group-ID.

The client and the server MAY support the Verify ID transaction. The service tree leaf that allows negotiation of this transaction is 'VRID'.

7.8.2 Error Conditions

- General address error (901)
- Unknown user (531).
- Contact list does not exist (700).
- Group does not exist (800).
- Domain not found (404).
- Service Not Supported (405)
- Service provider agreement missing (907)

7.8.3 Primitives and information elements

Primitive	Direction
VerifyIDRequest	Client → Server
Status	Client ← Server

Table 60: Primitive direction for Verify WV ID

Information Element	Req	Type	Description
Message-Type	M	VerifyIDRequest	Message identifier.
Transaction-ID	M	String	Identifies the transaction.
Session-ID	M	String	Identifier for the session.
ID-List	M	Structure	Includes a list of ID structures to be verified.

Table 61: Information elements in VerifyIDRequest primitive

8. Presence Feature

In order to achieve minimum level of interoperability both the client and the server MUST support the following functionalities:

- 8.3.2 Subscribed Presence Transactions

The rest of the presence-related functionalities are all OPTIONAL. The individual client or server implementations MAY decide whether support for a particular transaction is implemented or not.

8.1 Contact List

8.1.1 Contact List Properties

There are three properties for Contact List:

- DisplayName: a free text string given by user that can be presented in the user interface of the client.
- Default: a string set by user, ‘T’ (true) indicates that the particular contact list is the default contact list and ‘F’ (false) indicates that the list is not the default contact list.
- DoNotNotify: a Boolean value indicating whether the user added to a Contact List should be notified of such event. When the value is true (‘T’), the added user will not be notified. When the value is false (‘F’), the added user will be notified.

When the user creates his/her first contact list, the server MUST automatically set that contact list as the default (even if the user specifies that the ‘Default’ property SHOULD be set to ‘F’).

When the user has more than one contact list in the system, the user MAY set any of his/her contact lists as the default contact list (see ListManageRequest primitive). When the user sets ‘Default’ property of a contact list to ‘T’, the ‘Default’ property of the previously default contact list MUST be set to ‘F’ automatically by the server.

If the user tries to set the ‘Default’ property of the default contact list to ‘F’, the server MUST silently ignore this. If the user deletes the default contact list, the server MUST select another contact list as default. It is a server preference how the new default contact list is selected.

When the user creates a contact list, the server MAY set the DoNotNotify flag to true (‘T’) or false (‘F’).

8.1.2 Transactions

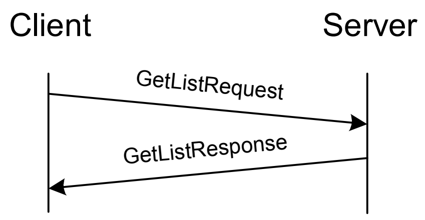

Figure 32: Get list of ContactList-IDs transaction

The user MAY retrieve the list of all his/her own ContactList-IDs at once. The default ContactList-ID MUST be indicated in a separate information element. The client MUST send the GetListRequest primitive to the server. The server MUST reply with a GetListResponse primitive, which MUST contain the list of all ContactList-IDs owned by the user, as well as the default ContactList-ID. In case there is some error, the server MUST respond with a Status primitive instead of the expected GetListResponse primitive.

The client and the server MAY support the get list of ContactList-IDs transaction. The service tree leaf that allows negotiation of this transaction is ‘GCLI’.

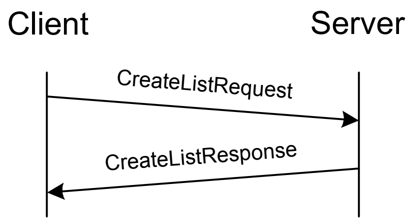

Figure 33: Create contact list transaction

A user MAY create more than one contact list, but there MAY be implementation-specific limitations for the maximum number of lists per user.

To create a contact list, the client MUST send the CreateListRequest primitive to the server including the ID of the contact list, and MAY include the initial contact list properties and a list of initial users that SHOULD be added to the list. The server MUST create the contact list, and respond with a CreateListResponse primitive, unless an error occurs.

The create contact list transaction MAY be supported by the client and the server. The service tree leaf that allows negotiation of this transaction is 'CCLI'.

In order to save extra transactions:

- the client MAY request a list of users to be added to the contact list initially by specifying those users in the User-Nick-List element of the request. For more information about adding users to the contact list please refer to 'Manage contact list transaction' in 8.1.2. If a list of initial users is supplied in the request, the server MUST add all of those users to the contact list. If the server is not able to add all users to the contact list due to some error, the server MUST reply with a '201' (partially successful) result code in the Status primitive.
- the client MAY request initial properties to be applied to a contact list by specifying those properties in the Contact-List-Props element of the request. For more information about applying properties to the contact list please refer to 'Manage contact list transaction' in 8.1.2. If any contact list properties are supplied in the request, the server MUST apply these to the list with one exception: If the list created is the first contact list of the user, the server MUST set the 'Default' contact list property to 'T' regardless of what the client requests. If the properties cannot be applied to the list due to some error, the server MUST reply with a '201' (partially successful) result code in the Status primitive.

If the contact list already exists on the server, the server MUST indicate the error in the Status primitive.

If the contact list does not exist on the server yet, the server MUST create it disregarding success or failure that originates from adding users and applying properties to the contact list.

The server MUST return all properties of the newly created Contact List in the CreateListResponse after it has been created successfully.

If an initial list of users have been added to the Contact List and the DoNotNotify property of the Contact List is false ('F'), the server MUST send notifications those other online users that have been added to the Contact List initially and are subscribed to such event using the General Notification Mechanism; see 7.2 General Notification Transactions.

Whenever a new Contact List has been created for the user, the server MUST send notification of the event to those other online clients of the user that are subscribed to such event using the General Notification Mechanism; see 7.2 General Notification Transactions.

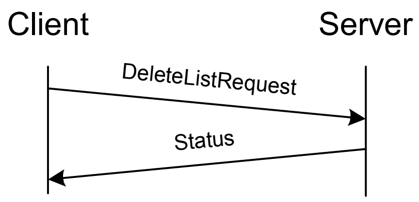

Figure 34: Delete contact list transaction

A user MAY delete a contact list at any time.

The client and the server MAY support the delete contact list transaction. The service tree leaf that allows negotiation of this transaction is ‘DCLI’.

The client sends the DeleteListRequest primitive to the server containing the ID of the contact list to be deleted. The server MUST remove the indicated contact list and respond with a Status primitive. If the contact list does not exist, the server MUST respond with a status code of 700.

For all clients of the user who are subscribed for presence to this contact list, the server MUST automatically unsubscribe the presence attributes associated with the subscription to the deleted contact list.

Whenever an existing Contact List has been deleted from the user, the server MUST send notification of the event to those other online clients of the user that are subscribed to such event using the General Notification Mechanism; see 7.2 General Notification Transactions.

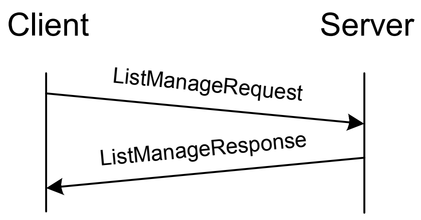

Figure 35: Manage contact list transaction

The user MAY retrieve one contact list at a time; add or remove members; change the name of a contact list and set the contact list as the default contact list. The client MUST send the ListManageRequest primitive to the server. The server MUST perform the requested operations, and then reply with a ListManageResponse primitive. If there is an error, the server MUST respond with a Status primitive instead of the expected ListManageResponse primitive.

The manage contact list transaction MAY be supported by the client and the server. The service leaf that allows negotiation of this transaction is ‘MCLI’.

The user MAY manage only his/her own contact list(s). The User-IDs MUST be used to manage the users on the contact list – the NickName part is OPTIONAL and informational.

If the request contains Add-Nick-List but does not contain Nickname for certain users, the server MUST automatically add a Nickname for those users. When the Public Profile of the added user is available, the FriendlyName MUST be added as Nickname. Otherwise, the server MUST add the User-ID as Nickname. When the server adds a Nickname, the response MUST contain the Nickname chosen by the server.

The request primitive MUST either contain:

- ContactList-ID and Receive-List (to retrieve the contact list) – Receive-List MUST be ‘T’ in this case, or
- ContactList-ID and Receive-List and Contact-List-Props (to update the properties), or
- ContactList-ID and Receive-List and Add-Nick-List (to add users), or
- ContactList-ID and Receive-List and Remove-Nick-List (to remove users).

If the Receive-List is ‘T’, the server MUST include the Nick-List containing the actual content of the contact list in the response. The Nick-List MUST reflect the contact list state after any other operations in the List-Manage operation has been carried out.

If the Receive-List is ‘F’, the server MUST NOT include the Nick-List in the response.

If the request contains:

- ContactList-ID and Receive-List (client requests retrieval of the contact list), the server MUST return the properties and the User-Nick-List of the requested contact list.
- ContactList-ID and Receive-List and Contact-List-Props (client requests updating some properties of the contact list), the server MUST apply the new properties to the requested contact list and the response

MUST contain the new properties. If a contact list's 'Default' property has been changed from 'F' to 'T', the previous default contact list's 'Default' property MUST be changed from 'T' to 'F'. The server MUST ignore requests from the client that attempts to set the 'Default' property of a contact list from 'T' to 'F'.

- ContactList-ID and Receive-List and Add-Nick-List (client requests adding users to the contact list), and
  - a requested User-ID does not exist in the contact list, the server MUST add the specified User-IDs and nicknames to the contact list.
  - a requested User-ID already exists in the contact list, the server MUST replace the NickName for those User-IDs that already exist in the contact list.
- ContactList-ID and Receive-List and Remove-Nick-List (client requests removing users from the contact list), the server MUST remove the specified User-IDs and their corresponding nicknames from the contact list. If there are User-IDs in the Remove-Nick-List that do not exist in the contact list, the transaction MUST NOT fail.

When a user is added to a subscribed contact list, the server MUST subscribe the added user to those presence attributes that have been subscribed for this contact list. When a user is removed from the contact list or the contact list is removed, the server MUST unsubscribe those presence attributes that have been subscribed for the removed user or removed contact list.

Whenever user(s) have been successfully added to the Contact List and the DoNotNotify property of the Contact List is false ('F'), the server MUST send notification(s) to those online user(s) that have been successfully added to the Contact List and are subscribed to such event using the General Notification Mechanism; see 7.2 General Notification Transactions.

Whenever a Contact List has been updated for the user (users have been added or removed or any property has been changed), the server MUST send notification of the event to those other online clients of the user that are subscribed to such event using the General Notification Mechanism; see 7.2 General Notification Transactions.

The client MAY request authorizing OnlineStatus presence attribute to all users on the contact list and addition of Contact-List-ID to the grant list using the AuthorizeAndGrant information element. The AuthorizeAndGrant is not applicable with value 'F', thus it MUST NOT be used. The server MUST ignore the AuthorizeAndGrant element when its value is 'F'. When the server receives the ListManageRequest primitive including AuthorizeAndGrant information element and its value is 'T', the server MUST:

- Add the Contact-List-ID to the grant list of the requested user. If the Contact-List-ID already exists in the grant list of the requesting user, the server MUST NOT add the Contact-List-ID again to the grant list.
- Add the OnlineStatus presence attribute to the presence attribute list that is assigned to the requested contact list. When the contact list does not have a presence attribute list assigned, the server MUST create a presence attribute list containing only the OnlineStatus attribute, and assign it to the requested contact list.

8.1.3 Error conditions

Generic error conditions:

- Not logged in. (604)
- Service not agreed. (506)
- Service not supported. (405)

GetListRequest error conditions:

- None except the generic error conditions.

CreateListRequest error conditions:

- Contact list already exists. (701)
- Invalid or unsupported contact list property. (752)
- Unknown User-ID. (531)
- The maximum number of contact lists has been reached for the user (753)

DeleteListRequest error conditions:

- Contact list does not exist. (700)

ListManageRequest error conditions:

- Contact list does not exist. (700)
- Invalid or unsupported contact list property. (752)
- Unknown User-ID. (531)
- The maximum number of contacts has been reached for the user (754)

8.1.4 Primitives and information elements

Primitive	Direction
GetListRequest	Client → Server
GetListResponse	Client ← Server
CreateListRequest	Client → Server
CreateListResponse	Client ← Server
DeleteListRequest	Client → Server
Status	Client ← Server
ListManageRequest	Client → Server
ListManageResponse	Client ← Server

Table 62: Primitive directions for contact list management

Information Element	Req	Type	Description
Message-Type	M	GetListRequest	Message identifier.
Transaction-ID	M	String	Identifies the transaction.
Session-ID	M	String	Identifier for the session.

Table 63: Information elements in GetListRequest primitive.

Information Element	Req	Type	Description
Message-Type	M	GetListResponse	Message identifier.
Transaction-ID	M	String	Identifies the transaction.
Session-ID	M	String	Identifier for the session.
Contact-List-ID-List	C	Structure	The list of all ContactList-IDs except the default contact list. If empty or omitted, no contact lists exist.
Default-CList-ID	C	String	Identifies the default contact list. If omitted, no default contact list exists.

Table 64: Information elements in GetListResponse primitive.

Information Element	Req	Type	Description
Message-Type	M	CreateList Request	Message identifier.
Transaction-ID	M	String	Identifies the transaction.
Session-ID	M	String	Identifier for the session.
Contact-List-ID	M	String	Identifies the contact list to be created.
Contact-List-Props	O	Structure	The initial properties of the contact list.

User-Nick-List	O	Structure	Identifies the initial users to be added to the contact list (User-ID, Nickname).

Table 65: Information elements in CreateListRequest primitive

Information Element	Req	Type	Description
Message-Type	M	CreateList Response	Message identifier.
Transaction-ID	M	String	Identifies the transaction.
Session-ID	M	String	Identifier for the session.
Contact-List-ID	M	String	Identifies the contact list to be created.
Contact-List-Props	M	Structure	The initial properties of the contact list.

Table 66: Information elements in CreateListResponse primitive

Information Element	Req	Type	Description
Message-Type	M	DeleteListRequest	Message identifier.
Transaction-ID	M	String	Identifies the transaction.
Session-ID	M	String	Identifier for the session.
Contact-List-ID	M	String	Identifies the contact list to be deleted.

Table 67: Information elements in DeleteListRequest primitive

Information Element	Req	Type	Description
Message-Type	M	ListManageRequest	Message identifier.
Transaction-ID	M	String	Identifies the transaction.
Segment-Info	O	Structure	Contains segmentation information when the primitive is segmented.
Session-ID	M	String	Identifier for the session.
Contact-List-ID	M	String	Identifies the contact list.
Add-Nick-List	C	Structure	Identifies the users to be added to the contact list. (User-ID, Nickname)
Remove-Nick-List	C	Structure	Identifies the users to be removed from the contact list (User-ID).
Contact-List-Props	C	Structure	The properties of the contact list to be set.
Receive-List	M	Boolean	Indicates if the client wants to receive the User-Nick-List in the ListManageResponse.
AuthorizeAndGrant	O	Boolean	Indicates if the contact list ID is to be added to the grant list. It also indicates that the online status presence attribute is to be added to the presence attributes list assigned to the contact list. The default value is false.

Table 68: Information elements in ListManageRequest primitive.

Information Element	Req	Type	Description
Message-Type	M	ListManage Response	Message identifier.
Transaction-ID	M	String	Identifies the transaction.
Segment-Info	O	Structure	Contains segmentation information when the primitive is segmented.
Session-ID	M	String	Identifier for the session.
Result	M	Structure	The result of the request.
Contact-List-Props	C	Structure	The properties of the contact list.
User-Nick-List	C	Structure	Contains the list of users on the contact list. (User-ID, Nickname)

Table 69: Information elements in ListManageResponse primitive.

8.2 Authorization of Presence Attributes

8.2.1 Overview

The authorization of presence attributes is divided into two models*: proactive authorization* in which the IMPS user authorizes presence attributes before anyone has requested the attributes, and *reactive authorization* in which the IMPS user authorizes presence attributes on request.

The model and tools for authorization of the presence attributes is presented in Figure 36 below.

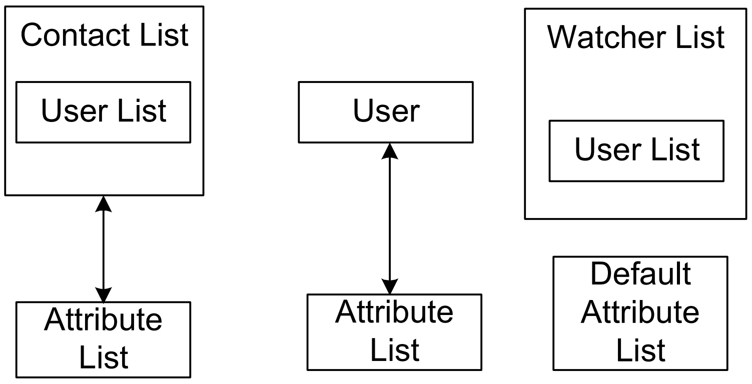

Figure 36: Authorization model for presence attributes

The authorization MAY be targeted to either individual users or to a group of users through a Contact List, or to the general public through the Default Attribute list.

The actual attributes that are authorized MUST be defined in an attribute list.

8.2.2 Proactive Authorization

In the proactive authorization model, the publisher MAY use *individual authorization*, *list authorization* or *default authorization.*

For list authorization, the publisher MUST specify the presence attributes to be associated with a Contact List. The presence attributes assigned to the Contact List MUST be accessible by all users in that Contact List except those users, that have individual authorization – individual authorization has the highest priority. There MUST NOT be more than one attribute list per Contact List.

When a user is in multiple Contact Lists that have attribute lists assigned, the union of all attributes in those attribute lists MUST be accessible to the user.

For individual authorization, the publisher MUST specify a list of presence attributes to be associated with a user. There MUST NOT be more than one attribute list per user.

When a user has been authorized individually, and he/she is also authorized via a Contact List, then the individual authorization MUST have priority over list authorization, i.e. the individual's attributes override the attributes of the Contact List.

For individual authorization, the publisher MAY specify that he wants to be notified if the authorized watcher request other attributes than the ones authorized. If the watcher requested other presence attributes than those authorized by the publisher, reactive authorization MUST be applied (see 8.2.3 Reactive Authorization)

The publisher MAY specify a list of default presence attributes. These presence attributes MUST only be available for all users who do not have individual or list authorization.

The following section describes how the authorization of presence attributes MUST be calculated. There are two users involved, the subscriber and the publisher. The subscriber is subscribing to or requesting presence information of the publisher.

The following sets are used:

$\boldsymbol{PA_{notif}}$: is the set of presence attributes the subscriber MUST get notifications for. If any of the publisher's presence attributes that are in this set have been changed, the subscriber MUST get a notification.

$\boldsymbol{PA_{subscribe}}$: is the set of the publisher's presence attributes that the subscriber has subscribed to.

$\boldsymbol{PA_{list}}$:is the set of presence attributes that the publisher has authorized to all of the users in a contact list via an attribute list that is attached to the contact list. If the set is empty, then $\boldsymbol{PA_{list}}$ is used for the purpose of blocking all users in the particular contact list from accessing any presence attributes. If there is no attribute list attached, authorization has not been done.

$\boldsymbol{PA_{individual}}$: is the set of presence attributes that the publisher has authorized for the subscriber individually via an attribute list that is assigned to the subscriber. If the set is empty, then $\boldsymbol{PA_{individual}}$ is used for the purpose of blocking the specific subscriber from accessing any presence attributes. If there is no attribute list assigned, authorization has not been done.

$\boldsymbol{PA_{default}}$: is the set of presence attributes that the publisher has authorized to all subscribers – the general public – which have not been authorized using individual or list authorization. If the set is empty or not present, the publisher has no default attribute list.

$\boldsymbol{PA_{authorize}}$: is the total set of presence attributes that the publisher has authorized the subscriber to access.

Calculating $\boldsymbol{PA_{notif}}$:

A subscriber MUST be notified of presence changes in the publisher's presence attributes according to the following formula:

$$PA_{notif} = PA_{subscribe} \cap PA_{authorize}$$

Figure 37: Calculating $\boldsymbol{PA_{notif}}$

Calculating $\boldsymbol{PA_{Authorize}}$:

The authorized presence attributes MUST be calculated as follows:

- If the publisher has associated an individual authorization with the subscriber, then

$$PA_{authorize} = PA_{individual}$$

Figure 38: Calculating $PA_{Authorize}$ for individual users

- Otherwise, find all the publisher's contact lists $x_1 \ldots x_n$ where subscriber is in the list of users and the publisher has associated an attribute list with the contact list. If any such contact list exists, then

$$PA_{authorize} = \bigcup_{i=1}^{n} PA_{list_{x_i}}$$

Figure 39: Calculating $PA_{Authorize}$ for contact lists

- If no such contact list exists, then

$$PA_{authorize} = PA_{default}$$

Figure 40: Calculating $PA_{Authorize}$ from default attribute list

The attribute list management functions – 8.2.4 Transactions – include creation/update/deletion of attribute lists to individual users/contact lists/the default list.

8.2.3 Reactive Authorization

If the publisher did not proactively authorize the subscriber in any way (no individual or list authorization and the requested presence attributes are not covered in the default presence attribute list) and the publisher requested to be notified about additional presence attribute requests than those authorized via the Default List, the server MUST notify the publisher, using the generic notification mechanism (see 7.2 General Notification Transactions). The notification MUST contain the User-ID of the subscriber and the list of requested presence attributes that are currently not authorized. The notified user MAY use proactive authorization to authorize the subscriber via individual, list, or Default List authorization. The Client SHOULD permit the notified user to trigger the proactive authorization with minimal user interaction.

If the publisher proactively authorized the subscriber via individual authorization with indication that he wants to be notified when the authorized subscriber requests other attributes than the ones that are already authorized, the server MUST notify the publisher using the generic notification mechanism (see 7.2 General Notification Transactions). The notification MUST contain the User-ID of the subscriber and the list of requested presence attributes, currently not authorized. The notified user MAY use proactive authorization to authorize the subscriber via individual, list, or Default List authorization. The Client SHOULD permit the notified user to trigger the proactive authorization with minimal user interaction.

If the publisher proactively authorized the subscriber via list authorization with indication that he wants to be notified when the authorized subscriber requests other attributes than the ones that are already authorized across all contact lists, the server MUST notify the publisher using the generic notification mechanism (see 7.2 General Notification Transactions). The notification MUST contain the User-ID of the subscriber and the list of requested presence attributes that are currently not authorized. The notified user MAY use proactive authorization to authorize the subscriber via individual, list, or Default List authorization. The Client SHOULD permit the notified user to trigger the proactive authorization with minimal user interaction.

8.2.4 Transactions

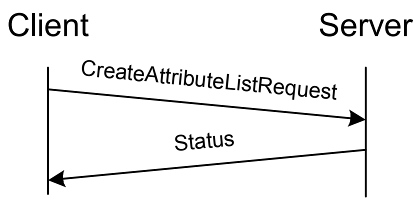

Figure 41: Create attribute list transaction

The user MAY create user or contact-list specific attribute list(s), but only one attribute lists per user or per contact list.

Changing an authorization MUST NOT cancel already active subscriptions. The subscriber MUST NOT receive notifications of attributes that are unauthorized, but if the attributes are reauthorized the subscriber MUST receive notifications without the need for re-subscription.

The create attribute list transaction MUST be supported by the client and the server, therefore its support is not negotiated.

If the requested attribute list does not exist on the server, the server MUST create it. In order to modify (update) an attribute list it MUST be overwritten by creating another attribute list for the same User-ID or ContactList-ID (e.g. there is no need to delete first). If the attribute list exists on the server, the server MUST overwrite it without indicating error.

If the ‘Default-List’ element is ‘T’, the server MUST associate the supplied attribute list with the default attribute list.

If the attribute list is empty (i.e.: it does not contain any presence attributes), the server MUST regard this as a valid – but empty – attribute list to be associated with the indicated user(s), contact list(s) and/or default list.

The server MUST persistently store:

- the Default-Notify flag for the Default-List,
- the User-Notify flag per user,
- ContactList-Notify flag per Contact-List.

When the client did not include User-Notify or ContactList-Notify in the request, the server MUST not alter the previous value.

Whenever an attribute list has been created for the user or an existing one has been overwritten, the server MUST send notification of the event to those other online clients of the user that are subscribed to such event using the General Notification Mechanism; see 7.2 General Notification Transactions.

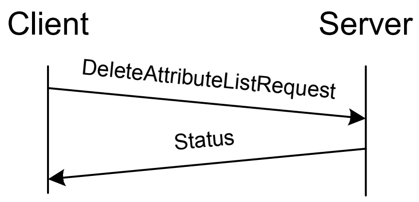

Figure 42: Delete attribute list transaction

A user MAY delete the default attribute list and /or the attribute list from a set of users and/or contact lists.

The client and the server MUST support the delete attribute list transaction, therefore its support is not negotiated.

The server MUST delete the attribute list from every user specified in the User-ID-List element and every user contact list specified in the Contact-List-ID-List element.

If the specified attribute list(s) does not exist on the server, it MUST be silently ignored without generating an error.

If the Default-List element indicates ‘T’, the server MUST delete the default attribute list.

Whenever an exiting attribute list has been deleted from the user, the server MUST send notification of the event to those other online clients of the user that are subscribed to such event using the General Notification Mechanism; see 7.2 General Notification Transactions.

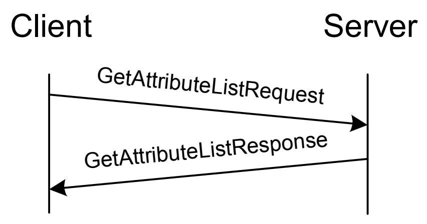

Figure 43: Get attribute list(s) transaction

The publisher MAY retrieve the attributes he/she associates with a specific contact list(s) or user(s), or the default attribute list.

The client and the server MUST support the get attribute list transaction, therefore its support is not negotiated.

If the Default-List element indicates 'T' the server MUST include the default attribute list in the response even if it is empty.

If the request contains:

- Contact-List-ID-List, the server MUST send only those attribute list(s) in the response that are associated with the particular contact list(s).
- User-ID-List, the server MUST send only those attribute list(s) in the response that are associated with the particular user(s).

If the request does not contain the Contact-List-ID-List element or the User-ID-List element, then the server MUST deliver all contact list and User-ID associations in the response.

The server MUST deliver the notification preferences (User-Notify, ContactList-Notify, Default-Notify) for the users/contact lists/default list within the requested lists respectively, however when a flag was not set previously, it MUST NOT deliver the particular flag.

8.2.5 Error conditions

Generic error conditions:

- Not logged in. (604)
- Service not agreed. (506)
- Service not supported. (405)

CreateAttributeListRequest error conditions:

- Unknown User-ID. (531)
- Contact list does not exist. (700)
- Unknown presence attribute (not defined in [PA]). (750)
- The maximum number of attribute lists has been reached for the user (755)

DeleteAttributeListRequest error conditions:

- Unknown User-ID. (531)
- Contact list does not exist. (700)

GetAttributeListRequest error conditions:

- Unknown User-ID. (531)
- Contact list does not exist. (700)

8.2.6 Primitives and information elements

Primitive	Direction
CreateAttributeListRequest	Client → Server
Status	Client ← Server
DeleteAttributeListRequest	Client → Server
Status	Client ← Server
GetAttributeListRequest	Client → Server
GetAttributeListResponse	Client ← Server

Table 70: Primitive directions for attribute list management

Information Element	Req	Type	Description
Message-Type	M	CreateAttribute ListRequest	Message identifier.
Transaction-ID	M	String	Identifies the transaction.
Session-ID	M	String	Identifier for the session.
Presence-Attribute-List	M	Structure	A list of presence attributes. These will be authorized to the user.
User-ID-List	O	Structure	Identifies the user(s) to assign the attribute list association.
Contact-List-ID-List	O	Structure	Identifies the contact list(s) to assign the attribute list association.
Default-List	M	Boolean	Indicates if the attributes are targeted to the default attribute list in addition to the lists specified by the fields User-ID-List and Contact-List-ID-List above.
Default-Notify	C	Boolean	Indicates if the user wants to be notified when subscriber without authorization requests other attributes than those available in the Default List.
User-Notify	C	Boolean	Indicates if the user wants to be notified when subscriber with individual authorization requests other attributes than those available in the individual authorization. The element MUST be present only when the primitive contains User-ID-List.
ContactList-Notify	C	Boolean	Indicates if the user wants to be notified when subscriber with contact list authorization requests other attributes than those available across all contact list authorizations. The element MUST be present only when the primitive contains Contact-List-ID-List.

Table 71: Information elements in CreateAttributeListRequest primitive

Information Element	Req	Type	Description
Message-Type	M	DeleteAttribute ListRequest	Message identifier.
Transaction-ID	M	String	Identifies the transaction.
Session-ID	M	String	Identifier for the session.
Contact-List-ID-List	O	Structure	Identifies the contact list(s) to remove the attribute list association

User-ID-List	O	Structure	Identifies the user(s) to remove the attribute list association.
Default-List	M	Boolean	Indicates if the default attribute list SHOULD be cleared.

Table 72: Information elements in DeleteAttributeListRequest primitive

Information Element	Req	Type	Description
Message-Type	M	GetAttributeList Request	Message identifier.
Transaction-ID	M	String	Identifies the transaction.
Session-ID	M	String	Identifier for the session.
Default-List	M	Boolean	Indicates if the default attribute list is requested.
Contact-List-ID-List	O	Structure	Identifies the contact list(s) to retrieve the associated attribute lists for.
User-ID-List	O	Structure	Identifies the user(s) to retrieve the associated attribute lists for.

Table 73: Information elements in GetAttributeListRequest primitive.

Information Element	Req	Type	Description
Message-Type	M	GetAttributeList Response	Message identifier.
Transaction-ID	M	String	Identifies the transaction.
Segment-Info	O	Structure	Contains segmentation information when the primitive is segmented.
Session-ID	M	String	Identifier for the session.
Result	M	Structure	Result of the request.
Attribute-Association-List	O	Structure	The list of user, and contact-list presence attribute associations.
Default-Attribute-List	O	Structure	The list of presence attributes associated with the default list.

Table 74: Information elements in GetAttributeListResponse primitive.

8.3 Presence Information Delivery

8.3.1 Presence Information

To ensure interoperability, sets of interoperable presence attributes are defined. This is accomplished by dividing the presence information into client status and user status classes by defining the most common presence attributes within these classes. See [PA] for more information.

8.3.2 Subscribed Presence Transactions

In order to achieve minimum level of interoperability both the client and the server MUST support the following functionalities:

- 8.3.2.1 Subscribe Presence transaction
- 8.3.2.1 Unsubscribe Presence transaction
- 8.3.2.1 Presence Notification transaction

The rest of the presence information delivery-related functionalities are all OPTIONAL. The individual client or server implementations MAY decide whether support for a particular transaction is implemented or not.

8.3.2.1 Transactions

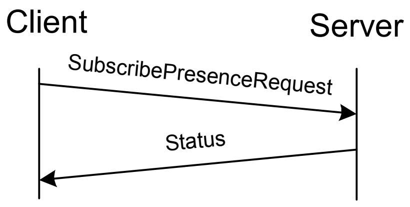

Figure 44: Subscribe presence transaction

The user MAY subscribe to presence information to receive updates whenever the subscribed presence information is updated. To subscribe presence information the requesting client MUST send a SubscribePresenceRequest primitive to the server. The server MUST respond to a SubscribePresenceRequest primitive with a Status primitive. After a successful subscription the server MUST initially send all available and authorized attributes (See 8.2 Authorization of Presence Attributes) to the requesting client using the 'presence notification transaction' – when there are no such attributes, the server MUST send an empty presence notification. While there are active subscriptions, the server MUST send subsequent presence notifications when the publisher's presence information is updated or when the publisher has newly authorized a presence attribute that was already subscribed earlier. The server MAY deliver only those presence attributes in the 'presence notification transaction' that have been updated or newly authorized.

The subscription MUST NOT be persistent through different sessions.

A subscription MUST be related the requesting client only and MUST NOT interfere with subscription of other clients of the same user.

The subscribe presence transaction MUST be supported by the client and the server, thus its support is not negotiated

The scope of the subscription MUST be either a set of users and/or a set of contact lists referring to multiple users, thus the request MUST refer to User-ID(s) and/or ContactList-IDs, but it MUST NOT refer to screen name(s).

The requesting user MAY subscribe also only a part of the presence information by including the Presence-Attribute-List element containing the desired attributes. Absence of the Presence-Attribute-List in the request indicates to the server that all available presence information is requested. The server MUST subscribe the user(s) to the set of presence attributes supplied in the request or, if the Presence-Attribute-List is absent from the request the server MUST subscribe all of the presence attributes.

The server MAY deliver some presence attributes with empty value – note that however the Qualifier of the presence attribute MUST be 'F' in this case, see Qualifier in [PA]. The empty values are typically used when the publisher did not give an initial value for the particular presence attribute, or the publisher did not authorize the subscriber and it does not wish to reveal the fact that there is no authorization.

If there are any presence attributes in the request that are already subscribed, the server MUST ignore those silently without generating an error.

In order to save bandwidth, the server MAY choose not to send the initial presence notification when – and only when – all presence attributes in the request are already subscribed.

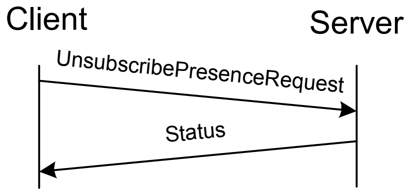

Figure 45: Unsubscribe presence transaction

When the requesting user does not want to receive presence notifications to a client anymore from an earlier subscription,, he/she MAY cancel the subscription by unsubscribing the presence information. To un-subscribe presence information the client MUST send UnsubscribePresenceRequest primitive to the server. Upon reception of such request the server MUST the server MUST stop delivering the presence attributes related to the earlier subscription. However the server MUST NOT unsubscribe presence attributes that have been subscribed via other ContactList-ID or User-ID based subscriptions of the same client or any subscriptions from another client. The server MUST respond to an UnsubscribePresenceRequest primitive with a Status primitive.

The scope of the unsubscribe presence transaction MUST be either a set of users and/or a set of contact lists referring to multiple users, thus the request MUST refer to User-ID(s) and/or ContactList-IDs, but it MUST NOT refer to screen name(s).

The unsubscribe presence transaction MUST be supported by the client and the server.

The server MUST stop delivering presence notifications of the un-subscribed users.

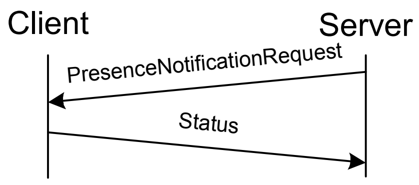

Figure 46: Presence notification

As long as an authorization and subscription is valid, the requesting user MUST receive presence information when the publisher's presence information is updated or when the publisher has newly authorized a presence attribute that was already subscribed earlier. The server MUST send the PresenceNotificationRequest primitive to the client containing the updated presence information. The client MUST respond to the PresenceNotificationRequest primitive with a Status primitive. The client and the server MUST fulfill the applicable content delivery related requirements in 5.6 Content for Presence/Instant Messaging.

The server MUST identify the users with User-IDs – ContactList-IDs. The server MUST NOT include users in the presence notification that have not been subscribed.

The client and the server MUST support the presence notification transaction, thus its support is not negotiated.

The server MAY deliver some presence attributes with empty value – note that however the Qualifier of the presence attribute MUST be 'F' in this case, see Qualifier in [PA]. The empty values are typically used when the publisher did not give an initial value for the particular presence attribute, or the publisher did not authorize the subscriber and it does not wish to reveal the fact that there is no authorization.

If the subscription did not contain the Presence-Attribute-List element, the PresenceNotificationRequest MUST contain all available and authorized presence attributes. If the subscription did contain the Presence-Attribute-List element, the server MUST deliver only the authorized subset of the requested attributes. See the 'subscribe presence transaction' for more information.

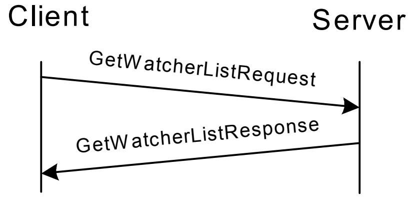

Figure 47: Get watcher list transaction

The user MAY get the list of watchers. The requesting client MUST send a GetWatcherListRequest primitive to the server. If no error occurs, the server MUST respond with a GetWatcherListResponse primitive including the list of the subscribing users, otherwise the server MUST respond with a Status primitive.

The client and the server MAY support the get watcher list transaction. The service leaf that allows negotiation of this transaction is ‘GETWL.

The server MUST include the User-ID and Watcher-Status of all subscribers if MaxWatcherList is not included in the GetWatcherListRequest primitive or the number of watchers does not exceed MaxWatcherList. If the number of watchers exceeds MaxWatcherList, the server MUST return a subset of all watchers, containing as many watchers as the MaxWatcherList value permits along with WatcherCount containing an integer value giving the total number of watchers. The server MAY also include the History-Period in the response.

Note: There are no ways of retrieving exactly what attributes every user subscribes to, but the GetAttributeList transaction can tell what attributes the users are authorized to see.

8.3.2.2 Error Conditions

Generic error conditions:

- Not logged in. (604)
- Service not agreed. (506)
- Service not supported. (405)

SubscribePresenceRequest error conditions:

- Unknown User-ID. (531)
- Contact list does not exist. (700)
- Unknown presence attribute (not defined in [PA]). (750)
- The maximum number of users in watcher list has been reached for the user. (758)
- Service provider agreement missing (907)

UnSubscribePresenceRequest error conditions:

- Unknown User-ID. (531)
- Contact list does not exist. (700)

PresenceNotificationRequest error conditions:

- Client MAY ignore any error and respond with Successful. (200)

GetWatcherListRequest error conditions:

- None except the generic error conditions.

8.3.2.3 Primitives and information elements

Primitive	Direction
SubscribePresenceRequest	Client → Server
Status	Client ← Server
UnSubscribePresenceRequest	Client → Server
Status	Client ← Server
PresenceNotificationRequest	Client ← Server
Status	Client → Server
GetWatcherListRequest	Client → Server
GetWatcherListResponse	Client ← Server

Table 75: Primitive directions

Information Element	Req	Type	Description
Message-Type	M	SubscribePresence Request	Message identifier.
Transaction-ID	M	String	Identifies the subscription request transaction.
Session-ID	M	String	Identifier for the session.
User-ID-List	C	Structure	Identifies the IM user(s).
Contact-List-ID-List	C	Structure	Identifies the set(s) of users for subscription.
Presence-Attribute-List	O	Structure	A list of presence attributes. An empty or missing list indicates all available presence attributes are desired.

Table 76: Information elements in SubscribePresenceRequest primitive

Information Element	Req	Type	Description
Message-Type	M	UnSubscribe PresenceRequest	Message identifier.
Transaction-ID	M	String	Identifies the transaction.
Session-ID	M	String	Identifier for the session.
User-ID-List	C	Structure	Identifies the IM users(s).
Contact-List-ID-List	C	Structure	Identifies the set of users to be un-subscribed.

Table 77: Information elements in UnsubscribePresenceRequest primitive

Information Element	Req	Type	Description
Message-Type	M	Presence NotificationRequest	Message identifier
Transaction-ID	M	String	Identifies the transaction.
Segment-Info	O	Structure	Contains segmentation information when the primitive is segmented.
Session-ID	M	String	Identifier for the session.
Presence-Value-List	M	Structure	List of User-IDs and its corresponding presence values.

Table 78: Information elements in PresenceNotificationRequest primitive

Information Element	Req	Type	Description
Message-Type	M	GetWatcherList Request	Message identifier.
Transaction-ID	M	String	Identifies the subscription request transaction.
Session-ID	M	String	Identifier for the session.
History-Period	O	Integer	Indicates the time period in seconds on the longest possible history of the watcher (from the time of request) that SHOULD be returned. In case of absence, it indicates the user request the watcher list at the time of the request only. The value 0 MUST NOT be used.
MaxWatcherList	O	Integer	Indicates the maximum number of Watcher elements in GetWatcherListResponse.

Table 79: Information elements in GetWatcherListRequest primitive

Information Element	Req	Type	Description
Message-Type	M	GetWatcherList Response	Message identifier.
Transaction-ID	M	String	Identifies the subscription request transaction.
Segment-Info	O	Structure	Contains segmentation information when the primitive is segmented.
Session-ID	M	String	Identifier for the session.
History-Period	O	Integer	Indicates the time period in seconds in which the watcher history has been accumulated. This value MUST NOT be larger than the requested period. Absence indicates that the Watcher information element (see below) returns only the current subscribers at the time of the request even if the history log is requested. The value 0 MUST NOT be used.
WatcherCount	C	Integer	Indicates the total number of watchers. WatcherCount is only returned if the number of entries is bigger than MaxWatcherList in the GetWatcherListRequest primitive.
Watcher	O	Structure	Identifies the watchers and their status from the history period. If this element is not present at all within the response, it indicates that the server does not give any watcher information at this time. The number of this element in GetWatcherListResponse MUST NOT be larger than Max Watcher List value in the corresponding GetWatcherListRequest.

Table 80: Information elements in GetWatcherListResponse primitive

8.3.3 Get Presence Transactions

8.3.3.1 Transactions

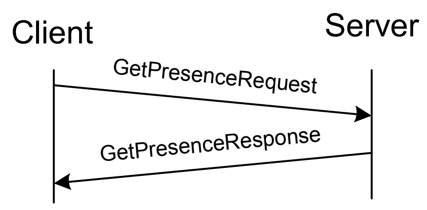

Figure 48: Get Presence transaction

A user MAY, if authorized, get another user's presence information at any time. To attempt to retrieve another user's presence information, the client MUST send a GetPresenceRequest primitive to the server containing a set of User-IDs and/or a set of ContactList-IDs, and MAY include the list of requested presence attributes. The client and the server MUST fulfill the applicable content delivery related requirements in 5.6 Content for Presence/Instant Messaging.

The request MAY refer to User-ID(s) and ContactList-IDs, but it MUST NOT refer to screen name(s).

The requesting user MAY retrieve only part of the presence information and, correspondingly, the user whose presence information is retrieved MAY allow only part of the presence information to be delivered. Absence of the Presence-Attribute-List element in the request indicates to the server that all available presence information is requested. If no information is available about a particular presence attribute, the corresponding presence attribute MUST NOT be returned in the GetPresenceResponse message. The server MUST distribute only those attributes that are authorized for the requesting user. If the request does not contain the Presence-Attribute-List element, the server MUST deliver all available and authorized presence attributes. If the request contains the Presence-Attribute-List element, the server MUST deliver only the available and authorized subset of the requested attributes.

In case of success, the server MUST respond with a GetPresenceResponse primitive containing the result of the request and the presence attributes for every requested User-ID(s). The server MUST resolve the requested ContactList-ID(s) into separate User-IDs that are used to identify the presence state.

If the Result element indicates unsuccessful transaction, the Presence-Value-List element MUST NOT be present in the response primitive.

The client and the server MAY support the get presence transaction. The service leaf that allows negotiation of this transaction is ‘GETPR’.

8.3.3.2 Error Conditions

Generic error conditions:

- Not logged in. (604)
- Service not agreed. (506)
- Service not supported. (405)

GetPresenceRequest error conditions:

- Unknown User-ID. (531)
- Contact list does not exist. (700)
- Unknown presence attribute (not defined in [PA]). (750)
- Service provider agreement missing (907)

8.3.3.3 Primitives and information elements

Primitive	Direction
GetPresenceRequest	Client → Server
GetPresenceResponse	Client ← Server

Table 81: Primitive directions for getting presence

Information Element	Req	Type	Description
Message-Type	M	GetPresenceRequest	Message identifier.
Transaction-ID	M	String	Identifies the transaction.
Session-ID	M	String	Identifier for the session.
User-ID-List	C	Structure	Identifies the set of User-IDs.
Contact-List-ID-List	C	Structure	Identifies the set of Contact-List-IDs.
Presence-Attribute-List	O	Structure	A list of presence attributes. An empty or missing list indicates all available presence attributes are desired.

Table 82: Information elements in GetPresenceRequest primitive

Information Element	Req	Type	Description
Message-Type	M	GetPresenceResponse	Message identifier.
Transaction-ID	M	String	Identifies the transaction.
Segment-Info	O	Structure	Contains segmentation information when the primitive is segmented.
Session-ID	M	String	Identifier for the session.
Result	M	Structure	Result of the request.
Presence-Value-List	O	Structure	List of User-IDs and its corresponding presence values.

Table 83: Information elements in GetPresenceResponse primitive

8.3.4 Update Presence Transactions

8.3.4.1 Transactions

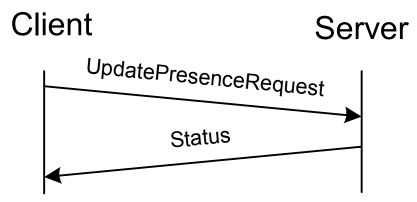

Figure 49: Update presence transaction

A publisher MAY update any of his/her own presence attributes and their values by sending an UpdatePresenceRequest primitive to the server. To update the presence information, the client MUST send the UpdatePresenceRequest primitive to the server. The server MUST respond to the UpdatePresenceRequest primitive with a Status primitive. The client and the server MUST fulfill the applicable content delivery related requirements in 5.6 Content for Presence/Instant Messaging.

Only the updated attributes and their values MUST be carried in this primitive, the omitted attributes MUST NOT be modified. The server MUST update the provided values into the PRSE.

The client and the server MAY support the update presence transaction. The service leaf that allows negotiation of this transaction is ‘UPDPR’.

8.3.4.2 Error Conditions

Generic error conditions:

- Not logged in. (604)
- Service not agreed. (506)
- Service not supported. (405)

UpdatePresenceRequest error conditions:

- Unknown presence attribute (not defined in [PA]). (750)
- Unknown presence value (not defined in [PA]). (751)

8.3.4.3 Primitives and information elements

Primitive	Direction
UpdatePresenceRequest	Client → Server
Status	Client ← Server

Table 84: Primitive directions for UpdatePresenceRequest

Information Element	Req	Type	Description
Message-Type	M	UpdatePresence Request	Message identifier
Transaction-ID	M	String	Identifies the transaction.
Segment-Info	O	Structure	Contains segmentation information when the primitive is segmented.
Session-ID	M	String	Identifier for the session.
Presence-Attribute-List	M	Structure	A list of presence values to update.

Table 85: Information elements in UpdatePresenceRequest

9. Instant Messaging Feature

In order to achieve minimum level of interoperability both the client and the server MUST support the following functionalities:

- 9.1.1 Send Message Transaction.
- Either 9.1.5 New Message Transactions or 9.1.8 Get Message Transactions - SMS transport is an exception.
- 9.2 Message-Info Structure.
- Support text/plain and the applicable requirements in 5.6 Content for Presence/Instant Messaging.

Additionally to the above, the server MUST support the following functionalities:

- 9.1.2 Set Delivery Method transaction.
- Both 9.1.5 New Message Transactions and 9.1.8 Get Message Transactions - SMS transport is an exception.
- 9.1.6 Message Notification Transactions.
- 9.1.9 Delivery Status Report Transaction

The rest of the instant messaging-related functionalities are all OPTIONAL. The individual client or server implementations MAY decide whether support for a particular transaction is implemented or not.

9.1 Delivery Transactions

9.1.1 Send Message Transaction

9.1.1.1 Transactions

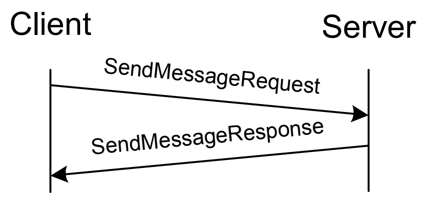

Figure 50: Send Message transaction

The user MAY send IM to other user(s), to a group or to other user(s) in a group. To send an IM, the client MUST send the SendMessageRequest primitive to the server. The client MUST include the Message-Info structure, and MAY include the Content element in the SendMessageRequest. The Message-Info structure MUST be conformant to the Message-Info structure requirements described in 9.2 Message-Info Structure.

The recipient(s) MUST be user(s) specified with User-ID(s) or screen names, ContactList-ID(s), or group(s) specified with Group-ID(s), or some combination of these that do not break the message context. See message context in 9.2 Message-Info Structure. The sender MUST be a single user specified either with User-ID or screen name. When User-ID is used to identify the sender/recipient, additionally the sending/recipient client(s) MAY be identified in the Message-Info structure with Client-ID. Similarly, when User-ID is used to identify the sender/recipient, additionally the sending/recipient application MAY be identified in the Message-Info structure with Application-ID. The client MUST NOT use both Client-ID and Application-ID to identify any sender/recipient User-ID(s) in a request.

The client MUST NOT attempt to send content types that have not been agreed during its client capability negotiation, as described in 5.6 Content for Presence/Instant Messaging. The client SHOULD NOT send content types that are not supported by the recipient(s) according to their limitations described in their ClientInfo presence attribute – unless the necessary presence information is not available (the recipient is a group or screen name, the client does not support presence, the requesting user is not authorized by the publisher, etc).

The client MUST send the instant message using a Message-URI or by including the content directly, as described in 9.2 Message-Info Structure. While in general the server SHOULD generate the instant messages according to the requirements described in 9.2 Message-Info Structure, the server MAY apply special behavior if desired:

- When the client sends the instant message using Message-URI, the server that handles the transaction MAY choose to retrieve the content from the specified URI and add it directly into the instant message. Such behavior MAY be desired when the Message-URI refers to a location that might not be available to some recipients because a 3rd party content server is used, or for security reasons when the URI includes a username/password combination. The server MUST NOT perform such behavior when the content specified by the Message-URI does not exist, or when any content policy limitation would apply for any of the recipients – which ultimately means that those servers without proper presence support and authorization MUST NOT perform this behavior.
- When the client sends the instant message by including the content directly, the server that handles the transaction MAY choose to place the content to a specific URI, and send the instant message using Message-URI. Such behavior MAY be desired when the instant message was requested to be delivered to a lot of users using the some rich content – typical use might be (but not restricted to) group messaging. The server MUST NOT perform such behavior when the content is “text/plain”, or when the sender or any of the recipients would not have access to the URI. When the server performs such behavior, the server MUST store the directly included content without changes to a location that is accessible by the sender and all of the recipients, and include the Message-URI that refers to the stored content in the response and the generated instant messages. The server MUST provide the storage space itself for the content stored using this behavior – the user MUST NOT lose any resources because of this behavior.

If the server accepted the message for delivery, the server MUST:

- respond with the SendMessageResponse message to the sending client including the Message-Info structure containing the generated Message-ID, and
- generate and send the instant messages to each requested recipient(s). The generated instant messages MUST contain the generated Message-ID, as well as all information elements from the SendMessageRequest primitive unchanged, unless:
    - content-transcoding takes place, or
    - the server uploaded the direct content to a URI,
    - or the server downloaded the URI and included the content directly.

If the server rejects the message delivery, the server MUST respond with a Status primitive. The server MAY reject the request for various reasons, however the request MUST be rejected if any of the following cases occur:

- The client placed the IM into a message context that the server is unable to handle.
- The Recipient element refers to a single ContactList-ID only, but the referenced contact list is empty.
- User-ID is included in the Sender element, but the User-ID does not match the User-ID of the requesting user.
- Client-ID is included in the Sender element, but the Client-ID does not match the Client-ID that the requesting client used during login.
- Application-ID is included in the Sender element, but the Application-ID does not match the Application-ID that has been registered for the requesting client during login.
- If the recipient has been specified by screen name(s), and the recipient user has not been joined to the group when the message delivery is imminent.
- If the message is targeted to a group and the sender has not joined to the group.
- The message is a private message within a group, however the private messaging for the group is disabled.
- The message is a private message within a group, however the private messaging for the recipient user is disabled.

The Send Message transaction is mandatory for both client and the server, thus its support is not negotiated.

9.1.1.2 Error conditions

Generic error conditions:

- Not logged in. (604)
- Service not agreed. (506)
- Service not supported. (405)

SendMessageRequest error conditions:

- Unsupported content-type. (415)
- Invalid User-ID. (427)
- Invalid Client-ID. (428)
- Invalid Application-ID. (451)
- Forbidden Application-ID. (452)
- Message queue full. (507)
- Unsupported message context. (508)
- Related client capabilities are missing (510)
- Domain not supported. (516)
- Recipient user does not exist. (531)
- Recipient user blocked the sender. (532)
- Recipient user is not logged in. (533)
- Message has been rejected due to limitations (540)
- Contact list does not exist. (700)
- Contact list is empty. (703)
- Recipient group does not exist. (800)
- Sender has not joined the group (or kicked). (808)
- Private messaging is disabled in the group. (812)
- Private messaging is disabled for the recipient. (813)
- Service provider agreement missing (907)

9.1.1.3 Primitives and information elements

Primitive	Direction
SendMessageRequest	Client → Server
SendMessageResponse	Client ← Server

Table 86: Primitive directions for Send Message Transaction

Information Element	Req	Type	Description
Message-Type	M	SendMessageRequest	Message identifier.
Transaction-ID	M	String	Identifies the transaction.
Session-ID	M	String	Identifier for the session.
Delivery-Report-Request	M	Boolean	Indicates if the user wants delivery report.
Message-Info	M	Structure	Message information data. See 9.2 Message-Info Structure.
Content	C	String | Binary data	The content of the instant message.

Table 87: Information elements in SendMessageRequest

Information Element	Req	Type	Description
Message-Type	M	SendMessageResponse	Message identifier.
Transaction-ID	M	String	Identifies the transaction.
Session-ID	M	String	Identifier for the session.
Result	M	Structure	Result of the SendMessageRequest
Message-ID	M	String	Server-generated Message-ID for this message.

Table 88: Information elements in SendMessageResponse primitive

9.1.2 Set Delivery Method transaction

9.1.2.1 Transactions

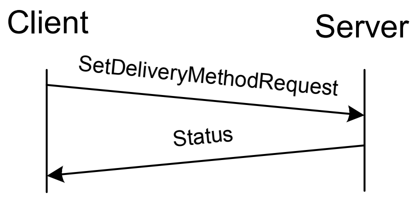

Figure 51: Set Delivery Method transaction

The recipient client MUST indicate the preferred initial IM delivery method in the Requested-Capabilities element during client capability negotiation. The client MAY change the message delivery method during the session by using the SetDeliveryMethodRequest primitive.

When SetDeliveryMethodRequest is sent to the server, the server MUST respond with a Status primitive.

If the Group-ID is present in the request, the delivery method MUST be applied to that specific group only, unless an error occurs. When Group-ID is present in the request, the client MAY also specify Group-Content-Limit. When Group-Content-Limit was specified in the request, the server MUST NOT deliver any instant messages within the active session context from the specified group that are larger than the value indicated in Group-Content-Limit.

The server MUST NOT accept the new delivery method when:

- the services for the new delivery method have not been agreed, or
- the user is rejected from the requested group, or
- the user in not a member of the requested 'Restricted' group.

When the server accepts changing the delivery method from "Notify/Get" to "Push", the server MUST deliver all instant messages waiting for delivery to the particular client.

At all times, the server MUST deliver the instant messages according to the requirements described in 5.6.4 Content delivery, unless the instant message content size is limited by the Group-Content-Limit setting.

The server MUST support the 'Set Delivery Method transaction'. The client MAY support the 'Set Delivery Method transaction'. The service tree leaf that allows negotiation of this transaction is 'SETD'.

9.1.2.2 Error Conditions

Generic error conditions:

- Not logged in. (604)
- Service not agreed. (506)
- Service not supported. (405)

SetDeliveryMethodRequest error conditions:

- Related services are missing (509)
- Group does not exist. (800)

- User has been rejected. (809)
- Not a group member. (810)
- Service provider agreement missing (907)

9.1.2.3 Primitives and information elements

Primitive	Direction
SetDeliveryMethodRequest	Client → Server
Status	Client ← Server

Table 89: Primitive directions for Set Delivery Method Transaction

Information Element	Req	Type	Description
Message-Type	M	SetDeliveryMethod Request	Message identifier.
Transaction-ID	M	String	Identifies the transaction.
Session-ID	M	String	Identifier for the session.
Delivery-Method	M	Enumerated character	Determines the type of message delivery. "Push" means that the complete message is transferred in the notification, see 9.1.5 New Message Transactions. "Notify/Get" means that only the Message-ID is transferred in the notification; the message is then retrieved using a get – see notification in 9.1.6 Message Notification Transactions and get in 9.1.8 Get Message Transactions.
Group-ID	O	String	Group-ID if delivery method refers to a group.
Group-Content-Limit	C	Integer	Maximum size of message that can be sent from the specified group. It MUST NOT be indicated when Group-ID is missing.

Table 90: Information elements in SetDeliveryMethodRequest

9.1.3 Get List of Messages Transactions

9.1.3.1 Transactions

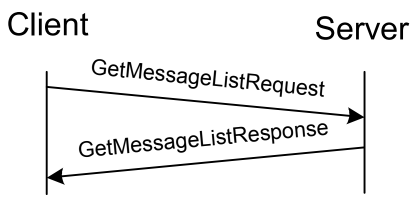

Figure 52: Get Message List transaction

The server MAY offer space where undelivered messages or group history can be stored. The client MAY discover support for offline storage using the client capability negotiation, see OfflineETEMHandling in 6.9 Service and Capability Negotiation. The client and the server MUST fulfill the applicable "Notify/Get" related requirements in 5.6.4 Content delivery.

The Get Message List transaction is used to retrieve the Message-Info structures of the undelivered instant messages. The client MAY use the retrieved Message-Info structures to inform the user about the new messages, or it MAY retrieve/reject/forward the instant messages automatically. To retrieve/reject/forward an instant message, the client MUST use a valid Message-ID from the received Message-Info structures in a GetMessageRequest/ RejectMessageRequest/ForwardMessageRequest primitive.

The Message-Info structure(s) MUST be conformant to the Message-Info structure requirements described in 9.2 Message-Info Structure.

If the Group-ID element is present in the GetMessageRequest, it MUST indicate a single Group-ID only. In this case the server MUST send back the Message-Info structures of those messages that has been sent to the specified group only. If the Group-ID element is not present, the server MUST send the Message-Info structures of all non-delivered instant messages from all users and groups, unless the Recipient element of the instant message(s) contains Client-ID/Application-ID and it does not match the Client-ID/Application-ID used by the client during login.

If the client specifies the Message-Count element in the GetMessageRequest primitive, the server MUST NOT send more Message-Info structures than the value specified in Message-Count, while when the number of instant messages on the server exceeds Message-Count, the server MUST include Total-Message-Count in the response. Whenever included, the value of Message-Count and Total-Message-Count MUST be higher than zero.

When there are no instant messages stored on the server, the transaction MUST fail with error code 908.

When GetMessageListRequest primitive is sent from the client to the server and no error occurs, the server MUST respond with a GetMessageListResponse primitive. The GetMessageListResponse primitive MUST contain the requested Message-Info structure(s), unless there are no messages.

The server and the client MAY support the Get Message List transaction. The service tree leaf that allows negotiation of this transaction is ‘GETLM’.

9.1.3.2 Error Conditions

Generic error conditions:

- Not logged in. (604)
- Service not agreed. (506)
- Service not supported. (405)

GetMessageListRequest error conditions:

- Message-Count exceeded (539)
- Group does not exist. (800)
- History is not supported. (821)
- There are no instant messages. (908)

9.1.3.3 Primitives and information elements

Primitive	Direction
GetMessageListRequest	Client → Server
GetMessageListResponse	Client ← Server

Table 91: Primitive directions for Get Message List transactions

Information Element	Req	Type	Description
Message-Type	M	GetMessageListRequest	Message identifier.
Transaction-ID	M	String	Identifies the transaction.
Session-ID	M	String	Identifier for the session.
Group-ID	C	String	Identifies the group to retrieve history.
Message-Count	O	Integer	The maximum number of Message-Info structures to be returned.

Table 92: Information elements in GetMessageListRequest

Information Element	Req	Type	Description
Message-Type	M	GetMessageList Response	Message identifier.
Transaction-ID	M	String	Identifies the transaction.
Segment-Info	O	Structure	Contains segmentation information when the primitive is segmented.
Session-ID	M	String	Identifier for the session.
Message-Info-List	C	Structure	Message information data for each message. See 9.2 Message-Info Structure.
Message-Total-Count	C	Integer	When the client included Message-Count in the request but there are more instant messages waiting than the client requested, the server MUST include this value indicating the total number of instant messages.

Table 93: Information elements in GetMessageListResponse

9.1.4 Reject Message Transactions

9.1.4.1 Transactions

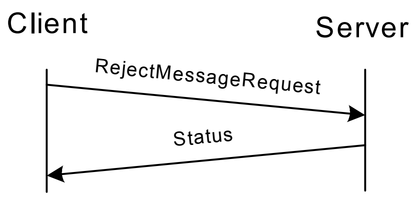

Figure 53: Reject Message transaction

The user MAY accumulate a number of unwanted messages. The user MAY request rejecting the accumulated messages.

To reject the accumulated messages the client MUST send the RejectMessageRequest primitive containing the ID(s) of the instant message(s) to the server. The server MUST respond with a Status primitive indicating the outcome and remove the requested messages from the server.

If the originating user requested delivery report, the server MUST be indicate for him/her that the message has been rejected.

The client and the server MAY support the Reject Message transaction. The service tree leaf that allows negotiation of this transaction is 'REJCM'.

9.1.4.2 Error Conditions

Generic error conditions:

- Not logged in. (604)
- Service not agreed. (506)
- Service not supported. (405)

RejectMessageRequest error conditions:

- Invalid Message-ID. (426)

9.1.4.3 Primitives and information elements

Primitive	Direction
RejectMessageRequest	Client → Server
Status	Client ← Server

Table 94: Primitive directions for Reject Message transactions

Information Element	Req	Type	Description
Message-Type	M	RejectMessageRequest	Message identifier.
Transaction-ID	M	String	Identifies the transaction.
Session-ID	M	Integer	Identifier for the session.
Message-ID-List	M	Structure	Identifies the messages to be removed using a list of Message-ID(s).

Table 95: Information elements in RejectMessageRequest

9.1.5 New Message Transactions

9.1.5.1 Transactions

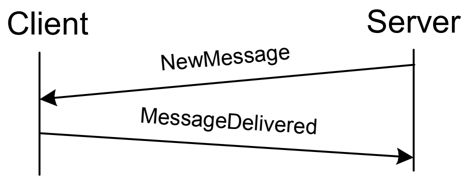

Figure 54: New Message transaction

The NewMessage primitive delivers new messages to the client. The client and the server MUST fulfill the applicable "Push" related requirements in 5.6.4 Content delivery. The Message-Info structure MUST be conformant to the Message-Info structure requirements described in 9.2 Message-Info Structure. The client MUST respond to the NewMessage primitive with a MessageDelivered primitive, unless an error occurs. The MessageDelivered primitive MUST contain Message-ID, which MUST be identical to the Message-ID from the NewMessage primitive.

The client MUST handle the received instant message according to the applicable message context and message content requirements. See message context in 9.2 Message-Info Structure, and see message content in 5.6 Content for Presence/Instant Messaging.

When the server has received the MessageDelivered primitive successfully, it MUST remove the instant message(s) referred to with Message-ID from the IMSE. Also, if the originator of the instant message requested delivery status report, the server MUST send delivery status report to the sender as described in 9.1.9 Delivery Status Report Transaction.

The server MUST support the NewMessage transaction. The client MAY support this transaction, however if it is not supported, the client MUST support 9.1.8 Get Message Transactions. The service tree leaf that allows negotiation of this transaction is 'NEWM'.

9.1.5.2 Error Conditions

Generic error conditions:

- Not logged in. (604)
- Service not agreed. (506)
- Service not supported. (405)

NewMessage error conditions:

- Client will not accept the message delivery. (410)

- Client does not support the content type. (415)

MessageDelivered error conditions:

- Invalid Message-ID. (426)

9.1.5.3 Primitives and information elements

Primitive	Direction
NewMessage	Client ← Server
MessageDelivered	Client → Server

Table 96: Primitive directions for Message Delivery Transactions

Information Element	Req	Type	Description
Message-Type	M	NewMessage	Message identifier.
Transaction-ID	M	String	Identifies the transaction.
Session-ID	M	String	Identifier for the session.
Message-Info	M	Structure	Message information data. See 9.2 Message-Info Structure.
Content	C	String | Binary data	Message data. Not present if Message-Info contains a Message-URI.

Table 97: Information elements in NewMessage

Information Element	Req	Type	Description
Message-Type	M	MessageDelivered	Message identifier.
Transaction-ID	M	String	Identifies the transaction. Either generated by client (after a GetMessageRequest) or by server when client is responding to a NewMessage.
Session-ID	M	String	Identifier for the session.
Message-ID	M	String	Message-ID of message that has been delivered successfully.

Table 98: Information elements in MessageDelivered

9.1.6 Message Notification Transactions

9.1.6.1 Transactions

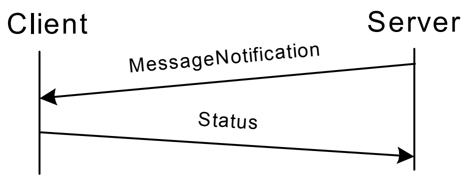

Figure 55: Message Notification transaction

The user MAY receive MessageNotification(s) from the server when the delivery method is "Notify/Get". The client and the server MUST fulfill the applicable "Notify/Get" related requirements in 5.6.4 Content delivery. The Message-Info structure MUST be conformant to the Message-Info structure requirements described in 9.2 Message-Info Structure. The client MUST respond to a MessageNotification primitive with a Status primitive.

The server MUST support the Message Notification transaction. The client MAY support the Message Notification transaction. The service tree leaf that allows negotiation of this transaction is 'NOTIF'. If the client doesn't support this, transaction, it MUST support the NewMessage transaction.

9.1.6.2 Error Conditions

MessageNotification error conditions:

- The client MAY ignore any error and respond with Successful. (200)

9.1.6.3 Primitives and information elements

Primitive	Direction
MessageNotification	Client ← Server
Status	Client → Server

Table 99: Primitive directions for Message Notification Transaction

Information Element	Req	Type	Description
Message-Type	M	Message Notification	Message identifier.
Transaction-ID	M	String	Identifies the transaction.
Session-ID	M	String	Identifier for the session.
Message-Info	M	Structure	Message information data. See 9.2 Message-Info Structure.

Table 100: Information elements in MessageNotification

9.1.7 Offline Message Notification

The IMPS system SHOULD support reception of, storage of, and subsequent delivery of Instant Messages from logged-in sender users to the logged-out recipient users – shortly offline messages.

When the recipient's home server supports offline messages, the home server - and only the home server - SHOULD send an offline message notification whenever the a new instant message arrives while the recipient is offline (not logged in) providing that the client requested such feature on behalf of the recipient during service negotiation and an offline bearer has been successfully agreed during client capability negotiation. The details of the offline notification are described in "Transport Binding for Offline Notifications" [CSP Trans].

The client MAY turn the offline notification on and off by performing service negotiation again, however the server MUST use the last valid value after the client has logged out/was disconnected.

The client and the server MAY support the offline notification. The service tree leaf that allows negotiation of the offline notification is 'OFFNOTIF'.

9.1.8 Get Message Transactions

9.1.8.1 Transactions

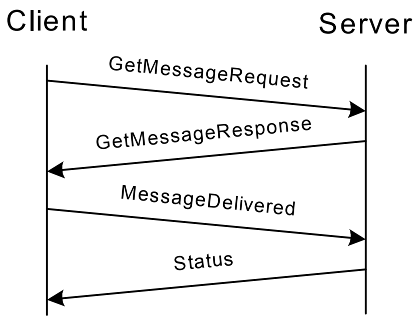

Figure 56: Get Message Transaction

The client MAY retrieve instant message stored on the server. To the retrieve a stored instant message, the client MUST use the GetMessageRequest primitive. The client and the server MUST fulfill the applicable “Notify/Get” related requirements in 5.6.4 Content delivery. The Message-Info structure MUST be conformant to the Message-Info structure requirements described in 9.2 Message-Info Structure.

When the client receives the GetMessageResponse primitive and no error occurs, the client MUST send MessageDelivered primitive to the server. The MessageDelivered primitive MUST contain Message-ID, which MUST be identical to the Message-ID from the GetMessageResponse primitive.

When the server has received the MessageDelivered primitive successfully, it MUST remove the instant message(s) referred to with Message-ID from the IMSE, and respond with a Status primitive to the client. Also, if the originator of the instant message requested delivery status report, the server MUST send delivery status report to the sender as described in 9.1.9 Delivery Status Report Transaction.

The client MUST handle the received instant message according to the applicable message context and message content requirements. See message context in 9.2 Message-Info Structure, and see message content in 5.6 Content for Presence/Instant Messaging.

The server MUST support the Get Message transaction. The client MAY support this transaction, however if it is not supported, the client MUST support 9.1.5 New Message Transactions. The service tree leaf that allows negotiation of this transaction is ‘GETM’.

9.1.8.2 Error Conditions

Generic error conditions:

- Not logged in. (604)
- Service not agreed. (506)
- Service not supported. (405)

GetMessageRequest error conditions:

- Invalid Message-ID. (426)
- Message cannot be delivered due to limitations (546)

9.1.8.3 Primitives and information elements

Primitive	Direction
GetMessageRequest	Client → Server
GetMessageResponse	Client ← Server
MessageDelivered	Client → Server
Status	Client ← Server

Table 101: Primitive directions for Get Message Transaction

Information Element	Req	Type	Description
Message-Type	M	GetMessageRequest	Message identifier.
Transaction-ID	M	String	Identifies the transaction.
Session-ID	M	String	Identifier for the session.
Message-ID	M	String	ID of the message to retrieve.

Table 102: Information elements in GetMessageRequest

Information Element	Req	Type	Description
Message-Type	M	GetMessage Response	Message identifier.
Transaction-ID	M	String	Identifies the transaction. Either generated by server (PUSH) or by client when doing a GetMessageRequest.
Session-ID	M	String	Identifier for the session.
Message-Info	M	Structure	Message information data. See 9.2 Message-Info Structure.
Content	C	String | Binary data	Message data. Not present if Message-Info contains a Message-URI.

Table 103: Information elements in GetMessageResponse

9.1.9 Delivery Status Report Transaction

9.1.9.1 Transactions

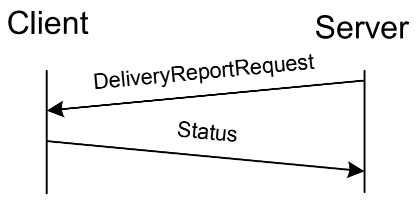

Figure 57: Delivery Status Report Transaction

Delivery status report MAY be requested when the instant message is sent; see 9.1.1 Send Message Transaction. If the sender did request a delivery report, the server

- MUST inform the client about successful delivery. In case of point-to-point messaging the delivery status report MUST be sent when the MessageDelivered primitive has been successfully received from the recipient client. In case of group messaging the delivery status report MUST be sent when the GRSE has accepted the message for delivery. In case of routing the message to multiple clients of a single recipient user in point-to-point messaging, a server MAY choose to send only one delivery report to the sender of the message.
- MAY inform the client about unsuccessful delivery attempt(s) due to various detected error conditions on the recipient client/server side.

To inform the client about delivery status report(s) the server MUST send the DeliveryReportRequest primitive to the client. The DeliveryReportRequest primitive MAY include Delivery-Time element containing the timestamp of the successful/unsuccessful delivery attempt. The DeliveryReportRequest primitive MUST include:

- Result element describing success/failure of the instant message, and
- Message-Info structure that provides information about the instant message in question. The Message-Info structure MUST be conformant to the Message-Info structure requirements described in 9.2 Message-Info Structure.

The client MUST respond to the DeliveryReportRequest primitive with a Status primitive.

If the client – to which delivery report sending is imminent – is offline, the server MUST send the delivery report according to the offline client's OfflineETEMHandling setting.

The server MUST support the Delivery Status Report transaction. The client MAY support the Delivery Status Report transaction. The service tree leaf that allows negotiation of this transaction is 'MDELIV'.

9.1.9.2 Error conditions

Generic error conditions:

- Not logged in. (604)
- Service not agreed. (506)
- Service not supported. (405)

DeliveryReportRequest error conditions:

- Client MAY ignore any error and respond with Successful. (200)

DeliveryReportRequest error indications:

- Unsupported content-type. (415)
- Domain not supported. (516)
- Contact list does not exist. (700)
- Recipient user does not exist. (531)
- Recipient user blocked the sender. (532)
- Recipient user is not logged in. (533)
- Message queue full. (507)
- Recipient group does not exist. (800)
- Sender has not joined the group (or kicked). (808)
- Private messaging is disabled in the group. (812)
- Private messaging is disabled for the recipient. (813)
- Message has been rejected (538)
- Message has been forwarded (541)
- Message has expired (not delivered) (542)

9.1.9.3 Primitives and information elements

Primitive	Direction
DeliveryReportRequest	Client ← Server
Status	Client → Server

Table 104: Primitive directions for delivery status report transaction

Information Element	Req	Type	Description
Message-Type	M	DeliveryReport Request	Message identifier.
Transaction-ID	M	String	Identifies the transaction requested.
Session-ID	M	String	Identifies the session.
Result	M	Structure	Result of the delivery.
Delivery-Time	O	DateTime	Date and time of delivery.
Message-Info	M	Structure	Message information data. See 9.2 Message-Info Structure.

Table 105: Information elements in DeliveryReportRequest primitive

9.1.10 Forward message transaction

9.1.10.1 Transactions

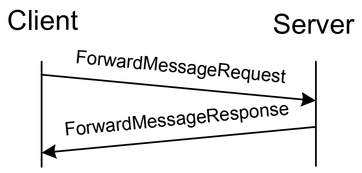

Figure 58: Forward message transaction

An instant message that was not retrieved yet and stored on the server MAY be forwarded to another recipient. . To forward an IM to another recipient, the client MUST send ForwardMessageRequest primitive to the server containing the Message-ID of the IM to be forwarded.

The recipient(s) MUST be user(s) specified with User-ID(s) or screen names, ContactList-ID(s), or group(s) specified with Group-ID(s), or some combination of these that do not break the message context. See message context in 9.2 Message-Info Structure. The sender MUST be a single user specified either with User-ID or screen name. When User-ID is used to identify the sender/recipient, additionally the sending/recipient client(s) MAY be identified in the Sender/Recipient element with Client-ID. Similarly, when User-ID is used to identify the sender/recipient, additionally the sending/recipient application MAY be identified in the Sender/Recipient element with Application-ID. The client MUST NOT use both Client-ID and Application-ID to identify any sender/recipient User-ID(s) in a request.

The client SHOULD NOT forward content types that are not supported by the recipient(s) according to their limitations described in their ClientInfo presence attribute – unless the necessary presence information is not available (the recipient is a group or screen name, the client does not support presence, the requesting user is not authorized by the publisher, etc).

If the server accepted the message for delivery, the server MUST:

- respond with the ForwardMessageResponse message to the sending client including the Message-Info structure containing the generated Message-ID, and
- remove the forwarded instant message from the user's IM storage, and
- generate and send the instant messages to each requested recipient(s). The generated instant messages MUST contain the generated Message-ID. The server MUST add the Sender/Recipient elements from ForwardMessageRequest primitive into the generated Message-Info structure(s). The generated Message-Info structure MUST be conformant to the Message-Info structure requirements described in 9.2 Message-Info Structure. The server MUST copy the rest of the information elements into the generated instant messages from the original instant message stored on the server, unless:
  - content-transcoding takes place, or
  - the server uploaded the direct content to a URI,
  - or the server downloaded the URI and included the content directly.

If the server rejects the message delivery, the server MUST respond with a Status primitive. The server MAY reject the request for various reasons, however the request MUST be rejected if any of the following cases occur:

- The client placed the IM into a message context that the server is unable to handle.
- The Recipient element refers to a single ContactList-ID only, but the referenced contact list is empty.
- User-ID is included in the Sender element, but the User-ID does not match the User-ID of the requesting user.
- Client-ID is included in the Sender element, but the Client-ID does not match the Client-ID that the requesting client used during login.
- Application-ID is included in the Sender element, but the Application-ID does not match the Application-ID that has been registered for the requesting client during login.

- If the recipient has been specified by screen name(s), and the recipient user has not been joined to the group when the message delivery is imminent.
- If the message is targeted to a group and the sender has not joined to the group.
- The message is a private message within a group, however the private messaging for the group is disabled.
- The message is a private message within a group, however the private messaging for the recipient user is disabled.

The server MUST support the Forward Message transaction. The client MAY support the Forward Message transaction. The service tree leaf that allows negotiation of this transaction is 'FWMSG'.

9.1.10.2 Error conditions

Generic error conditions:

- Not logged in. (604)
- Service not agreed. (506)
- Service not supported. (405)

DeliveryReportRequest error conditions:

- Unsupported content-type. (415)
- Invalid Message-ID. (426)
- Invalid User-ID. (427)
- Invalid Client-ID. (428)
- Invalid Application-ID. (451)
- Forbidden Application-ID. (452)
- Message queue full. (507)
- Unsupported message context. (508)
- Related client capabilities are missing (510)
- Domain not supported. (516)
- Recipient user does not exist. (531)
- Recipient user blocked the sender. (532)
- Recipient user is not logged in. (533)
- Contact list does not exist. (700)
- Contact list is empty. (703)
- Recipient group does not exist. (800)
- Sender has not joined the group (or kicked). (808)
- Private messaging is disabled in the group. (812)
- Private messaging is disabled for the recipient. (813)

9.1.10.3 Primitives and information elements

Primitive	Direction
ForwardMessageRequest	Client → Server
ForwardMessageResponse	Client ← Server

Table 106: Primitive directions for forward message transaction

Information Element	Req	Type	Description
Message-Type	M	ForwardMessage Request	Message identifier.
Transaction-ID	M	String	Identifies the transaction.
Session-ID	M	String	Identifier for the session.
Message-ID	M	String	Identifies the message to forward.
Sender	M	Structure	Identifies the user who forwarded the instant message. (User-ID, User-ID with Client-ID or User-ID with Application-ID.)
Recipient	M	Structure	Identifies the user(s) to whom the instant message SHOULD be forwarded. (User-ID, User-ID withClient-ID, User-ID with Application-ID, ContactList-ID, Group-ID, screen name, or any combination of these.)

Table 107: Information elements in ForwardMessageRequest

Information Element	Req	Type	Description
Message-Type	M	ForwardMessageResponse	Message identifier.
Transaction-ID	M	String	Identifies the transaction.
Session-ID	M	String	Identifier for the session.
Message-ID	M	String	Server-generated Message-ID the forwarded message.

Table 108: Information elements in ForwardMessageResponse primitive

9.1.11 Extend one-to-one IM conversation transactions

9.1.11.1 Transactions

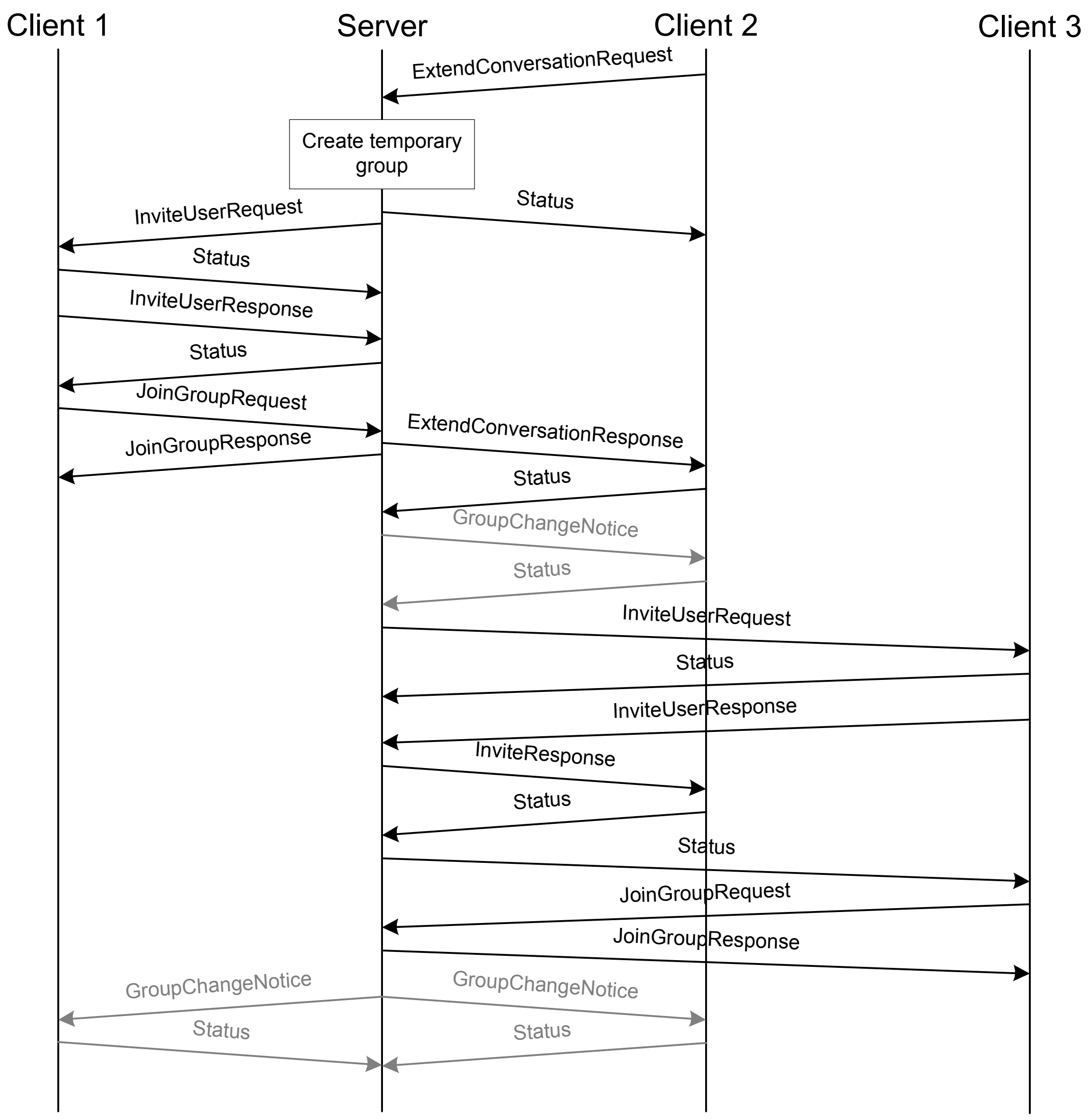

Figure 59: ExtendConversation transactions

A user can, while having an IM conversation with another user, decide to invite other user to the conversation. The initiating client, in above diagram Client2, sends an ExtendConversationRequest. The ExtendConversationRequest MUST contain a list of User IDs of the contacts that he/she wants to invite to the ongoing conversation. The ExtendConversationRequest also includes the ExtendConversationUser, which identifies the user, that Client2 is already having a conversation with, i.e. Client1. Optionally, the ExtendConversationUser can be specified with both a UserID and ClientID to route the invitation to the correct client. If the client is offline then OfflineETEMHandling MUST be used by the server to route the invitation. If a ClientID is not specified on the ExtendConversationUser then OnlineETEMHandling of that user MUST be used by the server to route the invitation.

The client MUST also include an ExtendConversation-ID in the ExtendConversationRequest. This ExtendConversation-ID MUST be unique in the scope of the session.

The client MAY request subscribing to group change notifications. If the Subscribe-Notif element is 'T' (true) in the request and the group change notification transaction was agreed during service negotiation, the server MUST subscribe the user to group change notification.

The client MAY define a welcome note in the ExtendConversationRequest. If no welcome note is defined, the server MAY set one.

The Client MUST also include the Own-Screen-Name element in the ExtendConversationRequest. The Own-Screen-Name element MUST contain the screen name of User2 and a “dummy” Group-ID that is generated by the Client. The server MUST ignore the “dummy” Group-ID when it receives an ExtendConversationRequest.

When the server receives ExtendConversationRequest it MUST create a group with all the following properties defined:

- Type: Public. The server owns the group. I.e. no users are allowed to delete the group or update the group properties.
- Accesstype: Open. All participants MUST be allowed to invite other users to join the *Private Group Conversation*.
- RequireInvitation: ‘T’. The server MUST only allow users who have explicitly been invited to the conversation to join the *Private Group Conversation*.
- AutoDelete: ‘T’. The server MUST delete the group as soon as the last participant leaves the *Private Group Conversation*.
- PrivateMessaging: ‘F’. Whispering MUST NOT be allowed within the *Private Group Conversation*.
- Searchable: ‘F’. The server MUST NOT include *Private Group Conversations* in search results.

When the server receives an ExtendConversationRequest primitive, it MUST check if Client1 supports ‘Extend one-to-one IM conversation’ functionality. In case Client1 does not support it, the server MUST reject the ExtendConversationRequest primitive by sending a Status primitive that includes the Result element with error code 909.

When the group has successfully been created the server MUST respond to Client 2 with Status primitive. If the server fails to create the group of any reasons (e.g. maximum number of groups reached) it MUST respond with a Status primitive including the proper error code.

When the server receives ExtendConversationRequest primitive and no error occurs, it MUST send InviteUserRequest to the user specified in the ExtendConversation-User-ID element, i.e. Client1. The server MUST set the Invite-Type in the InviteUserRequest to ExtendConversation (EC) in order to indicate to the receiving client to handle this invitation seamlessly in the scope of an active IM conversation. Also, the Invite-ID in the InviteUserRequest MUST be the same as the ExtendConversation-ID. If Client1 receives the request and no errors occur, Client1 MUST respond with a Status primitive. If the server receives InviteUserResponse sent from Client 1, where the Invite-ID refers to invitation with Invite-Type set to ‘EC’ (ExtendConversation) and the Invite-Acceptance is set to ‘F’ then the server MUST send a ExtendConversationResponse with status code 825. If the Invite-Acceptance is set to ‘T’ the server MUST ignore the response. . If Client1 has included a Response-Note in the InviteUserResponse, then the server MUST include this Response-Note in the ExtendConversationResponse.

If User1 accepts the invitation, then Client1 MUST sends a JoinGroupRequest to the server. When the server receives JoinGroupRequest it MUST respond to Client1 with JoinGroupResponse and MUST also send a ExtendConversationResponse to Client2. The ExtendConversationResponse MUST include the Group ID of the created group and the ExtendConversation-ID provided by the client in the ExtendConversationRequest. At this point both Client2 and the server MUST assume the initiating user to be joined in the group and hence all the messages sent between the two clients MUST be addressed to the Group ID.

When the server sends InviteUserRequest to Client1, it MUST set a timer supervising the response from Client1. The value of the timer is a server implementation issue and MAY differ.

If Client1 does not send a JoinGroupRequest to the server before the timer times out, then the IM conversation will not be extended to a *Private Group Conversation*. In this case the server MUST send ExtendConversationResponse to Client2 and declare the failure in the Result element. If this happens then the server MUST NOT send an InviteUserRequest to Client3.

Once the two initial users have joined the group, the server MUST send an InviteUserRequest with Invite-Type defined as ExtendGroup (EG), to all User IDs in the User-ID-List element, e.g. Client3. The Invite-ID in the InviteUserRequest MUST be the same as the ExtendConversation-ID. Before the server sends InviteUserRequest primitives to the invited users, it MUST check if the recipient client supports ‘Mandatory Group’ (MG), and 'Invitation' (INVIT) functionality (see 10.9 Subscribe to Group Change Notification) for each User ID in the User-ID-List. If a recipient client does not support this functionality the server MUST NOT send any InviteUserRequest to that specific client. If this happens the server MUST include in the ExtendConversationResponse that is sent to Client2 the User IDs of all those users who did not receive an InviteUserRequest because of lacking support for the required functionality in the detailed results with error code 909.

The server MUST NOT reveal any message history prior to Client 3 participation in the conversation. When Client3 receives such InviteUserRequest, it MUST handle it as any other group invitation.

If the Invite-ID refers to invitations with Invite-Type set to 'EG' (ExtendGroup), the server MUST forward InviteUserResponse primitives sent from User IDs in the User-ID-List element, e.g. Client3, by sending an InviteResponse back to Client2.

The server MUST support the ExtendConversation transactions. The client MAY support the ExtendConversation transactions. The service tree leaf that allows negotiation of this transaction is 'EXCON'.

9.1.11.2 Error conditions

Generic error conditions:

- Not logged in. (604)
- Service unavailable. (503)
- Service not agreed. (506)
- Service not supported. (405)

ExtendConversationRequest error conditions:

- Delivery to recipient not available. (410)
- Delivery to recipient domain not available. (516)
- Recipient unknown (UserID or screen-name). (531)
- The maximum number of groups has been reached (server-limit). (815)
- Extend conversation rejected. (825)
- Service provider agreement missing (907)
- Recipient does not support the requested functionality. (909)

9.1.11.3 Primitives and information elements

Primitive	Direction
ExtendConversationRequest	Client → Server
Status	Client ← Server
ExtendConversationResponse Status	Client ← Server Client → Server

Table 109: Primitive directions for extend conversation transaction

Information Element	Req	Type	Description
Message-Type	M	ExtendConversation Request	Message identifier.
Transaction-ID	M	String	Identifies the transaction.
Session-ID	M	String	Identifier for the session.
ExtendConversation-ID	M	String	Identifier for the Extended Conversation
User-ID-List	M	Structure	A list of User-Ids, which identifies the new users who are to be invited to the existing conversation
ExtendConversation-User-ID	M	Structure	Identifies the user who is already in the conversation.
Subscribe-Notif	M	Boolean	A flag indicating that the client wants to

Information Element	Req	Type	Description
			activate the group change notifications while joining the group.
Welcome-Text	O	String	A welcome message for the group.
Own-Screen-Name	M	Structure	The user's screen name from whom the Extended Conversation request originates.

Table 110: Information elements in ExtendConversationRequest

Information Element	Req	Type	Description
Message-Type	M	ExtendConversation Response	Message identifier.
Transaction-ID	M	String	Identifies the transaction.
Session-ID	M	String	Identifier for the session.
ExtendConversation-ID	M	String	Identifier for the Extended Conversation
Group-ID	M	String	Identifies the group that has been created.
Response-Note	C	String	Indicates the reason why Extended Conversation has been rejected or accepted by the other party.
Result	M	Structure	Result of the request.

Table 111: Information elements in ExtendConversationResponse

9.2 Message-Info Structure

The Message-Info structure is an information element that contains information about the characteristics of an IM.

This section lists the requirements for the Message-Info structure that MUST be fulfilled to guarantee interoperability of various client and server implementations. Both the client and the server MUST support the Message-Info element requirements. These are mandatory requirements and thus not negotiated.

The Message-Info structure MUST include the Recipient and Sender elements, while it MAY include MessageID, MessageURI, ContentType, ContentEncoding, ContentSize, ContentName, DateTime, Font, Validity – based on the primitive in which the Message-Info structure is used. The primitive-specific conditions are described in this section wherever such condition is applicable.

The Sender, Recipient and Validity elements, as well as the server routing requirements are described in 5.5.2 End-to-end message routing – all client and server implementations providing any kind of IM services MUST conform to these the IM requirements.

The Message-ID is a unique identifier of an instant message that is sent/received. The Message-ID element MUST NOT be present when the Message-Info structure is used in the SendMessageRequest primitive – the Message-ID is assigned by the server and returned in SendMessageResponse primitive unless the IM was not accepted for delivery as described in 9.1.1 Send Message Transaction. The server-generated Message-ID MUST be unique in the scope of the sender and all recipients. When the Content element is missing from the SendMessageRequest primitive, the client MUST include Message-URI in the Message-Info structure. In all other primitives than SendMessageRequest Message-ID MUST be included, unless an error occurs. The Message-URI SHOULD refer to an existing content that is accessible by the recipient(s). Whenever Message-URI is included, the specified URI MUST indicate the location of the content as well as the protocol that MUST be used to retrieve the content. Whenever the client receives an instant message containing Message-URI, the client SHOULD retrieve the actual content from the URI specified in Message-URI, and present the referred content to the user instead of presenting the URI itself.

ContentType SHOULD be indicated, however when it is not indicated, text/plain content type MUST be assumed. When the ContentType is indicated, it MUST contain the MIME type of the related content.

ContentEncoding MUST be indicated when the content in encoded with some encoding method, however when it is not indicated the content MUST be handled assuming that it is not transfer-encoded. When ContentEncoding is indicated, it MUST identify the transfer-encoding method that has been applied to the related content.

ContentSize MUST be always indicated. ContentSize MUST indicate the exact size of the related content in character count, however when the content is transfer-encoded then the ContentSize element MUST indicate the exact size of the related content in characters after the encoding took place.

If there is an original name associated with the multimedia content that will be sent in an IM, the client MAY include the original name of the multimedia content in the SendMessageRequest primitive utilizing ContentName in the Message-Info structure. When the server receives a SendMessageRequest primitive that includes the original name in the Message-Info structure, the server MUST include the original name of the multimedia content in the Message-Info structure of the generated IM(s) as well. When the client receives an IM, it MAY take use of the original name of the multimedia content in the Message-Info structure.

The client MUST send the IM in a certain message context. The message context is derived from the contents of the Sender and Recipient elements of the Message-Info structure in the SendMessageRequest primitive. The client MAY place the IM into any context, however the IMPS specifications define only a subset of all available combinations. It is RECOMMENDED that the client implementations do not deviate from the session contexts defined in the IMPS specifications, unless a proprietary agreement has been done using the extension framework, as described in 12 Extension Framework. The proprietary extensions MUST NOT re-define any of the message contexts defined in the IMPS specifications. The IMPS specification defines the following message contexts:

- IM from a user to other user(s). When an IM MUST be delivered to a user from other user(s), the Sender element MUST contain the User-ID of the sender and the Recipient element MUST contain the User-ID of the recipient(s) or ContactList-ID(s) as a reference(s) to list(s) of User-IDs.
- IM from a user to all joined users of a group. When an IM MUST be delivered to all joined user in a group, the Sender element MUST contain the Screen-Name of the sender in that group and the Recipient element MUST contain the Group-ID of the group.
- IM from a user to one specific user in a group (also known as whispering). When a private IM MUST be delivered to a user from another user in a group, the Sender element MUST contain the Screen-Name of the sender in that group and the Recipient element MUST contain the Screen-Name used by the recipient in that group.

When the client is unable to determine the message context of a received instant message based on the information above, the client MUST place the instant message into a message context based on the Recipient element. When the Recipient element is:

- User-ID with or without Client-ID/Application-ID, the instant message MUST be handled as “IM from a user to another user“.
- Group-ID, the instant message MUST be handled as “IM from a user to all joined users of a group“.
- Screen-Name, the instant message MUST be handled as “IM from a user to one specific user in a group”.

Before delivering the IM to the requested recipient the server MUST verify the access control rules of the recipient versus the Recipient and Sender elements as described in chapter 9.3 Access Control Transactions.

The DateTime MUST NOT be present when the structure is used SendMessageRequest primitive – it is the server that MUST add the DateTime element when the IM has been accepted for delivery. In other primitives DateTime MUST be present. The timestamp that the server adds to the IM MUST be based on the current time and time zone settings of the server that accepts the IM for delivery.

When the content type of the IM is "text/plain", the client MAY include font-formatting information in the SendMessageRequest primitive utilizing the Font element of the Message-Info structure. The Font element MAY include the size, the color and various style elements, and it MUST apply to entire content of the plain text message content. When the originating client includes formatting information, the server MUST deliver it to the recipient client(s) without changes. The recipient client(s) MAY ignore the formatting information. In order to avoid rendering unreadable text, it is RECOMMENDED that the recipient client verifies the specified text color versus the background color, and adjusts the text color when necessary.

The sender on an IM MAY request delivery report using the Delivery-Report-Request information element. The sending client MUST NOT request delivery report if the functionality (see 9.1.9 Delivery Status Report Transaction) was not agreed during service negotiation. If the Delivery-Report-Request element in the SendMessageRequest primitive indicates ‘T’ (true), and the server accepted the instant message for delivery, the server MUST inform the client using the Delivery Status Report Transaction whenever the IM has been successfully delivered to a recipient either using “Push” or “Notify/Get”.

The Validity element is described in detail in 5.6.4.2 Validity.

9.3 Access Control Transactions

9.3.1 Blocking Incoming Messages and Invitations Transaction

The concept of *blocking* means restricting message or invitation delivery from certain entities. These entities MAY be:

- user(s) specified by User-ID(s) or Screen-Name(s), and
- group(s) specified by Group-ID(s).
- contact list(s) specified by ContactList-ID(s).
- application(s) specified by Application-ID(s).

The server MUST NOT treat the entries in the BlockList and GrantList case sensitive.

The server MUST evaluate the conditions before delivering the IM or invitation to the user. The server MUST evaluate User-ID(s), Screen-Name(s) based on the Sender element. The server MUST evaluate Group-ID(s), Application-ID(s) based on the Recipient element. In invitation/cancel invitation requests the server MUST also evaluate Application-ID in the Invite-Application and Cancelled-Application elements as well.

The block/grant lists MAY include ContactList-IDs. Including/removing a contact list in/from the grant or block list affects all the users of the contact list(s) in the grant or block list. The server MUST use the up-to-date contents of the contact list(s) included in the grant or block list when it evaluates the grant/block decision procedure (see Figure 60: Blocking decision-tree).

If Blocked-Entity-List is in use, the server MUST remove all messages and invitations that are either coming from user(s) or are sent to specific Application-IDs on the Blocked-Entity-List (not to deliver).

If Granted-Entity-List is in use, the server MUST allow only those messages and invitations that are coming from user(s) and Application-IDs on the Granted-Entity-List, The server MUST remove all other messages and invitations (not to deliver).

Blocking is active for the blocked entities(s) until the user decides to turn off the use of the Blocked-Entity-List or to unblock (remove from the list) the particular entity.

Granting is active for the granted entities(s) until the user decides to turn off the use of the Granted-Entity-List or to un-grant (remove from the list) the particular entity.

If both Blocked-Entity-List and Granted-Entity-List are in use at the same time, the Blocked-Entity-List list MUST have higher priority over the Granted-Entity-List; see the decision tree below:

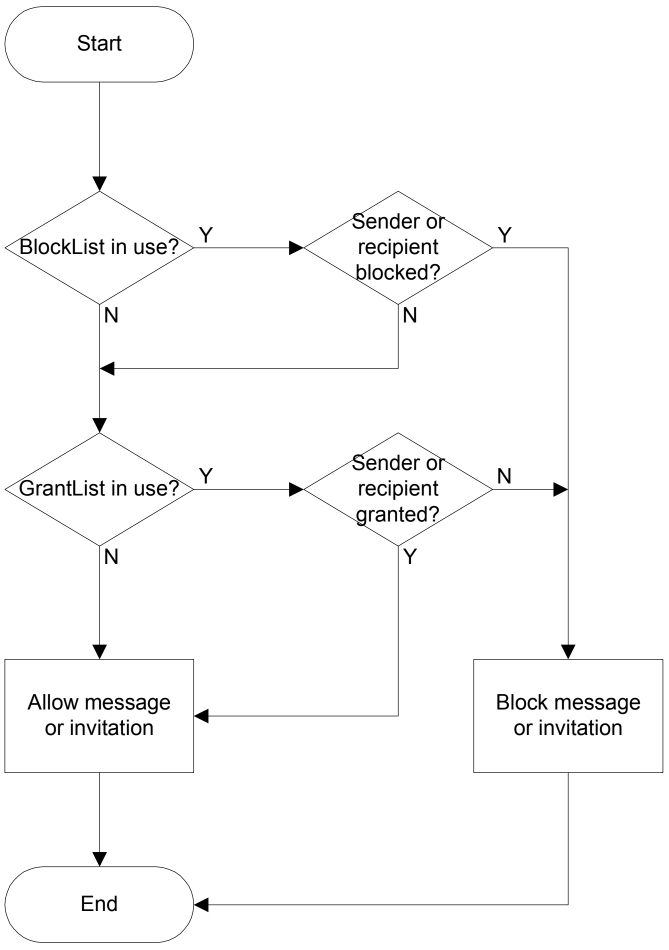
Start
BlockList in use?
Y
N
Sender or recipient blocked?
Y
N
GrantList in use?
Y
N
Sender or recipient granted?
N
Y
Allow message or invitation
Block message or invitation
End

Figure 60: Blocking decision-tree

9.3.1.1 Transactions

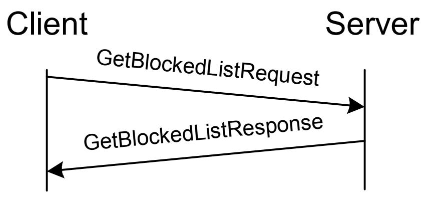

Figure 61: Get list of blocked entities transaction

A user MAY get his/her own list of blocked entities at any time. The client MUST send the GetBlockedListRequest primitive to the server. The server MUST respond with the GetBlockedListResponse primitive. If the user has block-list on the server, the response MUST contain Blocked-Entity-List, otherwise the response MUST NOT contain the list. If the user has grant-list on the server, the response from the server MUST contain Granted-Entity-List, otherwise the response MUST NOT contain the list. The Block-Entity-List and Grant-Entity-List MUST contain all the items the client(s) has added and not removed, including the exact list of wildcard character expressions, as defined by the user.

The client and the server MAY support the 'get list of blocked entities' transaction. The service tree leaf that allows negotiation of this transaction is 'GLBLU'.

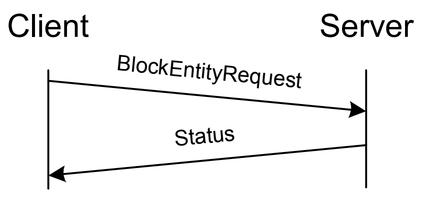

Figure 62: Block/unblock entities transactions

A user MAY block/un-block any other entity at any time. The client MUST send the BlockEntityRequest primitive to the server containing the list of entities to be blocked/unblocked and/or to be granted/un-granted. The server MUST respond with a Status primitive to the client.

The client and the server MAY support the 'block/unblock entities' transaction. The service tree leaf that allows negotiation of this transaction is 'BLENT'. The Block-Entity-List and Grant-Entity-List MUST be persistent even when the 'InUse' flags are turned off.

The server MAY allow wildcard characters to be used in the block/grant list entries. Wildcard characters are:

- question mark ('?') representing a single character and
- asterisk ('*') representing a sub-string.

The server MAY limit the use of wildcard characters. When a wildcard character expression is too complicated, the whole transaction MUST fail without adding any entry to the block/grant list, and the server MUST respond with error code 547. When the server does not allow the use of wildcard characters at all, the whole transaction MUST fail without adding any entry to the block/grant list, and the server MUST respond with error code 546. Client receiving error code 546:

- MAY repeat the request using full strings without wildcard character, and
- SHOULD remember that the server does not allow the use of wildcard characters, and
- SHOULD NOT request adding wildcard characters in the future.

The server MUST add those entities to the block-list that are specified in the Block-Entity-List element. The server MUST remove those entities from the block-list that are specified in the Unblock-Entity-List element.

The server MUST add those entities to the grant-list that are specified in the Grant-Entity-List element. The server MUST remove those entities from the grant-list that are specified in the Ungrant-Entity-List element.

The server MUST support turning on and off the use of Block-Entity-List and Grant-Entity-List.

Whenever the block list/grant list has been updated or any of the InUse flags have been altered, the server MUST send notification of the appropriate event to those other online clients of the user that are subscribed to such event using the General Notification Mechanism; see 7.2 General Notification Transactions.

Examples for wildcard character usage:

- *When the BlockList contains an Application-ID which is "*ApplicationBothersMe*", the server will block all of the following Application-IDs:*
  - *"thisapplicationbothersmev1.0",*
  - *"thisApplicationBothersMev2.0",*
  - *"thatApplicationBothersMe".*
- *When the BlockList contains an User-ID which is "wv:spammer*", the server will block all of the following User-IDs:*
  - *"wv:Spammer.Joe@foo.com",*
  - *"wv:spammerjoe@foo2.com",*
  - *"wv:SPAMMER".*
- *When the BlockList contains an ScreenName which is "Anonymous??" in group "wv:*chat*", the server will block all of the following ScreenNames:*

- *“Anonymous1“ in group “wv:mychatgroup@foo.com”,*
- *“anonymous99“ in group “wv:mychatgroup@foo.com”,*
- *“ANONYMOUS“ in group “wv:chat@foo.com”.*

9.3.1.2 Error conditions

Generic error conditions:

- Not logged in. (604)
- Service not agreed. (506)
- Service not supported. (405)

GetBlockedListRequest error conditions:

- None except the generic error conditions.

BlockEntityRequest error conditions:

- Unknown User-ID or ScreenName. (531)
- Wildcard characters not allowed (547)
- Wildcard expression is too complicated (548)
- Contact list does not exist. (700)
- The maximum number of users in grant list has been reached for the user. (756)
- The maximum number of users in block list has been reached for the user. (757)
- Unknown Group-ID. (800)

9.3.1.3 Primitives and information elements

Primitive	Direction
GetBlockedListRequest	Client → Server
GetBlockedListResponse	Client ← Server
BlockEntityRequest	Client → Server
Status	Client ← Server

Table 112: Primitive directions for block transactions

Information Element	Req	Type	Description
Message-Type	M	GetBlockedList Request	Message identifier.
Transaction-ID	M	String	Identifies the transaction requested.
Session-ID	M	String	Identifier for the session.

Table 113: Information elements in GetBlockedListRequest primitive

Information Element	Req	Type	Description
Message-Type	M	GetBlockedList Response	Message identifier.
Transaction-ID	M	String	Identifies the transaction requested.
Segment-Info	O	Structure	Contains segmentation information when the primitive is segmented.
Session-ID	M	String	Identifier for the session.

Information Element	Req	Type	Description
Blocked-Entity-List	C	Structure	The list of currently blocked entities.
Blocked-List-InUse	M	Boolean	Indicates if the list of blocked entities is currently in use (active).
Granted-Entity-List	C	Structure	The list of currently granted entities.
Granted-List-InUse	M	Boolean	Indicates if the list of granted entities is currently in use (active)

Table 114: Information elements in GetBlockedListResponse primitive

Information Element	Req	Type	Description
Message-Type	M	BlockEntity Request	Message identifier.
Transaction-ID	M	String	Identifies the transaction requested.
Segment-Info	O	Structure	Contains segmentation information when the primitive is segmented.
Session-ID	M	String	Identifier for the session.
Block-Entity-List	O	Structure	A list of entities to be added to or to be removed from the Blocked-Entity list.
Blocked-List-InUse	O	Boolean	Indicates if the list of blocked entities is currently in use (active).
Grant-Entity-List	O	Structure	A list of entities to be added to or to be removed from the Granted-Entity list.
Granted-List-InUse	O	Boolean	Indicates if the list of granted entities is currently in use (active)

Table 115: Information elements in BlockEntityRequest primitive

10.Group Feature

10.1 Group Models

The concept of *user group* means a discussion forum formed by two or more individuals (users) to exchange information, opinions, comments, thoughts about a particular issue, which is the *topic* of the particular group.

These user groups MAY be categorized by:

Ownership:

- Public (created and maintained by service provider),
- Private (created and maintained by a subscriber),

Membership:

- Open (any users MAY join the group),
- Restricted (only particular users MAY join the group).

The management of public groups is not in the scope of IMPS specifications.

In order to achieve minimum level of interoperability both the client and the server MUST support the following transactions:

- 10.4 Join Group Feature
- 10.5 Leave Group Feature

The rest of the group-related transactions are all OPTIONAL. The individual client or server implementations MAY decide whether support for a particular transaction is implemented or not.

The User group transactions are divided to two categories:

- more or less Static user group transactions,
- Dynamic user group transactions.

10.1.1.1 Static user group transactions

These transactions include user group management, and other transactions that are not used frequently:

- Creation, modification, and deletion of groups,
- Adding, removing group members,
- Setting access rights,
- Getting information about a group,
- Subscription to group change notification.

10.1.1.2 Dynamic user group transactions

These transactions include frequently used transactions:

- Joining, leaving a group,
- Inviting other users to a group,
- Rejecting users,
- Notification about group changes.

10.1.2 Private group model

A group MUST be considered as *private*, if an ordinary IM user has created it, and that user maintains it.

10.1.3 Public group model

A group MUST be considered as *public*, if the service provider has created it, and the service provider maintains it.

10.1.4 Access privileges

There are three levels of access privileges to the restricted groups:

- Administrator,
- Moderator,
- User.

Administrators MAY do anything in a group without restrictions.

The creator of the particular group MUST always have administrator privileges (administrator privileges MUST NOT be removed from the creator) as long as the group exists. A user MUST NOT be rejected in a group that belongs to him/her.

Moderators MAY add/remove/reject members who are ordinary users, but they MUST NOT add/remove/reject members who are moderators or administrators.

Users MUST NOT have any privileges other than to join/leave to/from the group, and to send/receive messages.

The following table describes the availability of transactions for each privilege level.

Y=available, N=not available, N/A=not applicable.

Name	Administrators	Moderators	Users
Send/receive messages	Y	Y	Y
Send/receive private messages	Y	Y	Y
Create group	N/A	N/A	N/A
Delete group	Y	N	N
Join/leave group	Y	Y	Y
Get/add/remove group members	Y	Y	N
Get group properties	Y	Y	Y
Set group properties	Y	N	N
Get/set own properties	Y	Y	Y
Get/modify reject list	Y	Y	N
Subscribe group change	Y	Y	Y
Group change notification	Y	Y	Y
Modify member access rights	Y	N	N
Get list of joined users	Y	Y	Y

Table 116: Availability of transactions for privilege levels

10.1.5 Multi-session and groups

Ownership of private groups, group access rights, group membership and rejection of users is all linked to a user and no particular client of the user. A server MUST send general notifications to all clients of the user which have subscribed previously to such notification type(s) if any of the above changes; see 7.2.

The act of joining a user to a group on particular client is done by a client and will only affect that particular client. Routing of group change notifications, forced leave from a group and group messages MUST be done towards the client the user joined the group on. The AutoJoin feature is also a setting for the user on a particular client and not the user in general.

10.1.6 Group properties

The values of the group properties MAY be defined by the owner, or by group member(s) with sufficient access rights. Only Administrators MAY modify these group property values. Each group MAY have the following properties:

- Name: a string that MAY be presented to the user as the name of the group (not necessarily same as Group-ID!).

  Default value MUST be an empty string. This is an OPTIONAL property (the client is NOT REQUIRED to specify it in the CreateGroupRequest primitive).

- Accesstype:
  - Open (for everyone) or
  - Restricted (members only).

  Default value MUST be 'open'. This is an OPTIONAL property (the client is NOT REQUIRED to specify it in the CreateGroupRequest primitive).

- Type[1]:
  - Public (maintained by service provider) or
  - Private (maintained by individual user(s)).

  For all user-created groups the server MUST set this property to 'private'.

- PrivateMessaging:
  - T (sending private messages is enabled) or
  - F (sending private messages is disabled).

  Default value MUST be "F". This is an OPTIONAL property (the client is NOT REQUIRED to specify it in the CreateGroupRequest primitive).

- Searchable:
  - T (the group MUST be subject to search) or
  - F (the group MUST NOT be included in searching).

  Default value MUST be "F". This is an OPTIONAL property (the client is NOT REQUIRED to specify it in the CreateGroupRequest primitive).

- Topic: a string that describes the subject of discussion in the group. The topic is subject to searching if allowed.

  Default value MUST be an empty string. This is an OPTIONAL property (the client is NOT REQUIRED to specify it in the CreateGroupRequest primitive).

- ActiveUsers[2]: an integer number that indicates the number of currently joined users.
- MaxActiveUsers: an integer number that indicates the maximum number of joined users at any given time.

  The Default value MUST be set by the GRSE. This is an OPTIONAL property (the client is NOT REQUIRED to specify it in the CreateGroupRequest primitive).

- WelcomeNote: a string that is presented as text to the user when he/she joins the group.

  Default value MUST be an empty structure. This is an OPTIONAL property (the client is NOT REQUIRED to specify it in the CreateGroupRequest primitive).

[1] This property is read-only (determined by the server) and it cannot be modified.

[2] This property is read-only (monitored by the server) and it cannot be modified.

- History:
    - T (message history is supported)
    - F (message history is not supported)

  If server supports the message history functionality, user/client MAY request it for a new or existing group.

- AutoDelete:
    - T (the group will be automatically deleted) - the server MUST verify the 'Validity' property. If the 'Validity' property was zero (e.g. the validity has expired), the server MUST automatically delete the group when all joined users have left. If the 'Validity' property was non-zero (e.g. the group is valid) the server MUST NOT delete the group.
    - F (group will not be automatically deleted) – the server MUST ignore the 'Validity' property. In this case, the group is considered permanent. The "permanency" of a group is subject to the local policy of maximum lifetime defined by server vendor or operator.

  Default value MUST be 'F'. This is an OPTIONAL property (the client is NOT REQUIRED to specify it in the CreateGroupRequest primitive).

- Validity: a non-negative integer number that indicates the time period (in minutes) for which the group is valid. When the value is zero the group MUST NOT be valid if and only if all joined user have left the group. The server MUST keep the value up-to-date and the value MUST reflect the remaining time period for which the group is valid. The 'Validity' MUST be ignored if 'AutoDelete' is 'F'. Note that the generic XML element of the value of the group property is defined as a String. For this particular 'Validity' property, the String MUST be the decimal representation of the non-negative integer number.

  Default value MUST be zero. This is an OPTIONAL property (the client is NOT REQUIRED to specify it in the CreateGroupRequest primitive).

- MinimumAge: an integer number that indicates the minimum age for joining/searching the group. The value of zero indicates that any user MAY join the group. A value that is higher than zero indicates that the group is restricted by age. If the group is restricted by age, special handling applies when the user is searching for the group, or attempts to join the group. These restrictions are described in 7.4 General Search Transactions and 10.4 Join Group Feature.

  Default value MUST be "0". This is an OPTIONAL property (the client is NOT REQUIRED to specify it in the CreateGroupRequest primitive).

- RequireInvitation[3]:
    - ·T Participants MUST be invited in order to be able to join the group. The server MUST allow users that unintentionally left the *Private Group Conversation* to rejoin it for a limited timeout period. After the expiration of the timeout period the server MUST NOT allow the non-intentionally (i.e. without sending a LeaveGroupRequest primitive to the server) dropped-out user to rejoin the on-going *Private Group Conversation* unless he/she is re-invited by one of the participants. The timeout period is a server-specific value. Server MUST NOT allow the users who have intentionally left the Group Conversation to re-join unless they are invited again by one of the participants.
    - ·F Participants MAY join the group without explicitly being invited.

Each user MAY have his/her own properties for each group individually. These properties are:

- PrivateMessaging:
    - T (sending private messages is enabled) or
    - F (sending private messages is disabled).

  Default value MUST be "F". This is an OPTIONAL property (the client does not have to specify it in the CreateGroupRequest primitive).

[3] This property is read-only (determined by the server) and it cannot be modified.

- IsMember[4]:
  - T (the user is a member of the group) or
  - F (the user is not member of the group).
- PrivilegeLevel[5]:
  - User (general user),
  - Mod (moderator),
  - Admin (administrator).
- AutoJoin
  - T (server joins the client automatically to the particular group)
  - F (server does not join client automatically to the particular group)

  Default value is “F” for every user in every group. This is an OPTIONAL property (the client is NOT REQUIRED to specify it in the CreateGroupRequest primitive).

- ShowID
  - T (server MAY disclosed the User-ID to other users joined to this group)
  - F (server MUST NOT disclose the User-ID to other users joined to this group)

  Default value is “F” for every user in every group. This is an OPTIONAL property (the client is NOT REQUIRED to specify it in the CreateGroupRequest primitive).

  Note that moderators and administrators MAY retrieve the User-IDs disregarding this setting. See ‘Get Joined Users’ transaction in chapter 10.6.

The XML Syntax DTD - see [CSP XMLS] - has been defined in a manner that allows custom group/own properties. The client and the server SHOULD ignore (without generating an error) the properties they are not able to process (not understood).

Both group members and joined users MAY have own group properties. Properties of non-members MAY be discarded after the user leaves the group whereas members’ properties MUST be kept on the server between separate group sessions.

10.2 Create Group Feature

10.2.1 Transactions

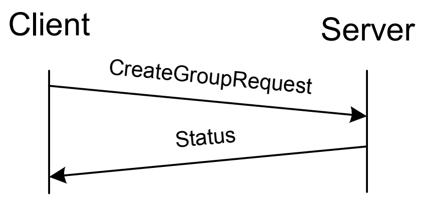

Figure 63: Create group transaction

A user MAY create a private user group at any time. The client MUST send the CreateGroupRequest primitive, which MUST contain the name (ID), the initial properties of the group (both own group properties and general group properties), a Boolean flag indicating whether the joining the group is requested a Boolean flag indicating whether subscribing to group change notification for the group is requested. The request MAY also contain the screen name. The server MUST create the group with the specified properties, and respond with a Status primitive.

[4] This property is read-only (determined by the server) and it cannot be modified.

[5] This property is read-only (determined by the server) and it cannot be modified.

The create group transaction MAY be supported by the client and the server. The service tree leaf that allows negotiation of this transaction is ‘CREAG’.

The server MUST grant the requesting user (owner) Administrator privileges for the newly created group.

If the client requested to join the group after creation – the value of the Join-Group flag is ‘T’ (true) – the server MUST join the user to the newly created group. If the flag indicates ‘F’ (false), the server MUST NOT join the user to the group.

If the client requested subscribing to group change notification – the value of the Subscribe-Notif flag is ‘T’ (true) – the server MUST subscribe the user to receive group change notifications from the newly created group. If the flag indicates ‘F’ (false), the server MUST NOT subscribe the user to receive group change notifications. The group change notifications MUST NOT be sent before the user has joined the group.

Whenever a new group has been created for the user, the server MUST send notification of the event to those other online clients of the user that are subscribed to such event using the General Notification Mechanism; see 7.2 General Notification Transactions.

10.2.2 Error Conditions

Generic error conditions:

- Not logged in. (604)
- Service not agreed. (506)
- Service not supported. (405)

CreateGroupRequest error conditions:

- Group already exists. (801)
- Invalid group attribute(s). (806)
- The maximum number of groups has been reached (user-limit). (814)
- The maximum number of groups has been reached (server-limit). (815)
- Cannot have searchable group without name or topic. (822)
- Service provider agreement missing (907)

10.2.3 Primitives and information elements

Primitive	Direction
CreateGroupRequest	Client → Server
Status	Client ← Server

Table 117: Primitive directions in create group transaction

Information Element	Req	Type	Description
Message-Type	M	CreateGroup Request	Message identifier.
Transaction-ID	M	String	Identifies the transaction requested.
Session-ID	M	String	Identifier for the session.
Group-ID	M	String	Identifies the group to be created.
Group-Props	M	Structure	The properties of the group.
Own-Props	M	Structure	The list of the users’ own group properties with the corresponding values.
Join-Group	M	Boolean	A flag indicating that the user creating the group joins the group at the same time.
Screen-Name	O	Structure	Screen name of the user in the group.

Subscribe-Notif	M	Boolean	A flag indicating that the user wants to activate the group change notifications while joining the group.

Table 118: Information elements in CreateGroupRequest primitive

10.3 Delete Group Feature

10.3.1 Transactions

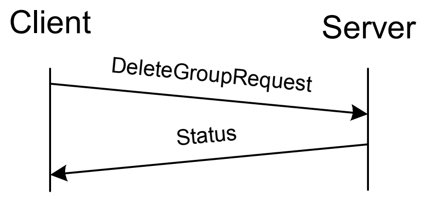

Figure 64: Delete group transaction

A user with sufficient access rights (see Table 116: Availability of transactions for privilege levels) MAY delete a private user group at any time. The client MUST send the DeleteGroupRequest primitive, which MUST contain the name (ID) of the group. The server SHOULD remove all currently joined users from the group (LeaveGroupResponse message), delete the specified group, and respond with a Status primitive.

The client and the server MAY support the delete group transaction. The service tree leaf that allows negotiation of this transaction is ‘DELGR’.

The client MUST refer to a private Group-ID in the request. The server MUST reject deletion requests that refer to public groups, or if the requesting user does not have Administrator privileges in the group.

Upon success the server MUST delete the requested group.

Whenever an existing group has been deleted from the user, the server MUST send notification of the event to those other online clients of the user that are subscribed to such event using the General Notification Mechanism; see 7.2 General Notification Transactions.

Whenever a group has been deleted for which the user is a member of, the server MUST send notification of the event to those other online clients of the user that are subscribed to such event using the General Notification Mechanism; see 7.2 General Notification Transactions.

10.3.2 Error Conditions

Generic error conditions:

- Not logged in. (604)
- Service not agreed. (506)
- Service not supported. (405)

DeleteGroupRequest error conditions:

- Group does not exist. (800)
- Group is public. (804)
- Insufficient user rights. (816)

10.3.3 Primitives and information elements

Primitive	Direction
DeleteGroupRequest	Client → Server
Status	Client ← Server

Table 119: Primitive directions in delete group transaction

Information Element	Req	Type	Description
Message-Type	M	DeleteGroup Request	Message identifier.
Transaction-ID	M	String	Identifies the transaction requested.
Session-ID	M	String	Identifier for the session.
Group-ID	M	String	Identifies the group.

Table 120: Information elements in DeleteGroupRequest primitive

10.4 Join Group Feature

A user MAY join a discussion group at any time. The user can join a group by sending the JoinGroupRequest (for details see next chapter) or by using the Auto-Join feature. If the support for Group Service is agreed upon and the Auto-Join feature is enabled – for one group at least – then the Auto-Join feature MUST take place after the service negotiation. The Auto-Join feature MUST NOT take place if the server and the client do not agree to support Group Service. If any error occurs during Auto-Join (group is not available, group is closed, not a member, etc), the IMPS server MUST NOT join the client to the group and it MUST NOT send any error messages to the client.

Upon auto-joining a user to a group, the IMPS server MUST try to join the user with a Screen Name equal to what the user had the last time he/she was joined to the group. If the Screen Name is not unique the server MUST create a unique Screen name. The IMPS server MUST also subscribe the user to notifications if the Subscribe Notification feature was in effect the last time the user was joined to the group. To allow for a client to be notified about getting auto joined to a group, the IMPS server MUST send a GroupChangeNotice to the client listing the the user with the chosen Screen Name in the Joined-Users-List, regardless of the Subscribe Notification feature setting. The initial GroupChangeNotice MUST only be sent if the client has agreed support for subscription on group changes (The service tree leaf that allows negotiation of this transaction is 'SUBGCN').

Since some users MAY be rejected and some groups MAY be restricted; a decision tree is provided to describe the behavior:

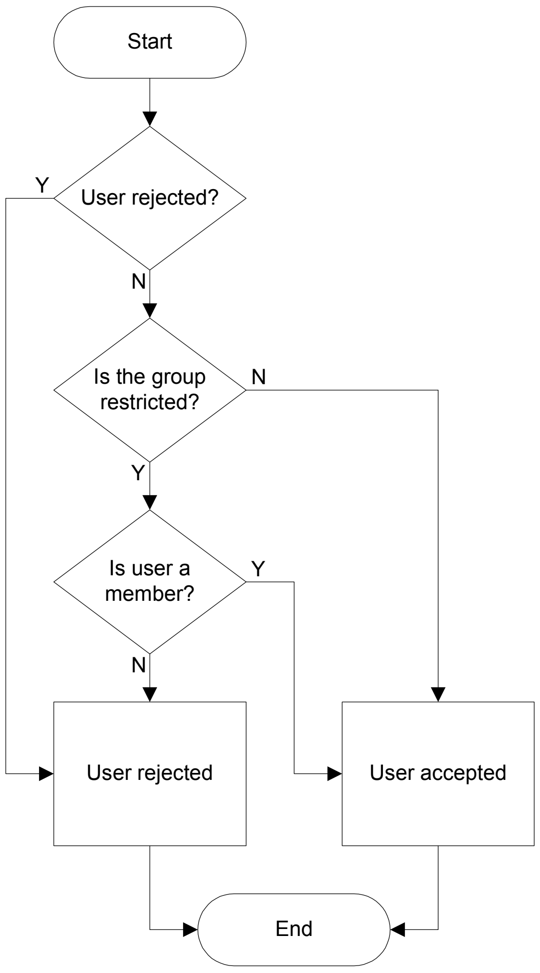

Figure 65: Joining decision-tree

10.4.1 Transactions

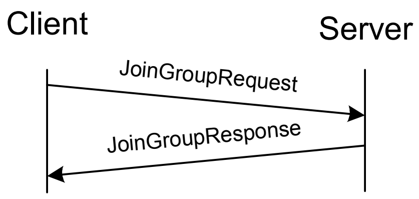

Figure 66: Join group transaction

To join a group, the client MUST send the JoinGroupRequest primitive to the server containing the ID of the group, his/her screen name shown during the discussion, the joined users' list request and the subscribe to group change notification request. The server MUST respond with the JoinGroupResponse primitive that MUST contain the list of joined users identified by screen names and User-IDs if requested, and MAY contain the Welcome–Note element. If there is an error, the server MUST respond with a Status primitive instead of the expected JoinGroupResponse primitive. Those User-IDs MUST be present only which users have set their ShowID own property to true.

If the Screen Name defined in the JoinGroupRequest is not unique, the server MUST create a unique Screen Name and inform the user about the changed Screen Name by including it in the JoinGroupResponse.

After the user successfully joins the group, the user MAY receive and send messages from/to the particular group.

To retrieve previous messages (history) from the group, the get message list transaction MAY be utilized.

The client and the server MUST support the join group transaction, thus its support is not negotiated.

In order to save extra transactions:

- the client MAY request the joined users' list to be returned in the response. If the Joined-Request element is 'T' (true) in the request, the server MUST include the joined users' list in the response. For more information about the joined users' list see the particulars of the 'Get joined users transaction' in 10.6.1 Transactions.
- the client MAY request subscribing to group change notifications. If the Subscribe-Notif element is 'T' (true) in the request and the group change notification transaction was agreed during service negotiation, the server MUST subscribe the user to group change notification.
- the client MAY include own group properties for the user in the request. If the own group properties are included in the request, the server MUST apply those properties.

If the group's MinimumAge property is higher than zero, the server MUST verify this value versus the joining user's age in his/her public profile, and:

- when the age field is missing from the public profile, the server MUST allow the user to join – providing that the rest of the conditions are met.
- when the age field is present in the public profile, and:
  - this age is smaller than the MinimumAge value, the server MUST NOT allow the user to join the group.
  - this age is equal or higher than the MinimumAge value, the server MUST allow the user to join the group– providing that the rest of the conditions are met.

The server MUST NOT allow rejected users to join the group from which they are expelled.

The server MUST allow any user – except the rejected users – to join an 'Open' group.

The server MUST allow only members to join a 'Restricted' group.

If the ScreenName in the JoinGroupRequest primitive is unique in the scope of the group, the server MUST apply the ScreenName for the user. If the ScreenName in the JoinGroupRequest primitive is not unique in the scope of the group, the server MUST create a ScreenName for the user and provide the generated ScreenName in the JoinGroupResponse primitive. The client MUST apply the returned ScreenName. The algorithm to generate a new screen name is an implementation issue, however the newly generated screen name MUST be unique within the group.

If a 'Welcome Note' was specified for the group, the server SHOULD include it in the response.

After the user has successfully joined the group, the server MUST start sending all messages that are sent to the group to the newly joined user. A user MAY have separate sessions at a time, thus the server MUST send the IMs from the group to those clients of the joined user only from which he/she has joined the group.

10.4.2 Error conditions

Generic error conditions:

- Not logged in. (604)
- Service not agreed. (506)
- Service not supported. (405)
- Invalid/unsupported group properties (806)

JoinGroupRequest error conditions:

- Group does not exist. (800)
- User already joined. (807)
- User has been rejected. (809)
- Not a member of the restricted group. (810)
- The maximum number of allowed users has been reached. (817)

- Minimum age requirement not fulfilled. (818)
- Service provider agreement missing. (907)

10.4.3 Primitives and information elements

Primitive	Direction
JoinGroupRequest	Client → Server
JoinGroupResponse	Client ← Server

Table 121: Primitive directions in join group transaction

Information Element	Req	Type	Description
Message-Type	M	JoinGroupRequest	Message identifier.
Transaction-ID	M	String	Identifies the transaction requested.
Session-ID	M	String	Identifier for the session.
Group-ID	M	String	Identifies the group in the transaction.
Screen-Name	M	Structure	Screen name of the user in the group.
Joined-Request	M	Boolean	Indicates if the user wants the list of currently joined users.
Subscribe-Notif	M	Boolean	A flag indicating that the user wants to activate the group change notifications while joining the group.
Own-Props	O	Structure	The list of the users' own group properties with the corresponding values.

Table 122: Information elements in JoinGroupRequest primitive

Information Element	Req	Type	Description
Message-Type	M	JoinGroupResponse	Message identifier.
Transaction-ID	M	String	Identifies the transaction requested.
Segment-Info	O	Structure	Contains segmentation information when the primitive is segmented.
Session-ID	M	String	Identifier for the session.
Joined-Users-List	C	Structure	The list of the currently joined users (screen name, User-ID). Present if it was requested.
Screen-Name	C	String	A unique screen name created by the server. Present if the Screen Name sent in the JoinGroupRequest was not unique.
Welcome-Text	O	Structure	A short text to be shown to the user when he/she has joined the group. The structure of the Welcome-Text includes {Content-type, OPTIONAL Content-encoding, and Content-Data}.
Joined-Blocked-Users-List	O	Structure	The list of the users who are currently joined in the group and who are blocked by the requesting user (Screen Name, User-ID).

Table 123: Information elements in JoinGroupResponse primitive

10.5 Leave Group Feature

10.5.1 Transactions

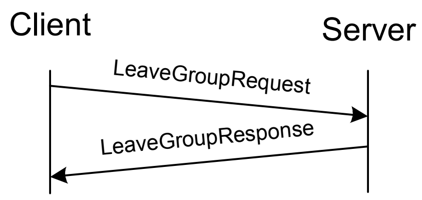

Figure 67: User initiated leave group transaction

A user MAY leave the joined discussion group at any time. The client MUST send the LeaveGroupRequest primitive to the server containing the ID of the group. The server MUST respond with a LeaveGroupResponse primitive containing the reason code, which is own request is this case. If there is an error, the server MUST respond with a Status primitive instead of the expected LeaveGroupResponse message.

The server MUST NOT include Group-ID in the response.

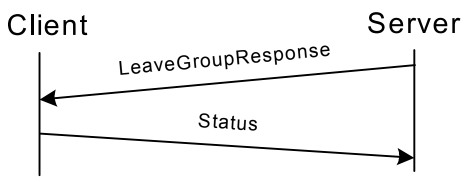

Figure 68: Server initiated leave group transaction

The server MAY initiate the group leaving also (user kicked out of the group, group deleted, etc.). In this case the server MUST send the LeaveGroupResponse primitive to the client containing the Group-ID and reason code.

After the user has left the group, the user MUST NOT receive/send messages from/to the particular group.

There is no difference between user and server-initiated leave group transactions from support point of view. The client and the server MUST support the leave group transaction, thus its support is not negotiated.

The client MAY receive LeaveGroupResponse primitive without prior request. The server MUST include the Group-ID and reason code in such primitives. The client MUST remove the user from the group – within the device – and reply with a Status primitive.

If the user was forced to leave the group because the group has been deleted or the user is a member of the deleted group, the server MUST send notification of the event to those online client(s) of the users that are subscribed to such event using the General Notification Mechanism; see 7.2 General Notification Transactions.

10.5.2 Error conditions

Generic error conditions:

- Not logged in. (604)
- Service not agreed. (506)
- Service not supported. (405)

LeaveGroupRequest error conditions:

- Group was not joined before transaction. (808)

LeaveGroupResponse error conditions:

- Client MAY ignore any error and respond with Successful. (200)

- Group does not exist. (800)
- Own request (824)

10.5.3 Primitives and information elements

Primitive	Direction
LeaveGroupRequest	Client → Server
LeaveGroupResponse	Client ← Server

Table 124: Primitive directions in leave group transaction

Information Element	Req	Type	Description
Message-Type	M	LeaveGroup Request	Message identifier.
Transaction-ID	M	String	Identifies the transaction requested.
Session-ID	M	String	Identifier for the session.
Group-ID	M	String	Identifies the requested content.

Table 125: Information elements in LeaveGroupRequest primitive

Information Element	Req	Type	Description
Message-Type	M	LeaveGroup Response	Message identifier.
Transaction-ID	M	String	Identifies the transaction requested.
Session-ID	M	String	Identifier for the session.
Result	M	Structure	Result that indicates why did the user leave the group. (Own request, rejected, etc.)
Group-ID	C	String	Identification of the- group that has been left. Not present if the client initiated the transaction.

Table 126: Information elements in LeaveGroupResponse primitive

10.6 Members' List Management

10.6.1 Transactions

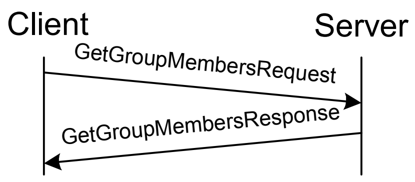

Figure 69: Get group members transaction

A user with sufficient access rights (see Table 116: Availability of transactions for privilege levels) MAY retrieve the member list of a group. The client MUST send the GetGroupMembersRequest primitive to the server containing the ID of the group. The server MUST responds with the GetGroupMembersResponse primitive containing the list of all group members. If there is an error, the server MUST respond with a Status primitive instead of the expected GetGroupMembersResponse primitive.

The client and the server MAY support the 'get group members' transaction. The service tree leaf that allows negotiation of this transaction is 'GETGM'.

The server MUST allow retrieval of the members' list only to those users who have Moderator or Administrator privileges in the particular group. If this is the case, the response MUST contain the members' list.

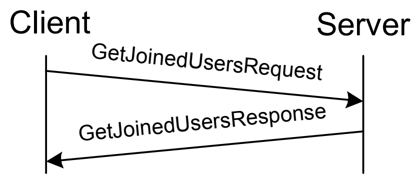

Figure 70: Get joined users transaction

A user MAY retrieve the list of users that are currently joined to the group. The client MUST send the GetJoinedUsersRequest primitive to the server containing the ID of the group. The server MUST respond with the GetJoinedUsersResponse primitive. If there is some error, the server MUST respond with a Status primitive instead of the expected GetJoinedUsersResponse primitive.

The client and the server MAY support the 'get joined users transaction'. The service tree leaf that allows negotiation of this transaction is 'GETJU'.

The server MUST reject the request if the requesting user is not joined to the group.

For those users who have joined the group and have their ShowID property set to 'T', the server MUST deliver their User-ID and screen name in the Joined-User-List element.

For those users who have joined the group and have their ShowID property set to 'F' the server MUST deliver their screen names and MUST NOT deliver the User-IDs in the Joined-User-List element.

If the requesting user has Administrator or Moderator privileges:

- The server MUST include the Admin-Map-List element in the response.
- The server MUST disregard the ShowID own group property of the users and send the User-IDs.

If the requesting user does not have Administrator or Moderator privileges:

- The server MUST include User-Map-List element in the response.
- The server MUST obey the ShowID own group property setting and include those User-IDs only that have the ShowID own group property turned on (True='T').

The server MUST also include the list of those joined users who have been blocked by the requesting user – in which case the server MUST regard the 'Show-ID' property of the blocked users as described above. The server MUST include users in the Joined-Blocked-Users-List or also in the Joined-Users-List element.

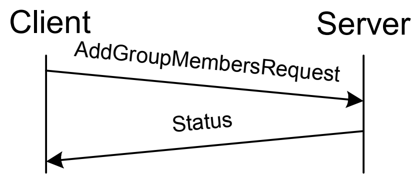

Figure 71: Add group members transaction

A user with sufficient access rights (see Table 116: Availability of transactions for privilege levels) MAY add user(s) to the member list of a group. The client MUST send the AddGroupMembersRequest primitive to the server containing the ID of the group, and the list(s) of users to be added. The server MUST respond with the Status primitive.

All of the newly added users MUST have the lowest privilege level: User.

The client and the server MAY support the 'add group members transaction'. The service tree leaf that allows negotiation of this transaction is 'ADDGM'.

The server MUST allow adding users to the members' list only to those users who have Moderator or Administrator privileges in the particular group. If this is the case, the specified users MUST be added to the members' list.

The server MUST ignore those users that are in the members' list already without generating an error.

Whenever group members have been added to a group, the server MUST send notification of the event to those other online client(s) of all Administrators and Moderators of the administered group that are subscribed to such event using the General Notification Mechanism; see 7.2 General Notification Transactions.

Whenever user(s) has been added to the members' list of a group, the server MUST send notification(s) of the event to those online client(s) of the newly added user(s) that are subscribed to such event using the General Notification Mechanism; see 7.2 General Notification Transactions.

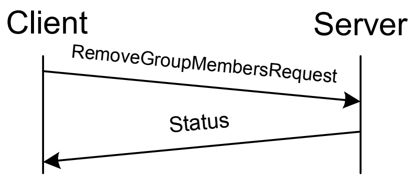

Figure 72: Remove group members transaction

A user with sufficient access rights (see Table 116: Availability of transactions for privilege levels) MAY remove user(s) from the member list of a group. The client MUST send the RemoveGroupMembersRequest primitive to the server containing the ID of the group, and the list of users to be removed. The server MUST respond with the Status primitive.

The client and the server MAY support the 'remove group members transaction'. The service tree leaf that allows negotiation of this transaction is 'RVMGM'.

The server MUST allow removing users from the members' list only to those users who have Moderator or Administrator privileges in the particular group. If this is the case, the specified users MUST be removed from the members' list.

The server MUST ignore the removal of those users that are not in the members' list without generating an error.

Whenever group members have been removed from a group, the server MUST send notification of the event to those other online client(s) of all Administrators and Moderators of the administered group that are subscribed to such event using the General Notification Mechanism; see 7.2 General Notification Transactions.

Whenever user(s) has been removed from the members' list of a group, the server MUST send notification(s) of the event to those online client(s) of the removed user(s) that are subscribed to such event using the General Notification Mechanism; see 7.2 General Notification Transactions.

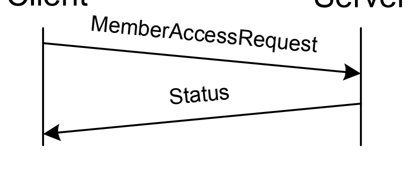

Figure 73: Member access rights transaction

A user with sufficient access rights (see Table 116: Availability of transactions for privilege levels) MAY change the access privileges of user(s). The client MUST send the MemberAccessRequest primitive to the server containing the ID of the group, and MAY contain the list of users to be set as administrator, moderator or ordinary user. The server MUST respond with a Status primitive.

Note for clarification: Being a group member does not have anything to do with being joined. Only members MUST be allowed to join a restricted group.

The client and the server MAY support the 'member access rights transaction'. The service tree leaf that allows negotiation of this transaction is 'MBRAC'.

The client and the server MUST identify the users using User-IDs.

The server MUST allow changing privileges only to those users who have Administrator privileges in the particular group. If this is the case, the access rights for the specified users MUST be updated exactly as requested.

The creator (owner) of the group MUST always have Administrator access rights to the group, therefore the server MUST NOT modify his/her access right.

Whenever the access rights of a member have been changed in a group, the server MUST send notification of the event to those other online clients of all Administrators of the administered group that are subscribed to such event using the General Notification Mechanism; see 7.2 General Notification Transactions.

10.6.2 Error Conditions

Generic error conditions:
- Not logged in. (604)
- Service not agreed. (506)
- Service not supported. (405)
- Service provider agreement missing (907)

GetGroupMembersRequest error conditions:
- Group does not exist. (800)
- Insufficient user rights. (816)

GetJoinedUsersRequest error conditions:
- Group does not exist. (800)
- Group was not joined before transaction. (808)

AddGroupMembersRequest error conditions:
- Group does not exist. (800)
- Insufficient user rights. (816)
- Unknown user. (531)
- The maximum number of group members has been reached (823)

RemoveGroupMembersRequest error conditions:
- Group does not exist. (800)
- Insufficient user rights. (816)
- Unknown user. (531)

MemberAccessRequest error conditions:
- Group does not exist. (800)
- Insufficient user rights. (816)
- Unknown user. (531)
- Not a group member (810)

10.6.3 Primitives and information elements

Primitive	Direction
GetGroupMembersRequest	Client → Server
GetGroupMembersResponse	Client ← Server
GetJoinedUsersRequest	Client → Server
GetJoinedUsersResponse	Client ← Server
AddGroupMembersRequest	Client → Server
Status	Client ← Server
RemoveGroupMembersRequest	Client → Server
Status	Client ← Server
MemberAccessRequest	Client → Server
Status	Client ← Server

Table 127: Primitive directions in add/remove user(s) to/from group transaction

Information Element	Req	Type	Description
Message-Type	M	GetGroupMembers Request	Message identifier.
Transaction-ID	M	String	Identifies the transaction requested.
Session-ID	M	String	Identifier for the session.
Group-ID	M	String	Identifies the group.

Table 128: Information elements in GetGroupMembersRequest primitive

Information Element	Req	Type	Description
Message-Type	M	GetGroupMembers Response	Message identifier.
Transaction-ID	M	String	Identifies the transaction requested.
Segment-Info	O	Structure	Contains segmentation information when the primitive is segmented.
Session-ID	M	String	Identifier for the session.
User-List-Adm	O	Structure	The list of users that are Administrators.
User-List-Mod	O	Structure	The list of users that are Moderators.
User-List	O	Structure	The list of users that are ordinary Users.

Table 129: Information elements in GetGroupMembersResponse primitive

Information Element	Req	Type	Description
Message-Type	M	GetJoinedUsers Request	Message identifier.
Transaction-ID	M	String	Identifies the transaction requested.
Session-ID	M	String	Identifier for the session.
Group-ID	M	String	Identifies the group.

Table 130: Information elements in GetJoinedUsersRequest primitive

Information Element	Req	Type	Description
Message-Type	M	GetJoinedUsers Response	Message identifier.
Transaction-ID	M	String	Identifies the transaction requested.
Segment-Info	O	Structure	Contains segmentation information when the primitive is segmented.
Session-ID	M	String	Identifier for the session.
Admin-Map-List	C	Structure	Administrators and moderators receive this list.
User-Map-List	C	Structure	Ordinary users receive this list.
Joined-Blocked-Users-List	O	Structure	The list of the users who are currently joined in the group and who are blocked by the requesting user (Screen Name, User-ID).

Table 131: Information elements in GetJoinedUsersResponse primitive

Information Element	Req	Type	Description
Message-Type	M	AddGroupMembers Request	Message identifier.
Transaction-ID	M	String	Identifies the transaction requested.
Segment-Info	O	Structure	Contains segmentation information when the primitive is segmented.
Session-ID	M	String	Identifier for the session.
Group-ID	M	String	Identification of the group.
User-List	M	Structure	The list of users to be added to the members' list.

Table 132: Information elements in AddGroupMembersRequest primitive

Information Element	Req	Type	Description
Message-Type	M	RemoveGroup MembersRequest	Message identifier.
Transaction-ID	M	String	Identifies the transaction requested.
Segment-Info	O	Structure	Contains segmentation information when the primitive is segmented.
Session-ID	M	String	Identifier for the session.
Group-ID	M	String	Identification of the group.
User-List	M	Structure	The list of members to be removed from the group specified by User-ID.

Table 133: Information elements in RemoveGroupMembersRequest primitive

Information Element	Req	Type	Description
Message-Type	M	MemberAccess Request	Message identifier.
Transaction-ID	M	String	Identifies the transaction requested.
Segment-Info	O	Structure	Contains segmentation information when the primitive is segmented.
Session-ID	M	String	Identifier for the session.
Group-ID	M	Group-ID	Identifies the group.

User-List-Admin	O	Structure	The list of users that are requested to be set as administrators.
User-List-Mod	O	Structure	The list of users that are requested to be set as moderators.
User-List	O	Structure	The list of users that are requested to be set as users.

Table 134: Information elements in MemberAccessRequest primitive

10.7 Modify Group Properties

10.7.1 Transactions

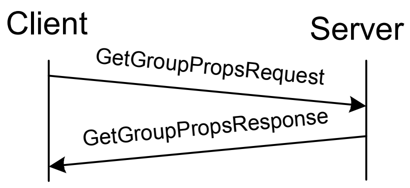

Figure 74: Get group properties transaction

A user MAY retrieve the properties of a group, as well as his/her own group properties. The server MAY choose to support the get group properties transaction and when it does, it MUST provide this functionality in restricted groups for all of the members, and in open groups for all users. The client MUST send the GetGroupPropsRequest primitive to the server containing the ID of the group. The server MUST respond with the GetGroupPropsResponse primitive, which MUST contain the properties of the specified group and MAY contain own properties of the user for the specified group. If there is an error, the server MUST responds with a Status primitive instead of the expected GetGroupPropsResponse primitive.

The client and the server MAY support the 'get group properties transaction'. The service tree leaf that allows negotiation of this transaction is 'GETGP'.

It is possible to extend the group properties with custom properties, however if the client or the server receives a group property that is not supported, it MUST ignore the particular group property without generating an error.

Upon success the server MUST send back the properties of the requested group to the client. If the user has own properties for the requested group, the server MUST include the own properties in the response.

The server MUST reject the transaction when the requested group is 'Restricted' and the requesting user is not in the members' list.

If the 'AutoDelete' property of the requested is 'T' (true), the server MUST re-calculate the 'Validity' property so that it contains the remaining time for which the group is valid.

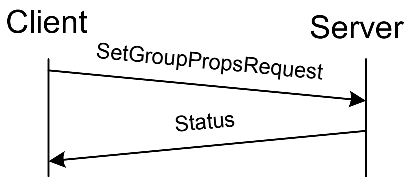

Figure 75: Set group properties transaction

A user with sufficient access rights (see Table 116: Availability of transactions for privilege levels) MAY update the properties of a group, or his/her own properties for a particular group. The server MAY choose to support the set group properties transaction and when it does, it MUST provide this functionality in restricted groups for all of the members, and in open groups for all users. The client MUST sends the SetGroupPropsRequest primitive to the server containing the ID and the new properties of the group and/or the new user properties. The server MUST respond with a Status primitive.

The client and the server MAY support the ‘set group properties transaction’. The service tree leaf that allows negotiation of this transaction is ‘SETGP’.

If the group is ‘Restricted’ and the requesting user is not in the members’ list, the server MUST reject the transaction.

Before updating the group properties the server MUST verify that the requesting user has Administrator privileges, and it MUST reject the request when the user does not have Administrator privileges. If the requesting user has Administrator privileges, the server MUST update the group properties.

The server MUST always update the own properties of the user on the server – except when the requesting user is not member f the ‘Restricted’ group.

The server MUST NOT update the read-only properties. See group properties in 10.1.6 Group properties.

The server MUST maintain the own group properties for members of an open or restricted group between group chat sessions – e.g. in an open or restricted group the own group properties of members MUST be persistent. The server MAY also maintain the own group properties of the users for open groups – e.g. the own group properties of a user in an open group MAY be persistent.

10.7.2 Error Conditions

Generic error conditions:

- Not logged in. (604)
- Service not agreed. (506)
- Service not supported. (405)

GetGroupPropsRequest error conditions:

- Group does not exist. (800)
- Insufficient user rights. (816)

SetGroupPropsRequest error conditions:

- Group does not exist. (800)
- Insufficient user rights. (816)
- Invalid group attribute(s). (806)
- Cannot have searchable group without name or topic. (822)
- Service provider agreement missing (907)

10.7.3 Primitives and information elements

Primitive	Direction
GetGroupPropsRequest	Client → Server
GetGroupPropsResponse	Client ← Server
SetGroupPropsRequest	Client → Server
Status	Client ← Server

Table 135: Primitive directions in modify group properties transaction

Information Element	Req	Type	Description
Message-Type	M	GetGroupProps Request	Message identifier.
Transaction-ID	M	String	Identifies the transaction requested.
Session-ID	M	String	Identifier for the session.
Group-ID	M	String	Identification of the group.

Table 136: Information elements in GetGroupPropsRequest primitive

Information Element	Req	Type	Description
Message-Type	M	GetGroupProps Response	Message identifier.
Transaction-ID	M	String	Identifies the transaction requested.
Session-ID	M	String	Identifier for the session.
Group-Props	M	Structure	The list of group properties with the corresponding values.
Own-Props	M	Structure	The list of the users' own group properties with the corresponding values.

Table 137: Information elements in GetGroupPropsResponse primitive

Information Element	Req	Type	Description
Message-Type	M	SetGroupProps Request	Message identifier.
Transaction-ID	M	String	Identifies the transaction requested.
Session-ID	M	String	Identifier for the session.
Group-ID	M	String	Identification of the group.
Group-Props	O	Structure	The new list of group properties with the corresponding values.
Own-Props	O	Structure	The list of the users' own group properties with the corresponding values.

Table 138: Information elements in SetGroupPropsRequest primitive

10.8 Rejecting User(s) from Group Feature

The concept of *rejecting* means that the server

- MUST force out users that are already joined in those groups from which they are being rejected, and
- MUST NOT allow rejected users to join any of the groups from which they have been rejected.

Rejecting is active for the rejected user(s) until another user with sufficient privileges removes him/her from the rejected users' list.

10.8.1 Transactions

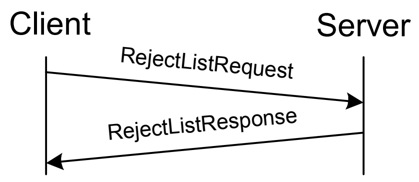

Figure 76: Reject user(s) from group transaction

A user with sufficient access rights (see Table 116: Availability of transactions for privilege levels) MAY retrieve/update the reject list of a group. The client MUST send the RejectListRequest primitive to the server, which MUST contain the ID of the group, and MAY contain the users to be added/removed to/from the reject list. The server MUST respond with the RejectListResponse primitive, which MUST contain the list of users that are rejected. If there is an error, the server MUST respond with a Status primitive instead of the expected RejectListResponse primitive.

The server MUST NOT allow users on the reject list to join the group.

Users in the reject list MUST be specified by their User-IDs.

The client MUST NOT include the same User-ID in the Add-Users-List and Remove-Users-List elements within the same transaction.

The client and the server MAY support the ‘reject user(s) from group transaction’. The service tree leaf that allows negotiation of this transaction is ‘REJEC’.

The server MUST allow rejecting user(s) only to those users who have Moderator or Administrator privileges in the particular group. If this is the case, the server MUST add the users in the Add-User-List element to the reject list of the particular group and remove the users in the Remove-User-List element from the reject list of the particular group.

The server MUST ignore adding a user who is already in the list and removing a user who is not in the list without generating an error.

If there are any users joined to the group that are added to the reject list, the server MUST remove (kick) those users from the group.

10.8.2 Error conditions

Generic error conditions:

- Not logged in. (604)
- Service not agreed. (506)
- Service not supported. (405)

RejectListRequest error conditions:

- User unknown. (531)
- Group does not exist. (800)
- Insufficient user rights. (816)
- Service provider agreement missing (907)

10.8.3 Primitives and information elements

Primitive	Direction
RejectListRequest	Client → Server
RejectListResponse	Client ← Server

Table 139: Primitive directions in reject user(s) from group transaction

Information Element	Req	Type	Description
Message-Type	M	RejectListRequest	Message identifier.
Transaction-ID	M	String	Identifies the transaction requested.
Segment-Info	O	Structure	Contains segmentation information when the primitive is segmented.
Session-ID	M	String	Identifier for the session.
Group-ID	M	String	Identification of the group.
Add-Users-List	O	Structure	The list of users that SHOULD be added to the reject list.
Remove-Users-List	O	Structure	The list of users that SHOULD be removed from the reject list.

Table 140: Information elements in RejectListRequest primitive

Information Element	Req	Type	Description
Message-Type	M	RejectListResponse	Message identifier.
Transaction-ID	M	String	Identifies the transaction requested.
Segment-Info	O	Structure	Contains segmentation information when the primitive is segmented.
Session-ID	M	String	Identifier for the session.
User-List	M	Structure	The list of users that are in the reject list.

Table 141: Information elements in RejectListResponse primitive

10.9 Subscribe to Group Change Notification

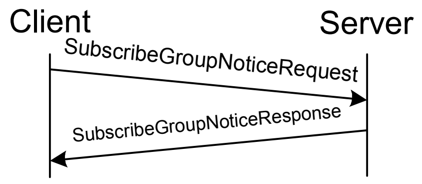

Figure 77: Subscribe group change notification transaction

A user MAY get/set/unset group change subscription status. The client MUST send the SubscribeGroupNoticeRequest primitive to the server. The primitive MUST contain the ID of the group and the 'Type' of the requested operation. The answer from the server for the get operation MUST be the SubscribeGroupNoticeResponse primitive, or Status if an error occurs. The answer from the server for the set/unset operation MUST be a Status primitive. While the subscription is active, the user MUST receive GroupChangeNotice primitives.

The client and the server MAY support the 'subscribe group change notification transaction'. The service tree leaf that allows negotiation of this transaction is 'SUBGCN'.

If the SubscribeGroupNoticeRequest primitive requests 'get' operation, the server MUST NOT update the subscription state on the server, and the current status of the subscription for the requested group MUST be sent in the response.

If the SubscribeGroupNoticeRequest primitive requests 'set' operation, the server SHOULD turn on the subscription state for the particular group.

If the SubscribeGroupNoticeRequest primitive requests 'unset' operation, the server MUST turn off the subscription state for the particular group.

The server MUST automatically turn the subscription status off when the joined user/member to whom the subscription belongs leaves (or removed from) the particular group.

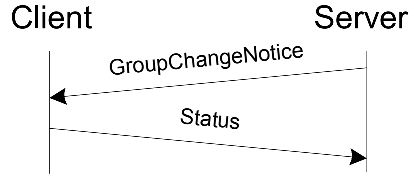

Figure 78: Group change notification

If the user subscribed to group change notifications, the server MUST send group change notification(s) to the user whenever some other user leaves or joins the group, or the group properties or the user's own properties have been changed. The server MUST send the GroupChangeNotice primitive to the users (whose group change subscription is active) containing the recently joined or left users, or the new properties of the group. The list of the recently joined or left users MUST contain ScreenName, and MUST contain the User-IDs for those users whose 'Show-ID' is 'T' (true), while MUST NOT contain the User-IDs for those users whose 'Show-ID' is 'F' (false) unless the receiving user is a Moderator/Administrator of the group. The server MUST also include the list of those recently joined/left users who have been blocked by the receiving user – in which case the server MUST regard the 'Show-ID' property of the blocked users as described previously. The server MUST include users in the Joined-Blocked-Users-List or Left-Blocked-Users-List also in the Joined-Users-List or Left-Users-List.

The client and the server MAY support the 'group change notification transaction'. The service tree leaf that allows negotiation of this transaction is 'GRCHN'.

When the 'Welcome Note' property of the group has been changed, the server MUST NOT send group change notification.

The server MAY maintain an internal 'Validity' property of the groups. When this internal 'Validity' has been updated, the server MUST NOT send group change notification.

When an administrator/moderator updates the Validity of a group, the server MUST send group change notification to all members (with active subscription) and to all joined users (with active subscription).

When any other properties of a group than 'Welcome Note' and 'Validity' (server internal update) have been changed, the server MUST send group change notification to all group members (with active subscription) and to all joined users (with active subscription) containing the changed properties.

When a user's own properties have been changed, the server MUST send group change notification only that particular user (if he/she has active group change subscription).

When some users have left or joined the group, the server MUST send group change notification to all joined users (with active subscription).

The group change notification MUST contain information about the changed properties and/or joined/left users – e.g. empty notifications MUST NOT occur.

10.9.1 Error Conditions

Generic error conditions:

- Not logged in. (604)
- Service not agreed. (506)
- Service not supported. (405)

SubscribeGroupNoticeRequest error conditions:

- Group does not exist. (800)
- Group was not joined before transaction. (808)

GroupChangeNotice error conditions:

- Client MAY ignore any error and respond with Successful. (200)
- Service provider agreement missing (907)

10.9.2 Primitives and information elements

Primitive	Direction
SubscribeGroupNoticeRequest	Client → Server
SubscribeGroupNoticeResponse	Client ← Server
GroupChangeNotice	Client ← Server
Status	Client → Server

Table 142: Primitive directions in subscribe group change notification transaction

Information Element	Req	Type	Description
Message-Type	M	SubscribeGroup NoticeRequest	Message identifier.
Transaction-ID	M	String	Identifies the transaction requested.
Session-ID	M	String	Identifier for the session.
Group-ID	M	String	Identification of the group.
Subscribe-Type	M	Enumerated character	Indicates the type of subscription request. (“G” for Get, “S” for Set, and “U” for Unset.)

Table 143: Information elements in SubscribeGroupNoticeRequest primitive

Information Element	Req	Type	Description
Message-Type	M	SubscribeGroup NoticeResponse	Message identifier.
Transaction-ID	M	String	Identifies the transaction requested.
Session-ID	M	String	Identifier for the session.
Subscription-State	M	Boolean	Indicates the status of subscription.

Table 144: Information elements in SubscribeGroupNoticeResponse primitive

Information Element	Req	Type	Description
Message-Type	M	GroupChange Notice	Message identifier.
Transaction-ID	M	String	Identifies the transaction requested.
Segment-Info	O	Structure	Contains segmentation information when the primitive is segmented.
Session-ID	M	String	Identifier for the session.
Group-ID	M	String	Identification of the group.
Joined-Users-List	O	Structure	A list of the users that recently joined the group (screen name, User-ID).
Left-Users-List	O	Structure	A list of screen names of the users that recently left the group.
Joined-Blocked-Users-List	O	Structure	A list of the blocked users that recently joined the group (Screen Name, User-ID),
Left-Blocked-Users-List	O	Structure	A list of the blocked users that recently left the group (Screen Name, User-ID).
Group-Props	O	Structure	The new properties of the group.
Own-Props	O	Structure	The new properties of the user in the group.

Table 145: Information elements in GroupChangeNotice primitive

11.Status Codes and Descriptions

CSP uses the concept and paradigm of HTTP/1.1 response to define the status code. However, there is no logical or semantic relationship between the status codes in CSP and the status codes in HTTP.

The following sections define the general categories as well as each status code.

11.1 1xx – Informational

The client MUST be prepared to accept one or more 1xx status codes prior to a regular response even if the client does not expect a 100 "Continue" status code. A user agent SHALL ignore unexpected 1xx status code. This category of the status codes does not finish a transaction.

11.1.1 100 – Continue

The client SHOULD continue with its request. The server has accepted the request for processing, but the processing has not been completed. The request might or might not eventually be successfully completed. The server MUST send a final response again upon completing the request. The "100" response is used when time of completion will be too long, possibly causing the server and client connection to break.

11.1.2 101 – Queued

The client SHOULD continue with its request. The server has accepted the request, but does not have resources to start processing. The request might or might not eventually be successfully completed. The server MUST send a final response again upon completing the request.

11.1.3 102 – Started

The client SHOULD continue with its request. The server has accepted the request for processing. The "102" response is used when server needs to start additional transactions in order to process the request. The server MUST send a final response again upon completing the request.

11.2 2xx – Successful

The 2xx class of status codes indicates that the client's request was successfully received, understood and accepted.

11.2.1 200 – Successful

This is used to indicate that the request succeeded.

11.2.2 201 – Partially successful

This is used to indicate that the request was successfully completed, but some parts were not completed due to certain errors. The details of the error case(s) are indicated in the response.

11.2.3 203 – Extension block ignored

The client/server requested a transaction that carries an extension block however the extension block was ignored on the terminating end – in an otherwise successful transaction. The originator of the request MUST NOT perform the behavior described in the proprietary solution as the requested proprietary functionality was ignored on the terminating end.

11.3 4xx – Client Error

The 4xx class of status codes is intended for cases in which the client seems to have erred. The server SHOULD include the explanation of the error situation including whether it is a temporary or permanent condition. The user agents SHOULD be able to display the error description to the user.

11.3.1 400 – Bad Request

The server could not understand the request due to the malformed syntax. The client MUST NOT repeat the request without modification.

11.3.2 401 – Unauthorized

When an authorization request is expected, the presence server will respond with this status code. Properties will contain details of available authorization schemes.

11.3.3 402 – Bad Parameter

The server cannot understand one of the parameters in the request. The client MUST NOT repeat the request without modification.

11.3.4 403 – Forbidden

The server understood the request, but the principal settings denied access to some of the presence, contact information or group. Authorization will not help and the request SHOULD NOT be repeated. This type of response can be returned if user not login in the network yet.

11.3.5 404 – Not Found

The server cannot find anything matching the request. No indication is given of whether the condition is temporary or permanent.

11.3.6 405 – Service Not Supported

The server does not support the service method in the request.

11.3.7 408 – Request Timeout

The client did not produce a request within the time the server was prepared to wait.

11.3.8 409 – Invalid password

The password provided by the client was incorrect; it does not match with the given User-ID. The client MUST NOT repeat the request without modification.

11.3.9 410 – Unable to Deliver

The server cannot deliver the request. The requested resource is no longer available at the server and no forwarding address is known.

11.3.10 411 – Unable to find suitable content type

The server cannot deliver the response because the client does not support any suitable content type. The client MUST NOT repeat the request without performing client capability negotiation where it agrees on a suitable content type.

11.3.11 415 – Unsupported Media Type

The server cannot deliver the request because the client cannot support the format of the entity that it requested.

11.3.12 420 – Invalid Transaction-ID

The server encountered an invalid transaction ID.

11.3.13 422 – User-ID and Client-ID do not match

The User-ID and the Client-ID do not match in the request.

11.3.14 423 – Invalid Invitation-ID

The server encountered an invalid invitation-ID.

11.3.15 424 – Invalid Search-ID

The server encountered an invalid search-ID.

11.3.16 425 – Invalid Search-Index

The server encountered an invalid search index.

11.3.17 426 – Invalid Message-ID

The server encountered an invalid Message-ID.

11.3.18 427 – Invalid User-ID

The server encountered an invalid User-ID.

11.3.19 428 – Invalid Client-ID

The server encountered an invalid Client-ID.

11.3.20 429 – Missing Group-ID

The server encountered a request without Group-ID while the Group-ID is necessary to fulfill the request.

11.3.21 431 – Unauthorized Group Membership

The user agent is not an authorized member of the group.

11.3.22 432 – Response too large

The response would be larger than the client is capable to handle according to the limitations agreed during client capability negotiation. The client MUST NOT repeat the request without performing client capability negotiation where it agrees on higher limitations.

11.3.23 433 – Invalid notification type

The server encountered an invalid notification type.

11.3.24 434 – Unknown Segment reference

The client or the server requested a segment, however the Transaction-ID in the Segment-ID was not recognized either because there is no such, or because it has been invalidated already. The originator MUST NOT repeat the request without modification.

11.3.25 435 – Segment index out of bounds

The client or the server requested a segment, however the segment index in the Segment-ID was not recognized either because there is no such, or because it has been invalidated already. The originator MUST NOT repeat the request without modification.

11.3.26 436 – System Message Response required

The server has sent a System Message notification to the client requiring response from the end-user. The client MUST NOT repeat the request without gathering feedback from the end-user and sending a System Message resposne to the server.

11.3.27 437 – Unknown System Message ID

The System Message response contains an unknown system message id.

11.3.28 438 – Incorrect Verification Key

The System Message response contains an incorrect verification key.

11.3.29 439 – Verification Mechanism Not Supported

The client does not support the verification mechanism required by the server in the System Message notification.

11.3.30 440 – Not allowed notification type

The client subscribed to an event of functionality not agreed during service negotiation.

11.3.31 441 – Number of characters exceeds the maximum number of characters

The client submitted a value where the number of characters is more than the allowed maximum numbers of characters for this value.

11.3.32 442 – Wrong value type

The type of the value submitted by the client does not match the required value types. E.g. the Age field MUST be an integer, etc.

11.3.33 450 – Missing Application-ID

The client attempted to accept an invitation without providing an Application-ID, thus the server rejected the request. The client MUST NOT repeat the request without modification.

11.3.34 451 – Invalid Application-ID

The client attempted to use an Application-ID that was not registered to it during login. The client MUST NOT repeat the request without modification during the active session.

11.3.35 452 – Forbidden Application-ID

The requesting user is blocking the requested Application-ID while the BlockList is in use, or the Application-ID is not permitted while the GrantList is in use. The client MUST NOT repeat the request without applying modifications to his/her access control rules.

11.3.36 480 – Segments dropped before delivery

The client has dropped the segment chain while the server was still retrieving it. The server MUST perform the segmented transaction using the successfully retrieved parts, unless mandatory or conditional elements are missing to perform the requested operation.

11.3.37 481 – Segments dropped before delivery with rollback

The client has dropped the segment chain while the server was still retrieving it. The server MUST NOT perform the segmented transaction using the successfully retrieved parts; it MUST roll back to the original state if necessary.

11.4 5xx – Server Error

The 5xx class of status codes is intended for cases in which the server is aware that it has erred or is incapable of performing the request.

11.4.1 500 – Internal server or network error

The server encountered an unexpected condition that prevented it from fulfilling the request.

11.4.2 501 – Not Implemented

The server does not support the functionality required to fulfil the request. This is the appropriate response when the server does not recognize the request method, and it is not capable of supporting it for any resources.

11.4.3 502 – Session could not be recovered

The server was not able to recover the session requested by the client.

11.4.4 503 – Service Unavailable

The server is currently unable to handle the request due to a temporary overloading or maintenance of the server.

11.4.5 504 – Timeout

The server could not produce a response within the time that it expected.

11.4.6 505 – Version Not Supported

The server does not support, or refuse to support, the request version that was used.

11.4.7 506 – Service not agreed

During service negotiation the server did not agree to provide the transaction that the client requests. The client MUST NOT repeat the request without a new service negotiation.

11.4.8 507 – Message queue is full

The server cannot fulfill the request, because its message queue is full. The client MAY repeat the request.

11.4.9 508 – Unsupported message context

The server is unable to understand the message context that the client has used. The client MUST NOT repeat the request without modification.

11.4.10 509 – Related services are missing

The server cannot perform the request, as some services that are indirectly related to the requested transaction have not been agreed during service negotiation. The client MUST NOT repeat the request until the indirectly required services have been agreed using a service negotiation.

11.4.11 510 – Related client capabilities are missing

The server cannot perform the request, as some client capabilities that are indirectly related to the requested transaction have not been agreed during client capability negotiation. The client MUST NOT repeat the request until the indirectly required capabilities have been agreed using a client capability negotiation.

11.4.12 515 – Application is forbidden

The client attempted to assign an Application-ID for the session during login that the service provider does not allow for some reason – such reasons might be that the *application* has been identified by service provider to be un-secure, non-compliant, spy-ware, etc –, thus the server rejected the login request. The client MUST NOT repeat the request without modification. The client MAY repeat the request without Application-ID, or using another Application-ID.

11.4.13 516 – Domain Not Supported

The server does not support forwarding to a different domain space.

11.4.14 517 – Location Not Supported

The server is unable to generate a map for the requested location for some reason. The client MUST NOT repeat the request without modification.

11.4.15 531 – Unknown user

The specified user is unknown/not valid User-ID was given.

11.4.16 532 – Recipient Blocked the Sender

The recipient of the message or invitation blocked the sender. Note that returning this error code reveals to the sender that the recipient has blocked it. It is up to the implementation and service provider to decide whether or not this error code SHOULD be returned. A WV server MAY instead report success, even though the message or invitation was discarded, to conceal this fact.

11.4.17 533 – Message Recipient Not Logged in

The recipient of the message is not logged in.

11.4.18 534 – Message Recipient Unauthorized

The recipient of the message is not authorized.

11.4.19 535 – Search timed out

The server has invalidated the requested search-request.

11.4.20 536 – Too many hits

The server performed the search successfully, but the server implementation limits the maximum number of hits – the server MAY discard the hits that are over its limits and the discard hits will not be available for the client.

11.4.21 537 – Too broad search criteria

The server did not perform the search – the server simply by analyzing the search criteria came to the conclusion that according to the requested criteria the search would give higher number of hits than the server is willing to handle. The client MAY repeat the request with narrowed search criteria.

11.4.22 538 – Message has been rejected

Recipient has rejected message. Note that returning this error code reveals to the sender that the recipient has rejected the message. It is up to the implementation and service provider to decide whether or not this error code SHOULD be returned. A WV server MAY instead report success, even though the message was discarded, to conceal this fact.

11.4.23 539 – Message-Count exceeded

The list of messages on the server exceeds the maximum number of Message-Info structures to be returned as specified in Message-Count.

11.4.24 540 – Message has been rejected due to limitations

The server (or another server on the route to the Recipient) has rejected the instant message because the instant message content is not allowed either because the content type is not supported, or because it is too large. The client MUST NOT repeat the transaction without modification. The client SHOULD check its own limitations from its agreed client capability settings, and the recipient's content type limitations using the ClientInfo presence attribute.

11.4.25 541 – Message has been forwarded

Recipient has forwarded message without retrieving it first. Note that returning this error code reveals to the sender that the recipient has forwarded the message. It is up to the implementation and service provider to decide whether or not this error code SHOULD be returned. A WV server MAY instead report success, even though the message was forwarded, to conceal this fact.

11.4.26 542 – Message has expired

Message has not been retrieved by the recipient in the specified time period and has been deleted from the server.

11.4.27 543 – No matching digest scheme supported

The server does not support any of the digest schemas that the client has requested.

11.4.28 544 – Too many elements in advanced criteria

The server did not perform the search – it has received an advanced search request, which includes advanced criteria with too many elements. The client MUST NOT repeat the request without modification.

11.4.29 545 – Too many levels of nesting in advanced criteria

The server did not perform the search – it has received an advanced search request, which includes advanced criteria with too many levels of nesting. The client MUST NOT repeat the request without modification.

11.4.30 546 – Message cannot be delivered due to limitations

The client attempted to retrieve an instant message where the content is not allowed either because the content type is not supported or it is too large according to the client's capability negotiation agreement. The client MUST NOT repeat the transaction without modification. The client SHOULD check its own limitations from its agreed client capability settings, and MAY perform client capability negotiation to agree on the necessary content type/limitation.

11.4.31 547 – Wildcard characters not allowed

The client attempted to use wildcard characters, however the server does not allow wildcard characters to be used. The client MUST NOT repeat the request without modification. The client MAY repeat the request with a full string without wildcard characters.

11.4.32 548 – Wildcard expression is too complicated

The client attempted to use wildcard characters, however the server finds the expression too complicated. The client MUST NOT repeat the request without modification. The client MAY repeat the request with a less complicated expression.

11.4.33 550 – Header encoding not supported

The requested SMS header encoding (UDH or textual) is not supported. The clients MUST NOT repeat the request without modification. The client MAY repeat the request with the opposite header encoding (UDH if it was textual, or vice versa). See [CSP PTS] for detail on how this status code is used.

11.4.34 560 – Unsupported search-element was requested

The client requested a search using a search-element that is not supported by the server. The client MUST NOT repeat the request without modification.

11.4.35 561 – Supported search-element

The server supports searching using the search-element marked with this error code. The client MAY attempt to search using this element.

11.4.36 562 – Unsupported search-element

The server does not support searching using the search-element marked with this error code. The client SHOULD NOT attempt to search using this element.

11.5 6xx – Session

The 6xx class status code indicates the session-related status.

11.5.1 600 – Session Expired

The client was disconnected because time-to-live parameter of user session has expired. The client MAY attempt to login at any time.

11.5.2 601 – Forced Logout

The server has disconnected the client for some reason. The client receiving this error code MUST NOT attempt login automatically, it MUST wait for user interaction. The status details SHOULD give sufficient information for the user when the next login attempt MAY be performed. The user MAY trigger the login manually.

11.5.3 604 – Invalid session (not logged in).

There is no such session. (Previously not logged in, disconnected, or logged out.)

11.5.4 605 – New value not accepted.

The server does not accept the new timeout value requested by the client, the old value MUST be used.

11.5.5 606 – Some services are not available

The server does not accept the session re-establishment request because some of the services that have been agreed during the terminated session are not available. The client MUST NOT repeat the re-establishment request without modification. The client MAY establish a new session.

11.5.6 607 – Too many non-conformant System Message replies

The server will not accept new requests from the client for a period of time (it is implementation-specific), because the client already attempted to respond a System Message with non-conformant replies too many times. The server cannot verify whether the user is making mistakes, or the client is not OMA IMPS compliant, but this error code allows the server to protect itself against undesired attempts. The server is NOT REQUIRED to validate any requests while this protection is active – it MAY respond any request without validation with a Status primitive using the same error code. The client MUST NOT repeat the request until the protection time indicated in the status details expires.

11.5.7 608 – Client-ID is not unique

The server did not accept the login request from the client, because the Client-ID that the client attempted to use during login is already in use.

11.5.8 610 – User session limitation reached

The user has already reached the limitation on the maximum number of concurrent sessions that are allowed across all SAPs that provide access to the service provider's domain. The client SHOULD NOT repeat the request for any SAPs.

11.6 7xx – Presence and Contact List

The 7xx class indicates the presence and contact list related status codes.

11.6.1 700 – Contact list does not exist

The contact list specified in the request does not exist.

11.6.2 701 – Contact list already exists

The contact list specified in the request already exists.

11.6.3 702 – Invalid or unsupported user properties

The user properties specified in the request are invalid, or not supported.

11.6.4 703 – Contact list is empty

The client attempted to use a contact list that is empty.

11.6.5 750 – Invalid or unsupported presence attribute

The presence attribute(s) specified in the request are invalid, or not supported.

11.6.6 751 – Invalid or unsupported presence value

The presence value(s) specified in the request are invalid, or not supported. The client SHOULD NOT repeat the request without modification.

11.6.7 752 – Invalid or unsupported contact list property

One or more contact list properties specified in the request are invalid or not supported. The client SHOULD NOT repeat the request without modification.

11.6.8 753 – The maximum number of contact lists has been reached for the user

The server limits the maximum number of contact lists per user. The limit has been reached; so additional contact lists cannot be created. The client SHOULD NOT repeat the request until a contact list that belongs to the particular user has been deleted.

11.6.9 754 – The maximum number of contacts has been reached for the user

The server limits the maximum number of contacts per user. The limit has been reached; so additional contacts cannot be created. The client SHOULD NOT repeat the request until a contact that belongs to the particular user has been deleted.

11.6.10 755 – The maximum number of attribute lists has been reached for the user

The server limits the maximum number of attribute lists per user. The limit has been reached; so additional attribute lists cannot be created. The client SHOULD NOT repeat the request until an attribute list that belongs to the particular user has been deleted.

11.6.11 756 – The maximum number of Users in grant list has been reached for the user

The server limits the maximum number of users in grant list per user. The limit has been reached; so the user cannot add new entities to the grant list. The client SHOULD NOT repeat the request until the user has removed entities from grant list.

11.6.12 757 – The maximum number of Users in block list has been reached for the user

The server limits the maximum number of users in block list per user. The limit has been reached; so the user cannot add new entities to the block list. The client SHOULD NOT repeat the request until the user has removed entities from block list.

11.6.13 758 – The maximum number of users in watcher list has been reached for the user

The server limits the maximum number of users in watcher list per user. The limit has been reached; so other users cannot subscribe the presence attributes of the user.

11.7 8xx – Groups

The 8xx class indicates the group-related status codes.

11.7.1 800 – Group does not exist

The group specified in the request does not exist.

11.7.2 801 – Group already exists

The group specified in the request already exists.

11.7.3 802 – Group is open

The group specified in the request is an open group.

11.7.4 803 – Group is restricted

The group specified in the request is a restricted group.

11.7.5 804 – Group is public

The group specified in the request is public.

11.7.6 805 – Group private

The group specified in the request is private.

11.7.7 806 – Invalid/unsupported group properties

The group properties specified in the request are invalid or not supported.

11.7.8 807 – Group is already joined

The group specified in the request is already joined. If the server does not allow the same user to join a group more than once then this error code is used to indicate that the user is already joined the particular group.

11.7.9 808 – Group is not joined

The request cannot be processed because it requires the user to be joined to the group.

11.7.10 809 – User has been rejected

The user has been rejected from the particular group. He/she is forced to leave the group and cannot join.

11.7.11 810 – Not a group member

The request cannot be processed because the user is not a member of the specified restricted group The client SHOULD NOT repeat the request until the user has been added to the group as a member.

11.7.12 812 – Private messaging is disabled for group

The client requested private message delivery but the private messaging is disabled in the particular group.

11.7.13 813 – Private messaging is disabled for user

The client requested private message delivery but the private messaging is disabled for the particular user.

11.7.14 814 – The maximum number of groups has been reached for the user

The server limits the maximum number of groups per user. The limit has been reached; additional groups cannot be created. The client SHOULD NOT repeat the request until a group that belongs to the particular user has been deleted.

11.7.15 815 – The maximum number of groups has been reached for the server

The maximum number of groups is limited on the server. The server limit has been reached; additional groups cannot be created. The client MAY repeat the request.

11.7.16 816 – Insufficient group privileges

The user is a member in the particular group, but does not have sufficient privileges group to perform the requested operation. The client SHOULD NOT repeat the request until the user has been authorized properly.

11.7.17 817 – The maximum number of joined users has been reached

The maximum number of joined users has been reached in the requested group. The client MAY repeat the request.

11.7.18 818 – Minimum age requirement not fulfilled

The group has an active age restriction limitation, and the requesting user does not fulfill the requirements needed to perform this transaction. The client SHOULD NOT repeat the request within a reasonable time period.

11.7.19 821 – History is not supported.

The server does not support group history. The client MUST NOT repeat the request.

11.7.20 822 – Cannot have searchable group without name or topic

The server cannot perform group search without group name or group topic. Either group name or group topic or both MUST be non-empty to support group search.

11.7.21 823 – The maximum number of group members has been reached

The server limits the maximum number of group members per group. The limit has been reached; so additional group members cannot be added. The client SHOULD NOT repeat the request until a group member has been removed from the group.

11.7.22 824 – Own Request

The reason code for the LeaveGroupResponse. Server sends this error code in the LeaveGroupResponse as a response to the client-initiated LeaveGroupRequest.

11.7.23 825 – Extend conversation rejected

The invitee rejected the invitation to the extended conversation.

11.8 9xx General Errors

The 9xx class indicates status codes too general to fit into other classes.

11.8.1 900 Multiple Errors

No part of the transaction was successfully processed for several reasons and thus one other status code cannot indicate the errors. The details of the error cases are indicated in the response.

11.8.2 901 General Address Error

The general address is not supported. No specific error is given due to security or privacy reason.

11.8.3 902 – Not enough credit to complete requested operation

The server cannot perform the requested operation since the user has not enough credit.

11.8.4 903 – Operation requires a higher class of service

The server cannot perform the requested operation since it requires a higher class of service. A class of service is a designation assigned by the service provider to describe the service treatment and privileges given to a particular user (e.g., premium, gold).

11.8.5 904 – Missing mandatory field(s) of requesting user

The requesting user did not fill in the mandatory fields of his/her public profile. The client MUST NOT repeat the request, and MAY receive a system message – see system message in 7.1 System Message.

11.8.6 905 – Public profile of requested user is not available

The requested user did not fill in the mandatory fields of his/her public profile, or the public profile is not available for some other reason. The client SHOULD NOT repeat the request within a reasonable time period.

11.8.7 906 – Too many public profiles requested

The client requested too many public profiles in a request. The server has successfully delivered as much public profiles as its implementation allows within a single transaction, however some public profiles have not been delivered due the limitations on server side. The client MAY retrieve the rest of the profiles, however it MUST retrieve the excess of public profiles in a separate transaction.

11.8.8 907 – Service provider agreement missing

The client attempted to perform an operation that involves another service provider, however the agreement between the related service providers prevents the server from performing the requested operation. The client SHOULD NOT repeat the request without modification.

11.8.9 908 – There are no instant messages

The client attempted to retrieve the list of instant message on the server, however there are no instant messages.

11.8.10 909 – Recipient does not support the requested functionality

The recipient client does not support the requested functionality.

11.8.11 920 – MSISDN error

The client attempted to use an MSISDN that is not used by the device, thus the server rejected the request. The client MUST NOT repeat the request without modification.

11.8.12 921 – Registration confirmation

This status code indicates that the registration is successful but extra registration information is required before the user can use the service and the client MUST NOT continue the login.

12.Extension Framework

CSP defines a framework that can be used by different vendors to extend the standard IMPS CSP protocol. This framework includes support for 3 basic operations:

- Extending existing primitives
- Introducing new primitives
- Extending end-to-end messages

It is RECOMMENDED that *applications* use the version discovery transaction for detecting which extensions do the server support and the service negotiation for negotiating these capabilities. The details of how to achieve these tasks are outside the scope of the IMPS protocol.

It is not possible to discover directly what kind of extensions a recipient client supports, however the ClientInfo presence attribute includes information that might include information that identifies the client/application itself. See the ClientInfo presence attribute in [PA].

The extension framework usage is described in detail in [CSP XMLS].

The server or client MAY return status code 501 to indicate that a particular transaction is not implemented within a recognized namespace.

12.1 Extending Existing Primitives

Extension blocks MAY be appended to existing primitives, designated with a namespace. Both clients and servers MUST ignore unrecognized extension blocks without generating an error.

12.2 Introducing New Primitives

The extension framework defines general primitives that serve as an envelope for proprietary operations. These primitives obey all rules set for the normal primitives with 1 exception:

- The actual content of these primitives are not defined.

Both clients and servers MUST ignore unrecognized primitive extensions without generating an error.

12.2.1 Extending end-to-end messages

Extension blocks MAY be added to end-to-end messages as well. The extension blocks within the end-to-end messages are for the recipients only, thus the server MUST send the extension block to all of the recipients unless the extension block has been identified as a security threat. The receiving clients MUST ignore unrecognized end-to-end message extensions without generating an error.

Appendix A. Change History (Informative)

A.1 Approved Version History

Reference	Date	Description
OMA-TS-IMPS_CSP-V1_3	23 Jan 2007	Status changed to Approved by TP TP Doc ref# OMA-TP-2006-0453R02

Appendix B. Static Conformance Requirements (Normative)

The notation used in this section is specified in [IOPPROC].

B.1 OMA IMPS Service Requirements

B.1.1 Clients

Item	Function	Reference	Status	Requirement
IMPS-CSP-SERV-C-001	Support of Service Access Point functionality	5.1	M	(IMPS-CSP-SERV-C-002 OR IMPS-CSP-SERV-C-003 OR IMPS-CSP-SERV-C-004 OR IMPS-CSP-SERV-C-005) AND IMPS-CSP-SAP-C-002 AND (IMPS-CSP-SAP-C-006 OR IMPS-CSP-SAP-C-007) AND IMPS-CSP-SAP-C-009 AND IMPS-CSP-SAP-C-010 AND IMPS-CSP-SAP-C-011 AND IMPS-CSP-SAP-C-013 AND IMPS-CSP-SAP-C-015 AND IMPS-CSP-SAP-C-017 AND IMPS-CSP-SAP-C-018 AND IMPS-CSP-SAP-C-019 AND IMPS-CSP-SAP-C-020 AND IMPS-CSP-SAP-C-021
IMPS-CSP-SERV-C-002	Support of Instant Messaging Service Element functionality	5.1	O	IMPS-CSP-IMSE-C-002 AND IMPS-CSP-IMSE-C-005 AND IMPS-CSP-IMSE-C-009
IMPS-CSP-SERV-C-003	Support of Presence Service Element functionality	5.1	O	IMPS-CSP-PRSE-C-005 AND IMPS-CSP-PRSE-C-006 AND IMPS-CSP-PRSE-C-007 AND IMPS-CSP-PRSE-C-008 AND IMPS-CSP-PRSE-C-010 AND IMPS-CSP-PRSE-C-012
IMPS-CSP-SERV-C-004	Support of Group Service Element functionality	5.1	O	IMPS-CSP-GRSE-C-011 AND IMPS-CSP-GRSE-C-012
IMPS-CSP-SERV-C-005	Support of Content Service Element functionality	5.1	O	
IMPS-CSP-SERV-C-006	Routing	5.5	M	
IMPS-CSP-SERV-C-007	Content	5.6	M	

B.1.2 Servers

Item	Function	Reference	Status	Requirement
IMPS-CSP-SERV-S-001	Support of Service Access Point functionality	5.1	M	(IMPS-CSP-SERV-S-002 OR IMPS-CSP-SERV-S-003 OR IMPS-CSP-SERV-S-004 OR IMPS-CSP-SERV-S-005) AND IMPS-CSP-SAP-S-001 AND IMPS-CSP-SAP-S-002 AND IMPS-CSP-SAP-S-004 AND IMPS-CSP-SAP-S-005 AND IMPS-CSP-SAP-S-006 AND IMPS-CSP-SAP-S-007 AND IMPS-CSP-SAP-S-009 AND IMPS-CSP-SAP-S-010 AND IMPS-CSP-SAP-S-011 AND IMPS-CSP-SAP-S-013 AND IMPS-CSP-SAP-S-014 AND IMPS-CSP-SAP-S-015 AND IMPS-CSP-SAP-S-017 AND IMPS-CSP-SAP-S-018 AND IMPS-CSP-SAP-S-019 AND IMPS-CSP-SAP-S-020 AND IMPS-CSP-SAP-S-021
IMPS-CSP-SERV-S-002	Support of Instant Messaging Service Element functionality	5.1	O	IMPS-CSP-IMSE-S-001 AND IMPS-CSP-IMSE-S-002 AND IMPS-CSP-IMSE-S-005 AND IMPS-CSP-IMSE-S-006 AND IMPS-CSP-IMSE-S-007 AND IMPS-CSP-IMSE-S-008 AND IMPS-CSP-IMSE-S-009 AND IMPS-CSP-IMSE-S-010
IMPS-CSP-SERV-S-003	Support of Presence Service Element functionality	5.1	O	IMPS-CSP-PRSE-S-005 AND IMPS-CSP-PRSE-S-006 AND IMPS-CSP-PRSE-S-007 AND IMPS-CSP-PRSE-S-008 AND IMPS-CSP-PRSE-S-010 AND IMPS-CSP-PRSE-S-012
IMPS-CSP-SERV-S-004	Support of Group Service Element functionality	5.1	O	IMPS-CSP-GRSE-S-011 AND IMPS-CSP-GRSE-S-012 AND IMPS-CSP-GRSE-S-016
IMPS-CSP-SERV-S-005	Support of Content Service Element functionality	5.1	O	
IMPS-CSP-SERV-S-006	Routing	5.5	M	
IMPS-CSP-SERV-S-007	Content	5.6	M	

B.2 Addressing Requirements

B.2.1 Clients

Item	Function	Reference	Status	Requirement
IMPS-CSP-ADDR-C-001	Support for local addressing	5.3.2	M	
IMPS-CSP-ADDR-C-002	Support for fully qualified addressing	5.3.2	M	

Item	Function	Reference	Status	Requirement
IMPS-CSP-ADDR-C-003	Support for client addressing.	5.3.7	M	
IMPS-CSP-ADDR-C-004	Users cannot refer to or access other users' contact lists.	5.3.5	M	
IMPS-CSP-ADDR-C-005	If the scheme part of a WV address is missing the default schema of "wv:" is assumed.	5.3.2	M	
IMPS-CSP-ADDR-C-006	The IMPS client supports other scheme than "wv:".	5.3.2	O	

B.2.2 Servers

Item	Function	Reference	Status	Requirement
IMPS-CSP-ADDR-S-001	Support for local addressing	5.3.2	M	
IMPS-CSP-ADDR-S-002	Support for fully qualified addressing	5.3.2	M	
IMPS-CSP-ADDR-S-003	Server maintains the same addressing format within a transaction.	5.3.1	M	
IMPS-CSP-ADDR-S-004	Support for client addressing.	5.3.7	M	
IMPS-CSP-ADDR-S-005	Users cannot refer to or access other users' contact lists.	5.3.5	M	
IMPS-CSP-ADDR-S-006	If the schema part of a WV address is missing the default scheme of "wv:" is assumed.	5.3.2	M	
IMPS-CSP-ADDR-S-007	The IMPS server supports other schemes than IMPS scheme	5.3.2	O	

B.3 Session Requirements

B.3.1 Clients

Item	Function	Reference	Status	Requirement
IMPS-CSP-SESSION-C-001	If the primitive is sent within a session, the Type of the session is 'Inband'.	5.1	M	
IMPS-CSP-SESSION-C-002	If the primitive is sent within a session, the ID element is present in the session identification element.	5.1	M	
IMPS-CSP-SESSION-C-003	If the primitive is sent without session, the Type of the session is 'Outband'	5.1	M	
IMPS-CSP-SESSION-C-004	If the primitive is sent without session, the ID element is not present in the session identification element.	5.1	M	
IMPS-CSP-SESSION-C-005	If the primitive is sent within the session, the Session-IDs of the originating message and the reply are equal.	5.1	M	

Item	Function	Reference	Status	Requirement
IMPS-CSP-SESSION-C-006	If the client logs in with binary message format, the client and server sends all primitives using binary message format throughout the session.	5.1	M	
IMPS-CSP-SESSION-C-007	The client uses only the protocol version accepted by the server in the login transaction throughout the whole session.	5.1	M	

B.3.2 Servers

Item	Function	Reference	Status	Requirement
IMPS-CSP-SESSION-S-001	If the primitive is sent within a session, the Type of the session is 'Inband'.	5.1	M	
IMPS-CSP-SESSION-S-002	If the primitive is sent within a session, the ID element is present in the session identification element.	5.1	M	
IMPS-CSP-SESSION-S-003	If the primitive is sent without session, the Type of the session is 'Outband'	5.1	M	
IMPS-CSP-SESSION-S-004	If the primitive is sent without session, the ID element is not present in the session identification element.	5.1	M	
IMPS-CSP-SESSION-S-005	If the primitive is sent within the session, the Session-IDs of the originating message and the reply are equal.	5.1	M	
IMPS-CSP-SESSION-S-006	The server does not disconnect session within the agreed KeepAliveTime.	6.7	M	
IMPS-CSP-SESSION-S-007	If TimeToLive is set in a client Request then server Response includes KeepAliveTime.	6.7	M	
IMPS-CSP-SESSION-S-008	The server can choose any timer value as KeepAliveTime The client MUST obey that. If the client has specified an infinite time to live in by not submitting the Time-to-Live element in LoginRequest the server can nevertheless specify a finite time using the KeepAliveTime element in LoginResponse.	6.4.6	M	
IMPS-CSP-SESSION-S-009	If the client logs in with binary message format, the client and server sends all primitives using binary message format throughout the session.	5.1	M	
IMPS-CSP-SESSION-S-010	The server accepts the protocol version used by the client in login transaction.	5.1	O	

B.4 Transaction Requirements

B.4.1 Clients

Item	Function	Reference	Status	Requirement
IMPS-CSP-TRACT-C-001	For each transaction pair the originating and resulting Transaction-IDs are equal.	5.4	M	
IMPS-CSP-TRACT-C-002	The same Transaction-ID is not used more than once in a session.	5.4	M	
IMPS-CSP-TRACT-C-003	When the corresponding response primitive does not contain a Result element, and an error occurs when processing the request, a Status primitive is returned instead of the corresponding response primitive.	5.4	M	
IMPS-CSP-TRACT-C-004	When the corresponding response primitive contains a Result element, and an error occurs when processing the request, either the corresponding response primitive or a Status primitive is returned. In either case, the response primitive indicates the error that occurred.	5.4	M	
IMPS-CSP-TRACT-C-005	When the response primitive to a transaction is replaced with Status primitive the response value indicates unsuccessful operation.	5.4	M	
IMPS-CSP-TRACT-C-006	All mandatory information elements are present in the primitives.	5.4	M	
IMPS-CSP-TRACT-C-007	All conditional information elements are present or absent according to the relevant SCR.	5.4	M	
IMPS-CSP-TRACT-C-008	If a transaction is completely successful, the Result element indicates successful completion, and detailed results are not included in the Result element.	6.1	M	
IMPS-CSP-TRACT-C-009	If a transaction is partially successful, the Result element does not indicate successful completion.	6.1	M	
IMPS-CSP-TRACT-C-010	If a transaction is partially successful, the details of the successfully completed transaction parts are included.	6.1	O	
IMPS-CSP-TRACT-C-011	If a transaction is partially successful, the details of the not completed transaction parts are included.	6.1	M	

B.4.2 Servers

Item	Function	Reference	Status	Requirement
IMPS-CSP-TRACT-S-001	For each transaction pair the originating and resulting Transaction-IDs are equal.	5.4	M	
IMPS-CSP-TRACT-S-002	The same Transaction-ID is not used more than once in a session.	5.4	M	
IMPS-CSP-TRACT-S-003	When the corresponding response primitive does not contain a Result element, and an error occurs when processing the request, a Status primitive is returned instead of the corresponding response primitive.	5.4	M	
IMPS-CSP-TRACT-S-004	When the corresponding response primitive contains a Result element, and an error occurs when processing the request, either the corresponding response primitive or a Status primitive is returned. In either case, the response primitive indicates the error that occurred.	5.4	M	
IMPS-CSP-TRACT-S-005	When the response primitive to a transaction is replaced with Status primitive the response value indicates unsuccessful operation.	5.4	M	
IMPS-CSP-TRACT-S-006	All mandatory information elements are present in the primitives.	5.4	M	
IMPS-CSP-TRACT-S-007	All conditional information elements are present or absent according to the relevant SCR.	5.4	M	
IMPS-CSP-TRACT-S-008	If a transaction is completely successful, the Result element indicates successful completion, and detailed results are not included in the Result element.	6.1	M	
IMPS-CSP-TRACT-S-009	If a transaction is partially successful, the Result element does not indicate successful completion.	6.1	M	
IMPS-CSP-TRACT-S-010	If a transaction is partially successful, the details of the successfully completed transaction parts are included.	6.1	O	
IMPS-CSP-TRACT-S-011	If a transaction is partially successful, the details of the not completed transaction parts are included.	6.1	M	

B.5 Service Access Point (SAP) Requirements

B.5.1 Functional requirements

B.5.1.1 Clients

Item	Function	Reference	Status	Conditional Requirement
IMPS-CSP-SAP-C-001	Support for session re-establishment	5.1	O	IMPS-CSP-SERV-C-001
IMPS-CSP-SAP-C-002	Support for Status primitive	6	M	IMPS-CSP-SERV-C-001
IMPS-CSP-SAP-C-003	Support for Communication Initiation Request and PollingRequest primitives	6	O	[CSP Trans] IMPS-CSP-TRANSP-C-002
IMPS-CSP-SAP-C-004	Support for version discovery transaction	6	O	
IMPS-CSP-SAP-C-005	Support for multiple concurrent sessions	6	O	
IMPS-CSP-SAP-C-006	Support for 2-way Login transaction	6	O	IMPS-CSP-SERV-C-001
IMPS-CSP-SAP-C-007	Support for 4-way Login transaction	6	O	IMPS-CSP-SERV-C-001
IMPS-CSP-SAP-C-008	Support for Auto-Registration	6	O	
IMPS-CSP-SAP-C-009	Support for Logout transaction originating from client	6	M	IMPS-CSP-SERV-C-001
IMPS-CSP-SAP-C-010	Support for Server originated disconnect	6	M	IMPS-CSP-SERV-C-001
IMPS-CSP-SAP-C-011	Support for Keep-Alive transaction	6	M	IMPS-CSP-SERV-C-001
IMPS-CSP-SAP-C-012	Support for Get Service Provider Info transaction	6	O	IMPS-CSP-SERV-C-001
IMPS-CSP-SAP-C-013	Support for Service negotiation transaction	6	M	IMPS-CSP-SERV-C-001
IMPS-CSP-SAP-C-014	Support for Client Capability negotiation transaction over SMS transport	6	O	IMPS-CSP-SERV-C-001
IMPS-CSP-SAP-C-015	Support for Client Capability negotiation transaction if any other transport/syntax than Plain Text Syntax over SMS transport is used	6	M	IMPS-CSP-SERV-C-001
IMPS-CSP-SAP-C-016	Support for segmentation mechanism	6	O	IMPS-CSP-SERV-C-001
IMPS-CSP-SAP-C-017	Support for System Message transactions	7	M	IMPS-CSP-SERV-C-001
IMPS-CSP-SAP-C-018	Support for General Notification transactions	7	M	IMPS-CSP-SERV-C-001
IMPS-CSP-SAP-C-019	Support for retrieving public profile	7	M	IMPS-CSP-SERV-C-001
IMPS-CSP-SAP-C-020	Support for clearing/updating the public profile	7	M	IMPS-CSP-SERV-C-001
IMPS-CSP-SAP-C-021	Support for delivery of Friendly Name with UserIDs	7	M	IMPS-CSP-SERV-C-001
IMPS-CSP-SAP-C-022	Support for public profile extension fields	7	O	IMPS-CSP-SERV-C-001

Item	Function	Reference	Status	Conditional Requirement
IMPS-CSP-SAP-C-023	Support for searching based on various private profile properties	7	O	IMPS-CSP-SERV-C-001 AND IMPS-CSP-SAP-C-026
IMPS-CSP-SAP-C-024	Support for searching based on various public profile properties	7	O	IMPS-CSP-SERV-C-001 AND IMPS-CSP-SAP-C-026
IMPS-CSP-SAP-C-025	Support for searching based on various group properties	7	O	IMPS-CSP-SERV-C-001 AND IMPS-CSP-SAP-C-026
IMPS-CSP-SAP-C-026	Support for stop search transaction	7	O	IMPS-CSP-SERV-C-001 AND (IMPS-CSP-SAP-C-023 OR IMPS-CSP-SAP-C-024 OR IMPS-CSP-SAP-C-025)
IMPS-CSP-SAP-C-027	Support for invitation transaction	7	O	IMPS-CSP-SERV-C-001
IMPS-CSP-SAP-C-028	Support for cancel invitation transaction	7	O	IMPS-CSP-SERV-C-001
IMPS-CSP-SAP-C-029	Support for Get Map transaction	7	O	IMPS-CSP-SERV-C-001
IMPS-CSP-SAP-C-030	Support for verify ID transaction	7	O	IMPS-CSP-SERV-C-001
IMPS-CSP-SAP-C-031	Support for advanced search	7	O	IMPS-CSP-SRCH-C-001 AND IMPS-CSP-SRCH-C-002 AND IMPS-CSP-SERV-C-001 AND (IMPS-CSP-SAP-C-023 OR IMPS-CSP-SAP-C-024 OR IMPS-CSP-SAP-C-025)

B.5.1.2 Servers

Item	Function	Reference	Status	Requirement
IMPS-CSP-SAP-S-001	Support for session re-establishment	5.1	M	IMPS-CSP-SERV-S-001
IMPS-CSP-SAP-S-002	Support for Status primitive	6	M	IMPS-CSP-SERV-S-001
IMPS-CSP-SAP-S-003	Support for Communication Initiation Request and PollingRequest primitives	6	O	IMPS-CSP-SERV-S-001 AND [CSP Trans] IMPS-CSP-TRANSP-S-002
IMPS-CSP-SAP-S-004	Support for version discovery transaction	6	M	IMPS-CSP-SERV-S-001
IMPS-CSP-SAP-S-005	Support for multiple concurrent sessions	6	M	IMPS-CSP-SERV-S-001
IMPS-CSP-SAP-S-006	Support for 2-way Login transaction	6	M	IMPS-CSP-SERV-S-001
IMPS-CSP-SAP-S-007	Support for 4-way Login transaction	6	M	IMPS-CSP-SERV-S-001
IMPS-CSP-SAP-S-008	Support for Auto-Registration	6	O	IMPS-CSP-SERV-S-001
IMPS-CSP-SAP-S-009	Support for Logout transaction originating from client	6	M	IMPS-CSP-SERV-S-001
IMPS-CSP-SAP-S-010	Support for Server originated disconnect	6	M	IMPS-CSP-SERV-S-001
IMPS-CSP-SAP-S-011	Support for Keep-Alive transaction	6	M	IMPS-CSP-SERV-S-001
IMPS-CSP-SAP-S-012	Support for Get Service Provider Info transaction	6	O	IMPS-CSP-SERV-S-001

Item	Function	Reference	Status	Requirement
IMPS-CSP-SAP-S-013	Support for Service negotiation transaction	6	M	IMPS-CSP-SERV-S-001
IMPS-CSP-SAP-S-014	Support for Client Capability negotiation transaction over SMS transport	6	M	IMPS-CSP-SERV-S-001 AND IMPS-CSP-CCAPAB-S-002 AND IMPS-CSP-CCAPAB-S-003
IMPS-CSP-SAP-S-015	Support for Client Capability negotiation transaction if any other transport/syntax than Plain Text Syntax over SMS transport is used	6	M	IMPS-CSP-SERV-S-001 AND IMPS-CSP-CCAPAB-S-002 AND IMPS-CSP-CCAPAB-S-003
IMPS-CSP-SAP-S-016	Support for segmentation mechanism	6	O	IMPS-CSP-SERV-S-001
IMPS-CSP-SAP-S-017	Support for System Message transactions	7	M	IMPS-CSP-SERV-S-001
IMPS-CSP-SAP-S-018	Support for General Notification transactions	7	M	IMPS-CSP-SERV-S-001
IMPS-CSP-SAP-S-019	Support for retrieving public profile	7	M	IMPS-CSP-SERV-S-001
IMPS-CSP-SAP-S-020	Support for clearing/updating the public profile	7	M	IMPS-CSP-SERV-S-001
IMPS-CSP-SAP-S-021	Support for delivery of Friendly Name with UserIDs	7	M	IMPS-CSP-SERV-S-001
IMPS-CSP-SAP-S-022	Support for public profile extension fields	7	O	IMPS-CSP-SERV-S-001
IMPS-CSP-SAP-S-023	Support for searching based on various private profile properties	7	O	IMPS-CSP-SERV-S-001 AND IMPS-CSP-SAP-S-026
IMPS-CSP-SAP-S-024	Support for searching based on various public profile properties	7	O	IMPS-CSP-SERV-S-001 AND IMPS-CSP-SAP-S-026
IMPS-CSP-SAP-S-025	Support for searching based on various group properties	7	O	IMPS-CSP-SERV-S-001 AND IMPS-CSP-SAP-S-026
IMPS-CSP-SAP-S-026	Support for stop search transaction	7	O	IMPS-CSP-SERV-S-001 AND (IMPS-CSP-SAP-S-023 OR IMPS-CSP-SAP-S-024 OR IMPS-CSP-SAP-S-025)
IMPS-CSP-SAP-S-027	Support for invitation transaction	7	O	IMPS-CSP-SERV-S-001
IMPS-CSP-SAP-S-028	Support for cancel invitation transaction	7	O	IMPS-CSP-SERV-S-001
IMPS-CSP-SAP-S-029	Support for Get Map transaction	7	O	IMPS-CSP-SERV-S-001
IMPS-CSP-SAP-S-030	Support for verify ID transaction	7	O	IMPS-CSP-SERV-S-001
IMPS-CSP-SAP-S-031	Support for advanced search	7	O	IMPS-CSP-SERV-S-001 AND IMPS-CSP-SAP-S-026 AND (IMPS-CSP-SAP-S-023 OR IMPS-CSP-SAP-S-024 OR IMPS-CSP-SAP-S-025)

B.5.2 Login transaction requirements

B.5.2.1 Clients

Item	Function	Reference	Status	Requirement
IMPS-CSP-LOGIN-C-001	If the Client-Capability-Request element in the LoginResponse primitive indicates 'F', the client initiates client capability negotiation after the successful login.	6.4	O	(IMPS-CSP-SAP-C-006 OR IMPS-CSP-SAP-C-007) AND (IMPS-CSP-SAP-C-014 OR IMPS-CSP-SAP-C-015)

B.5.2.2 Servers

There are not requirements for the server.

B.5.3 Logout transaction requirements

B.5.3.1 Clients

There are no requirements for the client.

B.5.3.2 Servers

Item	Function	Reference	Status	Requirement
IMPS-CSP-LOGOUT-S-001	If the time-to-live timer of the session is exceeded the server disconnects the client by sending a Disconnect primitive to the client.	6.6.1	O	IMPS-CSP-SAP-S-010

B.5.4 Keep-alive transaction requirements

B.5.4.1 Clients

There are not requirements for the client.

B.5.4.2 Servers

Item	Function	Reference	Status	Requirement
IMPS-CSP-NOOP-S-001	If the Time-To-Live element is present in the request primitive, the server changes the KeepAliveTime to the indicated value.	6.7.1	O	IMPS-CSP-SAP-S-011

B.5.5 Get Service Provider Info transaction requirements

B.5.5.1 Clients

There are no requirements for the client.

B.5.5.2 Servers

Item	Function	Reference	Status	Requirement
IMPS-CSP-GETSPI-S-001	The SAP supports GetSPInfoRequest as outband request	6.8.1	O	IMPS-CSP-SAP-S-012

B.5.6 Service negotiation transaction requirements

B.5.6.1 Clients

Item	Function	Reference	Status	Requirement
IMPS-CSP-SNEG-C-001	Support for repeating Service negotiation during a session.	6.9.1	O	IMPS-CSP-SAP-C-013

B.5.6.2 Servers

There are no requirements for the server.

B.5.7 Client capability negotiation transaction requirements

B.5.7.1 Clients

Item	Function	Reference	Status	Requirement
IMPS-CSP-CCAPAB-C-001	Support for repeating Client capability negotiation during a session.	6.9.1	O	IMPS-CSP-SAP-C-014 OR IMPS-CSP-SAP-C-015
IMPS-CSP-CCAPAB-C-002	If the client indicates the use of standalone UDP/IP binding for CIR, the client provides the UDP port in the request.	6.9.1	O	IMPS-CSP-SAP-C-015

B.5.7.2 Servers

Item	Function	Reference	Status	Requirement
IMPS-CSP-CCAPAB-S-001	The server maintains the client capabilities between sessions.	6.9.1	O	IMPS-CSP-SAP-S-014 OR IMPS-CSP-SAP-S-015
IMPS-CSP-CCAPAB-S-002	The server maintains the MSIDN persistently between sessions when SupportedOfflineBearer was agreed	6.9.1	M	(IMPS-CSP-SAP-S-014 OR IMPS-CSP-SAP-S-015) AND [CSP Trans] IMPS-CSP-TRANSP-S-017
IMPS-CSP-CCAPAB-S-003	The server maintains the OfflineETEMHandling value persistently between sessions when OfflineETEMHandling was agreed.	6.9.1	M	IMPS-CSP-SAP-S-014 OR IMPS-CSP-SAP-S-015

B.5.8 General search transaction requirements

B.5.8.1 Clients

Item	Function	Reference	Status	Requirement
IMPS-CSP-SRCH-C-001	The client sends StopSearchRequest primitive to the server when the search is needed no more.	7.4.1	O	IMPS-CSP-SAP-C-026
IMPS-CSP-SRCH-C-002	The client supports including more than one Search-Pair-Lists in the 1st SearchRequest.	7.4.1	O	IMPS-CSP-SAP-C-023 OR IMPS-CSP-SAP-C-024 OR IMPS-CSP-SAP-C-025

B.5.8.2 Servers

There are no requirements for the server.

B.5.9 Invitation transaction requirements

B.5.9.1 Clients

Item	Function	Reference	Status	Requirement
IMPS-CSP-INVIT-C-001	The client accepts or declines a received invitation.	7.5.1	O	IMPS-CSP-SAP-C-027

B.5.9.2 Servers

There are no requirements for the server.

B.6 Presence Service Element (PRSE) Requirements

B.6.1 Functional requirements

B.6.1.1 Clients

Item	Function	Reference	Status	Requirement
IMPS-CSP-PRSE-C-001	Support for get list of contact lists (IDs) transaction	8	O	IMPS-CSP-SERV-C-003
IMPS-CSP-PRSE-C-002	Support for create contact list transaction	8	O	IMPS-CSP-SERV-C-003
IMPS-CSP-PRSE-C-003	Support for delete contact list transaction	8	O	IMPS-CSP-SERV-C-003
IMPS-CSP-PRSE-C-004	Support for manage contact list transaction	8	O	IMPS-CSP-SERV-C-003
IMPS-CSP-PRSE-C-005	Support for create attribute list transaction	8	O	IMPS-CSP-SERV-C-003
IMPS-CSP-PRSE-C-006	Support for delete attribute list transaction	8	O	IMPS-CSP-SERV-C-003
IMPS-CSP-PRSE-C-007	Support for get attribute list transaction	8	O	IMPS-CSP-SERV-C-003
IMPS-CSP-PRSE-C-008	Support for subscribe presence transaction	8	O	IMPS-CSP-SERV-C-003
IMPS-CSP-PRSE-C-009	Support for automatic subscribe presence subscription	8	O	IMPS-CSP-SERV-C-003
IMPS-CSP-PRSE-C-010	Support for unsubscribe presence transaction	8	O	IMPS-CSP-SERV-C-003
IMPS-CSP-PRSE-C-011	Support for get watcher list transaction	8	O	IMPS-CSP-SERV-C-003
IMPS-CSP-PRSE-C-012	Support for presence notification transaction	8	O	IMPS-CSP-SERV-C-003
IMPS-CSP-PRSE-C-013	Support for get presence transaction	8	O	IMPS-CSP-SERV-C-003
IMPS-CSP-PRSE-C-014	Support for update presence transaction	8	O	IMPS-CSP-SERV-C-003

B.6.1.2 Servers

Item	Function	Reference	Status	Requirement
IMPS-CSP-PRSE-S-001	Support for get list of contact lists (IDs) transaction	8	O	IMPS-CSP-SERV-S-003
IMPS-CSP-PRSE-S-002	Support for create contact list transaction	8	O	IMPS-CSP-SERV-S-003
IMPS-CSP-PRSE-S-003	Support for delete contact list transaction	8	O	IMPS-CSP-SERV-S-003
IMPS-CSP-PRSE-S-004	Support for manage contact list transaction	8	O	IMPS-CSP-SERV-S-003
IMPS-CSP-PRSE-S-005	Support for create attribute list transaction	8	O	IMPS-CSP-SERV-S-003
IMPS-CSP-PRSE-S-006	Support for delete attribute list transaction	8	O	IMPS-CSP-SERV-S-003
IMPS-CSP-PRSE-S-007	Support for get attribute list transaction	8	O	IMPS-CSP-SERV-S-003
IMPS-CSP-PRSE-S-008	Support for subscribe presence transaction	8	O	IMPS-CSP-SERV-S-003
IMPS-CSP-PRSE-S-009	Support for automatic subscribe presence subscription	8	O	IMPS-CSP-SERV-S-003
IMPS-CSP-PRSE-S-010	Support for unsubscribe presence transaction	8	O	IMPS-CSP-SERV-S-003
IMPS-CSP-PRSE-S-011	Support for get watcher list transaction	8	O	IMPS-CSP-SERV-S-003
IMPS-CSP-PRSE-S-012	Support for presence notification transaction	8	O	IMPS-CSP-SERV-S-003
IMPS-CSP-PRSE-S-013	Support for get presence transaction	8	O	IMPS-CSP-SERV-S-003
IMPS-CSP-PRSE-S-014	Support for update presence transaction	8	O	IMPS-CSP-SERV-S-003

B.6.2 Create contact list transaction requirements

B.6.2.1 Clients

There are no requirements for the client.

B.6.2.2 Servers

Item	Function	Reference	Status	Requirement
IMPS-CSP-CCLI-S-001	Support for creating the contact list when the contact list does not exist on the server, and the server is unable to add all users in User-Nick-List or is unable to apply all property changes specified in Contact-List-Props.	8.1.2	O	IMPS-CSP-PRSE-S-002

B.6.3 Manage contact list transaction requirements

B.6.3.1 Clients

Item	Function	Reference	Status	Requirement
IMPS-CSP-MCLS-C-001	Support for providing Nickname with User-IDs.	8.1.2	O	IMPS-CSP-PRSE-C-004

B.6.3.2 Servers

There are no requirements for the server.

B.6.4 Get watcher list transaction requirements

B.6.4.1 Clients

There are no requirements for the client.

B.6.4.2 Servers

Item	Function	Reference	Status	Requirement
IMPS-CSP-GETWL-S-001	The server maintains the watcher history period and adds it in the GetWatcherListResponse.	8.3.1	O	IMPS-CSP-PRSE-S-011

B.7 Instant Messaging Service Element (IMSE) Requirements

B.7.1 Functional requirements

B.7.1.1 Clients

Item	Function	Reference	Status	Requirement
IMPS-CSP-IMSE-C-001	Support for setting delivery method	8	O	IMPS-CSP-SERV-C-002
IMPS-CSP-IMSE-C-002	Support for send message transaction	8	O	IMPS-CSP-SERV-C-002
IMPS-CSP-IMSE-C-003	Support for get list of messages transaction	8	O	IMPS-CSP-SERV-C-002
IMPS-CSP-IMSE-C-004	Support for reject message transaction	8	O	IMPS-CSP-SERV-C-002
IMPS-CSP-IMSE-C-005	Message-Info element requirements	8	O	IMPS-CSP-SERV-C-002
IMPS-CSP-IMSE-C-006	Support for new message transaction	8	O	IMPS-CSP-SERV-C-002
IMPS-CSP-IMSE-C-007	Support for message notification transaction	8	O	IMPS-CSP-SERV-C-002 AND IMPS-CSP-IMSE-C-008
IMPS-CSP-IMSE-C-008	Support for get message transaction	8	O	IMPS-CSP-SERV-C-002 AND IMPS-CSP-IMSE-C-007
IMPS-CSP-IMSE-C-009	Support for get message or new message transaction	8	O	IMPS-CSP-SERV-C-002
IMPS-CSP-IMSE-C-010	Support for delivery status report transaction	8	O	IMPS-CSP-SERV-C-002
IMPS-CSP-IMSE-C-011	Support for forward message transaction	8	O	IMPS-CSP-SERV-C-002
IMPS-CSP-IMSE-C-012	Support for get list of blocked entities transaction	8	O	IMPS-CSP-SERV-C-002
IMPS-CSP-IMSE-C-013	Support for block entity transaction	8	O	IMPS-CSP-SERV-C-002

B.7.1.2 Servers

Item	Function	Service Reference	Status	Requirement
IMPS-CSP-IMSE-S-001	Support for setting delivery method	8	O	IMPS-CSP-SERV-S-002
IMPS-CSP-IMSE-S-002	Support for send message transaction	8	O	IMPS-CSP-SERV-S-002
IMPS-CSP-IMSE-S-003	Support for get list of messages transaction	8	O	IMPS-CSP-SERV-S-002
IMPS-CSP-IMSE-S-004	Support for reject message transaction	8	O	IMPS-CSP-SERV-S-002
IMPS-CSP-IMSE-S-005	Message-Info element requirements	8	O	IMPS-CSP-SERV-S-002
IMPS-CSP-IMSE-S-006	Support for new message transaction	8	O	IMPS-CSP-SERV-S-002
IMPS-CSP-IMSE-S-007	Support for message notification transaction	8	O	IMPS-CSP-SERV-S-002
IMPS-CSP-IMSE-S-008	Support for get message transaction	8	O	IMPS-CSP-SERV-S-002
IMPS-CSP-IMSE-S-009	Support for delivery status report transaction	8	O	IMPS-CSP-SERV-S-002
IMPS-CSP-IMSE-S-010	Support for forward message transaction	8	O	IMPS-CSP-SERV-S-002
IMPS-CSP-IMSE-S-011	Support for get list of blocked entities transaction	8	O	IMPS-CSP-SERV-S-002
IMPS-CSP-IMSE-S-012	Support for block entity transaction	8	O	IMPS-CSP-SERV-S-002

B.7.2 Set delivery method transaction requirements

B.7.2.1 Clients

There are no requirements for the client.

B.7.2.2 Servers

Item	Function	Reference	Status	Requirement
IMPS-CSP-SETD-S-001	Support for applying the requested delivery method to the requested group only – when the Group-ID is present in the request.	9.1.2	O	IMPS-CSP-IMSE-S-1

B.7.3 Send message transaction requirements

B.7.3.1 Clients

Item	Function	Reference	Status	Requirement
IMPS-CSP-SENDM-C-001	Support for identifying the sender in the Message-Info structure.	9.1.1	O	IMPS-CSP-IMSE-C-002
IMPS-CSP-SENDM-C-002	Support for identifying the sending client using the Client-ID.	9.1.1	O	IMPS-CSP-IMSE-C-002

B.7.3.2 Servers

There are no requirements for the server.

B.7.4 Get list of messages transaction requirements

B.7.4.1 Clients

There are no requirements for the client.

B.7.4.2 Servers

Item	Function	Reference	Status	Requirement
IMPS-CSP-GETLM-S-001	Support for sending the Message-Info of those instant messages that has been sent to the specified group - when the Group-ID element is present in the request.	9.1.3	O	IMPS-CSP-IMSE-C-003
IMPS-CSP-GETLM-S-002	Support for sending the Message-Info of non-delivered instant messages that have been sent from any user or group - when the Group-ID element is not present in the request.	9.1.3	O	IMPS-CSP-IMSE-C-003

B.7.5 NewMessage primitive requirements

B.7.5.1 Clients

There are no requirements for the client.

B.7.5.2 Servers

Item	Function	Reference	Status	Requirement
IMPS-CSP-NEWM-S-001	Support for identifying the sending client using Client-ID.	9.1.5	O	IMPS-CSP-IMSE-C-006
IMPS-CSP-NEWM-S-002	Support for identifying the instant messages in the Message-Info structure using Message-URI.	9.1.5	O	IMPS-CSP-IMSE-C-006

B.7.6 Get message transaction requirements

B.7.6.1 Clients

There are no requirements for the client.

B.7.6.2 Servers

Item	Function	Reference	Status	Requirement
IMPS-CSP-GETM-S-001	Support for identifying the instant messages in the Message-Info structure using Message-URI.	9.1.8	O	IMPS-CSP-IMSE-C-008

B.7.7 Block entities transaction requirements

B.7.7.1 Clients

Item	Function	Reference	Status	Requirement
IMPS-CSP-BLENT-C-001	User-IDs, Screen-Names, Group-IDs and ContactList-IDs are supported.	9.3.1	O	IMPS-CSP-IMSE-S-012

B.7.7.2 Servers

There are no requirements for the server.

B.8 Group Service Element (GRSE) Requirements

B.8.1 Functional requirements

B.8.1.1 Clients

Item	Function	Reference	Status	Requirement
IMPS-CSP-GRSE-C-001	Support for group creation transaction	10.1	O	IMPS-CSP-SERV-C-004
IMPS-CSP-GRSE-C-002	Support for group deletion transaction	10.1	O	IMPS-CSP-SERV-C-004
IMPS-CSP-GRSE-C-003	Support for get group properties transaction	10.1	O	IMPS-CSP-SERV-C-004
IMPS-CSP-GRSE-C-004	Support for set group properties transaction	10.1	O	IMPS-CSP-SERV-C-004
IMPS-CSP-GRSE-C-005	Support for get group members transaction	10.1	O	IMPS-CSP-SERV-C-004
IMPS-CSP-GRSE-C-006	Support for add group members transaction	10.1	O	IMPS-CSP-SERV-C-004
IMPS-CSP-GRSE-C-007	Support for remove group members transaction	10.1	O	IMPS-CSP-SERV-C-004
IMPS-CSP-GRSE-C-008	Support for member access rights transaction	10.1	O	IMPS-CSP-SERV-C-004
IMPS-CSP-GRSE-C-009	Support for subscribe group notice transaction	10.1	O	IMPS-CSP-SERV-C-004
IMPS-CSP-GRSE-C-010	Support for group change notification transaction	10.1	O	IMPS-CSP-SERV-C-004
IMPS-CSP-GRSE-C-011	Support for join group transaction	10.1	O	IMPS-CSP-SERV-C-004
IMPS-CSP-GRSE-C-012	Support for leave group transaction	10.1	O	IMPS-CSP-SERV-C-004
IMPS-CSP-GRSE-C-013	Support for reject user(s) from group transaction	10.1	O	IMPS-CSP-SERV-C-004
IMPS-CSP-GRSE-C-014	Support for get joined users transaction	10.1	O	IMPS-CSP-SERV-C-004
IMPS-CSP-GRSE-C-015	Support for extended group properties.	10.1.6	O	IMPS-CSP-SERV-C-004
IMPS-CSP-GRSE-C-016	Support for extending one-to-one IM conversation to many-to-many IM conversation.	9.1.11	O	IMPS-CSP-SERV-C-002 AND IMPS-CSP-SERV-C-004 AND IMPS-CSP-SAP-C-027 AND IMPS-CSP-SAP-C-028 AND IMPS-CSP-GRSE-C-010

B.8.1.2 Servers

Item	Function	Reference	Status	Requirement
IMPS-CSP-GRSE-S-001	Support for group creation transaction	10.1	O	IMPS-CSP-SERV-S-004
IMPS-CSP-GRSE-S-002	Support for group deletion transaction	10.1	O	IMPS-CSP-SERV-S-004
IMPS-CSP-GRSE-S-003	Support for get group properties transaction	10.1	O	IMPS-CSP-SERV-S-004
IMPS-CSP-GRSE-S-004	Support for set group properties transaction	10.1	O	IMPS-CSP-SERV-S-004
IMPS-CSP-GRSE-S-005	Support for get group members transaction	10.1	O	IMPS-CSP-SERV-S-004
IMPS-CSP-GRSE-S-006	Support for add group members transaction	10.1	O	IMPS-CSP-SERV-S-004
IMPS-CSP-GRSE-S-007	Support for remove group members transaction	10.1	O	IMPS-CSP-SERV-S-004
IMPS-CSP-GRSE-S-008	Support for member access rights transaction	10.1	O	IMPS-CSP-SERV-S-004
IMPS-CSP-GRSE-S-009	Support for subscribe group notice transaction	10.1	O	IMPS-CSP-SERV-S-004
IMPS-CSP-GRSE-S-010	Support for group change notification transaction	10.1	O	IMPS-CSP-SERV-S-004
IMPS-CSP-GRSE-S-011	Support for join group transaction	10.1	O	IMPS-CSP-SERV-S-004 AND IMPS-CSP-JOING-S-001 AND IMPS-CSP-JOING-S-002
IMPS-CSP-GRSE-S-012	Support for leave group transaction	10.1	O	IMPS-CSP-SERV-S-004
IMPS-CSP-GRSE-S-013	Support for reject user(s) from group transaction	10.1	O	IMPS-CSP-SERV-S-004
IMPS-CSP-GRSE-S-014	Support for get joined users transaction	10.1	O	IMPS-CSP-SERV-S-004
IMPS-CSP-GRSE-S-015	Support for extended group properties.	10.1.6	O	IMPS-CSP-SERV-S-004
IMPS-CSP-GRSE-S-016	Support for extending one-to-one IM conversation to many-to-many IM conversation.	9.1.11	O	IMPS-CSP-SERV-S-002 AND IMPS-CSP-SERV-S-004 AND IMPS-CSP-SAP-S-027 AND IMPS-CSP-SAP-S-028 AND IMPS-CSP-GRSE-S-010

B.8.2 Join group transaction requirements

B.8.2.1 Clients

There are no requirements for the client.

B.8.2.2 Servers

Item	Function	Reference	Status	Requirement
IMPS-CSP-JOING-S-001	Support for using the screen name that the user requested.	10.4	O	IMPS-CSP-GRSE-S-011
IMPS-CSP-JOING-S-002	Support for assigning a screen name for the user automatically when it was not included in the request.	10.4	O	IMPS-CSP-GRSE-S-011

Item	Function	Reference	Status	Requirement
IMPS-CSP-JOING-S-003	Support for delivering the Welcome-Note in the JoinGroupResponse primitive.	10.4	O	IMPS-CSP-GRSE-S-011