SSP – Server to Server Protocol

Semantics Document

V1.0

WV Internal Tracking Number: WV-013

TABLE OF CONTENTS

Notice

Implementation of all or part of any Specification may require licenses under third party intellectual property rights, including without limitation, patent rights (such a third party may or may not be a Supporter). The Sponsors of the Specification are not responsible and shall not be held responsible in any manner for identifying or failing to identify any or all such third party intellectual property rights.

THIS DOCUMENT AND THE INFORMATION CONTAINED HEREIN ARE PROVIDED ON AN "AS IS" BASIS WITHOUT WARRANTY OF ANY KIND AND ERICSSON, MOTOROLA and NOKIA DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL ERICSSON, MOTOROLA or NOKIA BE LIABLE TO ANY PARTY FOR ANY LOSS OF PROFITS, LOSS OF BUSINESS, LOSS OF USE OF DATA, INTERRUPTION OF BUSINESS, OR FOR DIRECT, INDIRECT, SPECIAL OR EXEMPLARY, INCIDENTAL, PUNITIVE OR CONSEQUENTIAL DAMAGES OF ANY KIND IN CONNECTION WITH THIS DOCUMENT OR THE INFORMATION CONTAINED HEREIN, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH LOSS OR DAMAGE. The above notice and this paragraph must be included on all copies of this document that are made.

1. Document Information

1.1. Definitions

The following definitions define terms specific to the Wireless Village and general terms that may have some special context within the documentation. These definitions are provided to enhance the use of this documentation.

Home Domain refers to the home IMPS system, where the user subscribes to, and is authenticated and authorized to use IMPS services

Primary Service Element refers to a Service Element of an IMPS service for a client. PSE may be in the Home Domain of the client, or in the other domain.

Complementary Service refers to the situation in which the Primary Service Element (PSE) is NOT in the Home Domain. Instead, the PSE is in another domain.

Provider Server the WV server, which provides the services for the Requestor Server in the frame of a session after the successful service agreement

Requestor Server the WV server, which requests the services from the Provider Server in the frame of a session after the successful service agreement

Service Request it is initiated from the Requestor Server to the Provider Server

Service Notification it is initiated from the Provider Server to the Requestor Server

The terms MAY, SHOULD, MUST are consistent with the definition in RFC 2119.

1.2. Abbreviations

For the purposes of this specification the following abbreviations apply.

ARPA	Advanced Research Projects Agency An agency of the United States Department of Defense, ARPA underwrote the development of the Internet beginning in 1969. A precursor to IETF.
DTD	Document Type Definition
HTTP	Hypertext Transfer Protocol
IANA	Internet Assigned Number Authority
IETF	Internet Engineering Task Force A society of engineers and developers dedicated to designing and

	advancing standards for internet use.
WAP	Wireless Application Protocol A specification for a set of communication protocols to standardize the way that wireless devices, such as cellular telephones and radio transceivers, can be used for Internet access

1.3. References

WVARCH	"Wireless Village System Architecture Model v1.0", February 2002
WVFEAT	"Wireless Village Features and Functions v1.0", February 2002
WVPA	"Wireless Village Presence Attributes v1.0", February 2002
WVCSPS	"Wireless Village Client-Server Protocol – Session and Transactions v1.0", February 2002
WVSSPT	"Wireless Village Server-Server Protocol – Transport Binding v1.0", February 2002
WVSSPSCR	"Wireless Village Server-Server Protocol – Static Conformance Requirement v1.0", February 2002
[RFC822]	"Standard for the Format of ARPA Internet Text Messages", August 1982.
[RFC1321]	"The MD5 Message-Digest Algorithm", April 1992.
[RFC2045]	Multipurpose Internet Mail Extensions (MIME) Part one: Format of Internet Message Bodies. Section 6.8 "Base64 Content-Transfer-Encoding", November 1996
[RFC2046]	Borenstein N., and N. Freed, "MIME (Multipurpose Internet Mail Extensions) Part Two: Media Types", November 1996.
[RFC2119]	"Keywords for using RFCs to Indicate Requirements levels", March 1997
[RFC2616]	"Hypertext Transfer Protocol – HTTP/1.1", June 1999
[RFC2778]	"A Model for Presence and Instant Messaging", February 2000
[RFC2779]	"Instant Messaging / Presence Protocol Requirements", February 2000
[FIPS 180-1]	"Secure Hash Standard", April 1995
[IMPP-CPIM]	"A Common Profile for Instant Messaging (CPIM)", Internet Draft, November 2000
[E.164]	ITU-T Recommendation E.164 (05/97) The international Public Telecommunication Numbering Plan
[TS 22.121]	"Digital cellular telecommunications system (Phase 2+) (GSM); Universal Mobile Telecommunications System (UMTS); Service aspects; The Virtual Home Environment (3GPP TS 22.121 version 4.1.0 Release 4)"
[TS 23.127]	"Universal Mobile Telecommunications System (UMTS); Virtual Home Environment/Open Service Architecture (3GPP TS 23.127 version 4.2.0 Release 4)"
[WAPWAP]	"Wireless Application Protocol Architecture Specification", WAP Forum, June 2000
[WAPWSP]	"Wireless Session Protocol", WAP Forum, June 2000

[WAPWTLS] "Wireless Transport Layer Security", WAP Forum, June 2000
[WAPPush] "WAP Push Architectural Overview", WAP Forum, June 2000
[XML] "Extensible Markup Language 1.0 (Second Edition)", W3C recommendation, 6-October-2000
[UUID] Steven Miller, "DEC/HP Network Computing Architecture Remote Procedure Call Run-Time Extension Specification Version OSF TX1.0.11", July 23, 1992

2. Overview

Wireless Village (WV) Server-Server Protocol (SSP) provides the communication and interaction means between different IMPS service domains. SSP allows the WV clients to subscribe to the IMPS services provided by different service providers that are distributed across the network. SSP allows the WV clients to communicate with existing proprietary Instant Messaging networks through the Proprietary Gateway. The interoperability between different devices and service providers is achieved in a way that a user one that subscribes to Wireless Village services at Service Provider A can communicate with a user two that is a client of Service Provider B. The goal of SSP is to support the distributed interoperable complementary IMPS services across service provider domains.

2.1. SSP Interoperability Model

The term "Home Domain" is the domain where the client subscribes to, and is authenticated and authorized to use the IMPS services.

The term "Primary Service Element" (PSE) is the primary SE of an IMPS service for a client. PSE may be in the Home Domain of the client, or in a remote domain.

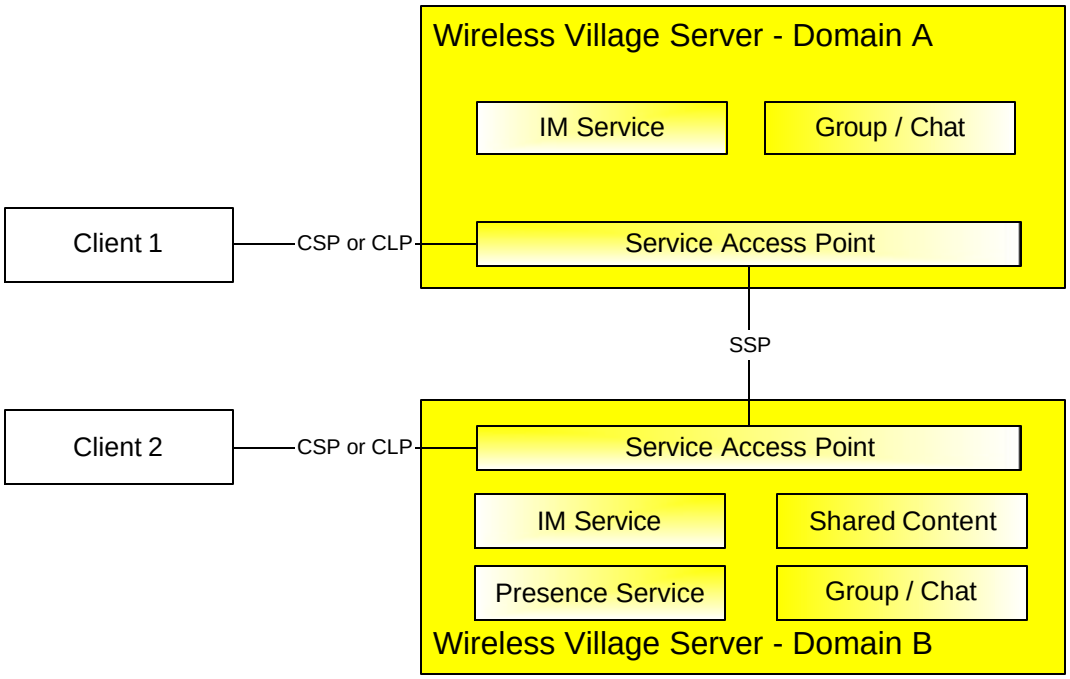

Figure 1. The SSP Minimum Interoperability Model

SSP supports server interoperability at different levels. At the lowest level, two users located at two different home domains are able to communicate with each other, as shown in Figure 1. At the highest level, SSP supports that a complete set of IMPS

services are assembled from complementary IMPS services across service provider domains, as shown in Figure 2. SSP defines the rules for the PSE to take appropriate actions to achieve the interoperability and provide distributed IMPS services.

In order for the service providers to have the flexibility to choose the appropriate level of interoperability and set up different service agreements between them, SSP mandates a minimum set of interoperable features and functions. To guarantee interoperability it is required that the two interacting servers provide the same subset of services.

In the example in Figure 1, client 1 is located in home domain A, and client 2 is located in home domain B. Domain A implements IM and Group service elements, and domain B implements the full set of Wireless Village service elements. The common subset of services is IM and Group, i.e. client 1 and client 2 are interacting across domains via the minimum set of interoperable IM and Group features and functions in SSP.

The full set of interoperability features includes the Interoperability Management and the IMPS Service Relay. The Interoperability Management includes a Security Model, Transaction Management, Session Management, Service Management and User Profile Management. The IMPS Service Relay includes Common IMPS Features, Contact List Features, Presence Features, Instant Messaging Features, Group Features and Shared Content Features.

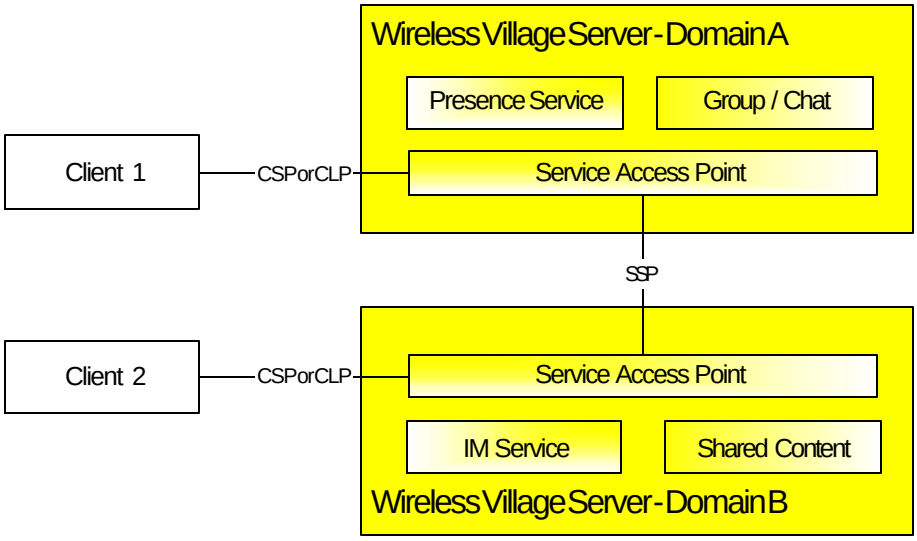

Figure 2. The SSP Full Interoperability Model

In the example in Figure 2, client 1 is located in home domain A, and Client 2 is located in home domain B. Domain A implements the presence and group service elements and domain B the IM and shared content service elements. The Wireless Village interoperability model allows client 1 and 2 to utilize the complete set of features and interact with each other via the SSP.

In SSP Interoperability, the Home Domains must have direct SSP connection if they want to interoperate with each other. However, SSP supports the routing of “Service Relay” between the Home Domain and the PSE. The route from Home Domain B to its PSE is shown in Figure 3, where the PSE domain that provides the actual service element, e.g. IM service, is at the end of the route. All intermediate domains are relaying the service request to the next hop. The intermediate nodes act the "logical" Service Provider role for each downstream domain, and act the “logical” Service Requestor role for each upstream domain.

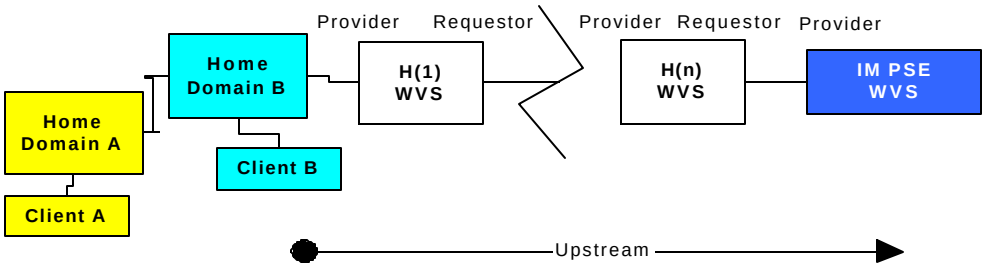

Figure 3. The SSP Service Relay

At each Wireless Village server, the Service Access Point (SAP) should maintain a Service Table that keeps track of the service agreements to appropriately relay the SSP service request on a per-service basis and forward the SSP service result on a per-domain basis. Being the “logical” Service Provider, the SAP should maintain a Session Record for each Service Requestor. Being the “logical” Service Requestor, the SAP should maintain a Transaction Record for each Service Provider. The SAP should maintain a Transaction Table to map each requested transaction from its Service Requestor to the initiated transaction to its Service Provider. The Transaction Table should be the uniquely one-one match. Therefore, the Service Relay flow and Result Forward flow at each SAP is clearly and uniquely identified by the transaction flows.

The SAP at Home Domain shall appropriately map the CSP/CLP service request from the client to the SSP service request, and/or map the SSP service result to CSP/CLP service result to the client.

2.2. SSP Interoperability Rules

In SSP Interoperability, the Home Domains must have direct SSP connection if they want to interoperate with each other. However, SSP supports the routing of “Service Relay” between the Home Domain and the PSE. The basic IOP rules are:

Rule 1: At the Home Domain, each user-initiated service request and the relayed service request from another Home Domain shall be routed / relayed from this Home Domain to its PSE for the first and primary processing. PSE is the primary and default service element to provide the user with the service.

Rule 2: If PSE needs more information from another SE in another Home Domain, but the service agreement between them does not support such information exchange, the PSE shall relay the service request to that Home Domain for further processing. Before a service request is relayed to a SE in another Home Domain, all information elements of local scope must be replaced with those of global scope. For example, a local User-ID is replaced with a global User-ID. Moreover, if the information element is a reference to a local object, it must be replaced by the actual information, e.g. a reference to a Contact-List must be replaced by a list of global User-ID's.

Rule 3: At the PSE, each PSE-initiated transaction shall be routed / relayed from the PSE back to its Home Domain, from which the PSE-initiated transaction is triggered (by the user-initiated or relayed service request). The PSE-initiated transaction shall be next relayed from the Home Domain to the destination Home Domain via the direct SSP connection between them (e.g. Figure 7 in section 2.4.4). If two Home Domains provide each other with the complementary PSE, the direct routing / relay is allowed from the complementary PSE to the destination domain (e.g. Figure 6 in section 2.4.3).

An intermediate domain shall route / relay the service request to the PSE and from the PSE based on its service agreement. A routing table is allowed in the intermediate domain. The routing table shall be offline configured based on service agreement. If the routing table is used in PSE, it shall override the routing Rule 3 (e.g. Figure 8 in section 2.4.5).

2.3. SSP Service Agreement and Routing

The exchange of messages between Wireless Village domains is normally performed in one hop over an established direct SSP connection. However, Wireless Village does support routing of messages between the Home Domain and the PSE. The SSP routing between domains is based on the SSP IOP rules and the business agreements between the domains. The business agreement must be established among all domains that are involved in the handling of SSP service relays between two end points.

After the business agreements are made between the domains, each domain shall be able to route and relay the services between the domains along the path. The routing table is created based on the business and service agreement.

In conclusion, the SSP IOP routing is defined by offline business agreement and service agreement that contains routing agreement and configuration. Each Wireless Village Server (WVS) holds a static list of direct connected neighbors. The list specifies the agreed domains that may be forwarded to one of the direct connected WVS’s.

2.4. SSP Interoperability Case Study

There are different situations in SSP interoperability. This section illustrates different interoperability models and the transaction flows based on the IOP rules described in 2.2.

2.4.1. Case 1 – Two Users are Located in different Home Domains. Each Home Domain has its own SE. Two Home Domains are Connected

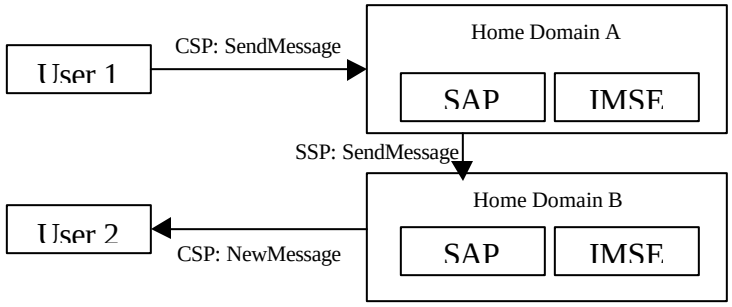

Figure 4. The SSP IOP Case One

In the example in Figure 4, client 1 is located in home domain A, and client 2 is located in home domain B. A's IM PSE is located in Domain A, and B's PSE is located in Domain B. This is the minimal interoperability case. The transaction flow of sending a message from client 1 to client2 is:

1. C1 -> DA: CSP-SendMessage
2. DA -> DB: SSP-SendMessage
3. DB -> C2, SSP-NewMessage (after checking block list etc.)

2.4.2. Case 2 – Two Users are Located in the same Home Domain

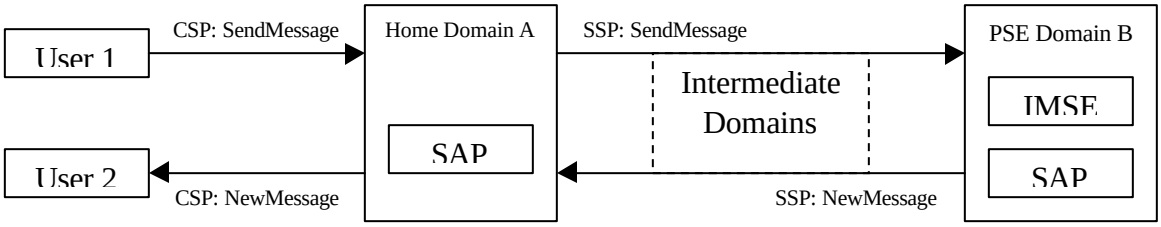

Figure 5. The SSP IOP Case Two

In the example in Figure 5, both client 1 and 2 are located in home domain A. The IM PSE is located in Domain B. Domain A and B are connected via some intermediate domains. The transaction flow of sending a message from client 1 to client2 is:

1. C1 -> DA: CSP-SendMessage
2. DA -> DB: SSP-SendMessage (through intermediate domains via routing)
3. DB -> DA, SSP-NewMessage (after checking block list etc.)
4. DA -> C2, CSP-NewMessage

If Domain A and Domain B are directly connected, there will be one SSP-SendMessage from A to B, and one SSP-NewMessage from B to A.

If Domain A and Domain B are connected through several intermediate domains, there will be several SSP-SendMessages from A to B, one for each hop. Each intermediate domain will relay the SSP-SendMessage to the next hop. There will also be several SSP-NewMessages from B to A, one for each hop. Each intermediate domain will forward the SSP-NewMessage to the next hop.

2.4.3. Case 3 – Domain A and C have Direct SSP Connection while Domain C Provides A with Complementary PSE

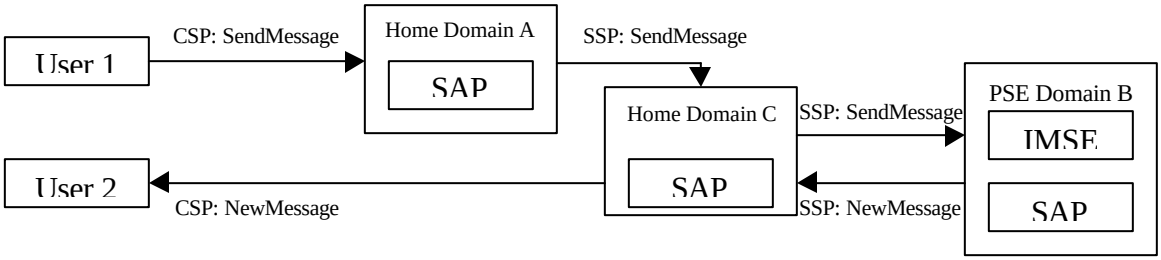

Figure 6. The SSP IOP Case Three

In the example in Figure 6, Domain A and C have direct SSP connection, and Domain C provides A with complementary IM PSE in Domain B. The transaction flow of sending a message from client 1 to client 2 is:

1. C1 -> DA: CSP-SendMessage
2. DA -> DC: SSP-SendMessage
3. DC -> DB: SSP-SendMessage (through intermediate domains via routing)
4. DB -> DC, SSP-NewMessage (after checking block list etc.)
5. DC -> C2, CSP-NewMessage

2.4.4. Case 4 – Two Users are Located in different Home Domains. Each Home Domain has its complementary PSE. Two Home Domains are Connected

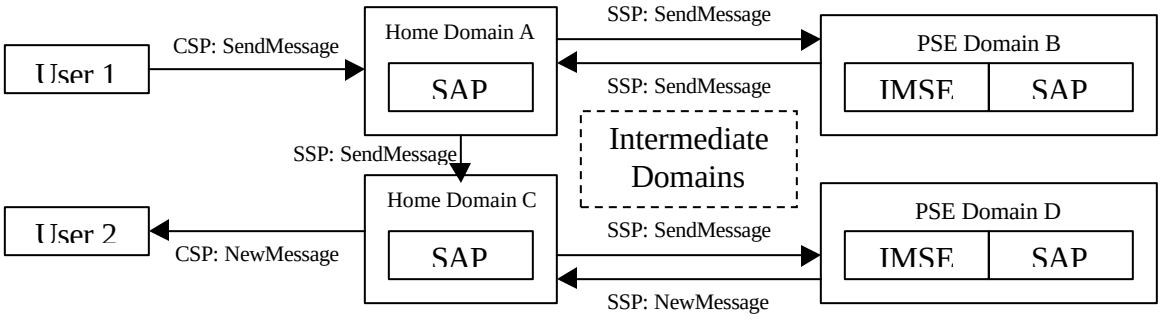

Figure 7. The SSP IOP Case Four

In the example in Figure 7, client 1 is located in home domain A, and client 2 is located in home domain C. A's IM PSE is located in Domain B, and C's PSE is located in

Domain D. Home Domain A and Home Domain C are connected via some intermediate domains. The transaction flow of sending a message from client 1 to client2 is:

1. C1 -> DA: CSP-SendMessage
2. DA -> DB: SSP-SendMessage (through intermediate domains via routing)
3. DB -> DA: SSP-SendMessage (through intermediate domains via routing)
4. DA -> DC: SSP-SendMessage
5. DC -> DD: SSP-SendMessage (through intermediate domains via routing)
6. DD -> DC, SSP-NewMessage (after checking block list etc.)
7. DC -> C2, CSP-NewMessage

2.4.5. Special Case Processing

The special cases include the situations that offline agreement overrides the IOP Rule 3. The following example illustrates the processing for this type of special cases.

Two Users are Located in different Home Domains. Both Home Domains Share the same PSE

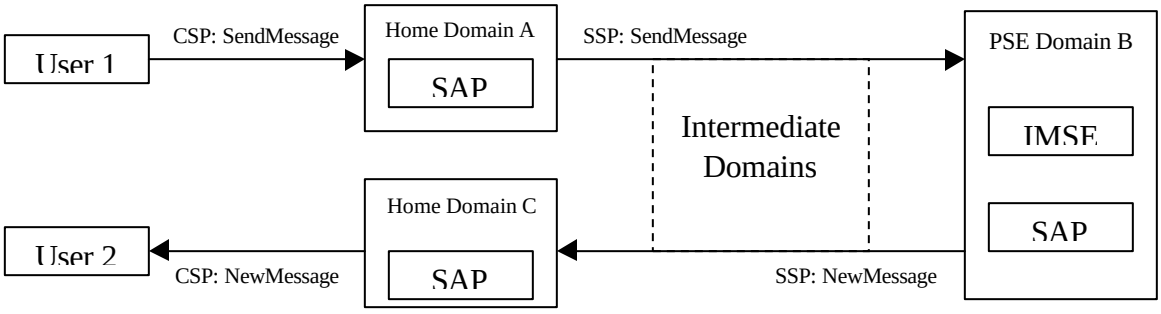

Figure 8. The SSP IOP Special Case

In the example in Figure 8, client 1 is located in home domain A, and client 2 is located in home domain C. Both Domain A and Domain C share the IM PSE located in Domain B. Domain A and B are connected via some intermediate domains. Domain C and B are connected via some intermediate domains. The transaction flow of sending a message from client 1 to client2 is:

1. C1 -> DA: CSP-SendMessage
2. DA -> DB: SSP-SendMessage (through intermediate domains via routing)
3. DB -> DC, SSP-NewMessage (after checking block list etc.)
4. DC -> C2, CSP-NewMessage

Note that the transaction flow is based on the offline configuration in PSE Domain B, which allows the direct relay from A to B to C without the direct SSP connection between Home Domain A and C based on their off-line routing agreement. IOP Rule 3 does not apply to this case.

If Domain A and Domain B are directly connected, there will be one SSP-SendMessage from A to B. If Domain A and Domain B are connected through several intermediate domains, there will be several SSP-SendMessages from A to B, one for each hop. Each intermediate domain will relay the SSP-SendMessage to the next hop.

If Domain C and Domain B are directly connected, there will be one SSP-NewMessage from B to C. If Domain C and Domain B are connected through several intermediate domains, there will be several SSP-NewMessages from B to C, one for each hop. Each intermediate domain will forward the SSP-NewMessage to the next hop.

2.5. SSP Protocol Stack

The SSP protocol stack is divided into three layers as follows.

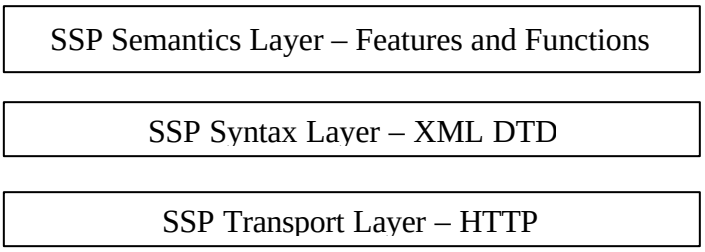

Figure 9. The SSP Protocol Stack

SSP Semantics Layer defines the complete set of features and functions that SSP intends to address in the full interoperability model among the WV domains. The nature of the features and functions, i.e. mandatory or optional or conditional, is also defined in the SSP Semantics Layer. The details of the features and functions are described in the transactions, primitives and information elements in the SSP Semantics Layer.

SSP Syntax Layer defines the “communication language” for the WV SAP’s to understand the information between each other and accomplish the interoperability of the features and functions defined in SSP Semantics Layer. SSP Syntax Layer v1.0 is the set of XML DTD specification.

SSP Transport Layer defines the “communication method” that conveys the “communication language” between the WV SAP’s to achieve the interoperability. SSP Transport Layer v1.0 is HTTP.

This document describes the SSP Semantics Layer.

The term “Server” in this document represents the logical server cluster in one service provider domain. The term “Server” is interpreted as the single access point of the domain, which may be physically a Local Director, or a Proxy, or a Routing Proxy, or anything else that represents the domain. The term “Server” is not interpreted as any physical server entity of the deployment within the domain.

3. Protocol Introduction

SSP is based on the architecture model described in the *"System Architecture Model"* document [WVARCH] and focuses on the communication and interaction among the WV domains. The semantics of SSP is consistent with the functional description of the Service Access Point (SAP) in the architecture model. The semantics of SSP implements the server interoperability described in the "*Features and Functions*" document [WVFEAT]. The semantics of SSP supports the semantics of Client to Server Protocol (CSP) [WVCSPS] in a distributed environment to achieve full interoperability.

3.1. Basics

3.1.1. Session

The server interoperability is accomplished in the frame of two SSP sessions. An SSP session is the period during which the servers conduct interactions and interoperations for the Service Provider to provide the Service Requestor with the negotiated IMPS services.

Each Provider Server maintains one session for each Requestor Server. There are two sessions between two domains. Each server maintains one session to provide the other with its own negotiated IMPS services.

3.1.2. Transaction

The SSP semantics are accomplished by "transactions". An SSP transaction is the sequence of interactions to complete a specific SSP feature or function. The SSP transactions include one-way transactions, two-way transactions, and multi-way transactions. A one-way transaction consists of a service request. A two-way transaction consists of a service request and a service response. A multi-way transaction consists of a sequence of service requests and responses.

3.1.3. Message

Both service requests and service responses are called SSP "messages". An SSP message is the syntax unit in one interaction.

An SSP message must contain some meta-information including the protocol information (e.g. version), the session information (e.g. Session-ID), the transaction information (e.g. Transaction-ID) and the attribute information (e.g. one-way / two-way, request / response). The "response" message in a two-way transaction must contain the same Transaction-ID as the corresponding "request" message. All transactions during one session must contain the same Session-ID.

3.1.4. Primitive

Each SSP message includes one or more SSP "primitives" with appropriate parameters. An SSP primitive is the semantics unit in one message.

Each service request message contains one functional primitive. Each service response message includes a status primitive as well as the optional, one or more SSP primitive(s).

3.2. Session Pair vs Connections

There are two sessions between two domains. Each domain maintains one session to provide the other with its own negotiated IMPS services. The two sessions are established through session establishment.

There are at least two physical connections, namely the connection pair, to carry the service traffic of the session pair. The servers may establish more than one connection pairs to support the same session pair.

The physical connection carries the service requests from the Requestor server to the Provider Server in one direction, and / or the notifications from the Provider Server to the Requestor Server in the other direction.

Connections are reusable. Each session may use some or all of the connections to transport its transactions. Each connection may be used by only one session, or reused by both sessions.

An SSP transaction (request and response) must be completed using the same connection pair.

Please refer to the SSP Transport Binding Document [WVSSPT] about how the connection (pair) is bound to the underlying transport.

3.3. Addressing

SSP addressing schema uses the uniform Wireless Village addressing model in a unique Wireless Village address space. SSP addressing schema is consistent with that in CSP.

The definition of SSP address is based on the URI [RFC 2396]. The addressable entities are:

- User
- Contact List
- Group (public and private)
- Content (public and private)
- Message
- Service (SSP unique)

The other address spaces may be used to interoperate with other systems. But it is up to the implementation and out of scope of Wireless Village.

3.3.1. General SSP Addressing Schema

The general SSP addressing schema is based on URI [RFC 2396]. The “wv” schema in the URI indicates the Wireless Village address space. The generic syntax is defined as follows:

WV-Address	= Service-ID | Message-ID | Other-Address
Other-Address	= “wv:” [User-ID] [“/” Resource] “@” Domain
Global-User-ID	= User-ID “@” Domain
Resource	= Group-ID | Contact-List-ID | Content-ID
Domain	= sub-domain *(“.” sub-domain)

where User-ID refers to the identification of the Wireless Village user inside the domain. Domain is a set of the Wireless Village entities that have the same “Domain” part in their Wireless Village addresses. Domain identifies the point of the Wireless Village server domain to which the IMPS service requests must be delivered if the requests refer to this domain. Resource further identifies the public or private resource within the domain. The sub-domain is defined in [RFC822]. The Service-ID is globally unique to identify a Server (either a WV server or a Proprietary Gateway), which is defined in section 3.3.7.

According to the URI specification [RFC2396], the “/” is a reserved character. So it cannot be used as a separator between User-ID and Resource address without encoding. It has to be substituted by the escaped octet encoded as a character triplet consisting of the percent character “%” followed by the two hexadecimal digits (2F) representing ACSII code of the slash character “/”.

When the Global-User-ID is present without the Resource, the address refers to the user. In SSP, the user is always identified in the global scope.

When the Global-User-ID is present with Resource, the address refers to the private resource of the user. When the User-ID is not present, the Domain and the Resource must always be present, and then the address refers to a public resource within the domain.

The domain must always be present in SSP addressing to globally identify the user or resources, and used for address resolution of those network entities.

The addresses are case insensitive.

3.3.2. User Addressing and Global-User-ID

SSP uses User-ID’s to uniquely identify a WV User. The User-ID either refers to the Internet-type address or to a mobile number of the user. If it refers to the mobile number of the user, the user name always starts either with digit or with '+' sign. User name referring to Internet-type address may not start with '+' sign or digit.

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

User-ID	= Mobile-Identity | Internet-Identity
Internet-Identity	= *alpha
Mobile-Identity	= (digit | "+") *digit
digit	= "0" | "1" | "2" |"3" |"4" |"5" |"6" |"7" |"8" | "9"
alpha	= Any ISO 8859-1 character except specials
specials	= "/" | "@" | "+" | " " | TAB

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

Examples of the User-ID's are:

Local-User-ID:	wv:Jon.Smith
	wv:+358503655121
	wv:0503655121
Global-User-ID:	wv:Jon.Smith@imps.com
	wv:+358503655121@imps.com
	wv:0503655121@imps.com

SSP always uses Global-User-ID to identify the users.

The users may also be identified by screen names, nicknames and aliases. These identifiers explicitly and implicitly refer to the User-ID.

ScreenName – the combination of a name a user chooses in a group session, and the Group-ID itself. The user may have different ScreenNames on different occasions as well as on different groups. The ScreenName is always connected to a group.

NickName – A name that is used internally in a client to hide the UserID of contacts. When ContactList is stored on the server, the NickName must have a space, but it is not possible to address a NickName.

Alias – The name a user suggest others to use as NickName. Part of the User Presence.

The definition of User-ID in SSP is consistent with that in CSP.

3.3.3. Contact List Addressing and Contact-List-ID

SSP uses Contact-List-ID's to uniquely identify any contact list of any user. The syntax of Contact-List-ID is defined as follows:

Contact-List-ID = *alpha

Examples of the contact list address with Contact-List-ID are:

wv:john%2Fcolleagues@imps.com
wv:%2Fmanagers@imps.com

SSP always identifies the contact list globally.

The definition of Contact-List-ID in SSP is consistent with that in CSP.

3.3.4. Group Addressing and Group-ID

SSP uses Group-ID's to uniquely identify any group. The syntax of the Group-ID is defined as follows:

Group-ID = *alpha

Examples of the group address with Group-ID are:

wv:john%2Fmygroup@imps.com
wv:%2Ftechnical_forum@imps.com

SSP always identifies the group globally.

The definition of Group-ID in SSP is consistent with that in CSP.

3.3.5. Content Addressing and Content-ID

SSP uses Content-ID's to uniquely identify any content. The syntax of the Content-ID is defined as follows:

Content-ID = *alpha

Examples of the content address with the Content-ID are:

wv:john%2FWV_presentation@imps.com
wv:%2Fwvspec@imps.com

SSP always identifies the content globally.

The definition of Content-ID in SSP is consistent with that in CSP.

3.3.6. Client Addressing and Client-ID

The Client-ID uniquely identifies the WV client as an application as well as its addressing that allows the access to the WV services. The client-ID is aimed to allow:

- Multiple accesses from the same user
- Direct application-to-application communication

The Client-ID consists of

- Optional application identifier such as a URL identifying the application and its addressing,
- Optional mobile device identity (such as international mobile number [E.164]).

The definition of Client-ID in SSP is consistent with that in CSP.

3.3.7. Service Addressing and Service-ID

The Service-ID in SSP is equivalent in the semantic role to the User-ID in CSP. The Service-ID in SSP uniquely identifies a Server. The syntax of Service-ID is defined as follows.

Service-ID = “wv:”@ Domain

Domain is a set of the WV entities that have the same Domain part in their WV addresses. The Domain is associated with one WV server (the unique access point) to which the IMPS service requests must be delivered if the addressed network entities refer to this Domain.

The Service-ID is used in the session establishment (refer to section 6.1.1, 6.2.2 and 6.3.1) and other SSP management functions.

The Service-ID is used as part of the meta-information in the SSP transactions (refer to section 5.1).

An examples of the Service-ID is:

Service-ID: wv:imps.com

3.3.8. Message and Message-ID

The Message-ID in SSP is globally unique to identify a message. The syntax of Message-ID is defined as follows.

Message-ID = Local-Message-ID “@” Domain

Where the “Local-Message-ID” uniquely identifies a message within the IMSE domain, and subject to the implementation.

An example of the Message-ID is: 12345678@imps.com.

The definition of Message-ID in SSP is consistent with that in CSP.

3.4. Data Types

SSP defines four basic data types, namely “Char”, “Integer”, “String” and “Boolean”, and three structured date types namely “Enum”, “DateTime” and “Structure”.

An information element is “String” type by default unless specified.

3.4.1. Char

A “Char” type element is a single character encoded in UTF-8.

3.4.2. Integer

An “Integer” type element is a 32-bit decimal number ranging in [0, 2^{32} - 1].

3.4.3. String

A “String” type element is a sequence of “Char” elements.

3.4.4. Boolean

A “Boolean” type element is either “True” or “False”.

3.4.5. Enum

An “Enum” type element is one of the pre-defined set of values.

3.4.6. DateTime

A “DateTime” type element follows the ISO-8601 specification and is expressed as a “String” type element. The date and time format shall be complete date and time using the basic format. There shall be no time-zone indication, but the time may indicate if the time is Coordinated Universal Time (UTC) or local time. The examples are:

Local time: 20011019T125031

UTC: 20011019T095031Z

3.4.7. Structure

A “Structure” type element is the combination of other types of elements as specified.

3.5. Infrastructure Elements

Infrastructure elements are required in the end-to-end solution of server interoperability. Infrastructure elements may not be carried within information elements in SSP protocol. However, the implementation shall be able to support the infrastructure elements to ensure the server interoperability.

3.5.1. Host-ID

The Host-ID is the primary (Master) host address of the SAP of the WV server or Proprietary Gateway. The Host-ID must be used for the session establishment with this WV server or Proprietary Gateway.

The Host-ID is referenced in the form of DNS host name. The Host-ID may be stored inside the environment for DNS A RR host address resolution, or may be retrieved from the Service-ID by the DNS SRV RR based address resolution.

The Host-ID cannot be changed during a session.

The example of Host-ID is: host1.imps.com

3.5.2. Redirect (Host) Name

When the WV server in a domain can be accessed through several SAP's distributed in different physical hosts, this WV server may provide a list of those hosts for the other WV server to share the load at the session establishment. This list is called Redirect List and contains the redirect host DNS names. A Redirect (Host) Name in SSP uniquely identifies a physical host in the WV Server or a Proprietary Gateway domain.

The Redirect (Host) Names may be configured statically based on offline agreement between two domains. The Redirect (Host) Addresses may be notified dynamically during session establishment over Master Connection Pair (6.1.1).

The example of a Redirect (Host) Address is: host2.serviceprovider.com.

3.6. Features and Functions

SSP supports the server interoperability features and functions defined and described in features and functions document.

3.6.1. Security

The scope of security in the server interoperability is the server-to-server communication at the IMPS application level, i.e. to ensure that the data sent and/or received on behalf of an End User in a given IMPS domain is actually originating from and/or terminating to the server in that domain.

SSP supports the security requirement in the server interoperability through the CALLBACK connection establishment and access control across session management and transaction management. Please refer to section 6.1.1 for details of CALLBACK connection establishment.

SSP supports the security requirement in the server interoperability through the underlying transport layer whenever possible.

The individual domain security enhances the overall security level in the server interoperability.

3.6.2. Connection Management

SSP connection management ensures the authenticated connections to transport SSP transactions during SSP sessions. Connection management includes connection establishment, connection termination and connection maintenance.

SSP supports CALLBACK connection establishment.

SSP supports the implicit connection termination and connection maintenance through session management. SSP session maintenance covers connection maintenance, and SSP session termination covers connection termination. Connection termination causes the session termination if no more connection exists.

3.6.3. Transaction Management

The transaction management defines the necessary common information elements in the service requests and service responses at transaction level, regulates the behavior in the transaction flows, and handles the exception and error conditions at transaction level.

3.6.4. Session Management

SSP supports the authentication among the WV SAP's. The WV SAP's must authenticate each other before they can provide each other with the IMPS services.

SSP supports the authorization and access control among the WV SAP's so that the servers and the gateways are allowed to access the IMPS services provided by each other.

SSP session management includes session establishment, session termination and session maintenance. The CALLBACK connection establishment shall be used in the session establishment. The access control is supported in the whole session management.

3.6.5. Service Management

SSP supports service discovery among the WV domains. The services include Common Services, Presence Service, Instant Messaging (IM) Service, Group Service and Shared Content Service that are defined in *"Features and Functions"* document. However, those services are discovered in the element level rather than the protocol level. SSP only provides a protocol method and facilitates the message exchange to support the service discovery.

SSP supports the service negotiation and agreement among the WV domains. The service agreement may be made either online or offline. The service agreement must be made before they can provide each other with the IMPS services.

3.6.6. User Profile Management

SSP supports the exchange of user profile information among the WV domains including the list of services to which a user subscribes, the service status (active / inactive), privacy status with regard to network service capabilities (e.g. user location, user interaction), terminal capabilities, the user account status etc.

User Profile Management features can support various functions based on the exchange of user profile information.

3.6.7. Service Relay

SSP supports the service relay among the WV domains including the functional relay of the common IMPS features, presence features, IM features, group features and shared content features that are defined in *"Features and Functions"* document. The goal of SSP is to support the distributed interoperable complementary IMPS services across service provider domains.

Because of the server interoperation nature, the SSP has its own requirement on meta-information and information elements in the primitives at transaction level. The complete primitives and transaction flows at SSP semantics level has been defined in the following sections including functional relay services.

Please refer to the CSP document so as to conclude how to relay the complete IMPS features from client-server interaction (CSP) to server-server interoperation (SSP).

4. Security

The scope of security in the server interoperability is the server-to-server communication at the IMPS application level, i.e. to ensure that the data sent and/or received on behalf of an End User in a given IMPS domain is actually originating from and/or terminating to the servers in that domain.

4.1. Trust Models

A TRUST model is assumed between the WV SAP and the Service Elements within a single IMPS domain.

A TRUST model is assumed for the network infrastructure such as DNS.

The TRUST model is mutual, i.e. A trusts B if and only if B trusts A.

The TRUST model is created between domain A and domain B if and only if they have been authenticated and authorized by each other. A TRUST model must be created between two domains before they can provide each other with interoperable complementary IMPS services.

4.2. Access Control

The authentication and authorization between the servers in different domains are accomplished by the access control at each server. The scope of access control covers online session management, transaction management and offline configuration agreement.

The online session management includes the initial CALLBACK connection establishment, authentication and authorization to start a session, session maintenance and session termination.

The transaction management supports the access control by the transaction authentication based on the information elements specified in each service request and service response.

The offline configuration agreement includes, but not limited to, server identity registration, Host-ID, account creation, password protection, configurable parameters, SAP Service Routing Table, etc. through provisioning and / or administration interface.

4.3. Transport Security

The security requirement in the transport layer and other underlying layers, such as data integrity and confidentiality, is out of the scope of SSP. However, whenever possible, current security approach including SSL / TLS, PGP, PKI, digital certificates, etc. in the underlying transport layer should be used to ensure the secure transmission in the

underlying layers to prevent from out-of-scope security issues. The deployed security technology is negotiated between the service providers through the offline configuration agreement.

4.4. Individual Domain Security

The security of individual domain enhances the inter-domain security. A single IMPS domain is encouraged to use firewalls or other precautions to ensure the highest possible level of security.

5. Transaction Management

The transaction management defines the necessary common information elements in the service requests and service responses at transaction level, regulates the behaviour in the transaction flows, and handles the exception and error conditions at transaction level.

5.1. Meta-Information

The SSP service requests must contain the meta-information as defined in table 1.

Information Element	Req	Type	Description
Client-Originated	M	Boolean	Indicates whether the request is originated from the client ("True") or from the service element ("False").
Session-ID	M	String	Identifies the session managed by the **Provider Server**.
Transaction-ID	M	String	Identifies the transaction originated from the transaction initiator (either requestor server, or provider server).
Service-ID	M	String	Identifies the initiator domain (and the service element if needed).
User-ID	C	String	Identifies the user represented by the requestor server domain. It is present if the request is originated from a client.
Client-ID	O	String	Identifies the Client-ID of the user. It optionally present if the request is originated from a client.

Table 1. Information elements in Meta-information primitive

The Session-ID is unique for each session at the Provider Server.

The Transaction-ID is unique for each transaction originated from the server that initiates the transaction.

An SSP service response in a two-way transaction must contain the same Session-ID and the Transaction-ID as those in the service request.

Some implementation notes are as follows.

1) The SAP at the service provider server should maintain a Session Record for each service requestor.

2) The SAP at the service requestor should maintain a Transaction Record for each service provider.

3) The SAP at each server should maintain a Transaction Table to map each requested transaction from its Service Requestor to the initiated transaction to its Service Provider. The Transaction Table should be the uniquely one-one match. Therefore, the Service Relay flow and Result Forward flow at each hop is clearly and uniquely identified by the transaction flows.

5.2. Status Primitive

The status primitive in the service response is defined as follows in table 2.

Information Element	Req	Type	Description
Session-ID	M	String	Identifies the session. It should be consistent with the Session-ID in the Meta-Information in the request.
Transaction-ID	M	String	Identified the transaction. It should be consistent with the Transaction-ID in the Meta-Information in the request.
Status code	M	String	Status code of the processing result.
Status description	O	String	Textual description of the status.

Table 2. Information elements in Status primitive

5.3. Asynchronous Transaction

The server shall support asynchronous transactions.

5.4. General Exception Handling

In two-way transactions, after a transaction is initiated, the originating server is expecting the response from the processing server. In multi-way transactions, after a transaction is initiated, one server is expecting the response from the other server.

Whenever error happens, the processing server shall handle the exception based on its own policy. In addition, the processing server shall inform the other server involved in this transaction of such exception by sending the Status primitive with appropriate Status Code and optional Status Description.

5.5. Invalid Transaction

A transaction is considered “valid” if the transaction completes within a reasonable period. The transaction validity time is the sum of the network latency, transaction processing time and an adjustable offset. Those three elements must be configurable at each service domain by the operator. Each operator shall define and configure the reasonable value of the three elements based on the network, hardware and software capacity to ensure the quality and performance of the service as well as the security.

A transaction is considered “invalid” if the transaction cannot complete within the validity time.

If an invalid transaction occurs, the service requestor shall not receive a response from the provider domain. The service requestor shall repeat the transaction for reasonable times until the transaction completes or the repeat times expire. If the transaction completes, the session shall go on for the future transactions. If the repeat times expire, the session shall be terminated by the requestor for security reason. In addition, the requestor-maintained session, which provides the other side with its own service, shall be terminated too.

The repeat times must be configurable at each service domain by the operator. Each operator shall define and configure a reasonable value of repeat times to ensure the quality and performance of the service as well as the security. The repeat times may be zero (0) if the security is the major concern.

5.6. Unknown Transaction

A transaction is considered “unknown” if (1) the request message has syntactic error (e.g. not XML well-formed, XML invalid, data value error); or (2) any of the information elements of the Meta-Information is invalid; or (3) the service request refers to a service which doesn't corresponds to the service agreement between the service requester and provider; or (4) the service response cannot be associated with the original service request.

If an unknown transaction happens in a service request, the provider domain shall return a status code indicating an “Unknown Transaction” error. If the unknown transaction happens frequently, the provider domain shall terminate the session as well as the session maintained by the requestor for security reason.

The definition of “Unknown Transaction Frequency” is up to the server implementation. However, the value of “Unknown Transaction Frequency” must be configurable at each service domain by the operator. Each operator shall define and configure a reasonable value of “Unknown Transaction Frequency” to ensure the quality and performance of the service as well as the security. The server may terminate the sessions immediately after an unknown transaction happens if the security is the major concern.

If an unknown transaction happens in a service response, the requestor shall perform the same behavior as that in handling “invalid transaction”.

5.7. General Status Code

All SSP transactions may be return the following status codes

- Continue (100) – for all complementary transactions
- Queued (101) – for all complementary transactions

- Started (102) – for all complementary transactions
- Server queued (104)
- Bad Request (400)
- Service not supported (405) – for all complementary transactions
- Service Unavailable (503)
- Invalid Timeout (504)
- Service not agreed (506) – except transactions required for the service agreement
- Internal Server Error (500)
- Unknown transaction (536)
- Invalid server session (620) – except transactions allowed outside of a session

6. Session Management

SSP session management includes session establishment, session termination and session maintenance. The CALLBACK connection establishment is used in the session establishment. The access control is supported in the whole session management.

6.1. Access Control

6.1.1. Session Establishment

The session is established through the connection establishment, and initial authentication and authorization between the servers in different domains.

The CALLBACK connection establishment is used in the session establishment. The basic session establishment with the CALLBACK connection works as follows.

Prerequisites:

- A-Host-ID represents the unique access point to domain A.
- B-Host-ID represents the unique access point to domain B.
- Offline configuration agreement has been established between Server A and Server B.
- In Server A, Server B's identity is registered with at least { B-Host-ID, B-Service-ID, B-password } tuple
- In Server B, Server A's identity is registered with at least { A-Host-ID, A-Service-ID, A-password } tuple
- Both servers has registered and supported a common digest schema such as MD5 or SHA.

The basic steps are:

1. Server A originates a connection 1 to Server B based on its own registration record about Server B, containing { A-Service-ID, A-secret-token} tuple.
2. Server B looks for { A-Service-ID} in its own registration record. If it is not found, Server B closes the connection.
3. Server B initiates connection 2 to the Server A containing { B-Service-ID, B-secret-token }.
4. Server A looks for { B-Service-ID } in its own registration record. If it is not found, Server A closes the connection.
5. Server A sends the LoginRequest to Server B through connection 1, containing { A Service-ID, A-password-digest }. The "A-password-digest" is generated with A-password and B-secret-token based on the common digest schema in the registration record.
6. Server B sends the LoginRequest to Server A through connection 2, containing { B-Service-ID, B-password-digest }. The "B-password-digest" is generated with B-password and A-secret-token based on the common digest schema in the registration record.
7. Server B verifies the A-password-digest. If the verification fails, it closes the connection.
8. Server B responses to Server A with the LoginResponse through connection 2, containing the status of the transaction and the new session information maintained by Server B. The LoginResponse may contain an optional list of Redirect (Host) Names. This is also called the Redirect List.
9. Server A verifies the B-password-digest. If the verification fails, it closes the connection.
10. Server A responses to Server B with the LoginResponse through connection 1, containing the status of the transaction and the new session information maintained by Server A. The

LoginResponse may contain an optional list of Redirect (Host) Names. This is also called the Redirect List.

The secret-token is a random string generated by the connection originator at each server.

After step 10 succeeds, two domains are authenticated with each other. The session pair between Server A and Server B are established with trust over two connections, i.e. the connection pair. The connection pair (1 and 2) between A-Host-ID and B-Host-ID is called “Master Connection Pair”.

The “Redirect List” reflects the server’s desire and capability to handle the redirect. If the server does not include the “Redirect List” in its LoginResponse, the server does not support the redirect, and the server intends to use the “Master Connection Pair” to support the session. In this case, the other server shall not try the connection pair establishment unless a new redirect process takes place. Therefore, even if the server does not have its own “Redirect List”, but if the server supports the redirect of the other server, it MUST provide a “Redirect” List in the LoginResponse. In this case, the “Redirect List” contains its original Host-ID only.

If the “Redirect List” is included in both of the LoginResponses, i.e. in both Step 8 and Step 10, the redirect takes place. Otherwise the Master Connection Pair (1 and 2) shall be used to support the session.

If the “Redirect List” is included in the LoginResponse in Step 8 and Step 10, both of the domains want to use the new “Redirect List” as the physical connections to support the session. The connection pair(s) shall be handed over to the actual physical nodes, and the Master Connection Pair (1 and 2) shall be disconnected. If there are more than one Redirect (Host) Names in either of the “Redirect List”, a mesh of redirect connection pairs shall be initiated to support the session pair.

After session establishment, there may be an optional online service negotiation and service agreement depending on the offline agreement between two domains. If the online service negotiation and service agreement is needed, it shall be the first transactions in the session pair.

Two servers will provide each other with the IMPS services after the authorization (i.e. online service negotiation and service agreement) if needed, or right after the session establishment otherwise.

There are at least two connections, the connection pair, to carry the session pair. The servers may establish more than one connection pairs to support the same session pair. The redirect connection pair between two redirect physical hosts in two domains is established through the same steps except that the redirect connection pair shall be bound to the existing session pair between two domains. The “Redirect List” in Step 8 and Step 10 of session establishment may have set up a mesh of more than one redirect connection pairs. Within the session, if additional (mesh of) redirect connection pair(s) is needed, the same Session Establishment steps with the “Redirect List” in Step 8 and Step 10 shall be

repeated except that the Master Connection Pair shall be bound to the existing session pair and no new session shall be created. The "Redirect List" shall initiate the establishment of a new mesh of redirect connection pairs. Note that the "Redirect List" is only allowed over Master Connection Pair. Also note that no new session shall be established when setting up redirect connection pairs. There is always one session pair between two domains no matter how many redirect connection pairs are created. In case of creating redirect connection pairs, it is not allowed to make online service negotiation and service agreement.

Connections are reusable. Each session may use some or all of the connections to transport its transactions. Each connection may be used by only one session, or reused by both sessions. In the simplest case, one possible situation is that Connection 1 will be used for the service session provided and managed by Server B, and connection 2 will be used for the service session provided and managed by Server A.

SSP Transport Binding document [WVSSPT] shall define how to bind session pair to reusable connections by the underlying transport.

6.1.2. Session Maintenance

Server A and Server B shall maintain the session and keep the session alive by exchanging the live traffic if needed during the session. The initial interval is negotiated during session establishment. The interval may be adjusted by negotiating a new interval when exchanging the live traffic.

The session maintenance may be required periodically in case that an intermediary (e.g. proxy) may break the connection, which results in terminating the session, if there is no data traffic for a reasonable time period. The session maintenance may also be required periodically in case that the server policy requires the termination of the session if there is no transaction activity for a reasonable time period. If session maintenance is required for one session, it is usually also required for the other session.

The interval must be configurable at each service domain by the operator. The operators shall define and configure a reasonable value of "interval" to ensure the quality and performance of the service as well as the security. The interval configuration must be adjustable on-the-fly.

The session maintenance shall be performed over all of the connections used by this session, thus covers the connection maintenance.

6.1.3. Session Termination

The session shall be able to be terminated by either Server A or Server B at any time. Both of the sessions managed by Server A and Server B must be terminated to ensure the security.

A session may be terminated normally. For example, the service agreement expires, or the session expires. If any of the service agreements expires, or any of the session expires, both of the sessions are terminated.

A session may be terminated abnormally. For example, an invalid session occurs, or the connection (due to the underlying transport) breaks. If all of the connections of one session break, both of the sessions are terminated. However, even if some connections are terminated due to load balancing or other reason, as long as there is at least one connection for each session, the session pair SHALL NOT be terminated.

The session termination covers and implies the connection termination. Whenever the session is terminated, all of the connections used by this session shall be terminated as well.

6.1.4. Session Re-establishment

If the sessions are terminated, two servers may re-establish the session based on their offline service agreement. The session re-establishment means creating a new session pair, and follows the same steps in the session establishment.

6.2. Primitives

6.2.1. The "SendSecretToken" Primitive

The "SendSecretToken" primitive is issued by the requestor server to send the secret token for the provider server as the first step of the CALLBACK connection establishment.

Information Element	Req	Type	Description
Message-Type	M	SendSecretToken	Message identifier
Transaction-ID	M	String	Identifies the transaction originated from the initiating provider server.
Service-ID	M	String	Identifies the requestor server.
Protocol	M	“WV-SSP”	SSP protocol.
Protocol-Version	M	“1.0”	SSP protocol version.
SecretToken	M	String	Secret token originated by the requestor.

Table 3. Information elements in SendSecretToken Primitive

6.2.2. The “LoginRequest” Primitive

The “*LoginRequest*“ primitive is issued from the requestor server to create a new session or a new connection pair inside the existing session with the provider server. The “*LoginRequest*“ primitive specifies initial status of the requestor server. The “*LoginRequest*“ primitive MAY also contain the “*time-to-live*“ attribute, which specifies

the time that the session or the connection will expire. If "*time-to-live*" attribute is omitted, the requestor server requests an infinite session or connection until service agreement expires.

Information Element	Req	Type	Description
Message-Type	M	LoginRequest	Message identifier
Session-ID	C	String	Identifies the session. It is present when creating additional redirect connection pairs within the existing session.
Transaction-ID	M	String	Identifies the transaction. It should be consistent with the Transaction-ID in the SendSecretToken originated from the provider server.
Service-ID	M	String	Identifies the requestor server.
Password-Digest	M	String	The password digest generated with password and secret token based on a common digest schema (MD5 or SHA).
Time-To-Live	O	Integer in Seconds	Interval for a valid session or connection before expired. If omitted, the requestor server requests an infinite session or connection.

Table 4. Information elements in LoginRequest Primitive

6.2.3. The "LoginResponse" Primitive

The "*LoginResponse*" primitive is issued from the provider server to accept the session creation or connection pair creation with the requestor server. In the response, the provider server MAY specify the "*time-to-live*" of the current session. This "*time-to-live*" may be different from that in the "*LoginRequest*" from the requestor server.

Information Element	Req	Type	Description
Message-Type	M	LoginResponse	Message identifier
Status-Info	M	Structure of Status-Primitive	The necessary status information in a service response defined in 5.2.
Time-To-Live	O	Integer in Seconds	Interval for a valid session or connection before expired. This time may be any value other than zero.
List-of-Hosts	O	Structure	"Redirect" list, which indicates the actual connection addresses in its own domain.

Table 5. Information elements in LoginResponse Primitive

6.2.4. The "LogoutRequest" Primitive

The *"LogoutRequest"* primitive is used for the requestor server to close the session with the provider server.

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

Table 6. Information elements in LogoutRequest

6.2.5. The "Disconnect" Primitive

The *"Disconnect"* primitive is used for the provider server to indicate that it accepts the *"LogoutRequest"* from the requestor server and closes the session.

If the provider server does not receive any session maintenance update within the time-to-live interval (see *"KeepAlive"* primitive) from requestor server, the provider server will also close this session by sending the *"Disconnect"* message to the requestor server.

Information Element	Req	Type	Description
Message-Type	M	Disconnect	Message identifier
Session-ID	C	String	Identifies the session. Present if the provider server initiates the Disconnect.
Transaction-ID	C	String	Identifies the transaction. Present if the provider server initiates the Disconnect.
Status-Info	C	Structure of Status-Primitive	The status information (see 5.2). Present if the requestor server Logout.

Table 7. Information Elements in Disconnect Primitive

6.2.6. The "KeepAliveRequest" Primitive

The *"KeepAliveRequest"* primitive is used for the requestor server to maintain the session and update the time-to-live interval with the provider server. The session maintenance shall be performed over all of the connections used by this session, thus implies and covers the connection maintenance for each connection. The TTL may have different value for different connection.

Information Element	Req	Type	Description
Message-Type	M	KeepAliveRequest	Message identifier
Session-ID	M	String	Identifies the session.
Transaction-ID	M	String	Identifies the transaction.
Time-to-live	O	Integer in Seconds	Indicates the time-to-live of the session over this connection.

Table 8. Information Elements in KeepAliveRequest Primitive

6.2.7. The "KeepAliveResponse" Primitive

The *"KeepAliveResponse"* primitive is used for the provider server to maintain the session and update the time-to-live interval with the requestor server. The session maintenance shall be performed over all of the connections used by this session, thus implies and covers the connection maintenance for each connection. The TTL may have different value for different connection.

Information Element	Req	Type	Description
Message-Type	M	KeepAliveResponse	Message identifier
Status-Info	M	Structure of Status-Primitive	The status information (see 5.2).
Time-to-live	O	Integer in Seconds	Indicates the time-to-live of the session over this connection.

Table 9. Information Elements in KeepAliveResponse Primitive

6.3. Transactions

6.3.1. The "Login" Transaction

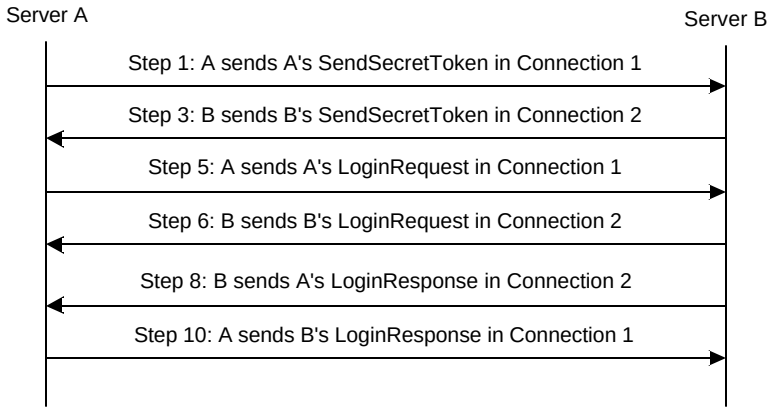

Figure 10. The "Login" Transaction

Session establishment and additional redirect connection establishment are achieved through "Login" transaction.

The Server A performs Step 1 and sends A's *'SendSecretToken*" to Server B through Connection 1. After the Server B performs Step 2, the Server B performs Step 3 and sends B's *'SendSecretToken*" to Server A through Connection 2. After the Server A performs Step 4, the Server A performs Step 5 and sends A's *"LoginRequest"* to Server B through Connection 1. The Server B performs Step 6 and sends B's "LoginRequest" to Server A through Connection 2. Finally, the Server B performs Steps 7 & 8, and replies

with A's *"LoginResponse"* to Server A through Connection 2, and A performs Steps 9 & 10 and replies with B's *"LoginResponse"* to Server B through Connection 1.

Step 1, Step 6 and Step 10 shares the same Transaction-ID that is generated by Server A in step 1.

Step 3, Step 5 and Step 8 shares the same Transaction-ID that is generated by Server B in step 3.

After step 10 succeeds, two domains are authenticated with each other. The session pair between Server A and Server B are established with trust over two connections, i.e. the connection pair. The connection pair (1 and 2) between A-Host-ID and B-Host-ID is called "Master Connection Pair".

The "Redirect List" reflects the server's desire and capability to handle the redirect. If the server does not include the "Redirect List" in its LoginResponse, the server does not support the redirect, and the server intends to use the "Master Connection Pair" to support the session. In this case, the other server shall not try the connection pair establishment unless a new redirect process takes place. Therefore, even if the server does not have its own "Redirect List", but if the server supports the redirect of the other server, it MUST provide a "Redirect" List in the LoginResponse. In this case, the "Redirect List" contains its original Host-ID only.

If the "Redirect List" is included in both of the LoginResponses, i.e. in both Step 8 and Step 10, the redirect takes place. Otherwise the Master Connection Pair (1 and 2) shall be used to support the session.

If the "Redirect List" is included in the LoginResponse in Step 8 and Step 10, both of the domains want to use the new "Redirect List" as the physical connections to support the session. The connection pair(s) shall be handed over to the actual physical nodes, and the Master Connection Pair (1 and 2) shall be disconnected. If there are more than one Redirect (Host) Names in either of the "Redirect List", a mesh of redirect connection pairs shall be initiated to support the session pair.

There are at least two connections, the connection pair, to carry the session pair. The servers may establish more than one connection pairs to support the same session pair. The redirect connection pair between two redirect physical hosts in two domains is established through the same steps except that the redirect connection pair shall be bound to the existing session pair between two domains. The "Redirect List" in Step 8 and Step 10 of session establishment may have set up a mesh of more than one redirect connection pairs. Within the session, if additional (mesh of) redirect connection pair(s) is needed, the same Session Establishment steps with the "Redirect List" in Step 8 and Step 10 shall be repeated except that the Master Connection Pair shall be bound to the existing session pair and no new session shall be created. The "Redirect List" shall initiate the establishment of a new mesh of redirect connection pairs. Note that the "Redirect List" is only allowed over Master Connection Pair. Also note that no new session shall be

established when setting up redirect connection pairs. There is always one session pair between two domains no matter how many redirect connection pairs are created. In case of creating redirect connection pairs, it is not allowed to make online service negotiation and service agreement.

Primitive	Direction
SendSecretToken	Requestor Server ← Provider Server
LoginRequest	Requestor Server → Provider Server
LoginResponse	Requestor Server ← Provider Server

Table 10. Primitive Directions for Login Transaction

6.3.2. The "Logout" Transaction

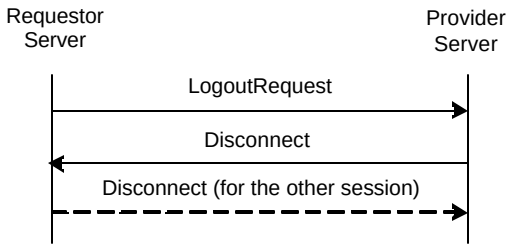

Figure 11. The "Logout" Transaction

Session termination is achieved through "Logout" and "Disconnect" transactions. All of the connections used by this session shall be terminated as well after the session is finished.

The requestor server can logout from the provider server and close the session through a "Logout" transaction. In addition, the requestor also shall terminate the other session through a "Disconnect" transaction that is illustrated in the dash line.

The requestor server sends a *"LogoutRequest"* request to the provider server. After the provider server finishes processing the request, it sends a *"Disconnect"* response to the requestor server to indicate the close of the session.

Primitive	Direction
LogoutRequest	Requestor Server → Provider Server
Disconnect	Requestor Server ← Provider Server

Table 11. Primitive Directions for Logout Transaction

6.3.3. The "Disconnect" Transaction

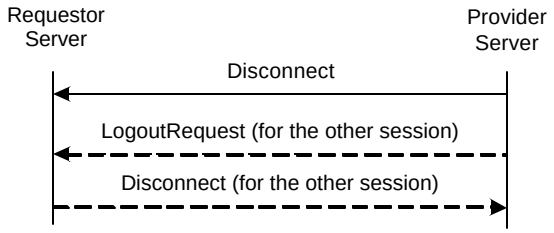

Figure 12. The "Disconnect" Transaction

The provider server may close the session through a "Disconnect" transaction. In addition, the provider also shall terminate the other session through a "Logout" transaction that is illustrated in the dash lines.

Primitive	Direction
Disconnect	Requestor Server ← Provider Server

Table 12. Primitive Directions for Disconnect Transaction

6.3.4. The "KeepAlive" Transaction

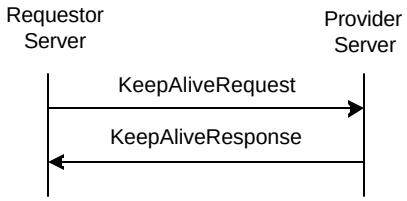

Figure 13. The "KeepAlive" Transaction

Session maintenance is achieved through "KeepAlive" transaction. "KeepAlive" transaction shall be performed over all of the connections used by this session, thus implies and covers the connection maintenance for each connection. The TTL may have different value for different connection.

The requestor server updates the time-to-live interval and keeps the session and the connection(s) alive through the "KeepAlive" transaction(s).

The requestor server sends a *"KeepAliveRequest"* request to the provider server. After the provider server finishes processing the request, it sends a *"KeepAliveResponse"* response to the requestor server to indicate the status of the session over this connection. The *"KeepAliveRequest"* may carry a new time-to-live interval. The time-to-live value returned in the *"KeepAliveResponse"* response may differ from that in the request.

The "KeepAlive" transaction may be required periodically in case that an intermediary (e.g. proxy) may break the connection, which results in terminating the session, if there is no data traffic for a reasonable time period.

The "KeepAlive" transaction may be required periodically in case that the server policy requires the termination of the session if there is no transaction activity for a reasonable time period.

If "KeepAlive" is required for one session, it is usually also required for the other session.

Primitive	Direction
KeepAliveRequest	Requestor Server → Provider Server
KeepAliveResponse	Requestor Server ← Provider Server

Table 13. Primitive Directions for KeepAlive Transaction

6.4. Status Code

6.4.1. "Login" Transaction

- Unknown Service-ID (606)
- Redirection refused (607)
- Invalid password. (608)

6.4.2. "Logout" / "Disconnect" Transaction

- Session Expired (600)
- Connection expired (609)

7. Service Management

The service management in SSP enables the Wireless Village servers to mutually agree on the usable Wireless Village services. The usable services offered by a server are arranged in a negotiation tree.

7.1. Service Structure

The Wireless Village services are organized in a hierarchy:

- Features – a specific set of related functionality
- Functions – defines a set of related transactions for each feature
- Transactions – defines a set of related primitives for each function
- Information Elements – the lowest level building blocks of the transactions

A Wireless Village server may support all or a subset of the features. However, if a WV server supports a feature, some functions and transactions must be supported to ensure minimal interoperability [WVSSPSCR]. The remaining functions and transactions are optional. Moreover, there are multiple choices in the semantics for some of the functions and transactions, e.g. the general search transaction with search-type USER-ID is mandatory while all other search types are optional.

The optional functions, transactions, and choices offered by a server are arranged in a service tree, as shown in Figure 14. Each node in the tree specifies the functions, transactions, and choices that must be supported by the server that includes that node in its Service-List.

Each node in the service tree defines a group of one or several transactions or choices. The content of each node and how the tree should be interpreted are described as follows. The transactions that are not described are considered mandatory functions that must be always supported in the servers.

General

If a *Feature* node is included in the Service-List, all mandatory requirements for that specific feature must be supported as specified in [WVSSPSCR].

If a lower level node is included in the Service-List, all transactions or choices specified by that node must be supported.

SAP Feature

- *Service Negotiation* node includes the following transactions
  - GetAvailableService
  - ServiceIndication

  - SetServiceAgreement
- *User Profile management* node includes the following transactions
  - GetUserProfile
  - UpdateUserProfile
- *Service Relay* node indicates if the SAP supports service relay including routing

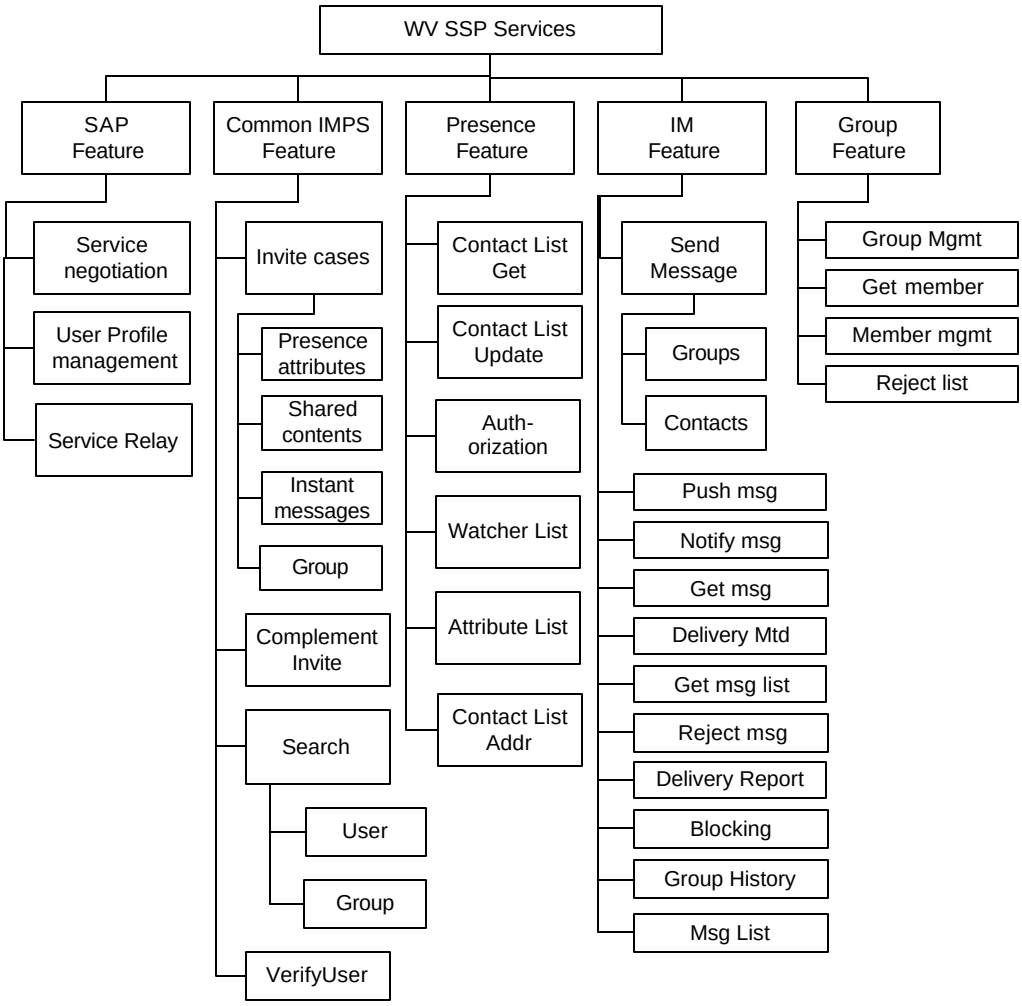

Figure 14: SSP Service tree

Common IMPS feature

- *Invite* node include the Invitation/Cancel-Invitation transactions
  - All supported invite types must be included in the Service List (Presence, IM, Shared Content, Group)
- *Complementary Invite* node includes the Complementary Invitation/Cancel-Invitation transactions
  - If the Complementary invite node is included in the Service-List, the Invite cases node must be included as well.

- *Search* node includes the optional choices for the GeneralSearch. All supported search types must be included in the Service List i.e.
  - User: Support Presence attributes criteria
  - Group: Support Group related criteria
- *VerifyUser* node includes the following transactions:
  - VerifyUserID

Presence Feature

- *Contact List Get* node includes the following transactions
  - GetContactList
  - GetListMember
  - GetListProperties
- *Contact List Update* node includes the following transactions
  - CreateContactList
  - DeleteContactList
  - AddListMember
  - RemoveListMember
  - SetListProperties
- *Authorization* node includes the following transactions
  - ReactiveAuthorizarion
  - CancelAuthorization
- *Watcher List* node includes the following transaction
  - GetWatcherList
- *Attribute List* node includes the following transactions
  - CreateAttributeList
  - DeleteAttributeList
  - GetAttributeList
- *Contact List Addr* node indicates if the contacts list is valid for addressing of users in the following transactions
  - Subscribe
  - UnSubscribe
  - GetPresence
  - UpdatePresence

IM Feature

- *Send Msg* node includes the optional choices for the SendMessage and ForwardMessage transactions. All supported ID types must be included in the Service List i.e.
  - Group-ID: Support recipient as Group-ID and addressing by screen name
  - ContactList-ID: Support recipients listed by Contact List ID
- *Push Msg* node include the following transaction
  - PushMessage
- *Notify Msg* node include the following transaction
  - MessageNotification

- *Get Msg* node include the following transaction
  - GetMessage
- *Delivery Mtd* node include the following transaction
  - SetMessageDeliveryMethod
- *Get Msg List* node include the following transaction
  - GetMessageList without group functionality
- *Reject Msg* node include the following transaction
  - RejectMessage
- *Delivery Report* node include the following transaction
  - NotifyDeliveryStatusReport
- *Blocking* node include the following transactions
  - BlockUser
  - GetBlockedList
- *Group History* node indicates if the IM service element supports group chat cashing functionality.
- *Msg List* node includes the optional choices for the GetMessageList transaction (Undelivered messages)

Group Feature

- *Group Mgmt* node include the following transactions
  - CreateGroup
  - DeleteGroup
- *Get Member* node include the following transaction
  - GetJoinedMember
- *Member mgmt* node include the following transactions
  - AddGroupMember
  - GetGroupMember
  - RemoveGroupMember
  - MemberAccess
- *Reject list* node include the following transactions
  - RejectList

7.2. Primitives

7.2.1. The "GetServiceRequest" Primitive

The *"GetServiceRequest"* primitive is issued from the requestor server to discover the available services provided by the provider server.

Information Element	Req	Type	Description
Message-Type	M	GetServiceRequest	Message identifier
Meta-Information	M	Structure of Meta-information	The necessary meta-information in a service request defined in 5.1.

Table 14. Information elements in GetServiceRequest Primitive

7.2.2. The "ServiceList" Primitive

The *"ServiceList"* primitive is issued from the provider server to indicate its available services.

Information Element	Req	Type	Description
Message-Type	M	ServiceList	Message identifier
Meta-Information	C	Structure of Meta-information	The necessary meta-information in a service request defined in 5.1. Present if the provider initiates ServiceIndication.
Status-Info	C	Structure of Status-Primitive	The status information (see 5.2). Present if the requestor initiates GetServiceRequest.
Service-List	M	Structure	List of available services in a tree structure.

Table 15. Information elements in ServiceList Primitive

7.2.3. The "ServiceNegotiation" Primitive

The *"ServiceNegotiation"* primitive is issued from the requestor server to negotiate the desired services that will be committed and provided by the provider server. The provider server sends the *"ServiceAgreement"* primitive to confirm the agreed services with the requestor server.

Information Element	Req	Type	Description
Message-Type	M	ServiceNegotiation	Message identifier
Meta-Information	M	Structure of Meta-information	The necessary meta-information in a service request defined in 5.1.
Desired-Service-List	M	Structure	List of desired services in a tree structure
Desired-Sub-Protocol	O	String	Desired sub-protocol and its version for proprietary protocol extensions
Time-to-live	O	Integer in Seconds	Indicates the desired time-to-live of the service agreement

Table 16. Information elements in ServiceNegotiation Primitive

7.2.4. The "ServiceAgreement" Primitive

After the provider server receives the *"ServiceNegotiation"* primitive from the requestor server, the provider server shall send the *"ServiceAgreement"* primitive to confirm the agreed services with the requestor server.

Information	Req	Type	Description

Element			
Message-Type	M	ServiceAgreement	Message identifier
Status-Info	M	Structure of Status-Primitive	The status information (see 5.2).
Agreed-Service-List	M	Structure	List of agreed services in a tree structure
Agreed-Sub-Protocol	O	String	Agreed sub-protocol and its version for proprietary protocol extensions
Agreed-Time-to-live	O	Integer in Seconds	Indicates the agreed time-to-live of the service agreement

Table 17. Information elements in ServiceAgreement Primitive

7.3. Transactions

7.3.1. The "GetAvailableService" Transaction

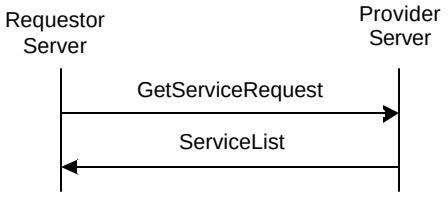

Figure 15. The "GetAvailableService" Transaction

SSP supports service discovery among the WV domains. The services include Common Features, Presence Service, Instant Messaging (IM) Service, Group Service and Shared Content Service that are defined in *"Features and Functions"* document.

The requestor server discovers the available services provided by the provider server through a "GetAvailableService" Transaction.

The requestor server sends a *"GetServiceRequest"* request to the provider server for the available services. After the provider server finishes processing the request, it sends a "*ServiceList*" response to the requestor server with the available service information.

Primitive	Direction
GetServiceRequest	Requestor Server → Provider Server
ServiceList	Requestor Server ← Provider Server

Table 18. Primitive Directions for GetAvailableService Transaction

7.3.2. The "ServiceIndication" Transaction

Figure 16. The "ServiceIndication" Transaction

The provider server also informs the requestor server of any change of the available services through a "ServiceIndication" Transaction. It depends on the offline service agreement between two domains to decide what are the following actions to be taken.

The provider server sends a *"ServiceList"* request to the requestor server and indicates the available services on-the-fly.

Primitive	Direction
ServiceList	Requestor Server ← Provider Server

Table 19. Primitive Directions for ServiceIndication Transaction

7.3.3. The "SetServiceAgreement" Transaction

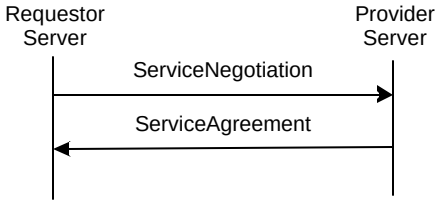

Figure 17. The "SetServiceAgreement" Transaction

The service agreement between the requestor and provider servers is established through a "SetServiceAgreement" Transaction.

The *"ServiceNegotiation"* request is issued from the requestor server to request and negotiate the agreement on the services that will be committed and provided by the provider server. The provider server sends the *"ServiceAgreement"* response to confirm the agreement with the requestor server.

After service agreement is confirmed, the servers may perform interoperable IMPS services.

Primitive	Direction
ServiceNegotiation	Requestor Server → Provider Server
ServiceAgreement	Requestor Server ← Provider Server

Table 20. Primitive Directions for SetServiceAgreement Transaction

7.4. Status Code

- Version Not Supported (505)

8. Interoperability Management – User Profile Management

These transactions are needed for the complementary services.

8.1. User Profile

User Profile consists of general user information and service-specific user information. The general user information includes the services to which the user subscribes, the service status (active / inactive), the privacy status with regard to network service capabilities (e.g. user location, user interaction), terminal capabilities, user account status etc. The service-specific user information includes the user-related information for each specific service element.

The general user information is defined as follows:

General UP Attribute	Value	Description
User.Account.Status	"ON" | "OFF"	Status of user account – active or inactive
User.Privacy.Location	"ON" | "OFF"	Status of location privacy – private or not
User.Privacy.Interaction	"ON" | "OFF"	Status of Interaction privacy – private or not
Services.Common	"YES" | "NO"	Whether or not Common service is subscribed
Services.Common.PSE	Domain	PSE of Common service. See 3.3.1 for Domain definition.
Services.Common.Status	"ON" | "OFF"	Status of Common service – active or inactive
Services.IM	"YES" | "NO"	Whether or not IM service is subscribed
Services.IM.PSE	Domain	PSE of IM service. See 3.3.1 for Domain definition.
Services.IM.Status	"ON" | "OFF"	Status of IM service – active or inactive
Services.Presence	"YES" | "NO"	Whether or not Presence service is subscribed
Services.Presence.PSE	Domain	PSE of Presence service. See 3.3.1 for Domain definition.
Services.Presence.Status	"ON" | "OFF"	Status of Presence service – active or inactive
Services.Group	"YES" | "NO"	Whether or not Group service is subscribed
Services.Group.PSE	Domain	PSE of Group service. See 3.3.1 for Domain definition.
Services.Group.Status	"ON" | "OFF"	Status of Group service – active or inactive
Services.Content	"YES" | "NO"	Whether or not Content service is subscribed
Services.Content.PSE	Domain	PSE of Content service. See 3.3.1 for Domain definition.
Services.Content.Status	"ON" | "OFF"	Status of Content service – active or inactive

Terminal.Delivery	"PUSH" | "NOTIFY"	Preferred message delivery method in client
Terminal.Content.type	MIME {, MIME }	Supported MIME types in client. See RFC 2045, RFC 2046 and WAP Forum for standard MIME.
Terminal.Content.encoding	encoding {, encoding }	Supported transfer encoding in client. See RFC 2045 for standard "transfer-encoding".
Terminal.Content.length	Integer in Byte	Supported message size in client for "PUSH"
Terminal.Content.protocol	Protocol {, Protocol }	Supported out-band protocol in client for binary message retrieval.
x.key	String	A service provider may define new key-values. These service provider specific keys are prefixed with x[.].

Table 21. General User Profile

Each piece of user profile information is organized in a "(name, value)" pair. The General User Profile is the list of "(name, value)" pairs, which are separated with "; ". An example of a General User Profile is as follows:

(User.Account.Status, ON); (Services.IM, ON); (Services.IM.PSE, im.wv.com); (Services.IM.Status, ON); (Terminal.Delivery, PUSH); (Terminal.Content.type, text/plain; charset=US-ASCII, text/xml; charset=UTF-8, image/wbmp); (Terminal.Content.encoding, BASE64); (Terminal.Content.length, 256); (Terminal.Content.protocol, HTTP, SIP, RTP, RTSP)); (x.MaxNumberOfConatctLists, 100)

8.2. Primitives

8.2.1. The "GetUserProfileRequest" Primitive

The "*GetUserProfileRequest*" primitive is issued to discover the available user profile information.

Information Element	Req	Type	Description
Message-Type	M	GetUserProfileRequest	Message identifier
Meta-Information	M	Structure of Meta-information	The necessary meta-information in a service request defined in 5.1.
User-ID-List	M	Structure	Identifies the users whose User Profiles are requested. If it is empty, all users' User Profiles are requested.

Table 22. Information elements in GetUserProfileRequest Primitive

8.2.2. The "UserProfile" Primitive

The *"UserProfile"* primitive is issued from the provider server to provide the user profile information.

Information Element	Req	Type	Description
Message-Type	M	UserProfile	Message identifier
Status-Info	M	Structure of Status-Primitive	The status information (see 5.2).
User-Profile-List	M	Structure of User-Profile	A list of User Profiles. Each User profile contains User-ID and a list of (name, value) pairs.

Table 23. Information elements in UserProfile Primitive

8.2.3. The "UpdateUserProfileRequest" Primitive

The *"UpdateUserProfileRequest"* primitive is issued to update the user profile information.

Information Element	Req	Type	Description
Message-Type	M	UpdateUserProfileRequest	Message identifier
Meta-Information	M	Structure of Meta-Information	The meta-information (see 5.1).
Updated-User-Profile-List	M	Structure of User-Profile	A list of User Profiles. Each User profile contains User-ID and a list of (name, value) pairs.

Table 24. Information elements in UpdateUserProfileRequest Primitive

8.3. Transactions

8.3.1. The "GetUserProfile" Transaction

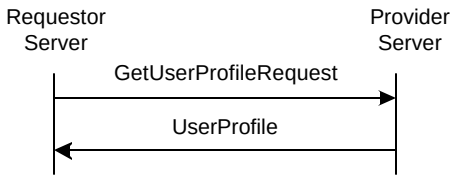

Figure 18. The "GetUserProfile" Transaction

SSP supports the exchange of user profile information among the WV domains including the list of services to which a user subscribes, the service status (active / inactive), privacy status with regard to network service capabilities (e.g. user location, user

interaction), terminal capabilities etc. The user profile information is discovered through a "GetUserProfile" transaction.

The *"GetUserProfileRequest"* request is issued from the requestor server to request the user profile information from the provider server. The provider server sends the "*UserProfile"* response to provide the requestor server with the user profile information.

Primitive	Direction
GetUserProfileRequest	Requestor Server → Provider Server
UserProfile	Requestor Server ← Provider Server

Table 25. Primitive Directions for GetUserProfile Transaction

8.3.2. The "UpdateUserProfile" Transaction

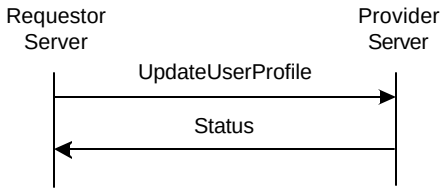

Figure 19. The "UpdateUserProfile" Transaction

The requestor server may update the user profile information in the provider server through an "UpdateUserProfile" Transaction.

The requestor server sends an *"UpdateUserProfile"* request to the provider server and provides the updated user profile information. After the provider server finishes processing the request, it sends a *"Status"* response to the requestor server and confirms that it has updated the user profile information.

Primitive	Direction
UpdateUserProfile	Requestor Server → Provider Server
Status	Requestor Server ← Provider Server

Table 26. Primitive Directions for UpdateUserProfile Transaction

8.4. Status Code

- Unknown user (531)

9. Service Relay – Common IMPS Features

SSP supports the service relay among the WV servers and the SSP Gateways including the functional relay of the common IMPS features, contact list, presence features, IM features, group features and shared content features that are defined in "*Features and Functions*" document.

9.1. Overview

This chapter focuses on the functional relay of common IMPS features. Because of the server interoperation nature, the SSP has its own requirement on meta-information and information elements in the primitives at transaction level. The complete primitives and transaction flows of common IMPS features at SSP semantics level has been defined in the following two sections.

Please refer to the CSP document so as to conclude how to relay the common IMPS features from client-server interaction (CSP) to server-server interoperation (SSP).

9.2. Primitives

9.2.1. The "SearchRequest" Primitive

The "*SearchRequest*" primitive is used for a user to search for users or groups based on different properties of the user or group. The user may limit the number of search results retrieved at one time. The user may continue the search and go through all the results.

The search is performed using a list of one or more Search-Pairs. 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, logical AND operation is assumed between the different pairs. Every Search-Element may be present only once within the same search request.

The result of a user search is always user-ID. Similarly, the result of a group search is always group-ID.

Search-Element for User Search (the result is always user-ID) is listed as follows:

Search-Element	Description
USER_ID	The *Search-String* is a substring of a user-ID.
USER_FIRST_NAME	The *Search-String* is a substring of a user's firstname.
USER_LAST_NAME	The *Search-String* is a substring of a user's lastname.
USER_EMAIL_ADDRESS	The *Search-String* is a substring of a user's e-mail address.
USER_ALIAS	The *Search-String* is a substring of a user's alias.

USER_MOBILE_NUMBER	The *Search-String* is a mobile number. [E.164].

Search-Element for Group Search (the result is always group-ID) is listed as follows:

Search-Element	Description
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* is a substring of a user-ID.

It is possible to search for any user or limit the search to users that are logged in to the system, which is specified by the Search-Online-Status information element. This element is ignored when there are no user related Search-Elements in the request. If the element is missing while there are user related Search-Elements in the request, both online and offline users are requested.

Information Element	Req	Type	Description
Message-Type	M	SearchRequest	Message identifier
Meta-Information	M	Structure of Meta-Information	The meta-information (see 5.1).
Search-Pair-List	C	Structure	Search criteria in terms of properties. It is present only in the 1st search request.
Search-Online-Status	C	Boolean	Retrieves only results for online users if value is TRUE. Present only in the 1st search request.
Search-Limit	C	Integer	Indicates the number of maximum search results that can be received at one time. It is 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 it is not present in the 1st search request.
Search-Index	C	Integer	Indicates that the results shall be sent starting from this particular index. It is present only when the search is continued.

Table 27. Information elements in SearchRequest Primitive

9.2.2. The "SearchResponse" Primitive

Information Element	Req	Type	Description
Message-Type	M	SearchResponse	Message identifier
Status-Info	M	Structure of Status-Primitive	The status information (see 5.2).
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 current findings.
Search-Index	M	Integer	Indicates the index of the last result. This provides the user with the information where to continue the next search.
Search-Results	C	Structure	Search results.

Table 28. Information elements in SearchResponse Primitive

9.2.3. The "StopSearchRequest" Primitive

The *"StopSearchRequest"* primitive is used for a user in the requestor server to indicate to the provider server that the search and / or its result is not needed any more from a previously issued search request.

Information Element	Req	Type	Description
Message-Type	M	StopSearchRequest	Message identifier
Meta-Information	M	Structure of Meta-Information	The meta-information (see 5.1).
Search-ID	M	String	Identifies the search to be invalidated.

Table 29. Information elements in StopSearchRequest Primitive

9.2.4. The "InviteRequest" Primitive

The *"InviteRequest"* primitive is used for the user in the requestor server to invite a list of other users to join a discussion / chat group, or to exchange messages, or to share presence information, or to share content.

The invited user may be a single user identified by its User-ID or Screen-Name. A list of users may be invited using a Contact-List-ID or Group-ID.

Information Element	Req	Type	Description
Message-Type	M	InviteRequest	Message identifier
Meta-Information	M	Structure of Meta-Information	The meta-information (see 5.1).
Invite-ID	M	String	Identifies this invitation.

Invite-Type	M	Enum {"GR", "IM", "PR", "SC" }	Inviting for Group/chat (GR), Messaging (IM), Presence (PR), or Content (SC).
Inviting-User	M	Structure	Identifies the requesting user who sends the invitation (User-ID and / or Screen-Name)
Invited-User	M	Structure	Identifies the user(s) to be invited (User-ID and / or Screen-Name, or Contact-List-ID)
Invite-Group-ID	C	String	Identifies the group. It is mandatory if InviteGroup (GR). Otherwise, not present.
Invite-Presence-Attribute-List	CO	Structure	Identifies the Presence Attributes that the inviter wants to share with the invitees. It is optional if InvitePresence (PR). Otherwise, not present.
Invite-Content-ID-List	CO	Structure	Identifies the related shared content as a list of URLs. It is optional if InviteContent (SC). Otherwise, not present.
Invite-Reason	O	String	Textual description of the invitation.
Validity	O	Integer in seconds	Indicates the interval in which the invitation is valid.

Table 30. Information elements in InviteRequest Primitive

9.2.5. The "InviteResponse" Primitive

The *"InviteResponse"* primitive is used for the provider server to return the result of the invitation to the requestor server, which represents the inviting user.

Information Element	Req	Type	Description
Message-Type	M	InviteResponse	Message identifier.
Status-Info	M	Structure of Status-Primitive	The status information (see 5.2).
Invite-ID	M	String	Identifies this invitation.
Inviting-User	M	Structure	Identifies the requesting user who sends the invitation (User-ID and / or Screen-Name)
Invite-Acceptance	M	Boolean	Indicates if the user accepts the invitation or not.
Responding-User	M	Structure	Identifies the responding invited user (User-ID and / or Screen-Name)

Invite-Response	O	String	Textual description, why the invited user accepted/rejected the invitation.

Table 31. Information elements in InviteResponse Primitive

Each tuple { Invite-Acceptance, Responding-User, Invite-Response } represents the response from one invitee. There may be multiple tuples { Invite-Acceptance, Responding-User, Invite-Response } in one "InviteResponse" primitive if the provider server is able to collect the response from the invited users in a reasonable time and combine the multiple responses in one primitive in order to reduce the traffic overhead between the servers.

9.2.6. The "InviteUserRequest" Primitive

The *"InviteUserRequest"* primitive is used for the provider server to invite the user(s) in the requestor server to join a discussion / chat group, or to exchange messages, or to share presence information, or to share content.

Information Element	Req	Type	Description
Message-Type	M	InviteUserRequest	Message identifier
Meta-Information	M	Structure of Meta-Information	The meta-information (see 5.1).
Invite-ID	M	String	Identifies this invitation.
Invite-Type	M	Enum { "GR", "IM", "PR", "SC" }	Inviting for Group/chat (GR), Messaging (IM), Presence (PR), or Content (SC).
Inviting-User	M	Structure	Identifies the requesting user who sends the invitation (User-ID and / or Screen-Name)
Invited-User	M	Structure	Identifies the user(s) to be invited (User-ID and / or Screen-Name, or List-of-User-IDs)
Invite-Group-ID	C	String	Identifies the group. It is mandatory if InviteGroup (GR). Otherwise, not present.
Invite-Presence-Attribute-List	CO	Structure	Identifies the Presence Attributes that the inviter wants to share with the invitees. It is optional if InvitePresence (PR). Otherwise, not present.
Invite-Content-ID-List	CO	Structure	Identifies the related shared content as a list of URLs. It is optional if InviteContent (SC). Otherwise, not present.
Invite-Reason	O	String	Textual description of the invitation.

Validity	O	Integer in seconds	Indicates the interval in which the invitation is valid.

Table 32. Information elements in InviteUserRequest Primitive

9.2.7. The "InviteUserResponse" Primitive

The *"InviteUserResponse"* primitive is used for the requestor server, which represents the invited users, to return the result of the invitation to the provider server.

Information Element	Req	Type	Description
Message-Type	M	InviteUserResponse	Message identifier
Status-Info	M	Structure of Status-Primitive	The status information (see 5.2).
Invite-ID	M	String	Identifies this invitation.
Inviting-User	M	Structure	Identifies the requesting user who sends the invitation (User-ID, Screen-Name)
Invite-Acceptance	M	Boolean	Indicates if the user accepts the invitation or not.
Responding-User	M	Structure	Identifies the responding invited user (User-ID and / or Screen-Name)
Invite-Response	O	String	Textual description, why the invited user accepted/rejected the invitation.

Table 33. Information elements in InviteUserResponse Primitive

Each tuple { Invite-Acceptance, Responding-User, Invite-Response } represents the response from one invitee. There may be multiple tuples { Invite-Acceptance, Responding-User, Invite-Response } in one "InviteUserResponse" primitive if the requestor server, which represents the invited users, is able to collect the response from the invited users in a reasonable time and combine the multiple responses in one primitive in order to reduce the traffic overhead between the servers.

9.2.8. The "CancelInviteRequest" Primitive

The *"CancelInviteRequest"* primitive is used for the user in the requestor server to cancel its previous invitation.

Information Element	Req	Type	Description
Message-Type	M	CancelInviteRequest	Message identifier
Meta-Information	M	Structure of Meta-Information	The meta-information (see 5.1).
Invite-ID	M	String	Identifies the invitation.

Canceling-User	M	Structure	Identifies the requesting user who cancels the invitation (User-ID and / or Screen-Name)
Canceled-User	M	Structure	Identifies the user(s) to whom the invitation will be canceled (User-ID and / or Screen-Name, or Contact-List-ID)
Canceled-Content-ID-List	C	Structure	Identifies the related shared content as a list of URLs which will be canceled.
Cancel-Reason	O	String	Textual description of the cancel.

Table 34. Information elements in CancelInviteRequest Primitive

9.2.9. The "CancelInviteUserRequest" Primitive

The *"CancelInviteUserRequest"* primitive is used for the provider server to cancel its previous invitation to the users in the requestor server.

Information Element	Req	Type	Description
Message-Type	M	CancelInviteUserRequest	Message identifier
Meta-Information	M	Structure of Meta-Information	The meta-information (see 5.1).
Invite-ID	M	String	Identifies the invitation.
Canceling-User	M	Structure	Identifies the requesting user who cancels the invitation (User-ID and / or Screen-Name)
Canceled-User	M	Structure	Identifies the user(s) to whom the invitation will be canceled (User-ID and / or Screen-Name, or List-of-User-IDs)
Canceled-Content-ID-List	C	Structure	Identifies the related shared content as a list of URLs which will be canceled.
Cancel-Reason	O	String	Textual description of the cancel.

Table 35. Information elements in CancelInviteUserRequest Primitive

9.2.10. The "VerifyUseridRequest" Primitive

The *"VerfiyUseridRequest"* primitive is used for the requestor server to verify that userid(s) are valid in the provider server.

Information Element	Req	Type	Description

Message-Type	M	VeifyUseridRequest	Message identifier
Meta-Information	M	Structure of Meta-Information	The meta-information (see 5.1).
Verify-User-ID-List	M	Structure	The list contains the User-ID's and optionally the time when the userid was created

Table 36. Information elements in VerfifyUseridRequest Primitive

9.2.11. The "VerifyUseridResponse" Primitive

The *"VerifyUseridResponse"* primitive is used for the provider server to return the valid userid's.

Information Element	Req	Type	Description
Message-Type	M	VerifyUseridResponse	Message identifier.
Status-Info	M	Structure of Status-Primitive	The status information (see 5.2).
Verify-User-ID-List	M	Structure	The list contains the userids that are in use and the time when the userid was created.

Table 37. Information elements in VerifyUseridResponse Primitive

9.3. Transactions

9.3.1. The "GeneralSearch" Transaction

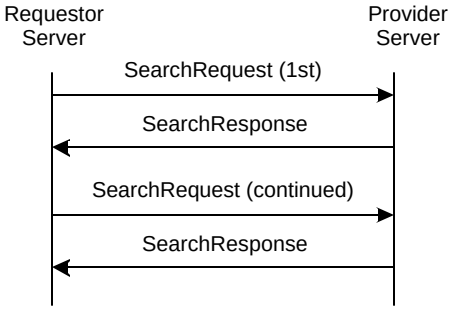

Figure 20. The "GeneralSearch" Transaction

The requestor server sends the *"SearchRequest"* message to the provider server including the Search-Pair-List, the Search-Online-Status (T-Online, F-Offline, N/A-both), the type of the search and the Search-Limit (maximum number of results at a time). The provider server responds with the *"SearchResponse"* message, which includes the Status of the search. If the search is successful, it includes 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).

The requestor server may continue the search. In this case the *"SearchRequest"* message includes only the Search-ID and the Search-Index. The provider server responds with the "*SearchResponse"*, but the message includes only the Result, the Search-Index, the Search-Findings and the Search-Results.

The requestor server may modify the Search-Index value, so that the search may be continued at a different place. The Search-Index is valid until a new search is performed or the session ends (a previous search is invalidated when a new search is started).

Primitive	Direction
SearchRequest	Requestor Server → Provider Server
SearchResponse	Requestor Server ← Provider Server

Table 38. Primitive Directions for GeneralSearch Transaction

9.3.2. The "StopSearch" Transaction

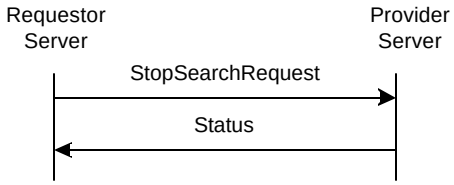

Figure 21. The "StopSearch" Transaction

The "StopSearch" transaction is used for the requestor server to indicate to the provider server that the search and / or the results are not needed any more from a previously issued search request. The requestor server sends the *"StopSearchRequest"* message to the provider server including the Search-ID. The provider server invalidates the indicated search, and replies with a Status message. The invalidated Search-ID cannot be used after invalidation.

Primitive	Direction
StopSearchRequest	Requestor Server → Provider Server
Status	Requestor Server ← Provider Server

Table 39. Primitive Directions for StopSearch Transaction

9.3.3. The "Invitation" Transaction

A user may invite other user(s) to join a discussion / chat group, or to exchange messages, or to share presence values list, or to share content.

There are two service models and the corresponding transaction flows.

9.3.3.1 Basic Invitation transaction

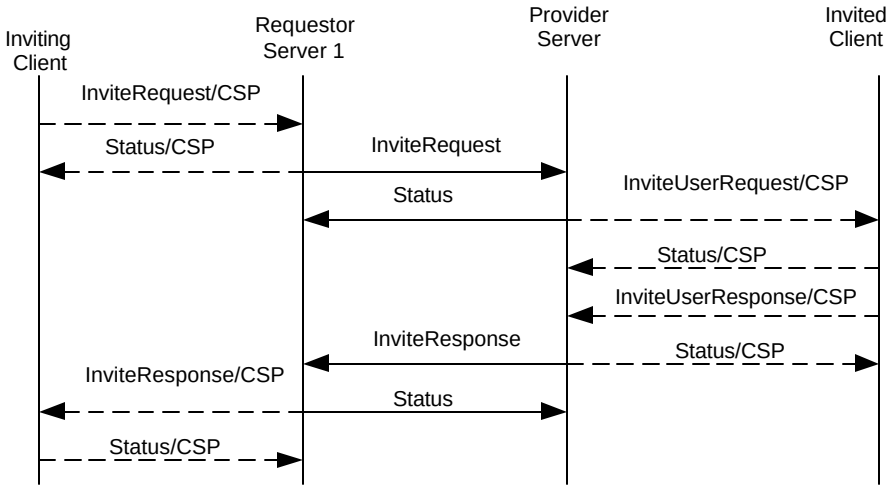

Figure 22. The “Basic Invitation” Transaction

The requestor server 1 is the Home Domain of the inviting user, the provider server is the Home Domain of the invited user.

Primitive	Direction
InviteRequest	Requestor Server 1 → Provider Server
Status	Requestor Server 1 ← Provider Server
InviteResponse	Requestor Server 1 ← Provider Server
Status	Requestor Server 1 → Provider Server

Table 40. Primitive Directions for Basic Invitation Transaction

9.3.3.2 Complementary Invitation transaction

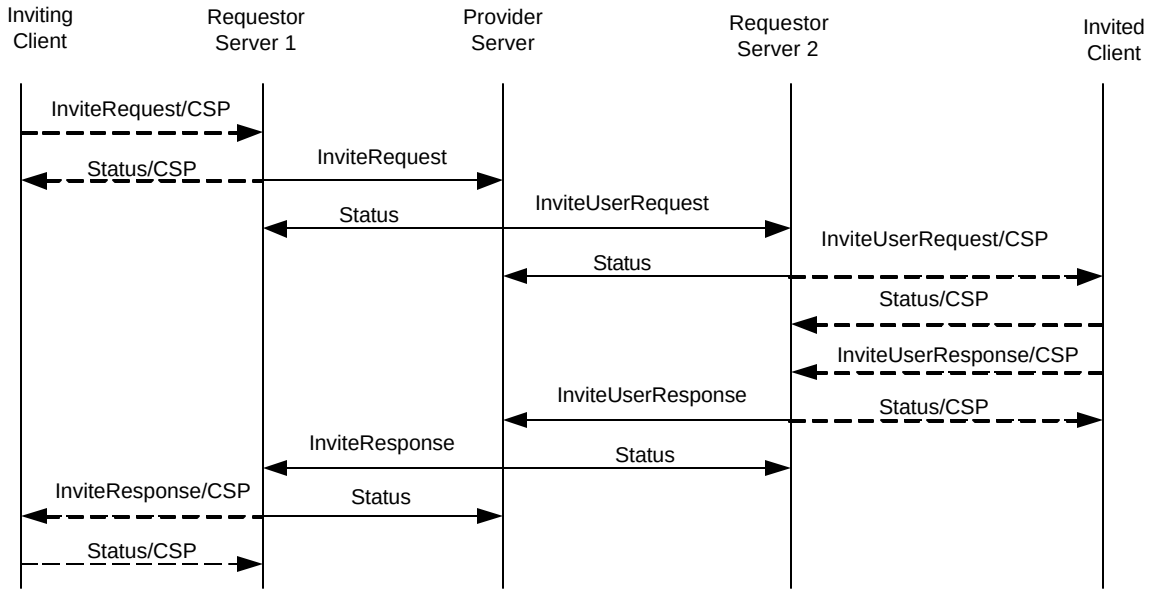

Figure 23. The "Complementary Invitation" Transaction

In this service model the requestor server 1 is the Home Domain of the inviting user, the provider server is the PSE of the invited user in another Domain, and the requestor server 2 is the Home Domain of the invited user. The transaction flow is as follows.

Primitive	Direction
InviteRequest	Requestor Server 1 → Provider Server
Status	Requestor Server 1 ← Provider Server
InviteUserRequest	Provider Server → Requestor Server 2
Status	Provider Server ← Requestor Server 2
InviteUserResponse	Provider Server ← Requestor Server 2
Status	Provider Server → Requestor Server 2
InviteResponse	Requestor Server 1 ← Provider Server
Status	Requestor Server 1 → Provider Server

Table 41. Primitive Directions for Complementary Invitation transaction

The general description of the transactions

The requestor server 1, which represents the inviting user, sends the provider server the "*InviteRequest*" message with the ID of the invitation, the invitation type, the inviting User-ID and/or Screen-Name, the list of user(s) to be invited specified by User-IDs and/or Screen-Names, the ID of the subject, and optionally the reason for the invitation (a short text).

The provider server responds the requestor server 1 with a Status message. The provider server also sends "*InviteUserRequest*" message to every requestor server 2, which represents one or several of the invited users. The "*InviteUserRequest*" message contains

the ID of the invitation, the invitation type, the inviting User-ID and/or Screen-Name, the list of user(s) to be invited specified by User-IDs and/or Screen-Names, the ID of the subject, and optionally the reason for the invitation (a short text).

Each requestor server 2 responds the provider server with a Status message.

The invited user may accept or reject the invitation, and the requestor server 2, which represents the invited users, responds the provider server with the *"InviteUserResponse"* message with the ID of the invitation, the acceptance indicator, the User-ID and/or Screen-Name of the responding invited user, and optionally the short response text.

The provider server responds the requestor server 2 with a Status message. The provider server will send the *"InviteResponse"* message to the requestor server 1, which represents the inviting user. The *"InviteResponse"* message contains the ID of the invitation, the acceptance indicator, the User-ID and/or Screen-Name of the responding invited user, and optionally the short response text.

The requestor server 1 responds the provider server with a Status message.

Each tuple { Invite-Acceptance, Responding-User, Invite-Response } represents the response from one invitee. There may be multiple tuples { Invite-Acceptance, Responding-User, Invite-Response } in one *"InviteUserResponse"* or *"InviteResponse"* primitive if the requestor server 2 or the provider server is able to collect the response from the invited users in a reasonable time and combine the multiple responses in one primitive in order to reduce the traffic overhead between the servers.

While in general there is no mandatory requirement about how an invited user shall act according to the acceptance indicator within its response in the scope of this function, it is recommended that the invited user should act consistently according to its response.

The subject of the invitation may be a group, messaging, a shared content, or presence. In case of presence the user may include a list of presence attributes that he/she is willing to share with the other party. Note that there is no actual presence attribute sharing that has been done, the transaction is only informational. Similarly, in case of group, messaging, or shared content invitations the actual action is not taken, it is up to the user to do it manually (the invitation is only informational).

9.3.4. The "CancelInvitation" Transaction

A user may cancel any previous invitations.

9.3.4.1 Basic Cancel Invitation transaction

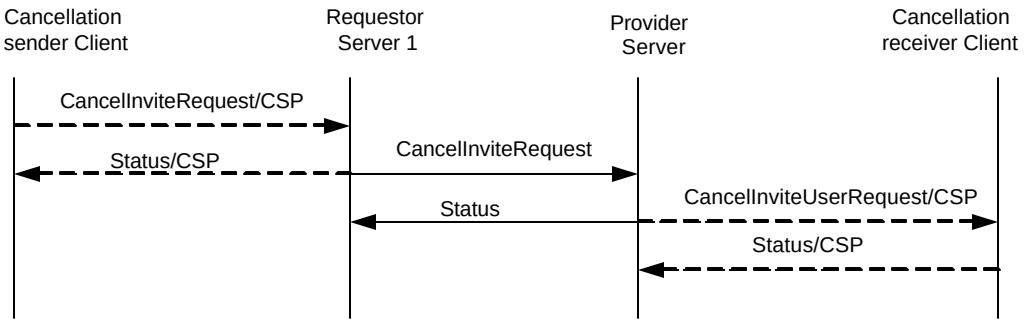

Figure 24. The "Basic CancelInvitation" Transaction

The requestor server 1 is the Home Domain of the invitation canceling user, the provider server is the Home Domain of the invitation cancellation receiver user.

Primitive	Direction
CancelInviteRequest	Requestor Server 1 → Provider Server
Status	Requestor Server 1 ← Provider Server

Table 42. Primitive Directions for Basic CancelInvitation Transaction

9.3.4.2 Complementary Cancel Invitation transaction

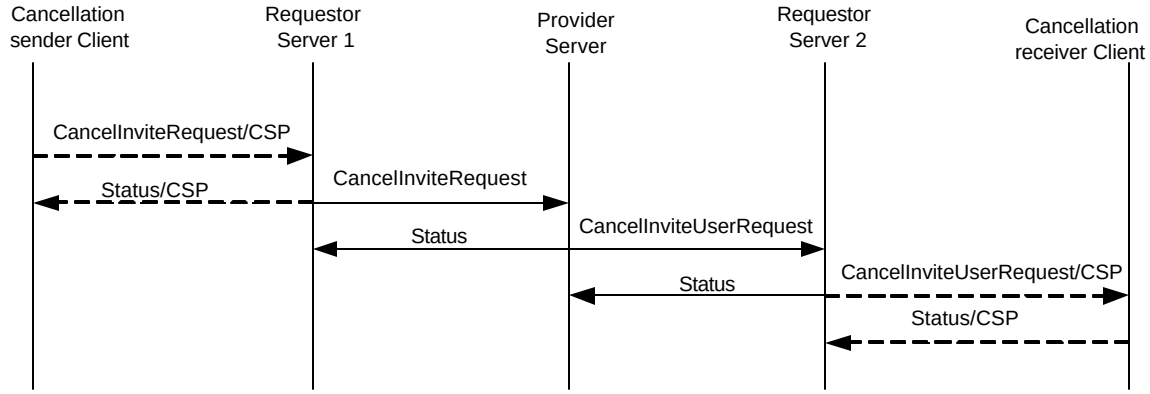

Figure 25. The "Complementary CancelInvitation" Transaction

In this service model the requestor server 1 is the Home Domain of the invitation canceling user, the provider server is the PSE of the invitation cancellation recipient in another Domain, and the requestor server 2 is the Home Domain of the invitation cancellation recipient. The transaction flow is as follows.

Primitive	Direction
CancelInviteRequest	Requestor Server 1 → Provider Server
Status	Requestor Server 1 ← Provider Server
CancelInviteUserRequest	Provider Server → Requestor Server 2
Status	Provider Server ← Requestor Server 2

Table 43. Primitive Directions for Complementary CancelInvitation Transaction

The general description of the transactions

The requestor server 1, which represents the inviting user, sends the provider server the "*CancelnviteRequest*" message with the ID of the invitation, the inviting User-ID and/or Screen-Name, the list of user(s) to be notified about the cancellation specified by User-IDs and/or Screen-Names, and optionally the reason for the cancellation (a short text).

The provider server responds the requestor server 1 with a Status message. The provider server also sends *"CancelnviteUserRequest"* message to every requestor server 2, which represents one or several of the invited users. The "*CancelnviteUserRequest*" message contains the ID of the invitation, the inviting User-ID and/or Screen-Name, the list of user(s) to be notified about the cancellation specified by User-IDs and/or Screen-Names, and optionally the reason for the invitation (a short text).

The requestor server 2, which represents the canceled users, responds the provider server with the Status message.

Note that the "CancelInvitation" transaction makes sense only for the scope of presence sharing and content sharing invitations.

9.3.5. The "VerifyUserid" Transaction

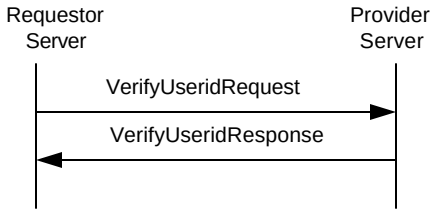

Figure 26. The "VerifyUserid" Transaction

The "VerifyUserid" transaction is used by the requestor server to verify that a User-ID is in use at the provider server, i.e. the Home Domain of the User-ID. The transaction is used before the User-ID is stored in the requestor sever to ensure that all locally stored User-ID's are valid. The VerifyUserid response contains the subset of User-ID'(s) in use and the time when the User-ID was created. The time information is used to verify that the locally stored User-ID belongs to the same end-user on both the requestor and provider server or if it has been recycled on the provider side and given to a new end-user. If the time is not present in the request it is assumed that the requestor server just want to verify that the User-ID is in use.

Primitive	Direction
VerifyUseridRequest	Requestor Server → Provider Server
VerifyUseridResponse	Requestor Server ← Provider Server

Table 44. Primitive Directions for the VerifyUserid Transaction

9.4. Status Code

9.4.1. "GeneralSearch" Transaction

- Unable to parse criteria. (Invalid Search-Element) (402)
- Service Not Supported (405)
- 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)
- Server search limit is exceeded (610)

9.4.2. "StopSearch" Transaction

- Service Not Supported (405)
- Invalid Search-ID (424)

9.4.3. "Invitation" Transaction

- Invalid invitation type(402).
- Service Not Supported (405)
- Unknown user (ID or screen-name) (531).
- Not logged in (604)
- Group does not exist (800).

9.4.4. "CancelInvitation" Transaction

- Invalid invitation type (402).
- Service Not Supported (405)
- Invalid invitation ID (423).
- Unknown user (ID or screen-name) (531).
- Not logged in (604)

10. Service Relay – Contact List Features

10.1. Overview

A "*contact list*" is a created and maintained by a User so that the User may send messages to the "*contact list*" as recipient. The message will be delivered to every member in the particular "*contact list*". However, except the owner User, the other members of the "c*ontact list*" do not have any knowledge about the "contact list". Nor do they conduct any group functions.

In concept, the "*contact list*" is a special case and subset of Private Group, and is also a special case of Restricted Group. In practice, the "*contact list*" means two cases:

- Address book – the "*contact list*" contains a list of addresses, nicknames, and other relevant information of family members, friends, colleagues or other frequently contacted persons.
- Presence – the "*contact list*" is closely tied to the presence service. It is used for proactive presence authorization (the people on the list can get these presence attributes), and presence update (presence attributes of the people on the list).

A user may have any number of contact lists, thus the contact lists has their own IDs. The users do not know about (and cannot access at all) each other's contact list(s).

There are two properties for Contact List:

- Display-Name: a free text string given by user that can be presented in the user interface of the client
- Default: a Boolean set by user that indicates that the particular contact list is the default contact list.

When the user creates his/her first contact list, the server automatically sets that contact list as the default. The server may also create the first list automatically.

When the user has more than one contact lists in the system, the user may set any of his/her contact lists as the default contact list. When the user sets "Default" property of a contact list to "True", the "Default" property of the previously default contact list must be set to "False" automatically by the server.

Watchers list is a system defined contact list with the functionality limited to hold users that have subscribed to presence information including the subscribed attributes.

All users that have subscribed to presence information are present in the Watchers list, i.e. a user that is present in a contact list and has subscribed to one or more presence attributes is always present in the watchers list. A user whose reactive authorization request is accepted shall also be present in the watchers list. If the user does not indicate

specific attributes in his reactive authorization request, the Default Public Attribute List will be used for this user. Otherwise, the specific attribute list shall be associated with the subscriber.

The server shall maintain one Watcher List for each user.

This chapter focuses on the functional relay of Contac List features. Because of the server interoperation nature, the SSP has its own requirement on meta-information and information elements in the primitives at transaction level. The complete primitives and transaction flows of Contact List features at SSP semantics level has been defined in the following two sections.

Please refer to the CSP document so as to conclude how to relay the Contact List features from client-server interaction (CSP) to server-server interoperation (SSP).

These transactions below belong to the complementary service.

10.2. Primitives

10.2.1. The "CreateContactListRequest" Primitive

The *"CreateContactListRequest"* primitive is used to create a contact list.

In addition to the "Contact-List-ID" which identifies the contact list, the "*CreateContactListRequest*" primitive contains the initial properties (Display-Name, Default) and a "User-List" which identifies the initial users to be added to the contact list (User-ID, Nickname).

Information Element	Req	Type	Description
Message-Type	M	CreateContactList Request	Message identifier
Meta-Information	M	Structure of Meta-Information	The meta-information (see 5.1).
Contact-List-ID	M	String	Identifies the contact list.
Contact-List-Props	O	Structure	The initial properties of the contact list (Display-Name, Default).
User-List	O	Structure	Identifies the initial users to be added to the contact list (User-ID, Nickname).

Table 45. Information elements in CreateContactListRequest Primitive

10.2.2. The "DeleteContactListRequest" Primitive

The *"DeleteContactListRequest"* primitive is used to delete the contact list(s).

Information Element	Req	Type	Description

Message-Type	M	DeleteContactList Request	Message identifier
Meta-Information	M	Structure of Meta-Information	The meta-information (see 5.1).
Contact-List-ID-List	M	Structure	Identifies the contact list(s).

Table 46. Information elements in DeleteContactListRequest Primitive

10.2.3. The "GetContactListRequest" Primitive

The *"GetContactListRequest"* primitive is used for a user in the requestor server to retrieve the list of all Contact-List-IDs.

Information Element	Req	Type	Description
Message-Type	M	GetContactListRequest	Message identifier
Meta-Information	M	Structure of Meta-Information	The meta-information (see 5.1).

Table 47. Information elements in GetContactListRequest Primitive

10.2.4. The "GetContactListResponse" Primitive

The *"GetContactListResponse"* primitive returns a list of all Contact-List-IDs.

Information Element	Req	Type	Description
Message-Type	M	GetContactListResponse	Message identifier
Status-Info	M	Structure of Status-Primitive	The status information (see 5.2).
Contact-List-ID-List	C	Structure	The list of the Contact-List-IDs.
Default-C-List-ID	C	String	Identifies the default contact list.

Table 48. Information elements in GetContactListResponse Primitive

10.2.5. The "GetListMemberRequest" Primitive

The *"GetListMemberRequest"* primitive is used to retrieve the all members of a contact list.

Information Element	Req	Type	Description
Message-Type	M	GetListMemberRequest	Message identifier
Meta-Information	M	Structure of Meta-Information	The meta-information (see 5.1).
Contact-List-ID	M	String	Identifies the contact list.

Table 49. Information elements in GetListMemberRequest Primitive

10.2.6. The "AddListMemberRequest" Primitive

The *"AddListMemberRequest"* primitive is used to add the members to a contact list.

Information Element	Req	Type	Description
Message-Type	M	AddListMemberRequest	Message identifier
Meta-Information	M	Structure of Meta-Information	The meta-information (see 5.1).
Contact-List-ID	M	String	Identifies the contact list.
User-List	M	Structure	Identifies the users to be added to the contact list (User-ID, Nickname).

Table 50. Information elements in AddListMemberRequest Primitive

10.2.7. The "RemoveListMemberRequest" Primitive

The *"RemoveListMemberRequest"* primitive is used to remove the members from the contact list.

Information Element	Req	Type	Description
Message-Type	M	RemoveListMemberRequest	Message identifier
Meta-Information	M	Structure of Meta-Information	The meta-information (see 5.1).
Contact-List-ID	M	String	Identifies the contact list.
User-List	M	Structure	Identifies the users to be removed from the contact list (User-ID, Nickname).

Table 51. Information elements in RemoveListMemberRequest Primitive

10.2.8. The "ContactListMemberResponse" Primitive

The *"ContactListMemberResponse"* primitive returns a list of all members in the contact list.

Information Element	Req	Type	Description
Message-Type	M	ContactListMemberResponse	Message identifier
Status-Info	M	Structure of Status-Primitive	Status information (see 5.2).
User-List	M	Structure	Identifies the users in the contact list (User-ID, Nickname).

Table 52. Information elements in ContactListMemberResponse Primitive

10.2.9. The "GetListPropsRequest" Primitive

The *"GetListPropRequest"* primitive is used to retrieve the properties of a contact list.

Information Element	Req	Type	Description
Message-Type	M	GetListPropsRequest	Message identifier
Meta-Information	M	Structure of Meta-Information	The meta-information (see 5.1).
Contact-List-ID	M	String	Identifies the contact list.

Table 53. Information elements in GetListPropsRequest Primitive

10.2.10. The "SetListPropsRequest" Primitive

The *"SetListPropRequest"* primitive is used to set the properties of a contact list.

Information Element	Req	Type	Description
Message-Type	M	SetListPropsRequest	Message identifier
Meta-Information	M	Structure of Meta-Information	The meta-information (see 5.1).
Contact-List-ID	M	String	Identifies the contact list.
Contact-List-Props	M	Structure	The properties (Display-Name, Default) to be set.

Table 54. Information elements in SetListPropsRequest Primitive

10.2.11. The "ContactListPropsResponse" Primitive

The *"ContactListPropsResponse"* primitive returns a list of all members in the contact list.

Information Element	Req	Type	Description
Message-Type	M	ContactListPropsResponse	Message identifier
Status-Info	M	Structure of Status-Primitive	Status information (see 5.2).
Contact-List-Props	M	Structure	The properties of the contact list (Display-Name, Default).

Table 55. Information elements in ContactListPropsResponse Primitive

10.2.12. The "CreateAttrListRequest" Primitive

The *"CreateAttrListRequest"* primitive is used to create an attribute list, and attach the attribute list to some contact list(s) and / or user(s).

Information Element	Req	Type	Description
Message-Type	M	CreateAttrListRequest	Message identifier
Meta-Information	M	Structure of Meta-Information	The meta-information (see 5.1).
Presence-Attribute-List	M	Structure	A list of presence attributes.
Default-List	M	"Yes" | "No"	Indicates if the attributes are targeted to the default attribute list instead of a separate attribute list.
Contact-List-ID-List	C	Structure	Contact list(s) which the attribute list should be attached to.
User-ID-List	C	Structure	User(s) which the attribute list should be attached to.

Table 56. Information elements in CreateAttrListRequest Primitive

10.2.13. The "DeleteAttrListRequest" Primitive

The *"DeleteAttrListRequest"* primitive is used to delete the attribute list(s).

Information Element	Req	Type	Description
Message-Type	M	DeleteAttrListRequest	Message identifier
Meta-Information	M	Structure of Meta-Information	The meta-information (see 5.1).
Default-List	M	"Yes" | "No"	Indicates if the default attribute list should be cleared.
Contact-List-ID-List	C	Structure	Identifies the contact list(s) to remove the attribute list association
User-ID-List	C	Structure	Identifies the user(s) to remove the attribute list association

Table 57. Information elements in DeleteAttrListRequest Primitive

10.2.14. The "GetAttrListRequest" Primitive

The *"GeAttrListRequest"* primitive is used to retrieve the published or subscribed attributes associated with specific contact list(s) and / or user(s). If the user(s) or contact list(s) are not specified, the response shall include all user-specific and contact list-specific attributes.

Information Element	Req	Type	Description
Message-Type	M	GetAttrListRequest	Message identifier
Meta-Information	M	Structure of Meta-	The meta-information (see 5.1).

		Information	
Default-List	M	"Yes" | "No"	Indicates if the default attribute list should be retrieved ("YES") or not.
Contact-List-ID-List	O	Structure	Identifies the contact list(s) to retrieve the attribute list association
User-ID-List	O	Structure	Identifies the user(s) to retrieve the attribute list association

Table 58. Information elements in GetAttrListRequest Primitive

10.2.15. The "GetAttrListResponse" Primitive

The *"GetAttrListResponse"* primitive returns the presence attributes.

Information Element	Req	Type	Description
Message-Type	M	GetAttrListResponse	Message identifier
Status-Info	M	Structure of Status-Primitive	The status information (see 5.2).
Attribute-Association-List	O	Structure	A list of attribute list associations with the user and / or the contact list.
Default-Association-List	O	Structure	The list of presence attributes associated with the default list.

Table 59. Information elements in GetAttrListResponse Primitive

10.3. Transactions

10.3.1. The "CreateContactList" Transaction

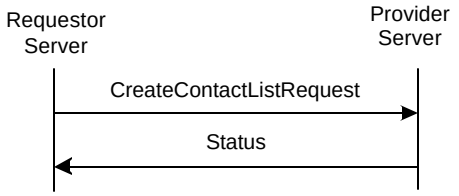

Figure 27. The "CreateContactList" Transaction

The requestor server sends a *"CreateContactListRequest"* to the provider server. The provider server shall create the contact list and respond with a Status message to the requestor server.

A user is able to create more than one contact list. There may be system specific limitations for the maximum number of lists per user. After the contact list is created, a user may create an attribute list for the contact list.

Primitive	Direction
CreateContactListRequest	Requestor Server → Provider Server

Status	Requestor Server ← Provider Server

Table 60. Primitive Directions for CreateContactList Transaction

10.3.2. The "DeleteContactList" Transaction

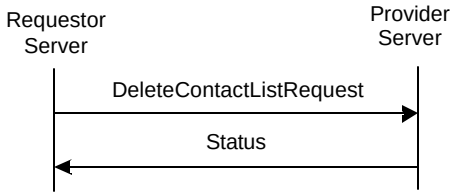

Figure 28. The "DeleteContactList" Transaction

The requestor server sends a *"DeleteContactListRequest"* to the provider server. The provider server shall delete the contact lists(s) and respond with a Status. The server should not unsubscribe the members implicitly, which means that if a contact list which has been subscribed to is deleted, the presence subscriptions should not be cancelled for the particular users.

A user may delete more than one contact list at one time.

Primitive	Direction
DeleteContactListRequest	Requestor Server → Provider Server
Status	Requestor Server ← Provider Server

Table 61. Primitive Directions for DeleteContactList Transaction

10.3.3. The "GetContactList" Transaction

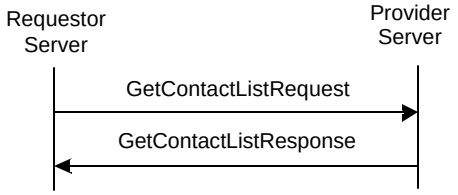

Figure 29. The "GetContactList" Transaction

The "GetContactList" transaction is used for the requestor server to retrieve the list of all Contact-List-IDs of the user. The requestor server sends a *"GetContactListRequest"* request. The provider server returns a *"GetContactListResponse"* primitive with a list of all Contact-List-ID's and the default contact list ID of the user.

Primitive	Direction
GetContactListRequest	Requestor Server → Provider Server
GetContactListResponse	Requestor Server ← Provider Server

Table 62. Primitive Directions for GetContactList Transaction

10.3.4. The "GetListMember" Transaction

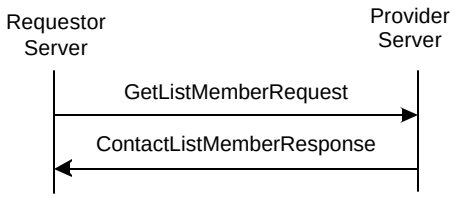

Figure 30. The "GetListMember" Transaction

The "GetListMember" transaction is used to retrieve all members of a contact list. The requestor server sends a *"GetListMemberRequest"* to the provider server. The provider responds with a *"ContactListMemberResponse"* to the requestor server containing the list of all members of the contact list.

Primitive	Direction
GetListMemberRequest	Requestor Server → Provider Server
ContactListMemberResponse	Requestor Server ← Provider Server

Table 63. Primitive Directions for GetListMember Transaction

10.3.5. The "AddListMember" Transaction

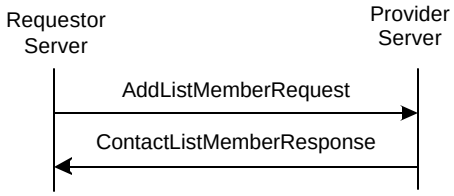

Figure 31. The "AddListMember" Transaction

The requestor server sends an *"AddListMemberRequest"* to the provider server to add one or more members in a contact list. The provider server shall respond with a *"ContactListMemberResponse"* to the requestor server containing the list of all members of the contact list.

Primitive	Direction
AddListMemberRequest	Requestor Server → Provider Server
ContactListMemberResponse	Requestor Server ← Provider Server

Table 64. Primitive Directions for AddListMember Transaction

10.3.6. The "RemoveListMember" Transaction

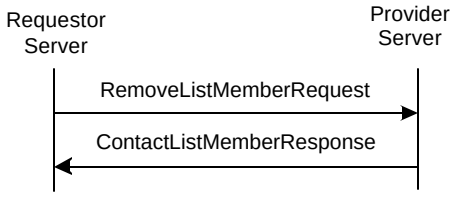

Figure 32. The "RemoveListMember" Transaction

The requestor server sends a *"RemoveListMemberRequest"* to the provider server. The provider server shall delete the user(s) from the specified contact list, and returns the list of all members of the contact list in the *"ContactListMemberResponse"*.

Primitive	Direction
RemoveListMemberRequest	Requestor Server → Provider Server
ContactListMemberResponse	Requestor Server ← Provider Server

Table 65. Primitive Directions for RemoveListMember Transaction

10.3.7. The "GetListProperties" Transaction

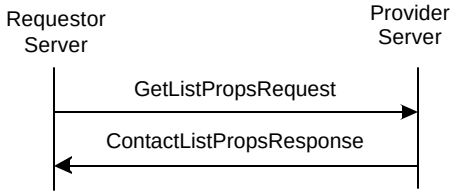

Figure 33. The "GetListProperties" Transaction

The "GetListProperties" transaction is used to retrieve the properties of a contact list (Display-Name, Default). The requestor server sends a *"GetListPropsRequest"* to the provider server. The provider responds with a *"ContactListPropsResponse"* to the requestor server containing the properties.

Primitive	Direction
GetListPropsRequest	Requestor Server → Provider Server
ContactListPropsResponse	Requestor Server ← Provider Server

Table 66. Primitive Directions for GetListProperties Transaction

10.3.8. The "SetListProperties" Transaction

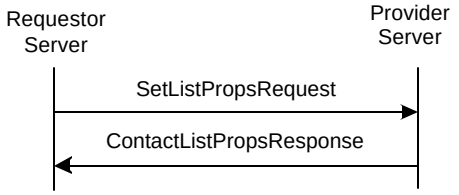

Figure 34. The "SetListProperties" Transaction

The "SetListProperties" transaction is used to set the properties of a contact list (Display-Name, Default), i.e. to set the display name, or to set a default contact list. The requestor server sends a *"SetListPropsRequest"* to the provider server. The provider responds with a *"ContactListPropsResponse"* to the requestor server containing the new properties.

Primitive	Direction
SetListPropsRequest	Requestor Server → Provider Server
ContactListPropsResponse	Requestor Server ← Provider Server

Table 67. Primitive Directions for SetListProperties Transaction

10.3.9. The "CreateAttributeList" Transaction

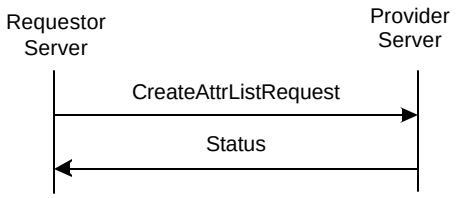

Figure 35. The "CreateAttributeList" Transaction

A user may create a specific attribute list for a contact list, or a member in a contact list through "CreateAttributeList" transaction. The requestor server sends a "*CreateAttrListRequest*" to the provider server. The provider server shall create an attribute list, and attach the attribute list to some contact list(s) and / or user(s).

In order to modify an attribute list, it can be simply overwritten by creating a new one for the same user or contact list. (So it is not necessary to delete it first.)

Primitive	Direction
CreateAttrListRequest	Requestor Server → Provider Server
Status	Requestor Server ← Provider Server

Table 68. Primitive Directions for CreateAttributeList Transaction

10.3.10. The "DeleteAttrList" Transaction

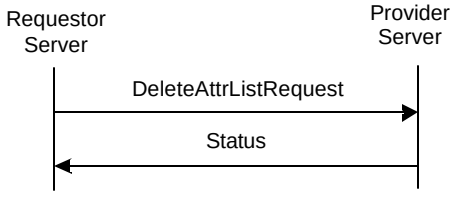

Figure 36. The "DeleteAttrList" Transaction

A user may delete an attribute list from a user and / or a contact list through "DeleteAttrList" transaction. The requestor server sends a *"DeleteAttrListRequest"* to the

provider server. The provider server shall remove the associations of the attribute lists with the contact list(s) and / or user(s). If an attribute list is not associated with any contact list or user, it shall be cleared from the provider server (garbage collection).

Primitive	Direction
DeleteAttrListRequest	Requestor Server → Provider Server
Status	Requestor Server ← Provider Server

Table 69. Primitive Directions for DeleteAttrList Transaction

10.3.11. The "GetAttrList" Transaction

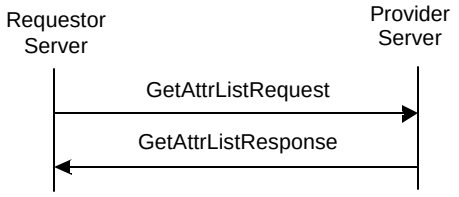

Figure 37. The "GetAttrList" Transaction

The "GetAttrList" transaction is used to retrieve the published or subscribed attributes associated with specific contact list(s) and / or user(s). The provider server returns the requested attributes. If the user(s) or contact list(s) are not specified in the request, the response shall include all user-specific and contact list-specific attributes.

Primitive	Direction
GetAttrListRequest	Requestor Server → Provider Server
GetAttrListResponse	Requestor Server ← Provider Server

Table 70. Primitive Directions for GetAttrList Transaction

10.4. Status Code

10.4.1. Contact List Transactions

- Service Not Supported (405)
- Unknown user ID (531)
- Contact list does not exist (700)
- Contact list already exists (701)
- Invalid or unsupported contact list property. (752)

10.4.2. Attribute List Transactions

- Service Not Supported (405)
- Unknown user ID (531)
- Contact list does not exist (701).
- Unknown presence attribute (not defined in [WVPA]) (750).

11. Service Relay – Presence Features

11.1. Overview

This chapter focuses on the functional relay of Presence features. Because of the server interoperation nature, the SSP has its own requirement on meta-information and information elements in the primitives at transaction level. The complete primitives and transaction flows of Presence features at SSP semantics level has been defined in the following two sections.

Please refer to the CSP document so as to conclude how to relay the Presence features from client-server interaction (CSP) to server-server interoperation (SSP).

11.2. Primitives

11.2.1. The "SubscribeRequest" Primitive

The *"SubscribeRequest"* primitive is used to create subscriptions to obtain notification about changes of the PRESENCE INFORMATION and attributes of other PRINCIPALS. The scope of subscription is either a single user or a contact list which refers to a list of users.

Information Element	Req	Type	Description
Message-Type	M	SubscribeRequest	Message identifier
Meta-Information	M	Structure of Meta-Information	The meta-information (see 5.1).
User-ID-List	C	Structure	Identifies the IM users to be subscribed.
Contact-List-ID-List	C	Structure	Identifies the set of users.
Presence-Attribute-List	O	Structure	A list of presence attributes to which are subscribed. An empty list or missing list indicates all presence attributes are desired.

Table 71. Information elements in SubscribeRequest Primitive

11.2.2. The "AuthorizationRequest" Primitive

The *"AuthorizationRequest"* primitive is used for the provider server to perform the reactive authorization with the requestor server that represents the publishing users.

Information Element	Req	Type	Description
Message-Type	M	AuthorizationRequest	Message identifier
Meta-Information	M	Structure of Meta-Information	The meta-information (see 5.1).

Authorizing-User-ID	M	String	Identifies the user who can grant the authorization to the requesting users.
List-of-Subscribing-User-ID-and-Presence-Attribute-List	M	Structure	A list of elements where each node specifies the user-ID and the presence attributes to which are subscribed. An empty attribute list indicates that all presence attributes are desired.

Table 72. Information elements in AuthorizationRequest Primitive

There may be multiple tuples { Authorizing-User-ID, List-of-Subscribing-User-ID-and-Presence-Attribute-List } in one *"AuthorizationRequest"* primitive if the provider server is able to combine the multiple reactive authorizations in one primitive in order to reduce the traffic overhead between the servers.

11.2.3. The "AuthorizationResponse" Primitive

The "*AuthorizationResponse*" primitive returns the authorization result from the responding authorizing users.

There may be multiple tuples { Authorizing-User-ID, Subscribing-User-IDs, Authorization-Result } in one *"AuthorizationResponse"* primitive if the provider server is able to collect the response from the authorizing users in a reasonable time and combine the multiple responses in one primitive in order to reduce the traffic overhead between the servers.

Information Element	Req	Type	Description
Message-Type	M	AuthorizationResponse	Message identifier
Meta-Information	M	Structure of Meta-Information	The meta-information (see 5.1).
Authorizing-User-ID	M	String	Identifies the user who can grant the authorization to the requesting users.
Subscribing-User-ID-List	M	Structure	Identifies the requesting users who want to subscribe
Authorization-Result(s)	M	"Yes" | "No"	Authorization results from the authorizing user per subscribing user.

Table 73. Information elements in AuthorizationResponse Primitive

11.2.4. The "UnsubscribeRequest" Primitive

The *"UnsubscribeRequest"* primitive is used to cancel the current subscription.

Information Element	Req	Type	Description
Message-Type	M	UnsubscribeRequest	Message identifier
Meta-Information	M	Structure of Meta-Information	The meta-information (see 5.1).
User-ID-List	C	Structure	Identifies the IM users to be unsubscribed.
Contact-List-ID-List	C	Structure	Identifies the set of users.

Table 74. Information elements in UnsubscribeRequest Primitive

11.2.5. The "PresenceNotification" Primitive

The "*PresenceNotification*" primitive is used for the provider server to send the notifications about changes of presence information to the requestor server.

Information Element	Req	Type	Description
Message-Type	M	PresenceNotification	Message identifier
Meta-Information	M	Structure of Meta-Information	The meta-information (see 5.1).
Subscribing-User-ID-List	M	Structure	Identifies the users who subscribed to the presence change.
Presence-Value-List	M	Structure	List of User IDs and corresponding presence values.

Table 75. Information elements in PresenceNotification Primitive

11.2.6. The "GetWatcherListRequest" Primitive

The "*GetWatcherListRequest*" primitive is used for the requestor server to retrieve the list of users that subscribed to its presence information.

Information Element	Req	Type	Description
Message-Type	M	GetWatcherListRequest	Message identifier
Meta-Information	M	Structure of Meta-Information	The meta-information (see 5.1).

Table 76. Information elements in GetWatcherRequest Primitive

11.2.7. The "GetWatcherListResponse" Primitive

The "*GetWatcherListResponse*" primitive is used for the provider server to return the subscriber list to the requestor server.

Information	Req	Type	Description

Element			
Message-Type	M	GetWatcherListResponse	Message identifier
Status-Info	M	Structure of Status-Primitive	Status information (see 5.2).
User-ID-List	C	Structure	Identifies the subscribers.

Table 77. Information elements in GetWatcherListResponse Primitive

11.2.8. The "GetPresenceRequest" Primitive

The *"GetPresenceRequest"* primitive is used for the requestor server to retrieve the updated presence information. If the presence attribute list is missing from the request, the server sends all available presence information.

Information Element	Req	Type	Description
Message-Type	M	GetPresenceRequest	Message identifier
Meta-Information	M	Structure of Meta-Information	The meta-information (see 5.1).
User-ID-List	C	Structure	Identifies the publishing users.
Contact-List-ID-List	C	Structure	Identifies the set of publishing users.
Presence-Attribute-List	O	Structure	A list of presence attributes to be retrieved. An empty or missing list indicates all presence attributes are desired.

Table 78. Information elements in GetPresenceRequest Primitive

11.2.9. The "GetPresenceResponse" Primitive

The *"GetPresenceResponse"* primitive is used for the provider server to send the updated presence information to the requestor server.

Information Element	Req	Type	Description
Message-Type	M	GetPresenceResponse	Message identifier
Status-Info	M	Structure of Status-Primitive	Status information (see 5.2).
Presence-Value-List	O	Structure	List of User IDs and corresponding presence values.

Table 79. Information elements in GetPresenceResponse Primitive

11.2.10. The "UpdatePresenceRequest" Primitive

The *"UpdatePresenceRequest"* primitive is used for the requestor server to update presence information for the publishing user. Only the updated attributes and their values need to be carried in this primitive, the omitted attributes are not modified.

Information Element	Req	Type	Description
Message-Type	M	UpdatePresenceRequest	Message identifier
Meta-Information	M	Structure of Meta-Information	The meta-information (see 5.1).
Presence-Value-List	M	Structure	A list of presence values to update.

Table 80. Information elements in UpdatePresenceRequest Primitive

11.2.11. The "CancelAuthRequest" Primitive

The *"CancelAuthRequest"* primitive is used for the publishing user to cancel its previous reactive authorizations, and remove the subscriber from its Watcher List.

Information Element	Req	Type	Description
Message-Type	M	CancelAuthRequest	Message identifier
Meta-Information	M	Structure of Meta-Information	The meta-information (see 5.1).
Canceled-User-ID-List	M	Structure	Identifies the users who will be cancelled authorization.

Table 81. Information elements in CancelAuthRequest Primitive

11.3. Transactions

11.3.1. The "Subscribe" Transaction

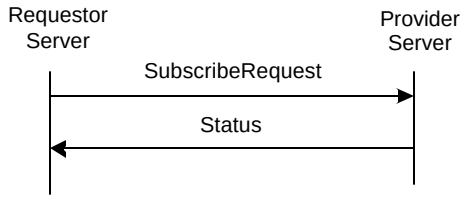

Figure 38. The "Subscribe" Transaction

The subscription to obtaining the notification about the changes of the presence information is accomplished through a "Subscribe" transaction.

The requestor server sends a *"SubscribeRequest"* request to the provider server for subscribing to the notification about the changes of the presence information of some publishing users. The provider server shall determine whether or not the reactive authorization is needed based on whether or not the subscribing user is proactively

authorized in the publishing user's contact list. The provider server shall return a "Status" message indicating that the provider server has accepted and processed the request.

The provider server shall perform "ReactiveAuthorization" transactions with the publishing users if the individual reactive authorizations are needed.

If the subscription succeeds, the requestor server shall receive the current presence information through a "PresenceNotification" transaction immediately. The requestor server shall also receive the presence changes in the future.

The scope of the subscription is either a single user or a contact list referring to multiple users. The requesting user may subscribe only part of the presence information and, correspondingly, the user whose presence information is subscribed may allow only part of the presence information to be delivered. The subscription may be persistent through different sessions.

Primitive	Direction
SubscribeRequest	Requestor Server → Provider Server
Status	Requestor Server ← Provider Server

Table 82. Primitive Directions for Subscribe Transaction

11.3.2. The "ReactiveAuthorization" Transaction

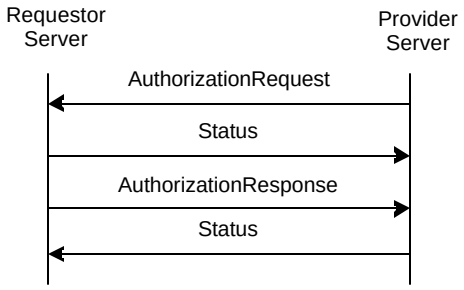

Figure 39. The "ReactiveAuthorization" Transaction

If the reactive authorization is needed in the "Subscribe" transaction from the subscribing user, the provider server shall perform the "ReactiveAuthorization" transactions with the requestor servers that represent the publishing users. The publishing user may accept or reject the request of authorization to subscribe to its presence information.

There may be multiple tuples { Authorizing-User-ID, List-of-Subscribing-User-ID-and-Presence-Attribute-List } in one *"AuthorizationRequest"* primitive if the provider server is able to combine the multiple reactive authorizations in one primitive in order to reduce the traffic overhead between the servers.

There may be multiple tuples { Authorizing-User-ID, Subscribing-User-IDs, Authorization-Result } in one *"AuthorizationResponse"* primitive if the provider server is

able to collect the response from the authorizing users in a reasonable time and combine the multiple responses in one primitive in order to reduce the traffic overhead between the servers.

A new authorization will overwrite the existing one.

This transaction belongs to the complementary service.

Primitive	Direction
AuthorizationRequest	Requestor Server ← Provider Server
Status	Requestor Server → Provider Server
AuthorizationResponse	Requestor Server → Provider Server
Status	Requestor Server ← Provider Server

Table 83. Primitive Directions for ReactiveAuthorization Transaction

11.3.3. The "Unsubscribe" Transaction

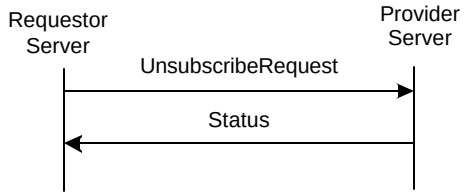

Figure 40. The "Unsubscribe" Transaction

The cancellation to current subscription is accomplished through an "Unsubscribe" transaction. The provider server shall return a "Status" message indicating that the provider server has accepted and processed the request.

Primitive	Direction
UnsubscribeRequest	Requestor Server → Provider Server
Status	Requestor Server ← Provider Server

Table 84. Primitive Directions for Unsubscribe Transaction

11.3.4. The "PresenceNotification" Transaction

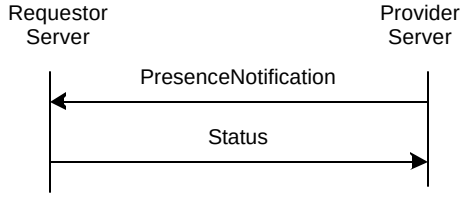

Figure 41. The "PresenceNotification" Transaction

The requestor server is informed of the change of the presence information through a "PresenceNotification" transaction originated from the provider server.

Primitive	Direction
PresenceNotification	Requestor Server ← Provider Server
Status	Requestor Server → Provider Server

Table 85. Primitive Directions for PresenceNotification Transaction

11.3.5. The "GetWatcherList" Transaction

Figure 42. The "GetWatcherList" Transaction

The purpose of the *"GetWatcherList"* transaction is for the requestor server to retrieve the list of users that subscribed to its presence information.

The requestor server sends a *"GetWatcherListRequest"* to the provider server. A *"GetWatcherListResponse"* message from the provider server contains a list of subscribers.

This transaction belongs to the complementary service.

Primitive	Direction
GetWatcherListRequest	Requestor Server → Provider Server
GetWatcherListRespons e	Requestor Server ← Provider Server

Table 86. Primitive Directions for GetWatcherList Transaction

11.3.6. The "GetPresence" Transaction

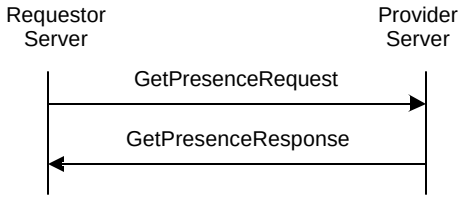

Figure 43. The "GetPresence" Transaction

The purpose of the *"GetPresence"* transaction is for the requestor server to retrieve the presence information of other users.

The requestor server sends a *"GetPresenceRequest"* to the provider server for the updated presence information of the publishing users. A *"GetPresenceResponse"* message from

the provider server will contain result code(s) and, if the request was successful, then it will relay the requested PRESENCE INFORMATION.

Primitive	Direction
GetPresenceRequest	Requestor Server → Provider Server
GetPresenceResponse	Requestor Server ← Provider Server

Table 87. Primitive Directions for GetPresence Transaction

11.3.7. The "UpdatePresence" Transaction

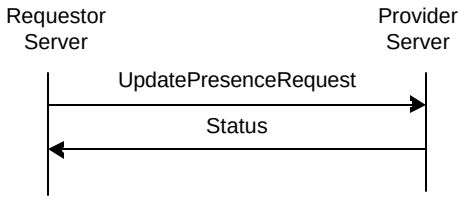

Figure 44. The "UpdatePresence" Transaction

An owner of the presence data or a user with sufficient privileges may update presence attributes and their values through a "UpdatePresence" transaction.

The requestor server sends an *"UpdatePresenceRequest"* message to the provider server. The provider server returns a Status response.

Primitive	Direction
UpdatePresenceRequest	Requestor Server → Provider Server
Status	Requestor Server ← Provider Server

Table 88. Primitive Directions for UpdatePresence Transaction

11.3.8. The "CancelAuthorization" Transaction

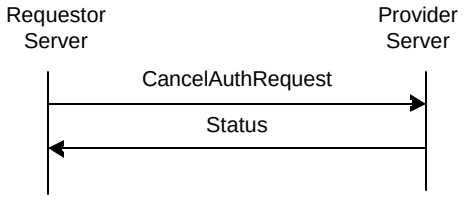

Figure 45. The "CancelAuthorization" Transaction

A publishing user may cancel the reactive authorization and subscription, and remove the subscriber from the Watcher List through "CancelAuthorization" transaction.

Please note that the proactive authorization is cancelled by removing the subscriber from the contact list, or by removing the associated attribute list, or by making the associated attribute list empty.

The requestor server sends a *"CancelAuthRequest"* message to the provider server. The provider server returns a Status response.

This transaction belongs to the complementary service.

Primitive	Direction
CancelAuthRequest	Requestor Server → Provider Server
Status	Requestor Server ← Provider Server

Table 89. Primitive Directions for CancelAuthorization Transaction

11.4. Status Code

11.4.1. "ReactiveAuthorization" Transaction

- Service Not Supported (405)
- Not logged in (604)
- Unknown presence attribute (not defined in [WVPA]). (750)

11.4.2. "GetPresence" Transaction

- Service Not Supported (405)
- Not logged in (604)
- Unknown presence attribute (not defined in [WVPA]) (750)
- Unknown presence value (not defined in [WVPA]) (751)

11.4.3. "UpdatePresence" Transaction

- Service Not Supported (405)
- Not logged in (604)
- Unknown presence attribute (not defined in [PresenceAttributes]) (750)
- Unknown presence value (not defined in [PresenceAttributes]) (751)

11.4.4. Other Presence Transactions

- Service Not Supported (405)
- Unknown user ID (531)
- Not logged in (604)
- Unknown contact list (700).
- Unknown presence attribute (not defined in [WVPA]). (750)
- Unknown presence value (not defined on the [WVPA])(751).

12. Service Relay – Instant Messaging Features

12.1. Overview

This chapter focuses on the functional relay of IM features. Because of the server interoperation nature, the SSP has its own requirement on meta-information and information elements in the primitives at transaction level. The complete primitives and transaction flows of IM features at SSP semantics level has been defined in the following two sections.

Please refer to the CSP document so as to conclude how to relay the IM features from client-server interaction (CSP) to server-server interoperation (SSP).

12.2. Primitives

12.2.1. The "SendMessageRequest" Primitive

The *"SendMessageRequest"* primitive is used for the requesting server to send the instant messages to the users through the requested server.

Information Element	Req	Type	Description
Message-Type	M	SendMessageRequest	Message identifier
Meta-Information	M	Structure of Meta-Information	The meta-information (see 5.1).
Delivery-Report-Request	M	"Yes" | "No"	Indicates if the user wants delivery report.
Message-Info	M	Structure	Message information data, including { Message-ID or Message-URI, Content-type / MIME, encoding, size, sender and recipients (User-ID and/or Client-ID and/or Screen-Name and/or Group-ID and/or Contact-List-ID), date and time, validity }. Message-ID is NOT present if the request is relayed from the user's Home Domain to its PSE. Otherwise, Message-ID is present.
Content	C	String or Binary data	The content of the instant message.

Table 90. Information elements in SendMessageRequest Primitive

12.2.2. The "SendMessageResponse" Primitive

The *"SendMessageResponse"* primitive is used for the requested server to inform the requesting server of the message sending result.

Information Element	Req	Type	Description
Message-Type	M	SendMessageResponse	Message identifier
Status-Info	M	Structure of Status-Primitive	Status information (see 5.2).
Message-ID	C	String	Server generated message id for this message.

Table 91. Information elements in SendMessageResponse Primitive

12.2.3. The "ForwardMessageRequest" Primitive

The *"ForwardMessageRequest"* primitive is used for the requesting server to forward the non-retrieved instant messages.

Information Element	Req	Type	Description
Message-Type	M	ForwardMessageRequest	Message identifier
Meta-Information	M	Structure of Meta-Information	The meta-information (see 5.1).
Message-ID	M	String	Identifies the message (either Message-ID or Message-URI).
Recipients	M	Structure	Identifies the users to whom the message is forwarded (User-ID-List, Contact-List-ID-List, Screen-Name-List and Group-ID-List)

Table 92. Information elements in ForwardMessageRequest Primitive

12.2.4. The "NewMessage" Primitive

The *"NewMessage"* primitive is used for the provider server to deliver the instant message to the users through the requestor server.

Information Element	Req	Type	Description
Message-Type	M	NewMessage	Message identifier.
Meta-Information	C	Structure of Meta-Information	The meta-information (see 5.1). Present if in PushMessage transaction.
Status-Info	C	Structure of Status-Primitive	Status information (see 5.2). Present if in GetMessage transaction.
Recipient-User-ID-List	M	Structure	Identifies the recipients with a list of User-ID's.

Message-Info	M	Structure	Message information data, including { Message-ID or Message-URI, Content-type / MIME, encoding, size, sender and recipients (User-ID and optionally the Client-ID and/or Screen-Name and/or Group-ID and/or Contact-List-ID), date and time, validity }.
Content	M	String or Binary data	Message data.

Table 93. Information elements in NewMessage Primitive

12.2.5. The "MessageDelivered" Primitive

The *"MessageDelivered"* primitive is used for the requestor server to confirm that the message has been delivered.

Information Element	Req	Type	Description
Message-Type	M	MessageDelivered	Message identifier.
Meta-Information	C	Structure of Meta-Information	The meta-information (see 5.1). Present if in GetMessage transaction.
Status-Info	C	Structure of Status-Primitive	Status information (see 5.2). Present if in PushMessage transaction.
Message-ID	M	String	ID of message that has been delivered

Table 94. Information elements in MessageDelivered Primitive

12.2.6. The "MessageNotification" Primitive

The *"MessageNotification"* primitive is used for the provider server to notify the user of the new messages through the requestor server.

Information Element	Req	Type	Description
Message-Type	M	MessageNotification	Message identifier
Meta-Information	M	Structure of Meta-Information	The meta-information (see 5.1).
Recipient-User-ID-List	M	Structure	Identifies the recipients with a list of User-ID's.
Message-Info	M	Structure	Message information data, including { Message-ID or Message-URI, Content-type / MIME, encoding, size, sender and recipients (User-ID and optionally the Client-ID and/or Screen-Name

			and/or Group-ID and/or Contact-List-ID), date and time, validity }.

Table 95. Information elements in MessageNotification Primitive

12.2.7. The "GetMessageRequest" Primitive

The *"GetMessageRequest"* primitive is used for the requestor server to get the instant message from the provider server.

Information Element	Req	Type	Description
Message-Type	M	GetMessageRequest	Message identifier
Meta-Information	M	Structure of Meta-Information	The meta-information (see 5.1).
Message-ID	M	String	ID of message to retrieve

Table 96. Information elements in GetMessageRequest Primitive

12.2.8. The "SetMessageDeliveryMethod" Primitive

The *"SetMessageDeliveryMethod"* primitive is used for user in the requestor server to set the instant message delivery method.

Information Element	Req	Type	Description
Message-Type	M	SetMessageDeliveryMethod	Message identifier
Meta-Information	M	Structure of Meta-Information	The meta-information (see 5.1).
Message-Delivery-Method	M	"Notify/Get" | "Push"	Determines the type of message delivery. Push means that complete message is transferred in the notification. Notify/Get means that only the message-ID or message-URI is transferred in the notification the message is then retrieved using a GetMessage transaction.
Accepted-Content-Length	O	Integer	Maximum size of message that can be pushed to the user.
Group-ID	O	String	Group ID if Delivery method refers to a group.

Table 97. Information elements in SetMessageDeliveryMethod Primitive

12.2.9. The "GetMessageListRequest" Primitive

If the provider server offers the space where messages are stored, the user can retrieve an undelivered message list or group history list. The *"GetMessageListRequest"* primitive is used for the requestor server to get the stored Message-ID's or Message-URI's so that they can be used in GetMessage or RejectMessage transactions. If "Group-ID" is present, the user will have the group history list. Otherwise, the user will have the undelivered message list.

Information Element	Req	Type	Description
Message-Type	M	GetMessageListRequest	Message identifier
Meta-Information	M	Structure of Meta-Information	The meta-information (see 5.1).
Group-ID	C	String	List the messages to the group(s) (to retrieve the history).
Message-Count	O	Integer	The maximum number of message-info structures to be returned.

Table 98. Information elements in GetMessageListRequest Primitive

12.2.10. The "GetMessageListResponse" Primitive

The *"GetMessageListResponse"* primitive is used for the provider server to return a list of message information.

Information Element	Req	Type	Description
Message-Type	M	GetMessageList Response	Message identifier.
Status-Info	M	Structure of Status-Primitive	Status information (see 5.2).
Message-Info-List	M	Structure	Message information data, including { Message-ID or Message-URI, Content-type / MIME, encoding, size, sender and recipients (User-ID and/or Client-ID and/or Screen-Name and/or Group-ID and/or Contact-List-ID), date and time, validity }.

Table 99. Information elements in GetMessageListResponse Primitive

12.2.11. The "RejectMessageRequest" Primitive

The *"RejectMessageRequest"* primitive is used for the requestor server to remove the unwanted and / or stored messages in the provider server.

Information Element	Req	Type	Description
Message-Type	M	RejectMessageRequest	Message identifier
Meta-Information	M	Structure of Meta-Information	The meta-information (see 5.1).
Message-ID-List	M	Structure	Identifies the messages (either Message-ID-List or Message-URI-List).

Table 100. Information elements in RejectMessageRequest Primitive

12.2.12. The "DeliveryStatusReport" Primitive

The *"DeliveryStatusReport"* primitive is used for the provider server to give the sender the message delivery status report. The delivery report can also inform the client about an unsuccessful delivery attempt due to detected error conditions on the receiving side.

Information Element	Req	Type	Description
Message-Type	M	DeliveryStatusReport	Message identifier.
Meta-Information	M	Structure of Meta-Information	Meta-information (see 5.1).
Delivery-Result	M	Structure of Status-Primitive	The delivery result shares the same structure as Status (see 5.2).
Message-Info	M	Structure	Message information data, including { Message-ID or Message-URI, Content-type / MIME, encoding, size, sender and recipients (User-ID and/or Client-ID and/or Screen-Name and/or Group-ID and/or Contact-List-ID), date and time, validity }.

Table 101. Information elements in DeliveryStatusReport Primitive

12.2.13. The "BlockUserRequest" Primitive

The *"BlockUserRequest"* primitive is used for the blocking user in the requesting server to prevent from message delivery from certain users. The blocked users always see that the blocking user is offline. None of the message from the blocked user will be delivered to the blocking user.

Information Element	Req	Type	Description
Message-Type	M	BlockUserRequest	Message identifier
Meta-Information	M	Structure of Meta-Information	The meta-information (see 5.1).
Block-User-ID-List	O	Structure	A list of users to be added to the

			block list.
Unblock-User-ID-List	O	Structure	A list of users to be removed from the block list.
Block-List-Status	M	"Active" | "Inactive"	Indicates if the block list is in use ("Active") or not ("Inactive").
Grant-User-ID-List	O	Structure	The list of users to be added to the grant list.
Ungrant-User-ID-List	O	Structure	The list of users to be removed from the grant list.
Grant-List-Status	M	"Active" | "Inactive"	Indicates if the grant list is in use ("Active") or not ("Inactive").

Table 102. Information elements in BlockUserRequest Primitive

12.2.14. The "GetBlockedRequest" Primitive

The *"GetBlockedRequest"* primitive is used for the blocking user in the requestor server to get its own list of blocked and granted users, and the status of the grant list and block list.

Information Element	Req	Type	Description
Message-Type	M	GetBlockedRequest	Message identifier
Meta-Information	M	Structure of Meta-Information	The meta-information (see 5.1).

Table 103. Information elements in GetBlockedRequest Primitive

12.2.15. The "GetBlockedResponse" Primitive

The *"GetBlockedResponse"* primitive is used for the provider server to return a list of blocked users and granted users, and the list status.

Information Element	Req	Type	Description
Message-Type	M	GetBlockedResponse	Message identifier
Status-Info	M	Structure of Status-Primitive	Status information (see 5.2).
Block-User-ID-List	C	Structure	The list of currently blocked users.
Block-List-Status	M	"Active" | "Inactive"	If the block list is in use ("Active") or not ("Inactive").
Grant-User-ID-List	C	Structure	The list of currently granted users.
Grant-List-Status	M	"Active" | "Inactive"	If the grant list is in use ("Active") or not ("Inactive").

Table 104. Information elements in GetBlockedResponse Primitive

12.3. Transactions

12.3.1. The "SendMessage" Transaction

Figure 46. The "SendMessage" Transaction

The purpose of "SendMessage" transaction is for the requestor server to send the instant messages through the provider server. The user may send message to a group or to other user(s) at any suitable time.

The requestor server sends a *"SendMessageRequest"* message to the provider server. The provider server returns a *"SendMessageRsponse"* response containing the result and the message ID.

Primitive	Direction
SendMessageRequest	Requestor Server → Provider Server
SendMessageResponse	Requestor Server ← Provider Server

Table 105. Primitive Directions for SendMessage Transaction

12.3.2. The "ForwardMessage" Transaction

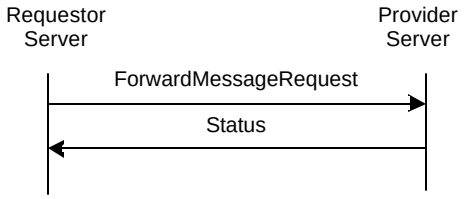

Figure 47. The "ForwardMessage" Transaction

Primitive	Direction
ForwardMessageRequest	Requestor Server → Provider Server
Status	Requestor Server ← Provider Server

Table 106. Primitive Directions for ForwardMessage Transaction

12.3.3. The "PushMessage" Transaction

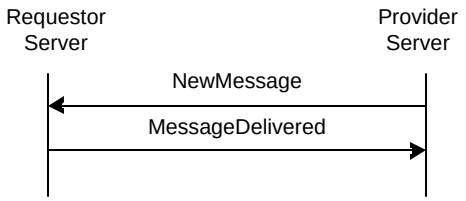

Figure 48. The "PushMessage" Transaction

The purpose of "PushMessage" transaction is for the provider server to deliver the messages to users through the requestor server.

The provider server sends a *"NewMessage"* primitive to the requestor server. The requestor server returns a *"MessageDelivered"* response containing the result and the message ID.

This transaction belongs to the complementary service.

Primitive	Direction
NewMessage	Requestor Server ← Provider Server
MessageDelivered	Requestor Server → Provider Server

Table 107. Primitive Directions for PushMessage Transaction

12.3.4. The "MessageNotification" Transaction

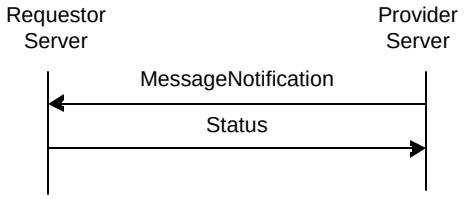

Figure 49. The "MessageNotification" Transaction

The purpose of "MessageNotification" transaction is for the provider server to notify the users of new messages through the requestor server.

The provider server sends a *"MessageNotification"* primitive to the requestor server. The requestor server returns a *"Status"* response.

This transaction belongs to the complementary service.

Primitive	Direction
MessageNotification	Requestor Server ← Provider Server
Status	Requestor Server → Provider Server

Table 108. Primitive Directions for MessageNotification Transaction

12.3.5. The "GetMessage" Transaction

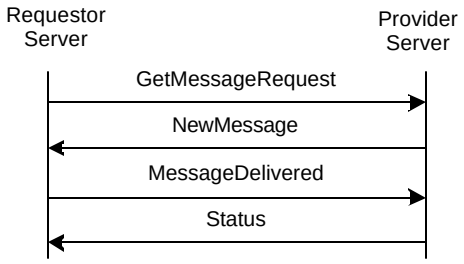

Figure 50. The "GetMessage" Transaction

The purpose of "GetMessage" transaction is for the requestor server to retrieves the new messages from the provider server.

The requestor server sends a *"GetMessageRequest"* message with a message ID to the provider server. The provider server returns a *"NewMessage"* response containing the new message.

This transaction belongs to the complementary service.

Primitive	Direction
GetMessageRequest	Requestor Server → Provider Server
NewMessage	Requestor Server ← Provider Server
MessageDelivered	Requestor Server → Provider Server
Status	Requestor Server ← Provider Server

Table 109. Primitive Directions for GetMessage Transaction

12.3.6. The "SetMessageDeliveryMethod" Transaction

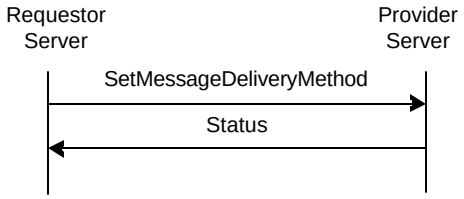

Figure 51. The "SetMessageDeliveryMethod" Transaction

The purpose of "SetMessageDeliveryMethod" transaction is for the user in the requestor server to set the appropriate message delivery method from the provider server.

The requestor server sends a *"SetMessageDeliveryMethod"* request to the provider server. The provider server returns a *"Status"* response.

This transaction belongs to the complementary service.

Primitive	Direction
SetMessageDeliveryMethod	Requestor Server → Provider Server
Status	Requestor Server ← Provider Server

Table 110. Primitive Directions for SetMessageDeliveryMethod Transaction

12.3.7. The "GetMessageList" Transaction

Figure 52. The "GetMessageList" Transaction

The purpose of "GetMessageList" transaction is for the requestor server to get the stored Message-ID's or Message-URI's so that they can be used in GetMessage or RejectMessage transactions. This transaction can be used to retrieve the message history of the group if the GetMessageListRequest contains the Group ID.

The requestor server sends a *"GetMessageListRequest"* to the provider server. The provider server returns a *"GetMessageListResponse"*.

This transaction belongs to the complementary service if the undelivered messages are requested.

Primitive	Direction
GetMessageListRequest	Requestor Server → Provider Server
GetMessageListResponse	Requestor Server ← Provider Server

Table 111. Primitive Directions for GetMessageList Transaction

12.3.8. The "RejectMessage" Transaction

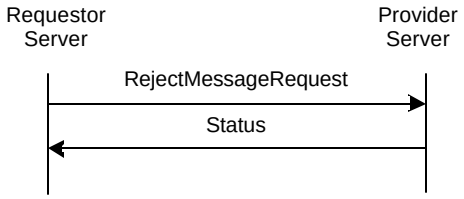

Figure 53. The "RejectMessage" Transaction

This transaction belongs to the complementary service.

Primitive	Direction

RejectMessageRequest	Requestor Server → Provider Server
Status	Requestor Server ← Provider Server

Table 112. Primitive Directions for RejectMessage Transaction

12.3.9. The "NotifyDeliveryStatusReport" Transaction

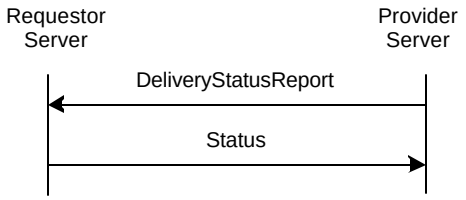

Figure 54. The "NotifyDeliveryStatusReport" Transaction

Primitive	Direction
DeliveryStatusReport	Requestor Server ← Provider Server
Status	Requestor Server → Provider Server

Table 113. Primitive Directions for NotifyDeliveryStatusReport Transaction

12.3.10. The "BlockUser" Transaction

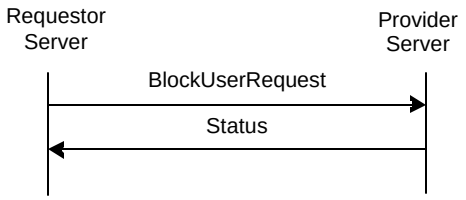

Figure 55. The "BlockUser" Transaction

A user may block/un-block any other user at any suitable time. The purpose of "BlockUser" transaction is for the blocking user in the requestor server to prevent from getting the messages from the blocked users in the provider server.

The requestor server sends a "*BlockUserRequest"* request to the provider server containing the list of users to be blocked / unblocked . The provider server returns a "*Status*" response.

This transaction belongs to the complementary service.

Primitive	Direction
BlockUserRequest	Requestor Server → Provider Server
Status	Requestor Server ← Provider Server

Table 114. Primitive Directions for BlockUser Transaction

12.3.11. The "GetBlockedList" Transaction

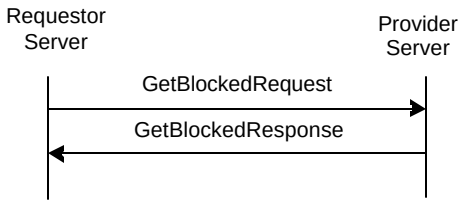

Figure 56. The "GetBlockedList" Transaction

A user may get its own list of blocked users at any suitable time. The purpose of "GetBlockedList" transaction is for the blocking user in the requestor server to get its own list of blocked users and granted users.

The requestor server sends a *"GetBlockedRequest"* request to the provider server. The provider server returns a *"GetBlockedResponse"* response containing the list of blocked users.

This transaction belongs to the complementary service.

Primitive	Direction
GetBlockedRequest	Requestor Server → Provider Server
GetBlockedResponse	Requestor Server ← Provider Server

Table 115. Primitive Directions for GetBlockedList Transaction

12.4. Status Code

12.4.1. "SendMessage" Transaction

- Service Not Supported (405)
- Unknown content-type (415)
- Message queue full (507)
- Recipient user does not exist. (531)
- Recipient user blocked the sender (532)
- Recipient user is not logged in (533)
- Not logged in (604)
- Contact list does not exist. (701)
- 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)

12.4.2. "SetMessageDeliveryMethod" Transaction

- Service Not Supported (405)
- Not logged in (604)

- Group does not exist. (800)

12.4.3. "GetMessageList" Transaction

- Service Not Supported (405)
- Not logged in (604)
- Group does not exist. (800)
- Group is not joined (808)
- History is not supported (821)

12.4.4. "RejectMessage" Transaction

- Service Not Supported (405)
- Invalid Message-ID (426)
- Not logged in (604)

12.4.5. "NewMessage" Transaction

- Service Not Supported (405)
- Invalid Message-ID (426)
- Not logged in (604)

12.4.6. "MessageNotification" Transaction

- Service Not Supported (405)
- Not logged in (604)

12.4.7. "GetMessage" Transaction

- Service Not Supported (405)
- Invalid Message-ID (426)
- Not logged in (604)

12.4.8. "NotifyDeliveryStatusReport" Transaction

- Service Not Supported (405)
- Not logged in (604)

12.4.9. "ForwardMessage" Transaction

- Service Not Supported (405)
- Unknown content-type. (415)
- Message queue full. (507)
- Recipient user does not exist. (531)
- Recipient user blocked the sender. (532)
- Recipient user is not logged in. (533)
- Not logged in (604)
- Contact list does not exist. (700)

- 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)

12.4.10. Block Transactions

- Service Not Supported (405)
- Unknown user ID (531)
- Not logged in (604)
- Unknown group-ID (800)

13. Service Relay – Group Features

This chapter focuses on the functional relay of Group features. Because of the server interoperation nature, the SSP has its own requirement on meta-information and information elements in the primitives at transaction level. The complete primitives and transaction flows of Group features at SSP semantics level has been defined in the following two sections.

Please refer to the CSP document so as to conclude how to relay the Group features from client-server interaction (CSP) to server-server interoperation (SSP).

13.1. Primitives

13.1.1. The "CreateGroupRequest" Primitive

The *"CreateGroupRequest"* primitive is used for the user in the requestor server to create a private user group at any suitable time. The *"CreateGroupRequest"* primitive contains the User-ID, Group-ID, and the initial properties of the group. The provider server creates the group with the specified properties, and responds with a Status message.

Information Element	Req	Type	Description
Message-Type	M	CreateGroupRequest	Message identifier
Meta-Information	M	Structure of Meta-Information	The meta-information (see 5.1).
Group-ID	M	String	Identifies the group
Group-Props	M	Structure	The properties of the group.

Table 116. Information elements in CreateGroupRequest Primitive

13.1.2. The "DeleteGroupRequest" Primitive

The *"DeleteGroupRequest"* primitive is used for the user with sufficient access rights in the requestor server to delete a private user group at any suitable time. The "*DeleteGroupRequest*" primitive contains the Group-ID. The provider server removes all currently joined users from the group (ServerInitiatedLeaveGroup transaction), deletes the specified group, and responds with a Status message.

Information Element	Req	Type	Description
Message-Type	M	DeleteGroupRequest	Message identifier
Meta-Information	M	Structure of Meta-Information	The meta-information (see 5.1).
Group-ID	M	String	Identifies the group

Table 117. Information elements in DeleteGroupRequest Primitive

13.1.3. The "JoinGroupRequest" Primitive

The *"JoinGroupRequest"* primitive is used for the user in the requestor server to join a discussion group at any suitable time. The *"JoinGroupRequest"* primitive contains the Group-ID, its screen name shown during the discussion, and the joined users' list request.

Information Element	Req	Type	Description
Message-Type	M	JoinGroupRequest	Message identifier
Meta-Information	M	Structure of Meta-Information	The meta-information (see 5.1).
Group-ID	M	String	Identifies the group
Joined-Request	M	"Yes" | "No"	Indicates if the user wants the list of currently joined users ("Yes") or not ("No").
Screen-Name	O	String	Screen name of the user in the group.

Table 118. Information elements in JoinGroupRequest Primitive

13.1.4. The "JoinGroupResponse" Primitive

The *"JoinGroupResponse"* primitive is used for the provider server to return the processing result with the list of currently joined users (if requested), and optionally a welcome note.

Information Element	Req	Type	Description
Message-Type	M	JoinGroupResponse	Message identifier
Status-Info	M	Structure of Status-Primitive	Status information (see 5.2).
Joined-User-Screen-Name-List	C	Structure	The list of currently joined users identified by their Screen-Name's. Present if it was requested.
Welcome-Text	O	String	A short text to be shown to the user when he/she has joined the group.

Table 119. Information elements in JoinGroupResponse Primitive

13.1.5. The "LeaveGroupRequest" Primitive

The *"LeaveGroupRequest"* primitive is used for the user in the requestor server to leave a discussion group at any suitable time. The *"LeaveGroupRequest"* primitive contains the Group-ID.

Information Element	Req	Type	Description
Message-Type	M	LeaveGroupRequest	Message identifier

Meta-Information	M	Structure of Meta-Information	The meta-information (see 5.1).
Group-ID	M	String	Identifies the group

Table 120. Information elements in LeaveGroupRequest Primitive

13.1.6. The "LeaveGroupIndication" Primitive

The "Leave*GroupIndication*" primitive is used for the provider server to return the group leaving result requested from the requestor server. The "Leave*GroupIndication*" primitive is also used for the provider server to initiate the group leaving due to user kickout, group deletion etc.

Information Element	Req	Type	Description
Message-Type	M	LeaveGroupIndication	Message identifier
Meta-Information	C	Structure of Meta-Information	Meta-information (see 5.1). Present if in ServerInitiatedLeaveGroup transaction
Status-Info	C	Structure of Status-Primitive	Status information (see 5.2). Present if in LeaveGroup transaction.
Reason-text	M	String	Indicate why the user has to leave.
Group-ID	C	String	Identification of the group that has been left. Present if in ServerInitiatedLeaveGroup transaction.

Table 121. Information elements in LeaveGroupIndication Primitive

13.1.7. The "GetJoinedMemberRequest" Primitive

The "*GetJoinedMemberRequest*" primitive is used for the requestor server to retrieve the joined member list of a group. This primitive (and transaction) has no corresponding CSP primitive (and transaction).

Information Element	Req	Type	Description
Message-Type	M	GetJoinedMemberRequest	Message identifier
Meta-Information	M	Structure of Meta-Information	The meta-information (see 5.1).
Group-ID	M	String	Identifies the group

Table 122. Information elements in GetJoinedMemberRequest Primitive

13.1.8. The "GetJoinedMemberResponse" Primitive

The *"GetJoinedMemberResponse"* primitive is used for the provider server to return the result with a list of joined group members.

Information Element	Req	Type	Description
Message-Type	M	GetJoinedMemberResponse	Message identifier
Status-Info	M	Structure of Status-Primitive	Status information (see 5.2).
Joined-User-List	M	Structure	A list of joined members identified by their { User-ID, Screen-Name } pairs.

Table 123. Information elements in GetJoinedMemberResponse Primitive

13.1.9. The "GetGroupMemberRequest" Primitive

The *"GetGroupMemberRequest"* primitive is used for the user with sufficient access rights in the requestor server to retrieve the member list of a group. The *"GetGroupMemberRequest"* primitive contains the Group-ID.

Information Element	Req	Type	Description
Message-Type	M	GetGroupMemberRequest	Message identifier
Meta-Information	M	Structure of Meta-Information	The meta-information (see 5.1).
Group-ID	M	String	Identifies the group

Table 124. Information elements in GetGroupMemberRequest Primitive

13.1.10. The "GetGroupMemberResponse" Primitive

The *"GetGroupMemberResponse"* primitive is used for the provider server to return the result with a list of all group members.

Information Element	Req	Type	Description
Message-Type	M	GetGroupMemberResponse	Message identifier
Status-Info	M	Structure of Status-Primitive	Status information (see 5.2).
User-ID-List-Adm	O	Structure	The list of users that are in the "Administrator" list.
User-ID-List-Mod	O	Structure	The list of users that are in the "Moderator" list.
User-ID-List	O	Structure	The list of users that are ordinary members.

Table 125. Information elements in GetGroupMemberResponse Primitive

13.1.11. The "AddGroupMemberRequest" Primitive

The *"AddGroupMemberRequest"* primitive is used for the user with sufficient access rights in the requestor server to add the other user(s) to a group. The *"AddGroupMemberRequest"* primitive contains the Group-ID and the list of user(s) to be added. All of the newly added users are the ordinary members.

Information Element	Req	Type	Description
Message-Type	M	AddGroupMemberRequest	Message identifier
Meta-Information	M	Structure of Meta-Information	The meta-information (see 5.1).
Group-ID	M	String	Identifies the group
User-ID-List	O	Structure	The list of users to be added.

Table 126. Information elements in AddGroupMemberRequest Primitive

13.1.12. The "RemoveGroupMemberRequest" Primitive

The *"RemoveGroupMemberRequest"* primitive is used for the user with sufficient access rights in the requestor server to remove some users from a group. The *"RemoveGroupMemberRequest"* primitive contains the Group-ID and the list of user(s) to be removed.

Information Element	Req	Type	Description
Message-Type	M	RemoveGroupMemberRequest	Message identifier
Meta-Information	M	Structure of Meta-Information	The meta-information (5.1).
Group-ID	M	String	Identifies the group
User-ID-List	M	Structure	A list of removed users.

Table 127. Information elements in RemoveGroupMemberRequest Primitive

13.1.13. The "MemberAccessRequest" Primitive

The *"MemberAccessRequest"* primitive is used for the user with sufficient access rights in the requestor server to change the access privileges of other users.

Information Element	Req	Type	Description
Message-Type	M	MemberAccessRequest	Message identifier
Meta-Information	M	Structure of Meta-Information	The meta-information (5.1).
Group-ID	M	String	Identifies the group
User-ID-List-Adm	O	Structure	The list of users to be set in the "Administrator" list.

User-ID-List-Mod	O	Structure	The list of users to be set in the "Moderator" list.
User-ID-List	O	Structure	The list of users to be set as ordinary members.

Table 128. Information elements in MemberAccessRequest Primitive

13.1.14. The "GetGroupPropsRequest" Primitive

The *"GetGroupPropsRequest"* primitive is used for the user with sufficient access rights in the requestor server to retrieve the properties of a group, and its own properties in that particular group. The *"GetGroupPropsRequest"* primitive contains the Group-ID.

Information Element	Req	Type	Description
Message-Type	M	GetGroupPropsRequest	Message identifier
Meta-Information	M	Structure of Meta-Information	The meta-information (see 5.1).
Group-ID	M	String	Identifies the group

Table 129. Information elements in GetGroupPropsRequest Primitive

13.1.15. The "GetGroupPropsResponse" Primitive

The *"GetGroupPropsResponse"* primitive is used for the provider server to return the result with a list of group properties and own properties of the specified group.

Information Element	Req	Type	Description
Message-Type	M	GetGroupPropsResponse	Message identifier
Status-Info	M	Structure of Status-Primitive	Status information (see 5.2).
Group-Prop-List	O	Structure	The list of group properties.
Own-Prop-List	O	Structure	The list of the user's properties in that group.

Table 130. Information elements in GetGroupPropsResponse Primitive

13.1.16. The "SetGroupPropsRequest" Primitive

The *"SetGroupPropsRequest"* primitive is used for the user with sufficient access rights in the requestor server to update the properties of a group, and/or its own properties in that particular group. The *"SetGroupPropsRequest"* primitive contains the Group-ID, the new properties of the group and/or the new user properties.

Information Element	Req	Type	Description
Message-Type	M	SetGroupPropsRequest	Message identifier
Meta-Information	M	Structure of Meta-	The meta-information (see

		Information	5.1).
Group-ID	M	String	Identifies the group
Group-Prop-List	O	Structure	The list of group properties.
Own-Prop-List	O	Structure	The list of the user's properties in that group.

Table 131. Information elements in SetGroupPropsRequest Primitive

13.1.17. The "RejectListRequest" Primitive

The "*RejectListRequest*" primitive is used for the user with sufficient access rights in the requestor server to retrieve / update the reject list of a group. Users on the reject list cannot join the group.

Information Element	Req	Type	Description
Message-Type	M	RejectListRequest	Message identifier
Meta-Information	M	Structure of Meta-Information	The meta-information (see 5.1).
Group-ID	M	String	Identifies the group
Add-User-ID-List	O	Structure	The list of users to be added to the reject list
Remove-User-ID-List	O	Structure	The list of users to be removed from the reject list.

Table 132. Information elements in RejectListRequest Primitive

13.1.18. The "RejectListResponse" Primitive

The "*RejectListResponse*" primitive is used for the provider server to return the reject list of the group.

Information Element	Req	Type	Description
Message-Type	M	RejectListResponse	Message identifier
Status-Info	M	Structure of Status-Primitive	Status information (see 5.2).
Reject-User-ID-List	O	Structure	A list of users in the reject list.

Table 133. Information elements in RejectListResponse Primitive

13.1.19. The "SubscribeGroupChangeRequest" Primitive

The "*SubscribeGroupChangeRequest*" primitive is used for the user in the requestor server to subscribe to a group change notice whenever the other user leaves or joins the group, or the group properties have been changed.

Information Element	Req	Type	Description
Message-Type	M	SubscribeGroupChangeRequest	Message identifier
Meta-Information	M	Structure of Meta-Information	The meta-information (see 5.1).
Group-ID	M	String	Identifies the group

Table 134. Information elements in SubscribeGroupChangeRequest Primitive

13.1.20. The "UnsubscribeGroupChangeRequest" Primitive

The *"UnsubscribeGroupChangeRequest"* primitive is used to cancel the current subscription.

Information Element	Req	Type	Description
Message-Type	M	UnsubscribeGroupChangeRequest	Message identifier
Meta-Information	M	Structure of Meta-Information	The meta-information (see 5.1).
Group-ID	M	String	Identifies the group

Table 135. Information elements in UnsubscribeGroupChangeRequest Primitive

13.1.21. The "GetGroupSubStatusRequest" Primitive

The *"GetGroupSubStatusRequest"* primitive is used for the user in the requestor server to retrieve its subscription status to the group change notice. The *"GetGroupSubStatusRequest"* primitive contains the Group-ID.

Information Element	Req	Type	Description
Message-Type	M	GetGroupSubStatusRequest	Message identifier
Meta-Information	M	Structure of Meta-Information	The meta-information (see 5.1).
Group-ID	M	String	Identifies the group

Table 136. Information elements in GetGroupSubStatusRequest Primitive

13.1.22. The "GetGroupSubStatusResponse" Primitive

The *"GetGroupSubStatusResponse"* primitive is used for the provider server to return the result with its current subscription status to a group change notice.

Information Element	Req	Type	Description
Message-Type	M	GetGroupSubStatusResponse	Message identifier

Status-Info	M	Structure of Status-Primitive	Status information (see 5.2).
Group-ID	M	String	Identifies the group
Subscription-Status	M	'S' | 'U'	Indicates the subscription status – subscribed ('S') or not ('U').

Table 137. Information elements in GetGroupSubStatusResponse Primitive

13.1.23. The "GroupChangeNotice" Primitive

The "*GroupChangeNotice*" primitive is used for the provider server to send the notifications to the subscribed users whenever some other user leaves or joins the group, or the group properties have been changed.

Information Element	Req	Type	Description
Message-Type	M	GroupChangeNotice	Message identifier
Meta-Information	M	Structure of Meta-Information	The meta-information (see 5.1).
Subscribing-User-ID-List	M	Structure	Identifies the users who subscribed to the group change.
Group-ID	M	String	Identification of the group.
Joined-User-Screen-Name-List	O	Structure	A list of users that have joined the group since last notification. The users are identified by their screen names
Left-User-Screen-Name-List	O	Structure	A list of users that have left the group since last notification. The users are identified by their screen names
Group-Prop-List	O	Structure	The new properties of the group.
Own-Props	O	Structure	The new properties of the user in the group.

Table 138. Information elements in GroupChangeNotice Primitive

13.2. Transactions

13.2.1. The "CreateGroup" Transaction

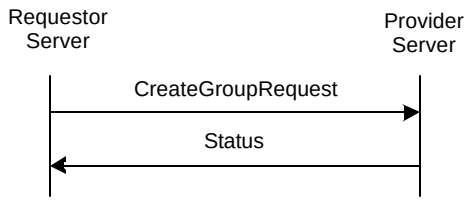

Figure 57. The "CreateGroup" Transaction

A user may create its own private group at any suitable time. The purpose of "CreateGroup" transaction is for the user in the requestor server to create its own private group.

The requestor server sends a *"CreateGroupRequest"* request to the provider server with the specified properties. The provider server returns a "*Status*" response.

This transaction belongs to the complementary service.

Primitive	Direction
CreateGroupRequest	Requestor Server → Provider Server
Status	Requestor Server ← Provider Server

Table 139. Primitive Directions for CreateGroup Transaction

13.2.2. The "DeleteGroup" Transaction

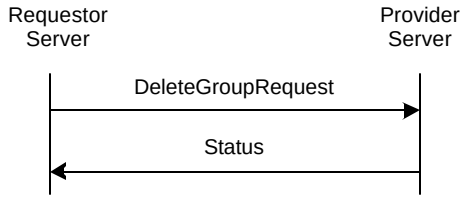

Figure 58. The "DeleteGroup" Transaction

A user with sufficient access rights may delete a private user group at any suitable time.

The requestor server sends a *"DeleteGroupRequest"* request to the provider server with the Group-ID. The provider server removes all currently joined users from the group (ServerInitiatedLeaveGroup transaction), deletes the specified group, and responds with a "*Status*" message.

This transaction belongs to the complementary service.

Primitive	Direction
DeleteGroupRequest	Requestor Server → Provider Server
Status	Requestor Server ← Provider Server

Table 140. Primitive Directions for DeleteGroup Transaction

13.2.3. The "JoinGroup" Transaction

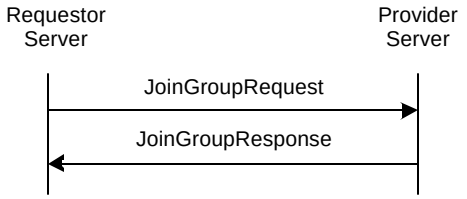

Figure 59. The "JoinGroup" Transaction

A user may join a discussion group at any suitable time.

The requestor server sends a *"JoinGroupRequest"* request to the provider server with the Group-ID, its screen name shown during the discussion, and the joined users' list request. The provider server returns a *"JoinGroupResponse"* response including the processing result with the list of currently joined users (if requested), and optionally a welcome note.

After a user successfully joined the group, the user may receive / send messages from / to the particular group.

Primitive	Direction
JoinGroupRequest	Requestor Server → Provider Server
JoinGroupResponse	Requestor Server ← Provider Server

Table 141. Primitive Directions for JoinGroup Transaction

13.2.4. The "LeaveGroup" Transaction

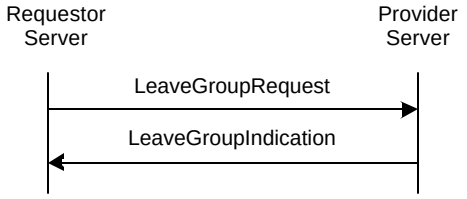

Figure 60. The "LeaveGroup" Transaction

A user may leave a discussion group at any suitable time.

The requestor server sends a *"LeaveGroupRequest"* request to the provider server with the Group-ID. The provider server returns a *"LeaveGroupIndication"* response.

Primitive	Direction
LeaveGroupRequest	Requestor Server → Provider Server
LeaveGroupIndication	Requestor Server ← Provider Server

Table 142. Primitive Directions for LeaveGroup Transaction

13.2.5. The "ServerInitiatedLeaveGroup" Transaction

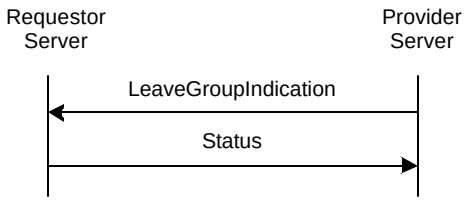

Figure 61. The "ServerInitiatedLeaveGroup" Transaction

A server may initiate a group leaving due to user kickout, group deletion etc.

The provider server sends a "*LeaveGroupIndication*" request to the requestor server.

Primitive	Direction
LeaveGroupIndication	Requestor Server ← Provider Server

Table 143. Primitive Directions for ServerInitiatedLeaveGroup Transaction

13.2.6. The "GetJoinedMember" Transaction

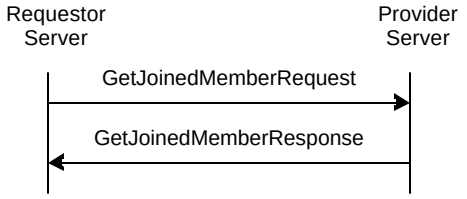

Figure 62. The "GetJoinedMember" Transaction

This transaction belongs to the complementary service.

Primitive	Direction
GetJoinedMemberRequest	Requestor Server → Provider Server
GetJoinedMemberResponse	Requestor Server ← Provider Server

Table 144. Primitive Directions for GetJoinedMember Transaction

13.2.7. The "GetGroupMember" Transaction

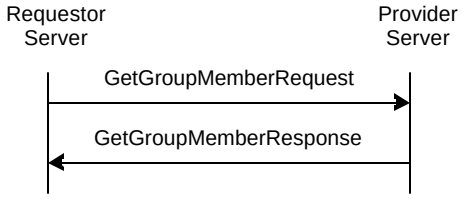

Figure 63. The "GetGroupMember" Transaction

A user with sufficient access rights may retrieve the member list of a group.

The requestor server sends a *"GetGroupMemberRequest"* request to the provider server with the Group-ID. The provider server returns a *"GetGroupMemberResponse"* response with the list of all group members.

This transaction belongs to the complementary service.

Primitive	Direction
GetGroupMemberRequest	Requestor Server → Provider Server
GetGroupMemberResponse	Requestor Server ← Provider Server

Table 145. Primitive Directions for GetGroupMember Transaction

13.2.8. The "AddGroupMember" Transaction

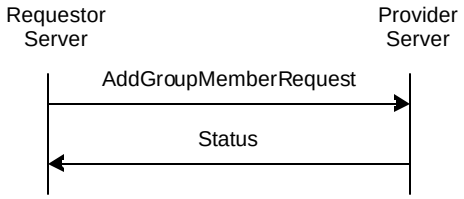

Figure 64. The "AddGroupMember" Transaction

A user with sufficient access rights may add user(s) to the member list of a group.

The requestor server sends a *"AddGroupMemberRequest"* request to the provider server with the Group-ID and the list(s) of users to be added. The provider server returns a "*Status*" response.

This transaction belongs to the complementary service.

Primitive	Direction
AddGroupMemberRequest	Requestor Server → Provider Server
Status	Requestor Server ← Provider Server

Table 146. Primitive Directions for AddGroupMember Transaction

13.2.9. The "RemoveGroupMember" Transaction

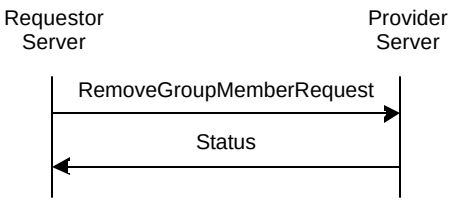

Figure 65. The "RemoveGroupMember" Transaction

A user with sufficient access rights may remove user(s) from the member list of a group.

The requestor server sends a "*RemoveGroupMemberRequest*" request to the provider server with the Group-ID and the list(s) of users to be removed. The provider server returns a "*Status*" response.

This transaction belongs to the complementary service.

Primitive	Direction
RemoveGroupMemberRequest	Requestor Server → Provider Server
Status	Requestor Server ← Provider Server

Table 147. Primitive Directions for RemoveGroupMember Transaction

13.2.10. The "MemberAccess" Transaction

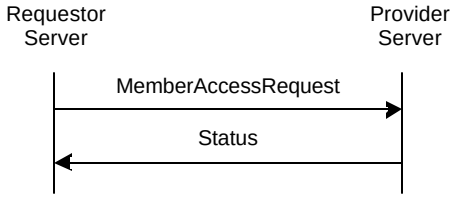

Figure 66. The "MemberAccess" Transaction

This transaction belongs to the complementary service.

Primitive	Direction
MemberAccessRequest	Requestor Server → Provider Server
Status	Requestor Server ← Provider Server

Table 148. Primitive Directions for MemberAccess Transaction

13.2.11. The "GetGroupProps" Transaction

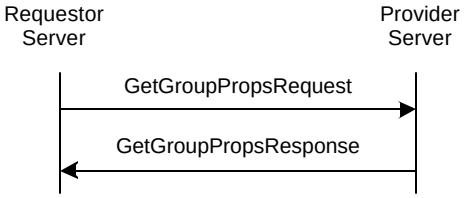

Figure 67. The "GetGroupProps" Transaction

A user with sufficient access rights may retrieve the properties of a group, and its own properties in that particular group.

The requestor server sends a "*GetGroupPropsRequest*" request to the provider server with the Group-ID. The provider server returns a "*GetGroupPropsResponse*" response with the list of group properties and own properties of the specified group.

Primitive	Direction
GetGroupPropsRequest	Requestor Server → Provider Server
GetGroupPropsResponse	Requestor Server ← Provider Server

Table 149. Primitive Directions for GetGroupProps Transaction

13.2.12. The "SetGroupProps" Transaction

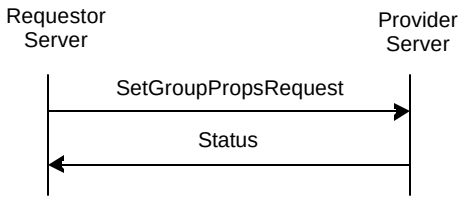

Figure 68. The "SetGroupProps" Transaction

A user with sufficient access rights may update the properties of a group, and/or its own properties in that particular group.

The requestor server sends a *"SetGroupPropsRequest"* request to the provider server with the Group-ID, the new properties of the group and/or the new user properties. The provider server returns a "*Status*" response.

Primitive	Direction
SetGroupPropsRequest	Requestor Server → Provider Server
Status	Requestor Server ← Provider Server

Table 150. Primitive Directions for SetGroupProps Transaction

13.2.13. The "RejectList" Transaction

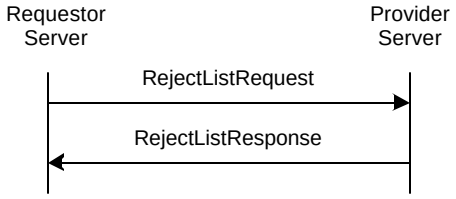

Figure 69. The "RejectList" Transaction

This transaction belongs to the complementary service.

Primitive	Direction
RejectListRequest	Requestor Server → Provider Server
RejectListResponse	Requestor Server ← Provider Server

Table 151. Primitive Directions for RejectList Transaction

13.2.14. The "SubscribeGroupChange" Transaction

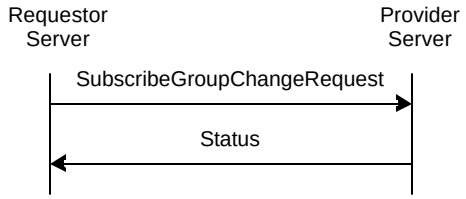

Figure 70. The "SubscribeGroupChange" Transaction

A user may subscribe to a group change notice whenever the other user leaves or joins the group, or the group properties have been changed.

The requestor server sends a *"SubscribeGroupChangeRequest"* request to the provider server with the Group-ID and an optional subscription expiration time. The provider server returns a *"Status"* response.

Primitive	Direction
SubscribeGroupChangeRequest	Requestor Server → Provider Server
Status	Requestor Server ← Provider Server

Table 152. Primitive Directions for SubscribeGroupChange Transaction

13.2.15. The "UnsubscribeGroupChange" Transaction

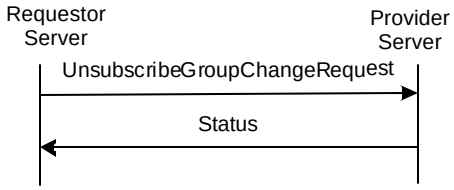

Figure 71. The "UnsubscribeGroupChange" Transaction

A user may cancel the subscription to the group change notice.

The requestor server sends a *"UnsubscribeGroupChangeRequest"* request to the provider server with the Group-ID. The provider server returns a *"Status"* response.

Primitive	Direction
UnsubscribeGroupChangeRequest	Requestor Server → Provider Server
Status	Requestor Server ← Provider Server

Table 153. Primitive Directions for UnsubscribeGroupChange Transaction

13.2.16. The "GetGroupSubStatus" Transaction

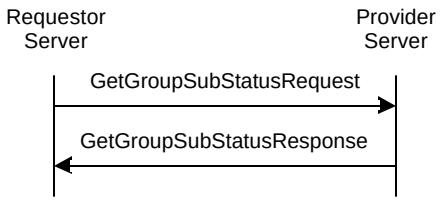

Figure 72. The "GetGroupSubStatus" Transaction

A user may retrieve its subscription status to a group change notice.

The requestor server sends a *"GetGroupSubStatusRequest"* request to the provider server with the Group-ID. The provider server returns a *"GetGroupSubStatusResponse"* response with its current subscription status to a group change notice.

Primitive	Direction
GetGroupSubStatusRequest	Requestor Server → Provider Server
GetGroupSubStatusResponse	Requestor Server ← Provider Server

Table 154. Primitive Directions for GetGroupSubStatus Transaction

13.2.17. The "NotifyGroupChange" Transaction

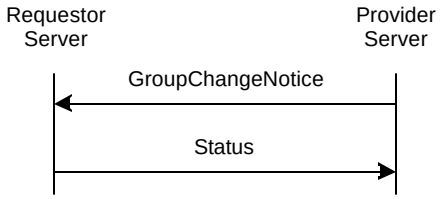

Figure 73. The "NotifyGroupChange" Transaction

The server may send group change notification(s) to the subscribed users whenever some other user leaves or joins the group, or the group properties have been changed.

The provider server sends a *"GroupChangeNotice"* request to the requestor server with a list of recently joined or left users, or the new properties of the group.

Primitive	Direction
GroupChangeNotice	Requestor Server ← Provider Server
Status	Requestor Server → Provider Server

Table 155. Primitive Directions for NotifyGroupChange Transaction

13.3. Status Code

13.3.1. "CreateGroup" Transaction

- Service Not Supported (405)
- Not logged in (604)

- 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 for the server (815)

13.3.2. "DeleteGroup" Transaction

- Service Not Supported (405)
- Not logged in (604)
- Group does not exist (800)
- Group is public (804)
- Insufficient group privileges (816)

13.3.3. "JoinGroup" Transaction

- Service Not Supported (405)
- Not logged in (604)
- Group does not exist (800)
- User already joined (807)
- Cannot join: "rejected"(809)
- Cannot join with the specified screen name; it is already in use (811)
- Insufficient group privileges (816)
- The maximum number of allowed users has been reached (817)

13.3.4. "LeaveGroup" Transaction

- Service Not Supported (405)
- Not logged in (604)
- Group was not joined before transaction (808)

13.3.5. Group Membership Transactions

- Service Not Supported (405)
- Unknown user (531)
- Not logged in (604)
- Group does not exist (800)
- Insufficient group privileges (816)

13.3.6. Group Properties Transactions

- Service Not Supported (405)
- Not logged in (604)
- Group does not exist (800).
- Invalid group attribute(s) (806).
- Insufficient group privileges (816).

13.3.7. "RejectList" Transaction

- Service Not Supported (405)
- User unknown (531).
- Not logged in (604)
- Group does not exist (800).
- Insufficient group privileges (816).

13.3.8. Group Change Transactions

- Service Not Supported (405)
- Not logged in (604)
- Group does not exist (800)

13.3.9. "GetJoinedMember" Transaction

- Group does not exist (800).

14. Status Codes and Descriptions

SSP 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 SSP and the status codes in HTTP. The following sections define the general categories as well as each status code.

14.1. 1xx – Informational

The client or server 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 client or server agent SHALL ignore unexpected 1xx status code. This category of the status codes does not finish a transaction.

14.1.1. 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.

14.1.2. 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.

14.1.3. 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.

14.1.4. 104 – Server Queued

The client MAY continue with its next requests. The server has accepted the request, but does not have resources to start processing. This status is used to indicate the overload of the server and therefore it is expected, that the client will (re)direct the next requests to other possible connections between the servers. The request processing will take place and the server MUST send a final response again upon completing the request.

14.2. 2xx – Successful

The 2xx class of status codes indicates that the client's request was successfully received, understood and accepted.

14.2.1. 200 – Successful

This is used to indicate that the request succeeded.

14.2.2. 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.

14.2.3. 202 – Accepted

This is used to indicate that server accepted the request, but not able to receive acknowledgment about delivery to client device. The request might or might not eventually be acted upon. There is no facility for re-sending a status code from an asynchronous operation such as this.

14.3. 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.

14.3.1. 400 – Bad Request

The server could not understand the request due to the malformed syntax. The client SHALL NOT repeat the request without modification.

14.3.2. 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.

14.3.3. 402 – Bad Parameter

The server cannot understand one of the parameters in the request. The client SHALL NOT repeat the request without modification.

14.3.4. 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 is also returned if user not login in the network yet.

14.3.5. 404 - Not Found

The server cannot find anything matching the request. No indication is given of whether the condition is temporary or permanent.

14.3.6. 405 – Service Not Supported

The server does not support the service method in the request.

14.3.7. 410 – Unable to Delivery

The server cannot deliver the request. The requested resource is no longer available at the server and no forwarding address is known.

14.3.8. 415 – Unsupported Media Type

The server cannot deliver the request, because the client cannot support the format of the entity that it requested.

14.3.9. 420 – Invalid Transaction-ID

The server encountered an invalid Transaction-ID.

14.3.10. 422 – User-ID and Client-ID Does Not Match

The User-ID and the Client-ID does not match in the request.

14.3.11. 423 – Invalid Invitation-ID

The server encountered an invalid invitation ID.

14.3.12. 424 – Invalid Search-ID

The server encountered an invalid search ID.

14.3.13. 425 – Invalid Search-Index

The server encountered an invalid search index.

14.3.14. 426 – Invalid Message-ID

The server encountered an invalid Message-ID.

14.3.15. 431 – Unauthorized Group Membership

The user agent is not an authorized member of the group.

14.4. 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.

14.4.1. 500 – Internal Server Error

The provider server encountered an unexpected condition, which prevented it from fulfilling the request.

14.4.2. 501 – Not Implemented

The server does not support the functionality required to fulfill 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.

14.4.3. 503 – Service Unavailable

The server is currently unable to handle the request due to a temporally overloading of the server.

14.4.4. 504 – Invalid Timeout

The provider server has not returned the response within the repeat time.

14.4.5. 505 – Version Not Supported

The server does not support, or refuse to support, the request version that was used. The response should contain the preferred supported version.

14.4.6. 506 – Service Not Agreed

The service request refers to a service which does not corresponds to the service agreement between the service requestor and provider server. The requestor server SHALL NOT repeat the request without a new service negotiation.

14.4.7. 507 – Message Queue is Full

The server cannot fulfill the request, because its message queue is full. The client MAY repeat the request.

14.4.8. 516 – Domain Not Supported

The server does not support forwarding to different domain space.

14.4.9. 521 – Unresponded Presence Request

The presence information provider does not respond the presence service specified in the request.

14.4.10. 522 – Unresponded Group Request

The group service provider does not respond the requested group transaction.

14.4.11. 531 – Unknown User

The specified user is unknown / User-ID is invalid.

14.4.12. 532 – Message Recipient Blocked the Sender

The recipient of the message blocked the sender.

14.4.13. 533 – Message Recipient Not Logged in

The recipient of the message is not logged in.

14.4.14. 534 – Message Recipient Unauthorized

The recipient of the message is not authorized.

14.4.15. 535 – Search Timed Out

The server has invalidated the requested search-request.

14.4.16. 536 – Unknown Transaction

In the request any of the information elements of the Meta-Information are invalid.

14.5. 6xx – Session

The 6xx class status code indicates the session-related status.

14.5.1. 600 – Session Expired

The server connection was disconnected because time-to-live parameter of provider session has expired.

14.5.2. 601 – Forced Logout

The provider server has disconnected the requestor server.

14.5.3. 604 – Invalid Session / Not Logged In

There is no such session. (Previously not logged in, disconnected, or logged out.)

14.5.4. 606 – Invalid Service-ID

Unknown Service-ID.

14.5.5. 607 – Redirection Refused

The redirected connection is refused.

14.5.6. 608 – Invalid Password

The password provided by the requestor server was incorrect, it does not match with the given Service-ID. The requestor SHALL NOT repeat the request without modification.

14.5.7. 609 – Connection Expired

The connection was disconnected because time-to-live parameter has expired. This is NOT the last active connection pair.

14.5.8. 610 – Server Search Limit is Exceeded

The search limit exceeds the server limit.

14.5.9. 620 – Invalid Server Session

There is no such session. (Previously not logged in, disconnected, or logged out.) If only the session-ID is invalid in the Meta-information, this error indication should be used instead of Unknown transaction.

14.6. 7xx – Presence and contact list

The 7xx class indicates the presence and contact list related status codes.

14.6.1. 700 – Contact List Does Not Exist

The contact list specified in the request does not exist.

14.6.2. 701 – Contact List Already Exists

The contact list specified in the request already exists.

14.6.3. 702 – Invalid or Unsupported User Properties

The user properties specified in the request are invalid, or not supported.

14.6.4. 750 – Invalid or Unsupported Presence Attributes

The presence attributes specified in the request are invalid, or not supported.

14.6.5. 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.

14.6.6. 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.

14.7. 8xx – Groups

The 8xx class indicates the group-related status codes.

14.7.1. 800 – Group Does Not Exist

The group specified in the request does not exist.

14.7.2. 801 – Group Already Exists

The group specified in the request already exists.

14.7.3. 802 – Group is Open

The group specified in the request is an open group.

14.7.4. 803 – Group is Closed

The group specified in the request is a closed group.

14.7.5. 804 – Group is Public

The group specified in the request is public.

14.7.6. 805 – Group Private

The group specified in the request is private.

14.7.7. 806 – Invalid / Unsupported Group Properties

The group properties specified in the request are invalid or not supported.

14.7.8. 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, this error code is used to indicate that the user is already joined the particular group.

14.7.9. 808 – Group is Not Joined

The request cannot be processed, because it requires the user to be joined to the group.

14.7.10. 809 – Rejected

The user has been rejected from the particular group. He/she is forced to leave the group, and cannot join.

14.7.11. 810 – Not a Group Member

The request cannot be processed, because the user is not a member of the specified closed group.

14.7.12. 811 – Screen Name Already in Use

The screen name specified in the request is already in use. If the server does not allow the same screen name to be used in a group more than once, then this error code is used to indicate that the screen name is already in use. The requesting user may try to change his/her screen name, and repeat the transaction.

14.7.13. 812 – Private Messaging is Disabled for Group

The client requested private message delivery, but the private messaging is disabled in the particular group.

14.7.14. 813 – Private Messaging is Disabled for User

The client requested private message delivery, but the private messaging is disabled for the particular user.

14.7.15. 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; so additional groups cannot be created. The client SHOULD NOT repeat the request until a group that belongs to the particular user has been deleted.

14.7.16. 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; so additional groups cannot be created. The client MAY repeat the request.

14.7.17. 816 – Insufficient Group Privileges

The user does not have sufficient privileges in the particular group to perform the requested operation. The client SHOULD NOT repeat the request until the user has been authorized properly.

14.7.18. 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.

14.7.19. 821 – History is Not Supported

The server does not support group message history caching.