# Server-Server Protocol Semantics

Approved Version 1.3 – 23 Jan 2007

**Open Mobile Alliance**

OMA-TS-IMPS_SSP-V1_3-20070123-A

# Contents

# Figures

# Tables

# 1 Scope

The OMA Instant Messaging and Presence Service (IMPS) includes four primary features:

- Presence
- Instant Messaging
- Groups
- Shared Content

Presence is the key enabling technology for IMPS. It includes client device availability (my phone is on/off, in a call), user status (available, unavailable, in a meeting), location, client device capabilities (voice, text, GPRS, multimedia) and searchable personal statuses such as mood (happy, angry) and hobbies (football, fishing, computing, dancing). Since presence information is personal, it is only made available according to the user's wishes - access control features put the control of the user presence information in the users' hands.

Instant Messaging (IM) is a familiar concept in both the mobile and desktop worlds. Desktop IM clients, two-way SMS and two-way paging are all forms of Instant Messaging. OMA IM will enable interoperable mobile IM in concert with other innovative features to provide an enhanced user experience.

Groups or chat are a fun and familiar concept on the Internet. Both operators and end-users are able to create and manage groups. Users can invite their friends and family to chat in group discussions. Operators can build common interest groups where end-users can meet each other online.

Shared Content allows users and operators to setup their own storage area where they can post pictures, music and other multimedia content while enabling the sharing with other individuals and groups in an IM or chat session.

These features, taken in part or as a whole, provide the basis for innovative new services that build upon a common interoperable framework.

# 2 References

## 2.1 Normative References


**[E.164]** ITU-T Recommendation E.164 (025/0597) The international public telecommunication numbering plan, URL: http://www.itu.int/search/searchredirect.asp?recommendation.asp?type=items&lang=E&parent=T-REC-E.164-200502-II

**[FIPS 180-1]** "Secure Hash Standard", April 1995, URL: http://csrc.nist.gov/publications/fips/fips180-1/fip180-1.pdf

**[IOPPROC]** "OMA Interoperability Policy and Process", Version 1.1, Open Mobile Alliance™, OMA-IOP-Process-V1_1, http://www.openmobilealliance.org/

**[RFC1320]** "The MD4 Message-Digest Algorithm", April 1992. URL: http://www.ietf.org/rfc/rfc1320.txt?number=1320

**[RFC1321]** "The MD5 Message-Digest Algorithm", April 1992. URL: http://www.ietf.org/rfc/rfc1321.txt?number=1321

**[RFC2045]** "Multipurpose Internet Mail Extensions (MIME) Part one: Format of Internet Message Bodies". Section 6.8 "Base64 Content-Transfer-Encoding", November 1996. URL: http://www.ietf.org/rfc/rfc2045.txt?number=2045

**[RFC2046]** "MIME (Multipurpose Internet Mail Extensions) Part Two: Media Types". Borenstein N., and N. Freed. November 1996. URL: http://www.ietf.org/rfc/rfc2046.txt?number=2046

**[RFC2119]** "Key words for use in RFCs to Indicate Requirement Levels". S. Bradner. March 1997. URL: http://www.ietf.org/rfc/rfc2119.txt

**[RFC2822]** "Standard for the Format of ARPA Internet Text Messages".August 1982. URL: http://www.ietf.org/rfc/rfc2822.txt?number=2822

**[XML]** "Extensible Markup Language 1.0 (Second Edition)", W3C recommendation, October 2000. URL: http://www.w3.org/TR/2000/REC-xml-20001006.pdf


## 2.2 Informative References


**[Arch]** "IMPS Architecture", Version 1.3, Open Mobile Alliance™, OMA-AD-IMPS-V1_3, URL: http://www.openmobilealliance.org

**[CSP]** "Client-Server Protocol Session and Transactions", Version 1.3, Open Mobile Alliance™, OMA-TS-IMPS_CSP-V1_3, URL: http://www.openmobilealliance.org

**[CSP DataType]** "Client-Server Protocol Data Types", Version 1.3, Open Mobile Alliance™, OMA-TS-IMPS_CSP_Data_Types-V1_3, URL: http://www.openmobilealliance.org

**[CSP PTS]** "Client-Server Protocol Plain Text Syntax", Version 1.3, Open Mobile Alliance™, OMA-TS-IMPS_CSP_PTS-V1_3, URL: http://www.openmobilealliance.org

**[CSP Trans]** "Client-Server Protocol Transport Bindings", Version 1.3, Open Mobile Alliance™, OMA-TS-IMPS_CSP_Transport-V1_3, URL: http://www.openmobilealliance.org

**[CSP WBXML]** "Client-Server Protocol Binary XML Definition and Examples", Version 1.3, Open Mobile Alliance™, OMA-TS-IMPS_CSP_WBXML-V1_3, URL: http://www.openmobilealliance.org

**[CSP XMLS]** "Client-Server Protocol XML Syntax", Version 1.3, Open Mobile Alliance™, OMA-TS-IMPS_CSP_XMLS, URL: http://www.openmobilealliance.org

**[PA]** "Presence Attributes", Version 1.3, Open Mobile Alliance™, OMA-TS-IMPS_PA-V1_3, URL: http://www.openmobilealliance.org

**[PA XMLS]** "Presence Attribute XML Syntax", Version 1.3, Open Mobile Alliance™, OMA-TS-IMPS_PA_XMLS-V1_3, URL: http://www.openmobilealliance.org

**[SSP]** "Server-Server Protocol Semantics", Version 1.3, Open Mobile Alliance™, OMA-TS-IMPS_SSP-V1_3, URL: http://www.openmobilealliance.org

**[SSP Syntax]** "Server-Server Protocol XML Syntax", Version 1.3, Open Mobile Alliance™, OMA-TS-IMPS_SSP_XMLS-V1_3, URL: http://www.openmobilealliance.org

**[SSP Trans]** "Server-Server Protocol Transport Binding", Version 1.3, Open Mobile Alliance™, OMA-TS-IMPS_SSP_Transport-V1_3, URL: http://www.openmobilealliance.org

# 3 Terminology and Conventions

## 3.1 Conventions

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

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

## 3.2 Definitions

The following definitions are for terms specific to IMPS 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, which the user subscribes to, and in which the user is authenticated and authorized to use IMPS services |
| **Primary Service Element** | refers to a Service Element of an IMPS service for a client. A PSE may be in the Home Domain of the client, or in the other domain. |
| **Complementary Service** | refers to a 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 is negotiated. |
| **Requestor Server** | the WV server, which requests the services from the Provider Server in the frame of a session after the successful service agreement is negotiated. |
| **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 definitions in RFC2119.

See also CSP Sessions and Transactions [CSP] for definitions common to CSP and SSP.

## 3.3 Abbreviations

| | |
|---|---|
| **ARPA** | Advanced Research Projects Agency |
| **DTD** | Document Type Definition |
| **HTTP** | Hypertext Transfer Protocol |
| **IANA** | Internet Assigned Number Authority |
| **IETF** | Internet Engineering Task Force |
| **WAP** | Wireless Application Protocol |

See also CSP Sessions and Transactions [CSP] for abgreviations common to CSP and SSP.

# 4 Introduction

The OMA IMPS Server-Server Protocol (SSP) provides the communication and interaction means between different IMPS service domains. SSP allows the OMA IMPS clients to subscribe to the IMPS services provided by different service providers that are distributed across the network. SSP allows the OMA IMPS 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 user #1 that subscribes to the OMA IMPS services at Service Provider A can communicate with user #2 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.

# 5 Server-Server Protocol

## 5.1 SSP Interoperability Model

The term “Home Domain” is the domain 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 be in a remote domain.

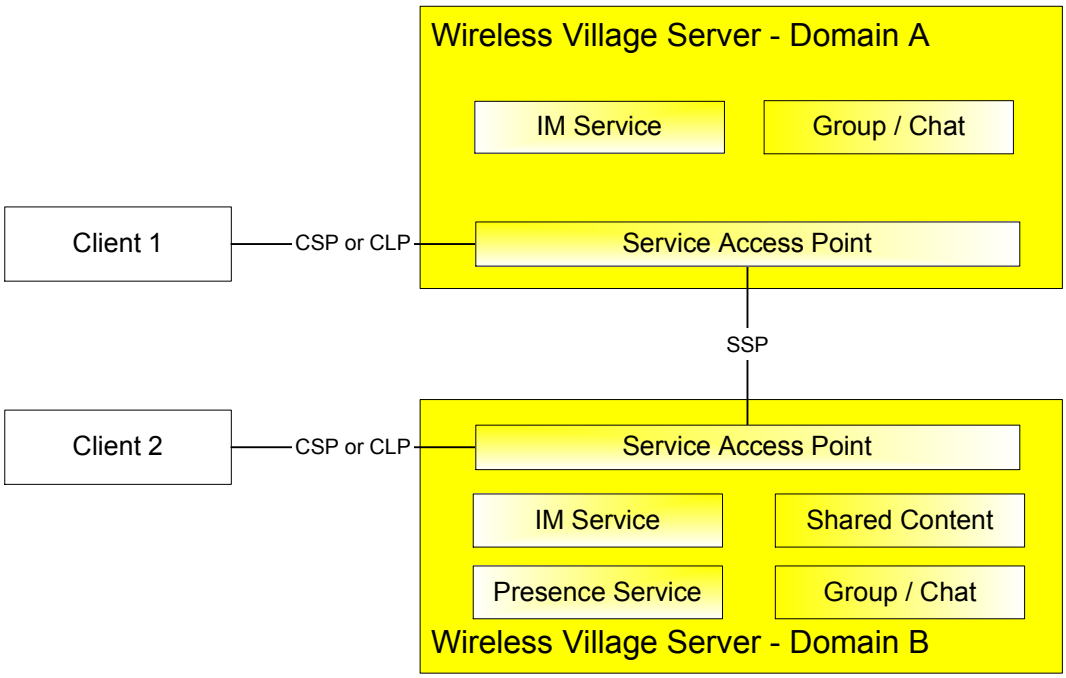


**Figure 1: The SSP Minimum Interoperability Model**

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 a complete set of IMPS services that 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.

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

In the example inFigure 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 OMA IMPS 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.

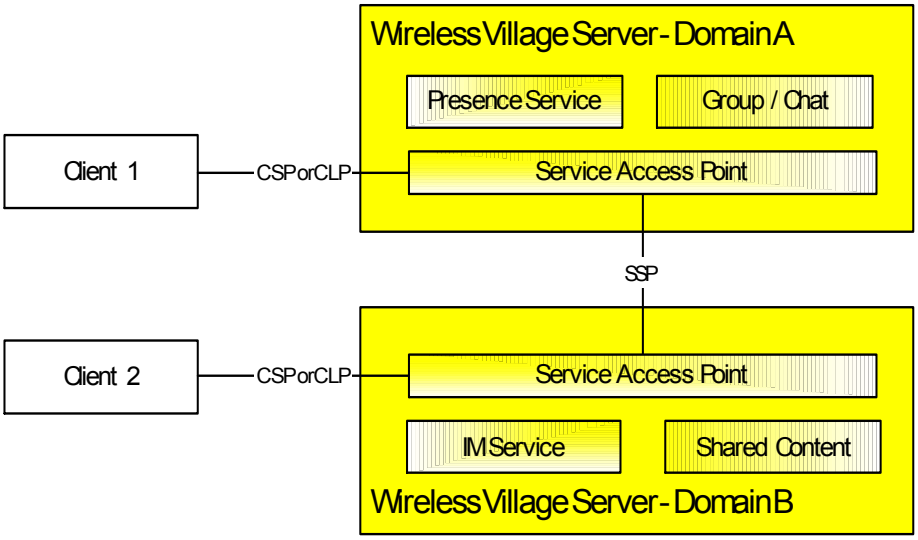


**Figure 2: The SSP Full Interoperability Model**

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 OMA IMPS SSP 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 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 as the "logical" Service Provider role for each downstream domain, and act as the "logical" Service Requestor role for each upstream domain.

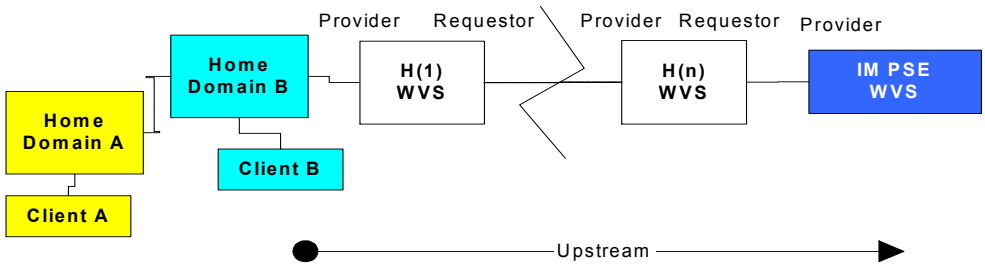


**Figure 3: The SSP Service Relay**

At each OMA IMPS 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 a Home Domain shall appropriately map the CSP service request from the client to the SSP service request, and/or map the SSP service result to CSP service result to the client.

## 5.2 SSP Interoperability Rules

In SSP Interoperability, the Home Domains MUST have direct SSP connections 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 MUST 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 MUST 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 MUST 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 5.4.4). 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 5.4.3).

An intermediate domain MUST 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 MUST be offline configured based on the service agreement. If the routing table is used in PSE, it MUST override the routing Rule 3 (e.g. Figure 8 in section 5.4.6).

If the group or presence service is complementary service, the Home Domain MUST forward the message to its complementary service in spite of the fact, that the network entity identifier (group-ID or ContactList-ID) addressed may contain another Domain part, than the Service-ID of the complementary service itself.

In minimal interoperability case, the Invite, CancelInvite, SendMessage and ForwardMessage transactions sent to screen name, distributed by the group owner Domain, MUST be accepted by the recipient Domains in spite of the fact, that it was initiated by an other Domain's user (indicated in the Meta Information element.

## 5.3 SSP Service Agreement and Routing

The exchange of messages between OMA IMPS domains is normally performed in one hop over an established direct SSP connection. However, OMA IMPS 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 agreements 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 agreements and service agreements that contains routing agreements and configuration. Each OMA IMPS Server holds a static list of direct connected neighbors. The list specifies the agreed domains that may be forwarded to one of the direct connected servers.

## 5.4 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 5.2.

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

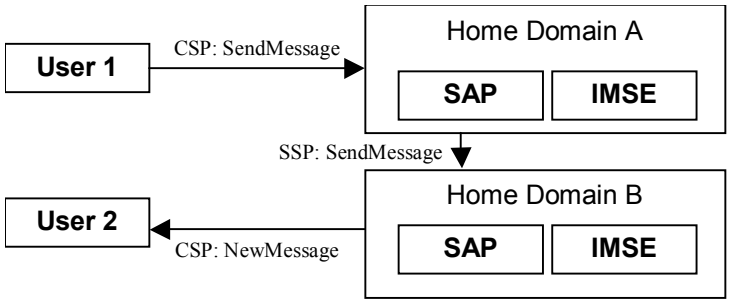


**Figure 4: The SSP IOP Case One**

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 client 2 is:

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

### 5.4.2 Case 2 – Two Users are Located in the same Home Domain

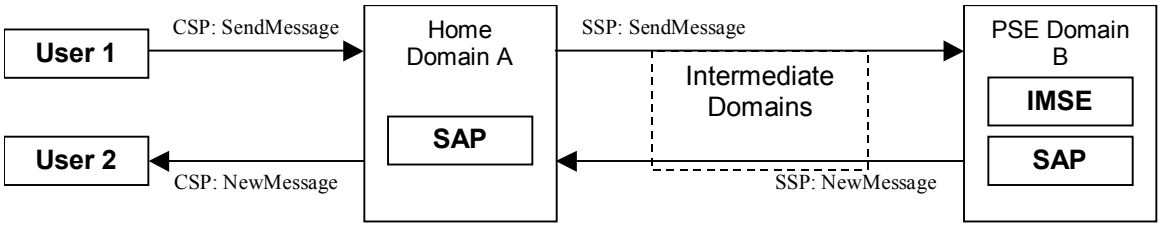


**Figure 5: The SSP IOP Case Two**

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.

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

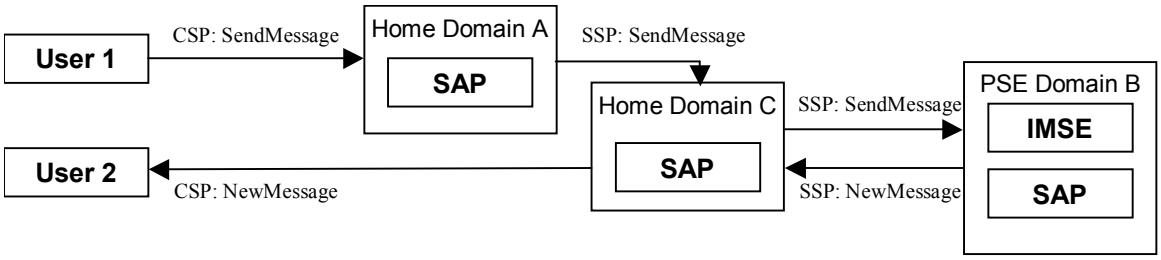


**Figure 6: The SSP IOP Case Three**

In the example in Figure 6, Domain A and C have a 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

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

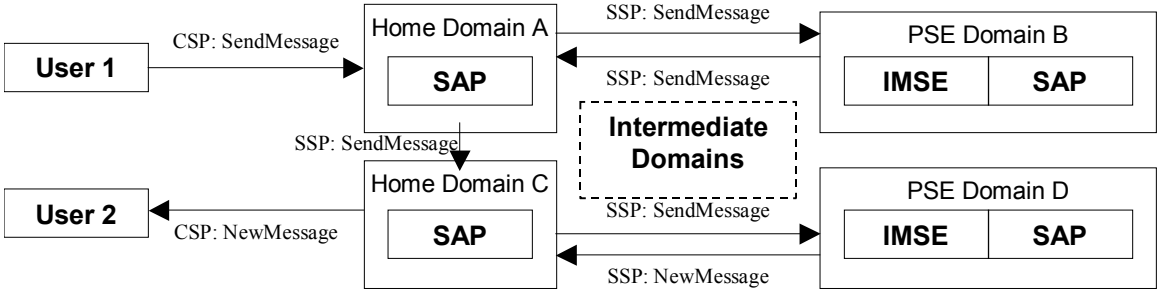


**Figure 7: The SSP IOP Case Four**

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 client 2 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

### 5.4.5 Special Case Processing

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

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

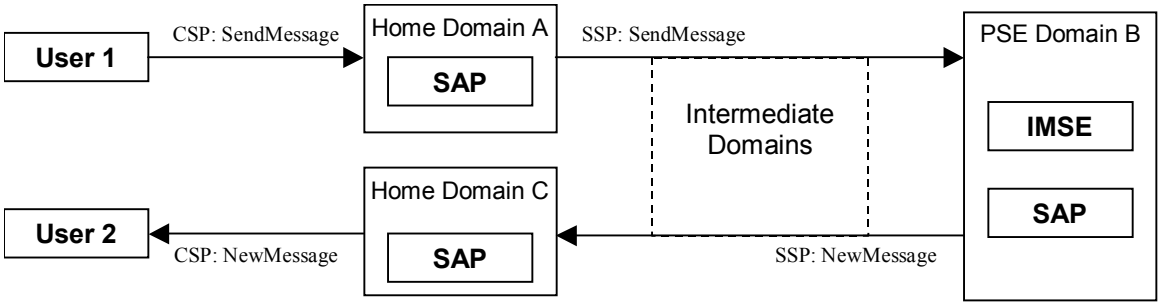


**Figure 8: The SSP IOP Special Case**

In the example in Figure 8, client 1 is located in home domain A, 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:

6. C1 -> DA: CSP-SendMessage
7. DA -> DB: SSP-SendMessage (through intermediate domains via routing)
8. DB -> DC, SSP-NewMessage (after checking block list etc.)
9. 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.

# 5.5 SSP Protocol Stack

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

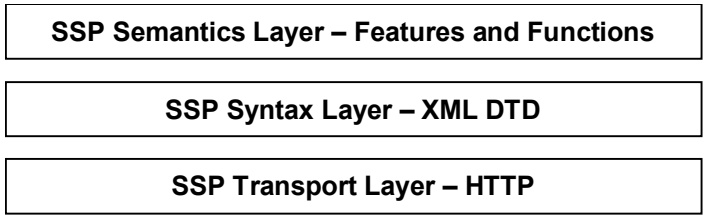


**Figure 9: The SSP Protocol Stack**

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 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 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.

# 6 Protocol Introduction

SSP is based on the architecture model described in the *"System Architecture Model"* document [Arch] and focuses on the communication and interaction among the OMA IMPS 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 [FeaFun]. The semantics of SSP supports the semantics of Client to Server Protocol (CSP) [CSP] in a distributed environment to achieve full interoperability.

## 6.1 Basics

### 6.1.1 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.

A session MUST be identified with a unique Session-ID. The Session-ID MUST be kept the same within the session. Each Provider Server MUST maintain one session for each Requestor Server. There MUST be at least two sessions between two domains. Each server MUSTmaintains one session to provide the other with its own negotiated IMPS services

### 6.1.2 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.

### 6.1.3 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.

### 6.1.4 Primitive

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

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

## 6.2 Session Pair vs. Connections

There MUST be 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 MUST be 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 pair 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 [SSP Trans] about how the connection (pair) is bound to the underlying transport.

# 6.3 Addressing

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

The definition of SSP address is based on the URI [RFC2396]. The addressable entities are:

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

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

IMPS server MUST forward invitations and instant messages containing non-IMPS address over SSP to a IWF to support delivery of these messages to non-IMPS systems.

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

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

## 6.3.1 Generic SSP address format

The generic SSP address syntax is based on URI [RFC2396]. The “wv” schema in the URI indicates the IMPS address space. The generic syntax is defined as follows:

```
IMPS-Address          = Service-ID | Message-ID | WV-URI
```

WV-URI specifies either a user or a resource and is defined in chapter 5.3.2 of [CSP]. ]. For SSP, only external addressing MUST be used. This makes the Domain-part mandatory.

The Service-ID is described in chapter 6.3.8

The Message-ID is described in chapter 6.3.9

The IMPS Servers MUST support the conversion from local addressing to external addressing.

## 6.3.2 Address encoding

See chapter 5.3.3 in [CSP]

### 6.3.3 User addressing

The User addressing is defined in chapter 5.3.4 of [CSP]. For SSP, only external addressing MUST be used. This makes the Domain-part mandatory.

### 6.3.4 Contact List Addressing and Contact-List-ID

The Contact List Addressing is defined in chapter 5.3.5 of [CSP]. ]. For SSP, only external addressing MUST be used. This makes the Domain-part mandatory.

### 6.3.5 Group Addressing

The Group Addressing is defined in chapter 5.3.6 of [CSP]. ]. For SSP, only external addressing MUST be used. This makes the Domain-part mandatory.

### 6.3.6 Client Addressing

The Client Addressing is defined in chapter 5.3.7 of [CSP].

### 6.3.7 Application Addressing

The Application Addressing is defined in chapter 5.3.8 of [CSP].

### 6.3.8 Service Addressing

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 IMPS entities that have the same Domain part in their IMPS addresses. The Domain is associated with one IMPS 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 9.1.1 and 9.2.1) and other SSP management functions.

The Service-ID is used as part of the meta-information in the SSP transactions (refer to section 8.1).

An examples of the Service-ID is:

```
Service-ID:          wv:@imps.com
```

### 6.3.9 Message Addressing

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.
```

## 6.4 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.

### 6.4.1 Char

A "*Char*" type element is a single character encoded in UTF-8. The "*Char*" type MUST be supported.

### 6.4.2 Integer

An "*Integer*" type element is a 32-bit decimal number ranging in [0, $2^{32}$ - 1]. The "*Integer*" type MUST be supported.

### 6.4.3 String

A "*String*" type element is a sequence of "*Char*" elements. The "*String*" type MUST be supported.

### 6.4.4 Boolean

A "*Boolean*" type element is either "True" or "False". The "*Boolean*" type MUST be supported.

### 6.4.5 Enum

An "*Enum*" type element is one of the pre-defined set of values. The "*Enum*" type MUST be supported.

### 6.4.6 DateTime

A "*DateTime*" type element MUST follow the ISO-8601 specification and is expressed in a "*String*" type element. The "*DateTime*" type MUST be supported. The date and time format shall be complete date and time using the basic format. The time SHALL always be indicated in Coordinated Universal Time (UTC). Example of UTC time is:

```
20011019T095031Z
```

### 6.4.7 Structure

A "*Structure*" type element is the combination of other types of elements as specified. The "*Structure*" MUST be supported.

## 6.5 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.

### 6.5.1 Host-ID

The Host-ID is the primary (Master) host address of the SAP of the IMPS server or Proprietary Gateway. The Host-ID MUST be used for establishing the session with this IMPS 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 MUST NOT be changed during a session.

An example of Host-ID is:

```
host1.imps.com
```

### 6.5.2 Redirect (Host) Name

When the IMPS server in a domain can be accessed through several SAP's distributed in different physical hosts, this IMPS server MAY provide a list of those hosts for the other IMPS 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 IMPS 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 (9.1.1).

An example of a Redirect (Host) Address is:

```
host2.serviceprovider.com.
```

# 6.6 Features and Functions

## 6.6.1 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 at the server in that domain.

SSP MUST support 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 MUST support the security requirement in the server interoperability through the underlying transport layer whenever although this is strangepossible.

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

## 6.6.2 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.

Servers MUST support the CALLBACK connection establishment.

SSP MUST support 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.

## 6.6.3 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.

## 6.6.4 Session Management

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

SSP MUST support the authorization and access control among the IMPS 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 MUST be used in the session establishment. The access control MUST be supported in the whole session management.

Servers MAY support remote user session management where the user session management is located in the provider server. The IMPS service consists of a local IMPS SAP responsible for connection handling and a remote IMPS SAP responsible for user session management. When the end-user logs in to the IMPS service, the two SAP entities communicate to establish a

session on behalf of the end-user. User Session Management relays the user session requests to the remote domain which is responsible for managing the user session instead of the requestor server domain.

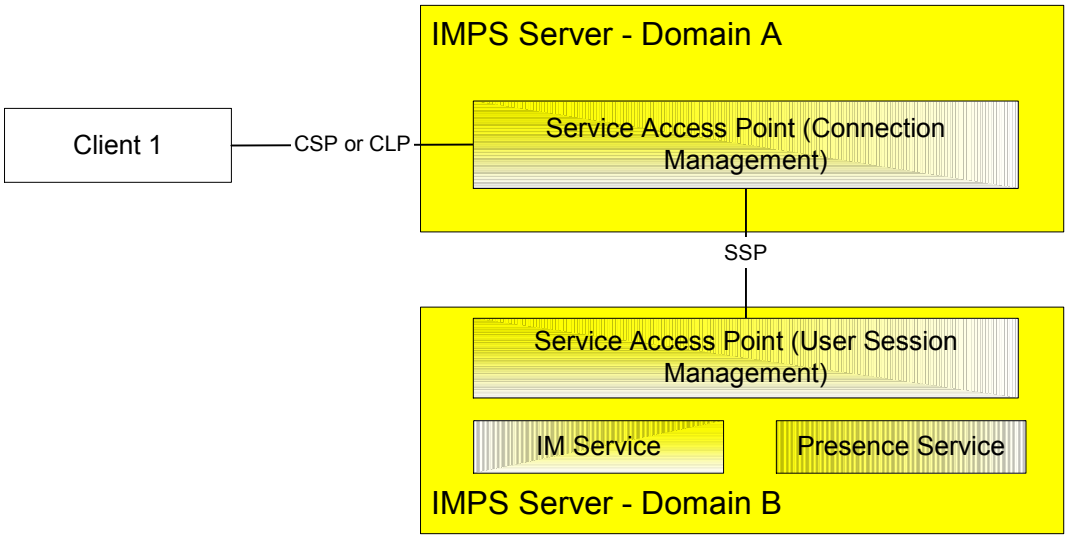


**Figure 10: Remote User Session Management**

## 6.6.5 Service Management

SSP MUST support service discovery among the IMPS domains. The services include Common Services, Presence Service, Instant Messaging (IM) Service, Group Service and Shared Content Service. 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 IMPS 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.

The SAP in the home domain or the IM Service Element of the MAY limit the content types and / or maximum sizes of messages sent through the system. The SAP in the home domain MAY discover the content handling policies of the IM Service Element either offline or online. Information about these policies is needed before clients can connect to the SAP in the home domain and perform client capability negotiation.

Either the SAP or the IM Service Element MAY support content transcoding. The SAP MAY add to or change the information provided by the IM Service Element to reflect the transcoding supported by the SAP.

If there is support for presence delivery on the server side, the SAP in the home domain MUST publish all of the agreed content types along with the related credentials in the ClientInfo/ClientContentLimit/ AcceptedContentType presence attribute after the service negotiation has been completed.

## 6.6.6 User Profile Management

SSP MUST support the exchange of user profile information among the IMPS 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.

### 6.6.7 Service Relay

SSP MAY support the service relay among the IMPS domains including the functional relay of the common IMPS features, presence features, IM features, group features and shared content features. The goal of SSP is to support the distributed interoperable complementary IMPS services across service provider domains.

A server implementation MAY forward messages using SSP for services the server itself does not support. I.e. if the server only supports SAP and PRSE, it MAY forward IMSE and GRSE messages to another server using SSP.

Due to the nature of the server interoperation, 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 have been defined in the following sections including functional relay services.

Servers MUST support semantics and syntax mapping between SSP primitive and CSP primitive of supported transactions. Primitives of supported transactions MUST be supported as a requestor serer and provider server.

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).

### 6.6.8 Segmentation Mechanism

IMPS servers MAY support the IMPS segmenetation mechanism including all related transactions – the related transactions MUST be negotiated as a whole. The service tree leaf that allows negotiation of the segmentation mechanism is ‘SGMNT’.. The SAP in the home domain is responsible for segmentation according to the limits set on the maximum transport message size set by the client in the client capability negotiation. Segmentation is not supported over the SSP protocol. The SAP in the home domain MUST map to and from the corresponding list transactions over SSP.

Please refer to the CSP document [CSP] for further information on the CSP segmentation mechanism.

# 7 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.

## 7.1 Trust Models

A TRUST model is assumed between the IMPS 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.

## 7.2 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 is 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.

## 7.3 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.

## 7.4 Individual Domain Security

The security of an 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.

# 8 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 the transaction level.

A transaction can be one-way, two-way or multiple-way. Servers MUST support all types of transactions.

All mandatory information elements MUST be present in the primitives as defined in this document. All conditional information elements MUST be present or absent according to the relevant SCR.

The Transaction-ID MUST be kept the same within a transaction.

## 8.1 Meta-Information

The SSP service requests MUST contain the meta-information as defined in Table 1. Servers MUST support the Meta-Information structure.

| 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 | Structure | 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 MUST be unique for each session at the Provider Server.

The Transaction-ID MUST be assigned by the transaction originator. The Transaction-ID MUST be unique for each transaction originated from the server that initiates the transaction.

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

Some implementation notes are as follows. Are the shoulds below ok or do they need to be MUST sometimes

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.

## 8.2 Status Primitive

The status primitive in the service response is defined as follows in Table 2. Servers MUST support the Status primitive. The status primitive MAY contain a System Message notification from the provider server.

| 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. |
| Detailed result | O | Structure | Detailed status code and status description |
| System-Message-List | O | Structure | The list of System Messages |

**Table 2. Information elements in Status primitive**

## 8.3 Asynchronous Transaction

The server MUST support asynchronous transactions.

## 8.4 General Error Handling

Servers MUST support the SSP General Error Handling procedure.

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 an error occurs, the processing server MUST handle the exception based on its own policy. In addition, the processing server MUST inform the other server involved in this transaction of such an exception by sending the Status primitive with an appropriate Status Code and optional Status Description. More precisely if the processing server sends Status Code 2XX then it SHALL be sent in the response primitive specified for the transaction. Otherwise Status primitive MUST be used.

## 8.5 Invalid Transaction

A transaction is considered “valid” if the transaction completes within a reasonable period. Servers MUST support the transaction validity time. 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 MUST not receive a response from the provider domain. The service requestor MUST repeat the transaction for reasonable times until the transaction completes or the repeat times expire. If the transaction completes, the session MUST go on for the future transactions. If the repeat times expire, the session MUST be terminated by the requestor for security reason. In addition, the requestor-maintained session, which provides the other side with its own service, MUST be terminated also.

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 security is the major concern.

## 8.6 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 that doesn't correspond 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 MUST return a status code indicating an "Unknown Transaction" error. If the unknown transaction happens frequently, the provider domain MUST terminate the session as well as the session maintained by the requestor for security reasons.

The definition of "Unknown Transaction Frequency" is up to each server implementation. However, the value of "Unknown Transaction Frequency" MUST be configurable at each service domain by the operator. Each operator MUST 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 security is the major concern.

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

# 8.7 General Status Code

All SSP transactions MAY 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)
- Invalid server session (620) – except transactions allowed outside of a session
- Multiple errors (900)
- Not logged in (604)
- Bad parameter (402)
- Forbidden (403)
- Not found (404)

# 9 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.

In order to achieve minimum level of interoperability both the request and provider server MUST support all of the Session Management transactions.

## 9.1 Access Control

### 9.1.1 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 is 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. An empty B-password is valid.
- In Server B, Server A's identity is registered with at least { A-Host-ID, A-Service-ID, A-password } tuple. An empty A-password is valid.
- Both servers has registered and supported a common digest schema such as MD5 (See [RFC1321])or SHA1(See [FIPS 180-1]) .

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 responds 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 responds 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. Servers MAY support redirects. 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 only its original `Host-ID`.

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 is 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. A mesh means that every single host connects to all remote hosts.

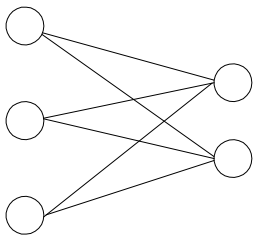

**Figure 11: Mesh of redirect connection pairs**

After establishing a session 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 transaction 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 otherwise right after the session establishment.

There are at least two connections, the connection pair, to carry the session pair. The servers may establish more than one connection pair 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 pair. 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 the

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. When creating redirect connection pairs online service negotiation and service agreements may not be made.

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 possiblity 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 [SSP Trans] shall define how to bind session pairs to reusable connections by the underlying transport.

### 9.1.2 Session Maintenance

Server A and Server B MUST maintain the session and keep the session alive by exchanging the live traffic if needed during the session. The initial interval MUST be 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 as an intermediary (e.g. proxy) MAY break the connection, resulting in terminating the session, if there is no data traffic for a reasonable time period. The session maintenance MAY also be required periodically in the case where 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 (reciprocal) session.

The interval MUST be configurable at each service domain by the operator. The operators MUST 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 provider server MUST NOT terminate the session within the agreed KeepAliveTime. The session maintenance MUST be performed over all of the connections used by the current session, thus covering the connection maintenance.

### 9.1.3 Session Termination

The session MUST 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 security.

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

A session MAY be terminated abnormally. For example, if 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 some other reason, as long as there is at least one connection for each session, the session pair MUST 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 MUST also be terminated. Additionally, if all connection pairs are terminated, the session pair MUST be terminated.

### 9.1.4 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 establishing the session.

# 9.2 Transactions

## 9.2.1 The "Login" Transaction

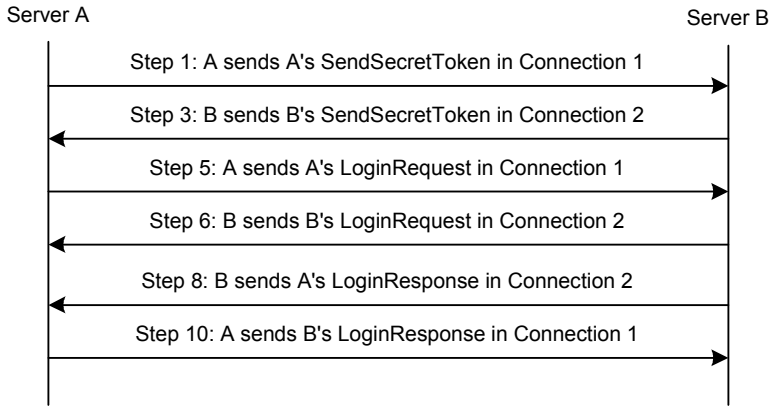


**Figure 12: The "Login" Transaction**

Session establishment and additional redirect connection establishment are achieved through a "**Login"** transaction. Servers MUST support the Login transaction. Servers MUST support primitives of the Login transaction as either an initiator server (Server A) or as a secondary server (Server B).

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 share the same `Transaction-ID` that is generated by Server A in step 1.

Step 3, Step 5 and Step 8 share 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 is 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. Servers MAY support a "Redirect List". 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 MUST NOT try a connection pair establishment unless a new redirect process takes place. Therefore 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) MUST 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 MUST use the new "Redirect List" as the physical connections to support the session. The connection pair(s) MUST be handed over to the actual physical nodes, and the Master Connection Pair (1 and 2) MUST be disconnected. If there are more than one Redirect (Host) Names in either of the "Redirect List", a mesh of redirect connection pairs MUST be initiated to support the session pair.

There MUST be at least two connections, the connection pair, to carry the session pair. The servers may establish more than one connection pair 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 the 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 pair. 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. While creating redirect connection pairs an online service negotiation and service agreement may not be made.

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

**Table 3. Primitive Directions for Login Transaction**

### 9.2.1.1 Primitives

#### 9.2.1.1.1 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 | "1.2" | SSP protocol version. |
| SecretToken | M | String | Secret token originated by the requestor. |

**Table 4. Information elements in SendSecretToken Primitive**

#### 9.2.1.1.2 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 the 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. |

| | | | |
|---|---|---|---|
| Redirect-HostID | O | String | Identifies the requestor host if the connection is a redirected connection pair. |
| 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 5. Information elements in LoginRequest Primitive**

#### 9.2.1.1.3 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. When LoginResponse indicates successful session establishment, the Session-ID MUST be present. When LoginResponse indicates failure in session establishment, the Session-ID MUST not be present. In case of setting up redirect connection pairs, the Session-ID must be the same as in the initial session establishment's LoginResponse if the transaction is successful.

In the response the provider server MAY specify the `time-to-live` of the current session. If TimeToLive is present in LoginResponse, it MUST be considered to be the valid session duration time. If TimeToLive is not present in LoginResponse, the TimeToLive in LoginRequest MUST be considered to be the valid session duration time (which may be infinite if it is not in LoginRequest either). This `time-to-live` may be different from that in the `LoginRequest` from the requestor server. If TimeToLive is requested in the LoginRequest during session / additional connection establishment, the provider server's LoginResponse MUST include KeepAliveTime.

| 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 8.2. |
| 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 6. Information elements in LoginResponse Primitive**

## 9.2.2 The "Logout" Transaction

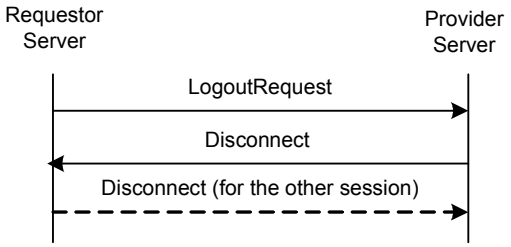


**Figure 13: The "Logout" Transaction**

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.

Servers MUST support the "Logout" transaction. The requestor server can logout from the provider server and close the session through a "Logout" transaction. In addition the requestor also MUST terminate the other session through a "Disconnect" transaction that is illustrated in the dashed 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 7. Primitive Directions for Logout Transaction**

#### 9.2.2.1 Primitives

##### 9.2.2.1.1 The "LogoutRequest" Primitive

The `LogoutRequest` primitive allows 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 8. Information elements in LogoutRequest**

### 9.2.3 The "Disconnect" Transaction

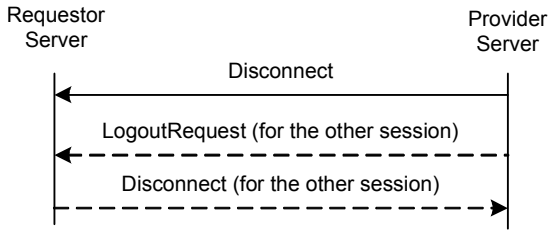


**Figure 14: "Disconnect" Transaction**

Servers MUST support the "Disconnect" transaction. The provider server may close the session through a "**Disconnect**" transaction. Under such conditions the provider also MUST terminate the other session through a "Logout" transaction that is illustrated in the dash lines.

If time out expires, the provider server MUST initiate Disconnect transaction to close the session or connection.

| Primitive | Direction |
|---|---|
| Disconnect | Requestor Server ← Provider Server |

**Table 9. Primitive Directions for Disconnect Transaction**

#### 9.2.3.1 Primitives

##### 9.2.3.1.1 The "Disconnect" Primitive

The `Disconnect` primitive allows 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 8.2). Present if the requestor server Logout. |

**Table 10. Information Elements in Disconnect Primitive**

## 9.2.4 The "KeepAlive" Transaction

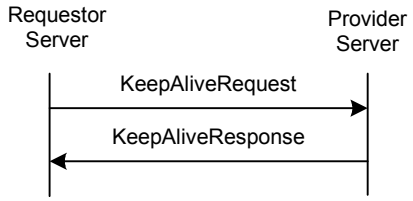


**Figure 15: The "KeepAlive" Transaction**

Session maintenance is achieved through the "**KeepAlive**" transaction. A "KeepAlive" transaction MUST 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.

Servers MUST support the "KeepAlive" transaction. 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 an intermediary (e.g. proxy) breaks the connection, resulting in terminating the session, if there is no data traffic for a reasonable time period.

The "KeepAlive" transaction MAY be required periodically in case 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, complementary, session.

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

**Table 11. Primitive Directions for KeepAlive Transaction**

### 9.2.4.1 Primitives

#### 9.2.4.1.1 The "KeepAliveRequest" Primitive

The "`KeepAliveRequest`" primitive allows the requestor server to maintain the session and update the time-to-live interval with the provider server. The session maintenance MUST 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 values for different connections. If TimeToLive is not present in KeepAliveRequest, or if it is zero, it MUST be considered to ask for an infinite session.

| 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 12. Information Elements in KeepAliveRequest Primitive**

#### 9.2.4.1.2 The "KeepAliveResponse" Primitive

The `KeepAliveResponse` primitive allows the provider server to maintain the session and update the time-to-live interval with the requestor server. The session maintenance MUST 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 values for different connection. If TimeToLive is present in KeepAliveResponse, it MUST be considered to be the valid session duration time. If TimeToLive is not present in KeepAliveResponse, the TimeToLive in KeepAliveRequest is considered to be the valid session duration time (which may be infinite if it is not in KeepAliveRequest either)

| Information Element | Req | Type | Description |
|---|---|---|---|
| Message-Type | M | KeepAliveResponse | Message identifier |
| Status-Info | M | Structure of Status-Primitive | The status information (see 8.2). |
| Time-to-live | O | Integer in Seconds | Indicates the time-to-live of the session over this connection. |

**Table 13. Information Elements in KeepAliveResponse Primitive**

# 9.3 Status Code

## 9.3.1 "Login" Transaction

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

### 9.3.2 “Logout” / “Disconnect” Transaction

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

# 10 Service Management

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

If the online service negotiation and service agreement is needed, it MUST be the first transactions in the session pair after the session is established through Login transaction. The agreed services MUST be provided by the Provider server. Servers MAY support Service re-negotiation during a session.

## 10.1 Service Structure

The IMPS 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 IMPS server MAY support all or a subset of the features. However, if a IMPS server supports a feature, some functions and transactions must be supported to ensure minimal interoperability [SSP SCR]. 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 16. 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 below. The transactions that are not described are considered mandatory functions that MUST be supported in the servers.

## 10.2 General

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

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

## 10.3 SAP Feature

Servers MUST support the SAP feature. The SAP feature consists of the following nodes in the SSP service tree.

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

- **User Profile Management** node includes the following transactions
  - GetUserProfile

- UpdateUserProfile
- Service Relay node indicates if the SAP supports service relay including routing

- **Remote User Session Management** node includes the following transactions
  - UserLogin
  - UserLogout
  - UserDisconnect
  - UserKeepAlive
  - UserGetSPInfo
  - UserServiceNegotation

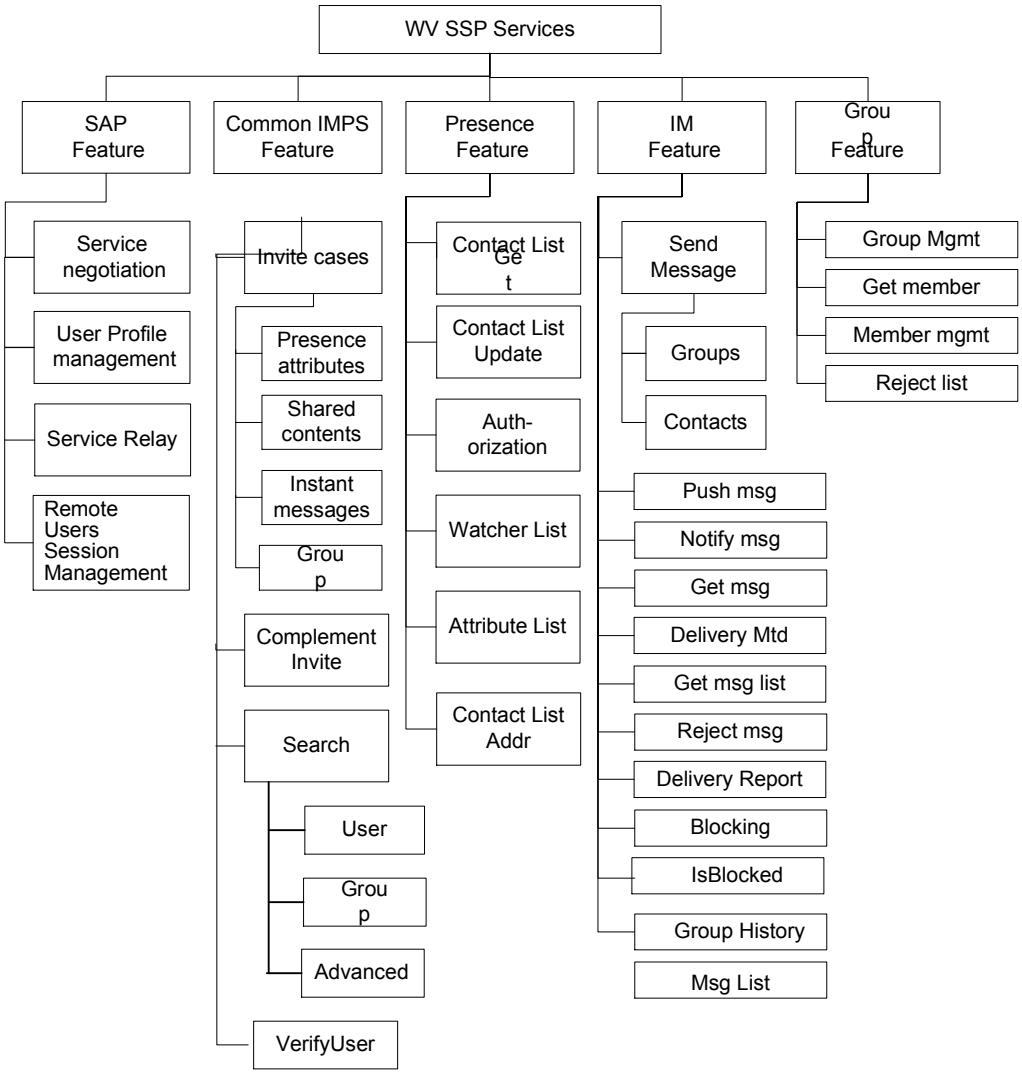


**Figure 16: SSP Service tree**

# 10.4 Common IMPS feature

Servers MAY support the common IMPS feature. The Common IMPS feature consists of the following nodes in the SSP Service tree:

- **Invite** node includes the Invitation/Cancel-Invitation transactions
  - All supported invite types must be included in the Service List (Presence, IM, Shared Content, Group, End-to-End Application)
- 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
  - Advanced: Support advanced search
- GetMap node includes the following transactions:
  - GetMap
- **VerifyUser** node includes the following transactions:
  - VerifyWVID
- SystemMessage node includes the following transactions:
  - SystemMessageRequest
  - SystemMessageUser

# 10.5 Presence Feature

Server MAY support the Presence feature. The Presence feature consists of the following nodes in the SSP Service tree:

- ***Contact List Get*** node includes the following transactions:
  - GetContactList
  - GetListMember
  - GetListProperties
- ***Contact List Update*** node includes the following transactions:
  - CreateContactList
  - DeleteContactList
  - AddListMember
  - RemoveListMember
  - SetListProperties
- ***Watcher List*** node includes the following transaction

  - GetWatcherList
- ***Contact List Addr*** node indicates if the contacts list is valid for addressing users in the following transactions
  - Subscribe
  - UnSubscribe
  - GetPresence
  - UpdatePresence
  - Suspend Presence

## 10.6 IM Feature

Servers MAY support the IM feature. The IM feature consists of the following nodes in the SSP Service tree:

- ***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 includes the following transaction
  - PushMessage
- ***Notify Msg*** node includes the following transaction
  - MessageNotification
- ***Get Msg*** node includes the following transaction
  - GetMessage
- ***Delivery Mtd*** node includes the following transaction
  - SetMessageDeliveryMethod
- ***Get Msg List*** node includes the following transaction
  - GetMessageList without group functionality
- ***Reject Msg*** node includes the following transaction
  - RejectMessage
- ***Delivery Report*** node includes the following transaction
  - NotifyDeliveryStatusReport
- ***Blocking*** node includes 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)

# 10.7 Group Feature

Servers MAY support the Group Feature. The Group Feature consists of the following nodes in the SSP service tree:

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

# 10.8 Transactions

## 10.8.1 The "GetAvailableService" Transaction

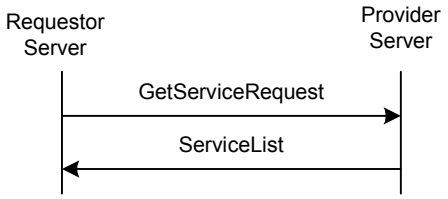


**Figure 17: The "GetAvailableService" Transaction**

SSP MAY support service discovery among the IMPS domains. The services include Common Features, Presence Service, Instant Messaging (IM) Service, Group Service and Shared Content Service.

Servers MAY support the "GetAvailableService" transaction. The service tree node that allows negotiation of this transaction is 'Service Negotiation'. 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 inquiring about 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 14. Primitive Directions for GetAvailableService Transaction**

### 10.8.1.1 Primitives

#### 10.8.1.1.1 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 8.1. |

**Table 15. Information elements in GetServiceRequest Primitive**

#### 10.8.1.1.2 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 8.1. Present if the provider initiates ServiceIndication. |
| Status-Info | C | Structure of Status-Primitive | The status information (see 8.2). Present if the requestor initiates GetServiceRequest. |
| Service-List | M | Structure | List of available services in a tree structure. |

**Table 16. Information elements in ServiceList Primitive**

## 10.8.2 The "ServiceIndication" Transaction


**Figure 18: The "ServiceIndication" Transaction**

Servers MAY support the "ServiceIndication" transaction. The service tree node that allows negotiation of this transaction is 'Service Negotiation'. The provider server also informs the requestor server of any change in the available services through a "**ServiceIndication**" Transaction. It depends on the offline service agreement between two domains to decide what the subsequent actions to be taken are.

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 17. Primitive Directions for ServiceIndication Transaction**

### 10.8.2.1 Primitives

See Section 10.8.1.1.2 for a description of the ServiceList primitive.

## 10.8.3 The "SetServiceAgreement" Transaction

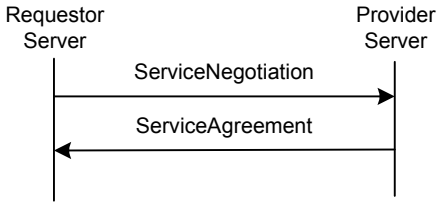


**Figure 18. The "SetServiceAgreement" Transaction**

The service agreement between the requestor and provider servers is established through a "**SetServiceAgreement**" Transaction. Servers MAY support the "SetServiceAgreement" transaction. The service tree node that allows negotiation of this transaction is 'Service Negotiation'.

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

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

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

**Table 18. Primitive Directions for SetServiceAgreement Transaction**

### 10.8.3.1 Primitives

#### 10.8.3.1.1 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 8.1. |
| 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 19. Information elements in ServiceNegotiation Primitive**

#### 10.8.3.1.2 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 Element | Req | Type | Description |
|---|---|---|---|
| Message-Type | M | ServiceAgreement | Message identifier |
| Status-Info | M | Structure of Status-Primitive | The status information (see 8.2). |

| | | | |
|---|---|---|---|
| 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 20. Information elements in ServiceAgreement Primitive**

## 10.8.4 The "GetAcceptedContentTypes" Transaction

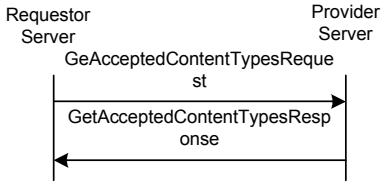


**Figure 19: The "GetAcceptedContentTypes" Transaction**

SSP MAY support retrieving information about accepted content types among the IMPS domains. This information is needed by the SAP in the home domain before client capability negotiation is performed, see section 6.9.1 in [CSP], and to be able to filter IMs based on accepted content types in the case of separated IM and Presence Service Elements, see section 7 in [PA].

Servers MAY support the "GetAcceptedContentTypes" transaction. The service tree node that allows negotiation of this transaction is 'Service Negotiation'. The requestor server discovers the acceptable content types and handling policies provided by the provider server through a "**GetAcceptedContentTypes**" Transaction.

The requestor server sends a `GetAcceptedContentTypesRequest primitive` to the provider server inquring about the accepted content types. After the provider server finishes processing the request, it sends a GetAcceptedContentTypesResponse primitive to the requestor server with the available content type handling policies.

| Primitive | Direction |
|---|---|
| GetAcceptedContentTypesRequest | Requestor Server → Provider Server |
| GetAcceptedContentTypesResponse | Requestor Server ← Provider Server |

**Table 21. Primitive Directions for GetAcceptedContentTypes Transaction**

### 10.8.4.1 Primitives

#### 10.8.4.1.1 The "GetAcceptedContentTypesRequest" Primitive

The `GetAcceptedContentTypesRequest` primitive is issued from the requestor server to discover the content handling policies at the provider server.

| Information Element | Req | Type | Description |
|---|---|---|---|
| Message-Type | M | GetAcceptedContentTypes Request | Message identifier |
| Meta-Information | M | Structure of Meta-information | The necessary meta-information in a service request defined in 8.1. |

**Table 22. Information elements in GetAcceptedContentTypesRequest Primitive**

#### 10.8.4.1.2 The "GetAcceptedContentTypesResponse" Primitive

The `GetAcceptedContentTypesResponse` primitive is issued from the provider server to indicate its content handling policies.

| Information Element | Req | Type | Description |
|---|---|---|---|
| Message-Type | M | GetAcceptedContentTypes Response | Message identifier |
| Meta-Information | M | Structure of Meta-information | The necessary meta-information in a service request defined in 8.1. |
| Accepted-Content-Types-List | O | Structure | Information about content types supported by the provider server and policies for those content types. |
| Any-Content | M | Boolean | Provider server supports all content types |

**Table 23. Information elements in GetAcceptedContentTypesResponse Primitive**

## 10.9 Status Code

- `Version Not Supported (505)`

# 11 Interoperability Management – User Profile Management

These transactions are needed for the complementary services.

## 11.1 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. Terminal capabilities information is associated with a specific client-id and each key may be repeated for each client in use by the end-user.

The general user information is defined as follows:

| General UP Attribute | Value | Description |
|---|---|---|
| MSISDN | String | International mobile number of the user |
| 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 6.3.1 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 6.3.1 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 6.3.1 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 6.3.1 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 6.3.1 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.encoding | encoding {, encoding } | Supported transfer encoding in client. See [RFC2045] for standard "transfer-encoding". |
| Terminal.TextContent.length | Integer in Byte | Supported message size in client for text messages |
| Terminal.Content.protocol | Protocol {, Protocol } | Supported out-band protocol in client for binary message retrieval. |
| Terminal.RichContent | ContentType, AcceptedRichContent Length, ContentPolicy, ContentPolicyLimit | Comma-separated list containing the media types and the related credentials of those content types that the client supports. See [CSP]. |
| Terminal.PullLength | Integer in character count | An integer number in character count that indicates the maximum length of content (either text/plain or multimedia object) that the client accepts for client-originated content delivery transactions. |

| General UP Attribute | Value | Description |
|---|---|---|
| Terminal.PushLength | Integer in character count | An integer number in character count that indicates the maximum length of content (either text/plain or multimedia object) that the client accepts for server-originated content delivery transactions |
| Terminal.Content.any | "YES" \| "NO" | A Boolean value indicating that the client accepts any rich content types. |
| Terminal.ClientType | String | The type of the client. See [PA]. |
| Terminal.DefaultLanguage | String | The language code is specifying that the client prefers to receive text information in the indicated language from the server. |
| Terminal.MultiTrans | Integer | The maximum number of open transactions from both client and server side at any given time. |
| Terminal.MultiTransPerMessage | Integer | Integer value indicating the maximum number of primitives that the client can handle within the same transport message at any given time. |
| Terminal. OfflineETEMHandling | "PRIORITYREJECT" \| "PRIORITYSTORE" \| "REJECT" \| "SENDREJECT" \| "SENDSTORE" | Enumerated value indicating how the client expects the server to handle end-to-end messages that have been addressed particularly to this client after the client logged out or was disconnected. See [CSP] |
| Terminal.OnlineETEMHandling | "DETECT" \| "SERVERLOGIC" \| "FORKALL" | Enumerated value indicating how the user expects the server to handle end-to-end messages that have not been addressed to any specific client/application of the user while the user is online. |
| Terminal.ParserSize | Integer in BYTE | The maximum character (byte) count of XML (WBXML, SMS - depending on the actual encoding) primitive size that the parser can handle. |
| Terminal.Session.priority | Float | A floating point number from 0 (lowest) to 1 (highest) indicating the priority of the session for end-to-end messages. |
| x.key | String | A service provider may define new key-values. These service provider specific keys are prefixed with x[.]. |

**Table 24. General User Profile**

Each piece of user profile information is organized in a “(name, value)” pair. Terminal capability information also contains a client-id for the client that originated that specific “(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.MaxNumberOfContactLists, 100)

# 11.2 Transactions

## 11.2.1 The "GetUserProfile" Transaction

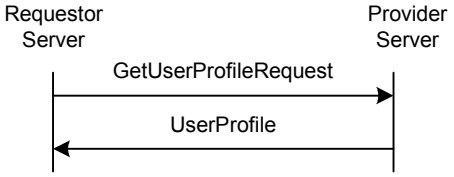


**Figure 20: The "GetUserProfile" Transaction**

SSP supports the exchange of user profile information among the IMPS 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. . Servers MAY support the "GetUserProfile" transaction. The service tree node that allows negotiation of this transaction is 'User Profile Management'.

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**

### 11.2.1.1 Primitives

#### 11.2.1.1.1 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 8.1. |
| User-ID-List | M | Structure | Identifies the users whose User Profiles are requested. If it is empty, all users' User Profiles are requested. |

**Table 26. Information elements in GetUserProfileRequest Primitive**

#### 11.2.1.1.2 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 8.2). |
| 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 27. Information elements in UserProfile Primitive**

## 11.2.2 The "UpdateUserProfile" Transaction

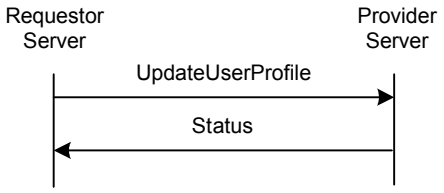


**Figure 21: The "UpdateUserProfile" Transaction**

Servers MAY support the "UpdateUserProfile" transaction. The service tree node that allows negotiation of this transaction is 'User Profile Management'. 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 28. Primitive Directions for UpdateUserProfile Transaction**

### 11.2.2.1 Primitives

#### 11.2.2.1.1 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 8.1). |
| 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 29. Information elements in UpdateUserProfileRequest Primitive**

# 11.3 Status Code

- `Unknown user (531)`

# 12 Remote Users Session Management

## 12.1 Transactions

### 12.1.1 The "UserLogin" Transaction

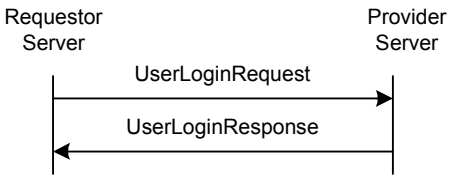


**Figure 22: The "UserLogin" Transaction**

The "UserLogin" transaction is used for the user in the requestor server to log into the provider server and creates a user session in the provider server. The transaction happens if the provider server holds the user's credential, and the user logs in through the requestor server. In this case, the user's home domain does not have the user's credential, thus cannot create the user session independently. So the home domain has to relay the user's login request to the provider server.

Servers that support Remote User Session Management MUST support the UserLogin transaction.

In the home domain, the requestor server MUST convert the CSP LoginRequest to SSP UserLoginRequest, and send to the provider server. However, the IE "Session-Cookie" is kept in the home domain so that the home domain may initiate the CIR to the client if needed. After the provider server completes the transaction and the user successfully logs in, the home domain shall create its own session with the client.

In the home domain, the requestor server MUST remove System Message responses from the System-Message-Response-List of the CSP LoginRequest that correspond to System Message notifications generated by the requestor server, before forwarding any additional System Messages responses to the provider server.

If the Login is optimized by including Service and Client Capability negotiation in the LoginRequest, the Requestor server MUST perform these transactions separately towards the Provider server.

The intermediate domains shall merely relay the transaction flow between the home domain and the provider server.

Please refer to CSP Sessions and Transactions Document [CSP][CSP] for the user login transaction details.

| Primitive | Direction |
|---|---|
| UserLoginRequest | Requestor Server → Provider Server |
| UserLoginResponse | Requestor Server ← Provider Server |

**Table 30. Primitive Directions for UserLogin Transaction**

#### 12.1.1.1 Primitives

##### 12.1.1.1.1 The "UserLoginRequest" Primitive

The "*UserLoginRequest*" primitive is used for a user to login into provider server domain. Generally, "Password-String", "Digest-Bytes", "Supported-Digest-Schema" and "Time-To-Live" are copied from user's CSP LoginRequest.

| Information Element | Req | Type | Description |
|---|---|---|---|
| Message-Type | M | UserLoginRequest | Message identifier |
| Meta-Information | M | Structure of Meta-Information | The meta-information (see 8.1). |
| Application-ID | O | String | Identifies the client application to be associated with the established session. |
| Password-String | C | String | The password string of User-ID (in Meta-information). |
| Digest-Bytes | C | String | The password digest generated based on a digest schema. The digest bytes are BASE64 encoded. |
| Supported-Digest-Schema | C | String | A list of Supported digest schema (SHA, MD4, MD5 etc). |
| Time-To-Live | O | Integer | Interval for a valid session before expired. If omitted, the requestor server requests an infinite session. |
| System-Message-Response-List | O | Structure | The list of System Message response(s) from the end-user. |

**Table 31. Information elements in UserLoginRequest Primitive**

#### 12.1.1.1.2 The "UserLoginResponse" Primitive

The "*UserLoginResponse*" primitive is issued from the provider server to accept user's LoginRequest, or requires further authentication.

| Information Element | Req | Type | Description |
|---|---|---|---|
| Message-Type | M | UserLoginRespons e | Message identifier |
| Status-Info | M | Structure of Status-Primitive | The necessary status information in a service response defined in 8.2. |
| User-ID | C | String | Identifies the requesting user. |
| Nonce | C | String | Random string generated by server for password digest. The string is not BASE64 encoded. |
| Digest-Schema | C | String | Type of digest schema to use. |
| Keep-Alive-Time | C | Integer | Interval for a valid session before expired. This time may be any value other than zero. |
| Client-Capability-Request | C | Boolean | Informs the Client that it needs to perform a Client Capability Request transaction. MUST be present only if login was successful. |

**Table 32. Information elements in UserLoginResponse Primitive**

## 12.1.2 The "UserLogout" Transaction

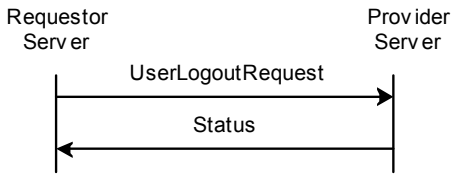


**Figure 23: The "UserLogout" Transaction**

Please refer to CSP Sessions and Transactions Document [CSP] for the user logout transaction details.

Servers that support Remote User Session Management MUST support the UserLogout transaction.

| Primitive | Direction |
|---|---|
| UserLogoutRequest | Requestor Server → Provider Server |
| Status | Requestor Server ← Provider Server |

**Table 33. Primitive Directions for UserLogout Transaction**

### 12.1.2.1 Primitives

#### 12.1.2.1.1 The "UserLogoutRequest" Primitive

The "*UserLogoutRequest*" primitive is used for a user to log out from the provider server domain

| Information Element | Req | Type | Description |
|---|---|---|---|
| Message-Type | M | UserLogoutRequest | Message identifier |
| Meta-Information | M | Structure of Meta-Information | The meta-information (see 8.1). |

**Table 34. Information elements in UserLogoutRequest Primitive**

## 12.1.3 The "UserDisconnect" Transaction

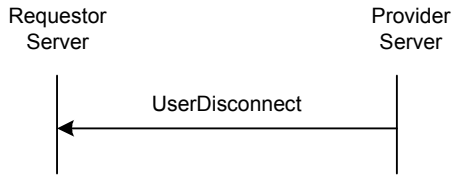


**Figure 24: The "UserDisconnect" Transaction**

Please refer to CSP Sessions and Transactions Document [CSP] for the user server disconnect transaction details.

Servers that support Remote User Session Management MUST support the UserDisconnect transaction.

| Primitive | Direction |
|---|---|
| UserDisconnect | Requestor Server ← Provider Server |

**Table 35. Primitive Directions for UserDisconnect Transaction**

#### 12.1.3.1 Primitives

##### 12.1.3.1.1 The "UserDisconnect" Primitive

The "*UserDisconnect*" primitive is used for the provider server to disconnect the user session.

| Information Element | Req | Type | Description |
|---|---|---|---|
| Message-Type | M | UserDisconnect | Message identifier |
| Meta-Information | M | Structure of Meta-Information | The meta-information (see 8.1). |
| Status-Info | M | Structure of Status-Primitive | The Status information (see 8.2). |

**Table 36. Information elements in UserDisconnect Primitive**

### 12.1.4 The "UserKeepAlive" Transaction

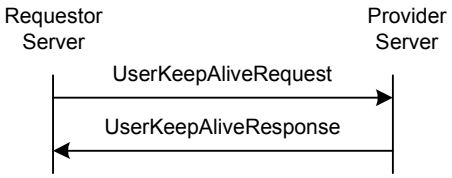


**Figure 25: The "UserKeepAlive" Transaction**

Please refer to CSP Sessions and Transactions Document [CSP] for the user keepalive transaction details.

Servers that support Remote User Session Management MUST support the UserKeepAlive transaction.

| Primitive | Direction |
|---|---|
| UserKeepAliveRequest | Requestor Server → Provider Server |
| UserKeepAliveResponse | Requestor Server ← Provider Server |

**Table 37. Primitive Directions for UserKeepAlive Transaction**

#### 12.1.4.1 Primitives

##### 12.1.4.1.1 The "UserKeepAliveRequest" Primitive

The "*UserKeepAliveRequest*" primitive is used for the requestor server to maintain a user session and update the time-to-live interval with the provider server.

| Information Element | Req | Type | Description |
|---|---|---|---|
| Message-Type | M | UserKeepAliveRequest | Message identifier |
| Meta-Information | M | Structure of Meta-Information | The meta-information (see 8.1). |
| Time-to-live | O | Integer | Indicates the time-to-live of the user session. |

**Table 38. Information Elements in UserKeepAliveRequest Primitive**

#### 12.1.4.1.2 The "UserKeepAliveResponse" Primitive

The "*UserKeepAliveResponse*" primitive is issued from the provider server to accept UserKeepAliveRequest.

| Information Element | Req | Type | Description |
|---|---|---|---|
| Message-Type | M | UserKeepAliveResponse | Message identifier |
| Status-Info | M | Structure of Status-Primitive | The necessary status information in a service response defined in 8.2. |
| Keep-Alive-Time | O | Integer | Interval for a valid session before expired. This time may be any value other than zero. |

**Table 39. Information elements in UserKeepAliveResponse Primitive**

## 12.1.5 The "UserGetSPInfo" Transaction

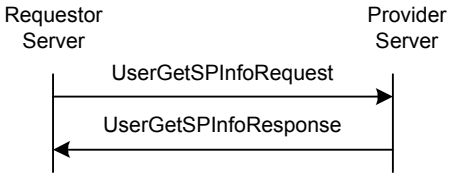


**Figure 26: The "UserGetSPInfo" Transaction**

Please refer to CSP Sessions and Transactions Document [CSP] for the user get service provider information transaction details.

Servers that support Remote User Session Management MAY support the UserGetSPInfo transaction.

| Primitive | Direction |
|---|---|
| UserGetSPInfoRequest | Requestor Server → Provider Server |
| UserGetSPInfoResponse | Requestor Server ← Provider Server |

**Table 40. Primitive Directions for UserGetSPInfo Transaction**

### 12.1.5.1 Primitives

#### 12.1.5.1.1 The "UserGetSPInfoRequest" Primitive

The "*UserGetSPInfoRequest*" primitive is used for the user in the requestor server to get the useful branding information.

| Information Element | Req | Type | Description |
|---|---|---|---|
| Message-Type | M | UserGetSPInfoRequest | Message identifier |
| Meta-Information | M | Structure of Meta-Information | The meta-information (see 8.1). |

**Table 41. Information Elements in UserGetSPInfoRequest Primitive**

#### 12.1.5.1.2 The "UserGetSPInfoResponse" Primitive

The "*UserGetSPInfoResponse*" primitive is used for the provider server to return the branding information.

| Information Element | Req | Type | Description |
|---|---|---|---|
| Message-Type | M | UserGetSPInfoResponse | Message identifier |
| Status-Info | M | Structure of Status-Primitive | The Status information (see 8.2). |
| Name | M | String | Name of the service provider. |
| Logo | O | Structure | Service-provider specific image. (e.g. logo) |
| Text | O | String | Descriptive text. |
| URL | O | String | Link to a web page. |

**Table 42. Information elements in UserGetSPInfoResponse Primitive**

## 12.1.6 The "UserServiceNegotiation" Transaction

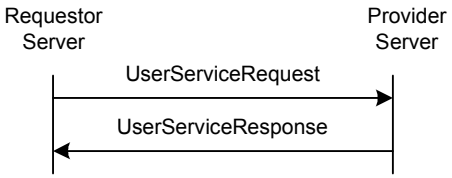


**Figure 27: The "UserServiceNegotiation" Transaction**

Please refer to CSP Sessions and Transactions Document [CSP] for the user service negotiation transaction details.

Servers that support Remote User Session Management MUST support the UserServiceNegotiation transaction.

| Primitive | Direction |
|---|---|
| UserServiceRequest | Requestor Server → Provider Server |
| UserServiceResponse | Requestor Server ← Provider Server |

**Table 43. Primitive Directions for UserServiceNegotiation Transaction**

### 12.1.6.1 Primitives

#### 12.1.6.1.1 The "UserServiceRequest" Primitive

The "*UserServiceRequest*" primitive is used for the user in the requestor server to get the service information supported in the provider server. The user MAY also ask for the whole services that are supported in the server.

| Information Element | Req | Type | Description |
|---|---|---|---|
| Message-Type | M | UserServiceRequest | Message identifier |
| Meta-Information | M | Structure of Meta-Information | The meta-information (see 8.1). |
| Requested-Functions | M | Structure | Identifies the service elements and functions the client requests. |
| All-Functions-Request | M | Boolean | Request the server to send all services that it supports in the reply. |

**Table 44. Information Elements in UserServiceRequest Primitive**

#### 12.1.6.1.2 The "UserServiceResponse" Primitive

The "*UserServiceResponse*" primitive returns services that are NOT supported by the server but requested by the client, i.e., the delta between requested and supported functions. If all requested functions are supported by the server the response

MUST contain an empty service set. If the user has requested all services, it MUST also return all services that the server supports.

| Information Element | Req | Type | Description |
|---|---|---|---|
| Message-Type | M | UserServiceResponse | Message identifier |
| Status-Info | M | Structure of Status-Primitive | The Status information (8.2). |
| Not-Available-Functions | C | Structure | Identifies the delta of the client requested and what is available for that user. |
| All-Functions | C | Structure | Identifies all of the functions that the server supports. |

**Table 45. Information elements in UserServiceResponse Primitive**

# 12.2 Status Code

## 12.2.1 "UserLogin" Transaction

- Further authorization needed to use the server (401)
- Invalid password (409)
- The particular user is not allowed to use the server (403)
- The server could not recover the session (502)
- *Application* is forbidden by operator (515)
- Unknown user (531)
- Session-ID, User-ID and Client-ID not matching (422)
- No matching digest scheme supported (543)
- Client-ID is not unique (638)
- User session limitation reached (609)
- MSISDN error (902)
- Registration confirmation (903)

## 12.2.2 "UserLogout" Transaction

- Not logged in (634)

## 12.2.3 "UserDisconnect" Transaction

- User Session - Forced Logout (631)
- User Session Expired (630)

## 12.2.4 "UserKeepAlive" Transaction

- Not logged in (634)
- New timeout value not accepted – old value is in use (635)

## 12.2.5 "UserGetSPInfo" Transaction

- Not logged in (634)

- Client-ID not matching this user (422)

### 12.2.6 “UserServiceNegotiation” Transaction

- Not logged in (634)
- Client-ID not matching this user (422)

# 13 Service Relay – Common IMPS Features

SSP MAY support the service relay among the IMPS 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.

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

- The “GetPublicProfile” Transaction
- The “UpdatePublicProfile” Transaction
- The “GeneralNotification” Transactions
- The “GeneralSearch” Transaction
- The “StopSearch” Transaction
- The “Invitation” Transaction
- The “CancelInvitation” Transaction
- The “SystemMessageRequest” Transaction
- The “SystemMessageUser” Transaction

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

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 are defined in the following two sections.

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

## 13.1 Transactions

### 13.1.1 The “GeneralNotification” Transactions

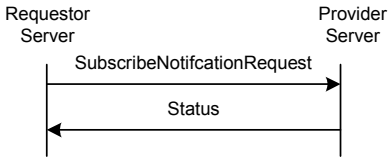


**Figure 28: The “SubscribeNotificationRequest” Transaction**

The client MAY be interested in receiving notifications about specific events on the server. To receive these notifications, the Client MUST send a SubscribeNotificationRequest to the server including the list of notification types he would like to receive. If the list of types is not present, then all types are requested. The requestor server MUST route the subscription request to the provider server. This may result in multiple subscription requests from the SAP in the home domain to the provider servers if the different service elements corresponding to the notification types are located on multiple provider servers. The provider server receiving a subscription request MUST respond with a Status.

After successful subscription, the provider server(s) MUST send the requested notifications.

The subscription(s) MUST NOT be persistent through different user sessions. The SAP handling the user session MUST remove all notification subscriptions for the particular client that was logged out or was disconnected.

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

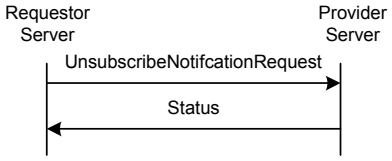


**Figure 29: The "UnsubscribeNotificationRequest" Transaction**

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

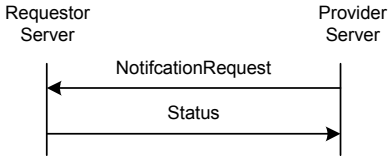


**Figure 30: The "NotifcationRequest" Transaction**

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

The different notification types are defined in the CSP Sessions and Transactions Document [CSP].

The "GeneralNotification" transactions MUST be supported by the IMPS server.

### 13.1.1.1 Primitives

#### 13.1.1.1.1 The "SubscribeNotificationRequest" Primitive

The requestor server sends the SubscribeNotificationRequest primitive to the provider server to subscribe to a particular notification type.

| Information Element | Req | Type | Description |
|---|---|---|---|
| Message-Type | M | SubscribeNotification Request | Message identifier |
| Meta-Information | M | Structure of Meta-Information | The meta-information (see 8.1). |
| Notification-Type-List | O | Structure | A list of notification types. An empty or missing list indicates all available types are desired. |

**Table 46. Information elements in SubscribeNotificationRequest Primitive**

#### 13.1.1.1.2 The "UnsubscribeNotificationRequest" Primitive

The requestor server sends the UnsubscribeNotificationRequest primitive to the provider server to unsubscribe to a particular notification type.

| Information Element | Req | Type | Description |
|---|---|---|---|
| Message-Type | M | UnsubscribeNotificationRequest | Message identifier |
| Meta-Information | M | Structure of Meta-Information | The meta-information (see 8.1). |
| Notification-Type-List | O | Structure | A list of notification types. An empty or missing list indicates all available types are desired. |

**Table 47. Information elements in UnsubscribeNotificationRequest Primitive**

#### 13.1.1.1.3 The "NotificationRequest" Primitive

The provider server sends the NotificationRequest primitive to the SAP in the home domain to send notifications to the client that has subscribed to notifications of that specific type.

| Information Element | Req | Type | Description |
|---|---|---|---|
| Message-Type | M | NotificationRequest | Message identifier |
| Meta-Information | M | Structure of Meta-Information | The meta-information (see 8.1). |
| Notification-Type | M | Enumerated String | Identifies the type of notification. |
| Blocked-List-Inuse | C | Boolean | Indicates if the list of blocked entities is currently in use (active). |
| Group-ID | C | String | Identifies the group:<br>of which the user is either a member of or joined to, have been deleted by the server, or<br>of the user that has been created/deleted for/from the user, or<br>of the user for which the memberships have been updated. |
| ContactList-ID-List | C | Structure | Identifies the contact list(s):<br>which have been created, or<br>which have been changed, or<br>which have been deleted, or<br>for which the authorizations have been updated, or.<br>for which additional reactive authorization is needed |
| Default-List | C | Boolean | Indicates<br>the attributes included in Presence-Attribute-List have been applied to the Default-List, or<br>additional reactive authorization is needed for the Default-List. |
| Granted-List-Inuse | C | Boolean | Indicates if the list of blocked entities is currently in use (active). |
| Invite-ID | C | String | Identifies the invitation that has been:<br>accepted, or<br>rejected, or<br>cancelled. |
| OnlineETEMHandling | C | String | The newly set OnlineETEMHandling setting for the user. |
| Presence-Attribute-List | C | Structure | The presence attribute list that have been assigned to the indicated user(s), contact list(s) and/or the Default-List. |
| Session-Priority | C | Integer | Indicates the newly assigned session priority value. |

| Information Element | Req | Type | Description |
|---|---|---|---|
| User-ID-List | C | Structure | Identifies the user(s):<br>who added the user to their contact list(s), or<br>to whom the presence authorizations have been updated, or<br>to whom additional reactive authorization is needed. |

**Table 48. Information elements in NotificationRequest Primitive**

## 13.1.2 The "GetPublicProfile" Transaction

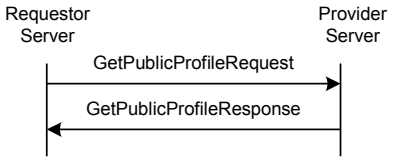


**Figure 31: The "GetPublicProfile" Transaction**

The IMPS system MUST support the "GetPublicProfile" transaction. A user MAY retrieve the public profile for a list of users through the "GetPublicProfile" transaction. The requestor server sends a GetPublicProfileRequest to the provider server. The provider server returns the public profiles of the users using the GetPublicProfileResponse.

Please refer to CSP Sessions and Transactions Document [CSP] for details on the Public Profile and the GetPublicProfile transaction.

| Primitive | Direction |
|---|---|
| GetPublicProfileRequest | Requestor Server → Provider Server |
| GetPublicProfileResponse | Requestor Server ← Provider Server |

**Table 49. Primitive Directions for GetPublicProfile Transaction**

### 13.1.2.1.1 The "GetPublicProfileRequest" Primitive

The GetPublicProfileRequest primitive is used to retrieve the public profile of a list of user.

| Information Element | Req | Type | Description |
|---|---|---|---|
| Message-Type | M | GetPublicProfileRequest | Message identifier |
| Meta-Information | M | Structure of Meta-Information | The meta-information (see 8.1). |
| User-ID-List | M | Structure | Identifies the user(s) whose public profile is requested. |

**Table 50. Information elements in GetPublicProfileRequest Primitive**

### 13.1.2.1.2 The "GetPublicProfileResponse" Primitive

The GetPublicProfileResponse primitive is used to return the public profile of a list of users.

| Information Element | Req | Type | Description |
|---|---|---|---|
| Message-Type | M | GetPublicProfileResponse | Message identifier |
| Meta-Information | M | Structure of Meta-Information | The meta-information (see 8.1). |
| Status-Info | M | Structure of Status-Primitive | The status information (see 8.2). |
| Public-Profile-List | M | Structure | The list of Public profiles per UserID. |

**Table 51. Information elements in GetPublicProfileResponse Primitive**

## 13.1.3 The "UpdatePublicProfile" Transaction

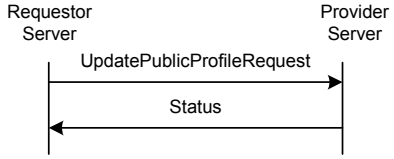


**Figure 32: The "UpdatePublicProfile" Transaction**

The IMPS system MUST support the "UpdatePublicProfile" transaction. A user MAY update his public profile through the "UpdatePublicProfile" transaction. The requestor server sends a UpdatePublicProfileRequest to the provider server. The provider server returns the Status primitive with the result of the transaction.

Please refer to CSP Sessions and Transactions Document [CSP] for details on the Public Profile and on the UpdatePublicProfile transaction.

| Primitive | Direction |
|---|---|
| UpdatePublicProfileRequest | Requestor Server → Provider Server |
| Status | Requestor Server ← Provider Server |

**Table 52. Primitive Directions for UpdatePublicProfile Transaction**

### 13.1.3.1.1 The "UpdatePublicProfileRequest" Primitive

The UpdatePublicProfileRequest primitive is used to update the the public profile of a of user.

| Information Element | Req | Type | Description |
|---|---|---|---|
| Message-Type | M | UpdatePublicProfile Request | Message identifier |
| Meta-Information | M | Structure of Meta-Information | The meta-information (see 8.1). |
| Clear-Public-Profile | M | Boolean | Indicates whether the public profile is cleared or not. |
| Public-Profile | O | Structure | The public profile fields to be updated for the requesting user. |

**Table 53. Information elements in UpdatePublicProfileRequest Primitive**

## 13.1.4 The "GeneralSearch" Transaction

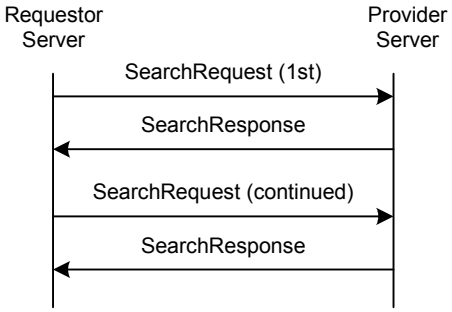


**Figure 33: The "GeneralSearch" Transaction**

The `SearchRequest` primitive allows a user to search for users or groups based on different properties of the user or group and MUST be supported by the IMPS server. 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 MUST be searched for the Search-String. When more than one search pair is specified in the primitive without using advanced search, a logical AND operation MUST be assumed among the different pairs. The IMPS Server MUST at least support USER-ID as criterion. Other Search criteria (see table 27 and 28) MAY also be supported. The service tree nodes that allows negotiation of optional feature for this transaction are 'Search' and sub nodes 'User' and 'Group'. The advanced search feature allows defining a more flexible search criteria. The advanced search is described in details in chapter 13.1.5

The search result is restricted in the same manner presence information is restricted when requested. If the searching user is not proactively authorized to see certain presence values for a user included in the search result, that presence value MUST not be included. If the unauthorized presence attribute is part of the search criteria, that user MUST not be included in the search result at all. Users that want to have certain presence attributes searchable SHOULD expose them through their default attribute list.

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

A server implementation SHOULD NOT allow searching based on Public Profile if the requesting user has not filled out the mandatory fields of his/her public profile. If the server does not allow it, the transaction MUST fail – in this case the server MUST respond with a Status primitive – when any Search-Element refers to a public profile field, and the requesting user did not fill out the mandatory fields of his/her own public profile. After the Status primitive the server MAY also send a System Message notification (see 13.1.11) that describes to the user that he/she SHOULD fill out the public profile as well as explaining the privacy issues related to filling in the Public Profile

The result of a user search MUST always be user-IDs and Friendly Name. If the Friendly Name exists in the public profile of the user then the IMPS server MUST include the Friendly Name from the public profile, otherwise the UserID MUST be included in the Friendly Name field instead. Similarly, the result of a group search MUST always be group-IDs.

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

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

- MUST verify the age field in the public profile of the users that have been found matching according the Search-Pair-List element, and:
  - if there is a local age limitation policy – to provide child protection – on the server, it MUST exclude those users from the search results who are under the age limitation according to their public profile.
  - if there is no local age limitation policy on the server, it MUST include all users in the search results.

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

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

The IMPS server MAY forward search request over SSP to IWF to extend and continue searching over non-IMPS technology domain as well.

| Search-Element | Search-String |
|---|---|
| USER_AGE_MIN | The *Search-String* is a decimal integer. |
| USER_AGE_MAX | The *Search-String* is a decimal integer. |
| USER_COUNTRY | The *Search-String* is a Country code as defined in [ISO3166-1]. |
| USER_FRIENDLY_NAME | The *Search-String* is a user's Friendly Name – free text. |
| USER_CITY | The *Search-String* is a city name – free text. |
| USER_GENDER | The *Search-String* is an enumerated value as 'Gender' described in chapter 7.3 of [CSP] . Using the 'U' (Unspecified) value is NOT RECOMMENDED – clients SHOULD exclude the criteria instead. |
| USER_INTENTION | The *Search-String* is a substring of a user's intention. |
| USER_INTERESTS_HOBBIES | The *Search-String* is a substring of a user's interests or hobbies. |
| USER_MARITAL_STATUS | The *Search-String* is an enumerated value as 'Marital status' described in chapter 7.3 of [CSP]. Using the 'U' (Unspecified) value is NOT RECOMMENDED – clients SHOULD exclude the criteria instead. |

**Table 54. Search elements for public profile-based user search**

| 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]. |
| USER_ONLINE_STATUS | The *Search-String* is an online status value. |

**Table 55. Search elements for private profile-based user search**

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* MUST be the searching user's own User-ID. The group property Searchable MUST be ignored and all groups MUST be treated as searchable. |
| GROUP_USER_ID_OWNER | The *Search-String* MUST be the searching user's own User-ID. Search result contains the list of groups owned by the specified user. The group property Searchable MUST be ignored and all groups MUST be treated as searchable. |
| GROUP_USER_ID_AUTOJOIN | The *Search-String* MUST be the searching user's own User-ID. Search result MUST contain the list of groups that have AutoJoin property set to "T" for the requesting user. The group property Searchable MUST be ignored and all groups MUST be treated as searchable. |

**Table 56. Search elements for group search**

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

**Table 57. Primitive Directions for GeneralSearch Transaction**

### 13.1.5 Advanced search mechanism

The advanced search mechanism allows a user to compose a free-form search criteria where not only logical AND, but also logical NOT and OR operations as well as nesting is allowed. The advanced search mechanism uses exactly the same transactions as the general search transaction, thus it inherits the same requirements – these requirements are not described here, please refer to chapter 13.1.4. The only difference between the advanced search and the general search mechanism is that when advanced search is performed, the 1st SearchRequest primitive MUST include additional information – the rest of the search mechanism is identical, including the StopSearch transaction.

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

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

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

An example

User wants to find his friend whose last name he knows, however he is not sure about how he registered his first name. The search elements will be:
USER_LAST_NAME = "Smith" with ID assigned to 0.
USER_FIRST_NAME = "John" with ID assigned to 1.
USER_FIRST_NAME = "Johnny" with ID assigned to 2.

The logical expression will be:
0+[1|2]

The server – upon receiving this request – will return all User-IDs and Friendly-Names that have "Smith" as their last name, and either "John" or "Johnny" as their first name.

### 13.1.5.1 Primitives

#### 13.1.5.1.1 The "SearchRequest" Primitive

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 MUST include only the Search-ID and the Search-Index. The provider server responds with the `SearchResponse`, but the message MUST include 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 MUST be valid until a new search is performed or the session ends (a previous search is invalidated when a new search is started).

| Information Element | Req | Type | Description |
|---|---|---|---|
| Message-Type | M | SearchRequest | Message identifier |
| Meta-Information | M | Structure of Meta-Information | The meta-information (see 8.1). |
| Search-Pair-List | C | Structure | Search criteria in terms of properties. For advanced searches, each search pair MUST have its own unique identifier included It is present only in the 1st search request. |
| Advanced-Criteria | C | String | The advanced search criteria in form of a logical expression. |
| Search-Limit | C | Integer | Indicates the number of maximum search results 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 58. Information elements in SearchRequest Primitive**

#### 13.1.5.1.2 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 8.2). |
| 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. |

| | | | |
|---|---|---|---|
| Completed | M | Boolean | Indicates if the client can expect new results. 'No' if server may provide new results (still searching), 'Yes' if new results will not be provided. |
| Search-Index | M | Integer | Indicates the index of the last result. This provides the user with the information of where to continue the next search. |
| Search-Results | C | Structure | Search results. |

**Table 59. Information elements in SearchResponse Primitive**

## 13.1.6 The "StopSearch" Transaction

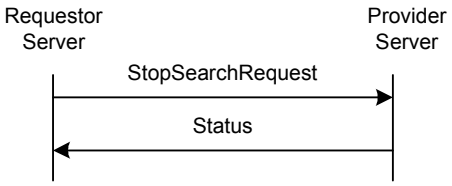


**Figure 34: The "StopSearch" Transaction**

The "**StopSearch**" transaction allows the requestor server to indicate to the provider server that the search and / or the results are not needed from a previously issued search request. The "**StopSearch**" transaction MUST be supported by the IMPS server. 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 60. Primitive Directions for StopSearch Transaction**

### 13.1.6.1 Primitives

#### 13.1.6.1.1 The "StopSearchRequest" Primitive

The `StopSearchRequest` primitive allows 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 8.1). |
| Search-ID | M | String | Identifies the search to be invalidated. |

**Table 61. Information elements in StopSearchRequest Primitive**

## 13.1.7 The "Invitation" Transaction

A user MAY invite other user(s) to join a discussion / chat group, to exchange messages, to share presence values list, to use end-to-end application communication, to initiate *Private Group Conversation*, to share content and to request to be added to a group's member list.

There are two service models with corresponding transaction flows.

### 13.1.7.1 Basic Invitation transaction

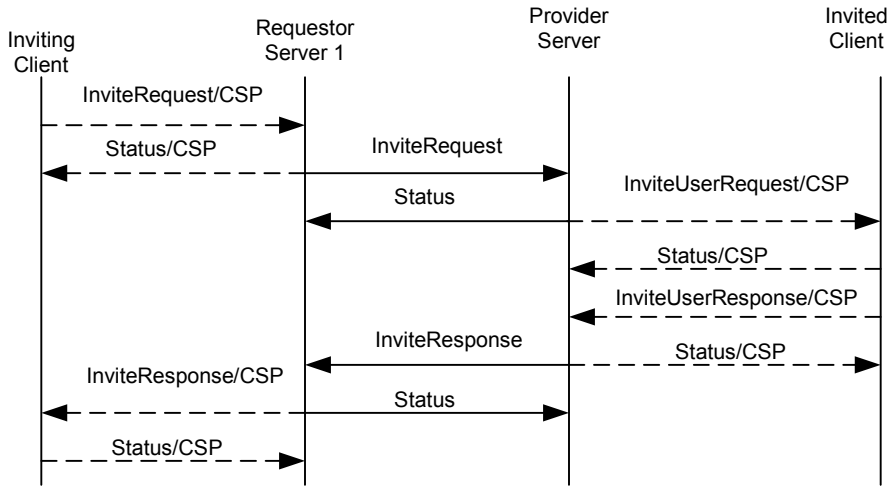


**Figure 35: The "Basic Invitation" Transaction**

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

The IMPS system MUST support the Basic Invitation Transaction

The transaction flow is as follows.

| 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 62. Primitive Directions for Basic 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 MUST respond to the requestor server 1 with a `Status` message and sends an invitation to all of the users indicated in the request.

The invited user MAY accept or reject the invitation. The provider server MUST 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 MUST respond to 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 `InviteResponse` primitive if the the provider server is able to collect the responses 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.

The Invite-ID MUST be the same in the InviteRequest, InviteUserRequest, InviteUserResponse and InviteResponse messages.

The provider server MUST support the Server Requirements of the corresponding transaction as defined in [CSP]. The provider server and requestor server MUST support the semantics and syntax mapping between SSP primitive and CSP primitive for the supported transactions.

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, the invited user MUST act consistently in accordance with its response.

The Provider Server MUST support all invitation cases, which IMPS service elements are supported.

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 share presence attributes manually (the invitation is only informational).

### 13.1.7.2 Complementary Invitation transaction

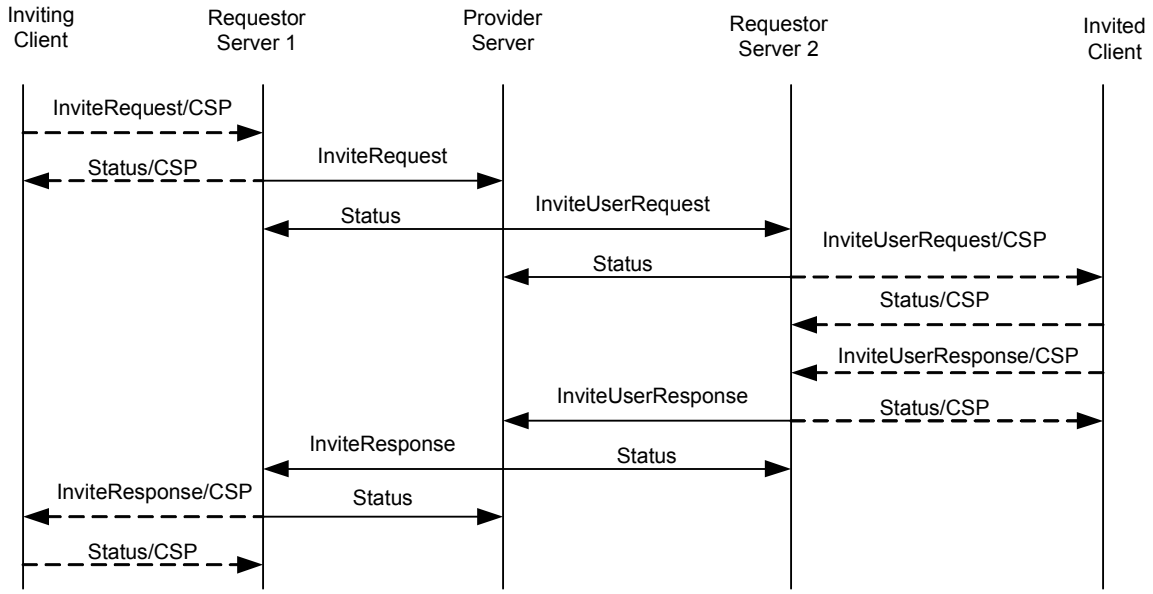


**Figure 36: The "Complementary Invitation" Transaction**

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 IMPS system MAY support the Complementary Invitation Transaction. The service tree node that allows negotiation of the this transaction is 'Complement Invite'

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 63. 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 MUST respond to the requestor server 1 with a `Status` message. The provider server MUST also send `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 MUST respond to the provider server with a `Status` message.

The invited user MAY accept or reject the invitation. The requestor server 2, which represents the invited users, SHALL respond to 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 MUST respond to the requestor server 2 with a Status message. The provider server MUST 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 MUST respond to 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 responses 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

The Invite-ID MUST be the same in the InviteRequest, InviteUserRequest, InviteUserResponse and InviteResponse messages

The provider server MUST support the Server Requirements of the corresponding transaction as defined in [CSP] and the requestor server 2 MUST the client requirements of the corresponding transaction as defined in [CSP]. The provider server and requestor servers MUST support the semantics and syntax mapping between SSP primitive and CSP primitive for the supported transactions

While in general there is no mandatory requirement about how an invited user MUST 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 in accordance with its response.

The subject of the invitation MAY be a group, messaging, a shared content, to use end-to-end application communication, to initiate *Private Group Conversation,* 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 share presence attributes manually (the invitation is only informational).

### 13.1.7.3 Primitives

#### 13.1.7.3.1 The “InviteRequest” Primitive

The `InviteRequest` primitive allows 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. If Invite-Type is GM, the Invited-User is the group ID.

| Information Element | Req | Type | Description |
|---|---|---|---|
| Message-Type | M | InviteRequest | Message identifier |
| Meta-Information | M | Structure of Meta-Information | The meta-information (see 8.1). |
| Invite-ID | M | String | Identifies this invitation. |
| Invite-Type | M | Enum {“GR”, “IM”, “PR”, “SC”, “GM”, “EC”, “EG“, “AP”} | Inviting for Group/chat (GR), Messaging (IM), Presence (PR), Content (SC) ), Group Membership (GM) , ExtendConversation (EC), ExtendGroup (EG) or End-to-End Application (AP). |
| Inviting-User | M | Structure | Identifies the requesting user who sends the invitation (User-ID (and optional Client-ID or Application-ID ) and optional Screen-Name). The Friendly Name is added as described in 6.3 |
| Invited-User | M | Structure | Identifies the user(s) to be invited (User-ID (and optional Client-ID or Application-ID), or Contact-List-ID). If Invite-Type is GM, identifies the group for which the group membership is requested. The Friendly Name(s) are addedas described in 6.3 |
| Invite-Group-ID | C | String | Identifies the group. It is mandatory if Invite-Type is ‘GR’, ‘GM’, 'EC', or EG. OPTIONAL if Invite-Type is ‘AP’. Otherwise it is not present. |
| Invite-Application-ID | C | Structure | Identifies the Application-ID. It is mandatory if invitation to End-to-End Application (AP), otherwise it is not present. |
| Invite-Presence-Attribute-List | C | Structure | Identifies the Presence Attributes that the inviter wants to share with the invitees. It is optional if Invite-Type is ‘PR’ or ‘AP’, otherwise it is not present. |
| Invite-Content-ID-List | C | Structure | Identifies the related shared content as a list of URLs. (Mandatory if Invite-Type is ‘SC’, OPTIONAL if Invite-Type is ‘AP’,otherwise it is not present. |
| Invite-Reason | O | String | Textual description of the invitation. |
| Own-Screen-Name | O | Structure | The user’s screen name from whom the invitation originates, to inform the invited user about it when desired. |
| Validity | O | Integer in seconds | Indicates the interval over which the invitation is valid. |

**Table 64. Information elements in InviteRequest Primitive**

### 13.1.7.3.2 The “InviteResponse” Primitive

The `InviteResponse` primitive allows the provider server to return the result of the invitation to the requestor server, representing the inviting user.

| Information Element | Req | Type | Description |
|---|---|---|---|
| Message-Type | M | InviteResponse | Message identifier. |

| Information Element | Req | Type | Description |
|---|---|---|---|
| Status-Info | M | Structure of Status-Primitive | The status information (see 8.2). |
| Invite-ID | M | String | Identifies this invitation. |
| Inviting-User | M | Structure | Identifies the user from whom the invitation originates. (User-ID, User-ID with Client-ID or User-ID with Application-ID). This value MUST be the same as the Inviting-User element received in InviteUserResponse unless the element contained ContactList-ID(s) or the server adds Friendly Name as described in 6.3. |
| Invite-Acceptance | M | Boolean | Indicates if the user accepts the invitation or not. |
| Responding-User | M | Structure | Identifies the responding user. (User-ID, User-ID with Client-ID or User-ID with Application-ID and Friendly Name if available). If Invite-Type is GM, identifies the group for which the membership is requested. This value MUST be the same as the Responding-User element received in InviteUserResponse unless the server adds Friendly Name to the User-ID(s)as described in 6.3. If Invite-Type was GM, identifies the group for which the group membership is requested. |
| Invite-Response | O | String | Textual description, why the invited user accepted/rejected the invitation. |
| Own-Screen-Name | O | Structure | The invited user's screen name to inform the user from whim the invitation originates about it when desired. |

**Table 65. 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.

#### 13.1.7.3.3 The "InviteUserRequest" Primitive

The `InviteUserRequest` primitive allows 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 or to become a group member.

| Information Element | Req | Type | Description |
|---|---|---|---|
| Message-Type | M | InviteUserRequest | Message identifier |
| Meta-Information | M | Structure of Meta-Information | The meta-information (see 8.1). |
| Invite-ID | M | String | Identifies this invitation. |
| Invite-Type | M | Enumerated String | Inviting for Group/chat (GR), Messaging (IM), Presence (PR), Content (SC), Group Membership (GM), ExtendConversation (EC), ExtendGroup (EG), or End-to-End Application (AP). |

| Information Element | Req | Type | Description |
|---|---|---|---|
| Inviting-User | M | Structure | Identifies user from whom the invitation originates (User-ID (and optional Client-ID or Application-ID) and optional Screen-Name). The Friendly Name is added as described in 6.3 |
| Invited-User | M | Structure | Identifies the user(s) to be invited (User-ID (and optional Client-ID or Application-ID), or List-of-User-IDs). The Friendly Name is added as described in 6.3 |
| Invite-Group-ID | C | String | Identifies the group. It is mandatory if Invite Type is 'GR', 'GM', 'EC' or 'EG'. OPTIONAL if Invite-Type is 'AP', otherwise it is not present. |
| Invite-Application-ID | C | Structure | Identifies the Application-ID. It is mandatory if invitation to End-to-End Application (AP), otherwise it is 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 Invite-Type is 'PR' or 'AP', otherwise it is not present. |
| Invite-Content-ID-List | CO | Structure | Identifies the related shared content as a list of URLs. . (Mandatory if Invite-Type is 'SC', OPTIONAL if Invite-Type is 'AP', otherwise it is not present. |
| Invite-Reason | O | String | Textual description of the invitation. |
| Own-Screen-Name | O | Structure | The user's screen name from whom the invitation originates, to inform the invited user about it when desired. |
| Validity | O | Integer in seconds | Indicates the interval in which the invitation is valid. |

**Table 66. Information elements in InviteUserRequest Primitive**

#### 13.1.7.3.4 The "InviteUserResponse" Primitive

The `InviteUserResponse` primitive allows the requestor server, representing 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 8.2). |
| Invite-ID | M | String | Identifies this invitation. |
| Inviting-User | M | Structure | Identifies the user from whom the invitation originates (User-ID (and optionally Client-ID or Application-ID)). The Friendly Name is added as described in 6.3 |

| Information Element | Req | Type | Description |
|---|---|---|---|
| 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 optionally Client-ID or Application.ID)) and optional Screen-Name). If Invite-Type was GM, identifies the group for which the group membership is requested. |
| Invite-Response | O | String | Textual description, why the invited user accepted/rejected the invitation. |
| Own-Screen-Name | O | Structure | The invited user's screen name to inform the user from whom the invitation originates about it when desired. |

**Table 67. 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.

## 13.1.8 The "CancelInvitation" Transaction

The IMPS system MUST support the cancellation of any previous invitations.

### 13.1.8.1 Basic Cancel Invitation transaction

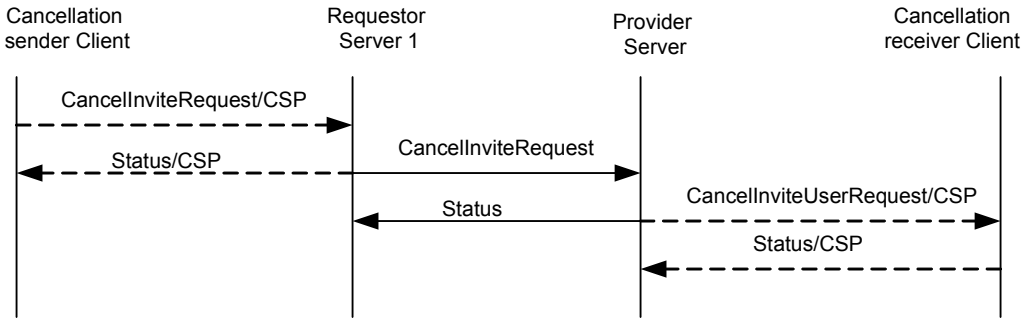


**Figure 37: The "Basic CancelInvitation" Transaction**

The requestor server 1 is the Home Domain of the invitation cancelling 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 68. Primitive Directions for Basic 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 MUST respond to the requestor server 1 with a `Status` message. The provider server MUST also send `CancelnviteUserRequest(CSP)` message to every 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 provider server, which represents the canceled users, MUST respond to the provider server with the Status message.

The Invite-ID MUST refer to to a previously sent out invitation and MUST be the same in the CancelInviteRequest and CancelInviteUserRequest

The provider server MUST support the Server Requirements of the corresponding transaction as defined in [CSP]. The provider server and requestor server MUST support the semantics and syntax mapping between SSP primitive and CSP primitive for the supported transactions

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

### 13.1.8.2 Complementary Cancel Invitation transaction

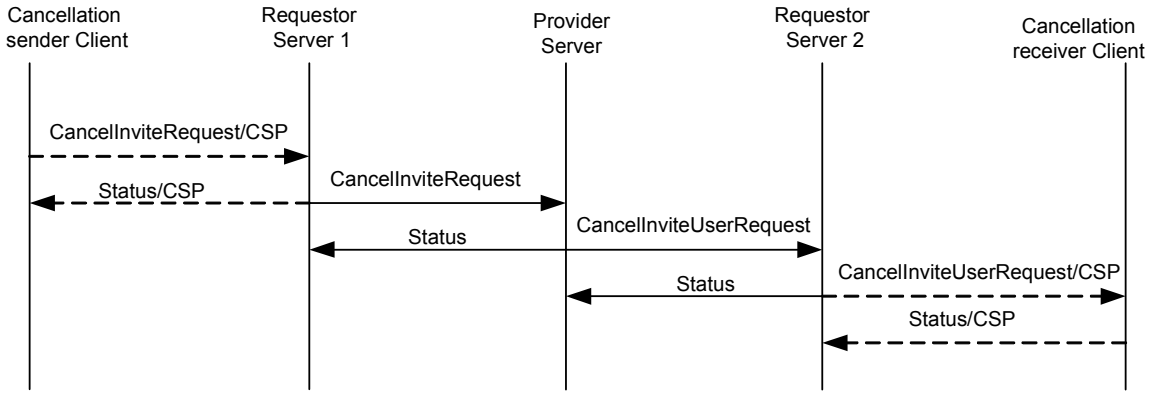


**Figure 38: The "Complementary CancelInvitation" Transaction**

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 69. 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 MUST respond to the requestor server 1 with a `Status` message. The provider server MUST also send `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, MUST respond to the provider server with the Status message.

The Invite-ID MUST refer to to a previously sent out invitation and MUST be the same in the InviteRequest, InviteUserRequest, InviteUserResponse and InviteResponse messages

The provider server MUST support the Server Requirements of the corresponding transaction as defined in [CSP] and the requestor server 2 MUST the client requirements of the corresponding transaction as defined in [CSP]. The provider server and requestor servers MUST support the semantics and syntax mapping between SSP primitive and CSP primitive for the supported transactions

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

### 13.1.8.3 Primitives

#### 13.1.8.3.1 The "CancelInviteRequest" Primitive

The `CancelInviteRequest` primitive allows 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 8.1). |
| Invite-ID | M | String | Identifies the invitation. |
| Canceling-User | M | Structure | Identifies the requesting user who cancels the invitation (User-ID (and optionally Client-ID or Application-ID) and optional Screen-Name). The Friendly Name is added as described in 6.3 |
| Canceled-User | M | Structure | Identifies the user(s) to whom the invitation will be canceled (User-ID(and optionally Client-ID or Application-ID) or Contact-List-ID). The Friendly Name is added as described in 6.3. If Invite-Type of the related invitation was 'GM', identifies the group for which the membership request is canceled. |
| Canceled-Group-ID | C | String | Identifies the group. It is mandatory if Invite-Type is 'GR', 'GM', 'EG', or 'EC'; OPTIONAL if Invite-Type is 'AP', otherwise it is not present. |
| Canceled-Application-ID | C | Structure | Identifies the Application-ID. It is mandatory if invitation to End-to-End Application (AP), otherwise it is not present. |
| Canceled-Presence-Attribute-List | C | Structure | Identifies the Presence Attributes that the inviter wants to share with the invitees. It is optional if Invite-Type is 'PR' or 'AP', otherwise it is not present. |

| Information Element | Req | Type | Description |
|---|---|---|---|
| Canceled-Content-ID-List | C | Structure | Identifies the related shared content as a list of URLs, which will be canceled. (Mandatory if Invite-Type of the related invitation was 'SC', OPTIONAL if Invite-Type of the related invitation was 'AP', otherwise not present.) |
| Cancel-Reason | O | String | Textual description of the cancel. |
| Own-Screen-Name | O | Structure | The user's screen name from whom the cancelling request originates; to inform the canceled user about it when desired. |

**Table 70. Information elements in CancelInviteRequest Primitive**

### 13.1.8.3.2 The "CancelInviteUserRequest" Primitive

The `CancelInviteUserRequest` primitive allows 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 8.1). |
| Invite-ID | M | String | Identifies the invitation. |
| Canceling-User | M | Structure | Identifies the requesting user who cancels the invitation (User-ID (and optional Client-ID or Application-ID) and / or Screen-Name) |
| Canceled-User | C | Structure | Identifies the user(s) to whom the invitation will be canceled (User-ID and / or Screen-Name, or List-of-User-IDs). The Friendly Name is added as described in 6.3. Not present if the invitation was to a group membership |
| Canceled-Group-ID | C | String | Identifies the group. It is mandatory if Invite-Type is 'GR', 'GM', 'EC' or 'EG'; OPTIONAL if Invite-Type is 'AP', otherwise it is not present. |
| Canceled-Application-ID | C | Structure | Identifies the Application-ID. It is mandatory if invitation to End-to-End Application (AP), otherwise it is not present. |
| Canceled-Presence-Attribute-List | C | Structure | Identifies the Presence Attributes that the inviter wants to share with the invitees. It is optional if Invite-Type is 'PR' or 'AP', otherwise it is not present. |
| Canceled-Content-ID-List | C | Structure | Identifies the related shared content as a list of URLs, which will be canceled. (Mandatory if Invite-Type of the related invitation was 'SC', OPTIONAL if Invite-Type of the related invitation was 'AP', otherwise not present.) |

| Information Element | Req | Type | Description |
|---|---|---|---|
| Cancel-Reason | O | String | Textual description of the cancel. |
| Own-Screen-Name | O | Structure | The user's screen name from whom the cancelling request originates; to inform the canceled user about it when desired. |

**Table 71. Information elements in CancelInviteUserRequest Primitive**

### 13.1.9 The "GetMap" Transaction

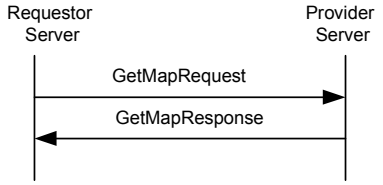


**Figure 39: The "GetMap" Transaction**

The "GetMap" transaction is used by the requestor server to retrieve map information based on location information or UserID.

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

When the requestor server requests map information based on UserID, the GetMapRequest primitive MUST include the UserID.

The requestor server MUST NOT include both GeoLocation and UserID elements in the request.

When the request contains UserID, the provider server MUST verify that the requested user has authorized the requesting user for the GeoLocation presence attribute, and the transaction MUST fail if the requesting user is not authorized – the provider server MUST NOT reveal the fact that the requested user did not authorize the requesting user the GeoLocation attribute, thus the server MUST reply with error code 517 in this case.

Upon success the provider server MUST return either the map or a URL allowing the client to retrieve the requested map.

If the provider server is unable to generate a map or a URL for the requested location, it MUST reply with error code 517.

The GetMapRequest primitive SHOULD contain information about the content type and size limitations of the client that is requesting the map information. If the provider server returns a map, it MUST select a content type and size based on this information that the client is able to handle. If the provider server is not able to deliver such contentmedia type, it MUST respond with error 411. If the provider server found a suitable media type, but the size limitation of the media type is too small to fit the content, it SHOULD attempt to find another content type and convert the media to that type. If the provider server did not find another media type where the content would fit, it MAY attempt to scale the content down to a smaller size. If the provider server cannot scale the content down or the scaling would produce content with unreasonable quality, the provider server MUST respond with error 432.

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

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

The "GetMap" transaction MAY be supported by the IMPS system. The service tree node that allows negotiation of this transaction is 'VerifyUser'

| Primitive | Direction |
|---|---|
| GetMapRequest | Requestor Server → Provider Server |
| GetMapResponse | Requestor Server ← Provider Server |

**Table 72. Primitive Directions for the GetMap Transaction**

### 13.1.9.1 Primitives

#### 13.1.9.1.1 The "GetMapRequest" Primitive

The GetMapRequest primitive allows the requestor server retrieve map information based on location information or UserID..

| Information Element | Req | Type | Description |
|---|---|---|---|
| Message-Type | M | GetMapRequest | Message identifier |
| Meta-Information | M | Structure of Meta-Information | The meta-information (see 8.1). |
| Accepted-Content-Type | O | Structure | Structure(s) containing the media types and the related credentials of those content types that the client supports. |
| Geo-Location | C | Structure | Contains a GeoLocation presence attribute. It is present only when UserID is not present. |
| User-ID | C | String | Identifies the user whose map information is requested. It is present only when GeoLocation is not present. |

**Tabel 73. Information elements in GetMapRequest Primitive**

#### 13.1.9.1.2 The "GetMapResponse" Primitive

The `GetMapResponse` primitive allows the provider server to return the map information to the requester server.

| Information Element | Req | Type | Description |
|---|---|---|---|
| Message-Type | M | GetMapResponse | Message identifier. |
| Status-Info | M | Structure of Status-Primitive | The status information (see 8.2). |
| Map | C | Structure | An image that contains the map of the requested location or user. |
| URL | C | String | The URL allowing the client to retrieve the requested map. |

**Tabel 74. Information elements in GetMapResponse Primitive**

## 13.1.10 The "VerifyID" Transaction



**Figure 40: The "VerifyID" Transaction**

The "VerifyID" transaction is used by the requestor server to verify that a list of WV IDs is in use and are valid at the provider server. The transaction is used before the WV ID is stored in the requestor sever to ensure that all locally stored WV IDs are valid. The VerifyID response contains the result of the verification, and a list of WV IDs along with the time when the valid WV ID was created. The time information is used to verify that the locally stored WV 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 if the WV IDsare valid and in use.

The "VerifyID" transaction MAY be supported by the IMPS system. The service tree node that allows negotiation of this transaction is 'VerifyUser'

| Primitive | Direction |
|---|---|
| VerifyIDRequest | Requestor Server → Provider Server |
| VerifyIDResponse | Requestor Server ← Provider Server |

**Table 75. Primitive Directions for the VerifyUserid Transaction**

### 13.1.10.1 Primitives

#### 13.1.10.1.1 The "VerifyIDRequest" Primitive

The `VerfiyIDRequest` primitive allows the requestor server to verify that userid(s) are valid in the provider server.

| Information Element | Req | Type | Description |
|---|---|---|---|
| Message-Type | M | VeifyIDRequest | Message identifier |
| Meta-Information | M | Structure of Meta-Information | The meta-information (see 8.1). |
| WV-ID-List | M | Structure | The list contains the WV-ID's to be verified, and optionally the time when the WV ID was created |

**Table 76. Information elements in VerfifyIDRequest Primitive**

#### 13.1.10.1.2 The "VerifyIDResponse" Primitive

The `VerifyIDResponse` primitive allows the provider server to return the result of the verification, and the list of valid WV IDs along with the time when the valid WV ID was created.

| Information Element | Req | Type | Description |
|---|---|---|---|
| Message-Type | M | VerifyIDResponse | Message identifier. |
| Status-Info | M | Structure of Status-Primitive | The status information (see 8.2). |
| WV-ID-List | M | Structure | The list contains the valid WV Ids along with the time when the valid WV ID was created. |

**Table 77. Information elements in VerifyIDResponse Primitive**

## 13.1.11 The "SystemMessageRequest" Transaction

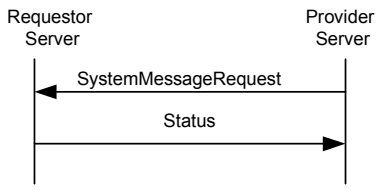


**Figure 41: The "SystemMessageRequest" Transaction**

The provider server MAY send a System Message notification to the user using the SystemMessageRequest transaction. The SystemMessageRequest transaction supports sending a System Message notification to the client triggered by a server event. The provider server MAY also send a System Message notification in any response to a client-triggered event through the Status element in those transactions that contain a Status element [see **Error! Reference source not found.**].

Please refer to CSP Sessions and Transactions Document [CSP] for the details on the SystemMessageRequest transaction.

Suppor for the SystemMessageRequest transaction is mandatory for requestor and provider servers.

| Primitive | Direction |
|---|---|
| SystemMessageRequest | Requestor Server ← Provider Server |
| Status | Requestor Server → Provider Server |

**Table 78. Primitive Directions for the SystemMessageRequest Transaction**

### 13.1.11.1 Primitives

#### 13.1.11.1.1 The "SystemMessageRequest" Primitive

The SystemMessageRequest primitive allows the provider server to send a list of system message notifications to the end-user.

| Information Element | Req | Type | Description |
|---|---|---|---|
| Message-Type | M | SystemMessageRequest | Message identifier |
| Meta-Information | M | Structure of Meta-Information | The meta-information (see 8.1). |
| System-Message-List | M | Structure | System Message(s) from provider server |

**Table 79. Information elements in SystemMessageRequest Primitive**

## 13.1.12 The "SystemMessageUser" Transaction

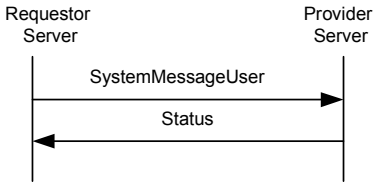


**Figure 42: The "SystemMessageUser" Transaction**

The requestor server sends an end-user system message response to the requestor server using the SystemMessageUser transaction.

Please refer to CSP Sessions and Transactions Document [CSP] for the details on the SystemMessageUser transaction.

Support for the SystemMessageUser transaction is mandatory for requestor and provider servers.

| Primitive | Direction |
|---|---|
| SystemMessageUser | Requestor Server → Provider Server |
| Status | Requestor Server ← Provider Server |

**Table 80. Primitive Directions for the SystemMessageUser Transaction**

#### 13.1.12.1 Primitives

##### 13.1.12.1.1 The "SystemMessageUser" Primitive

The SystemMessageUser primitive allows the requestor server to send a list of end-user system message responses to the requestor server.

| Information Element | Req | Type | Description |
|---|---|---|---|
| Message-Type | M | SystemMessageUser | Message identifier |
| Meta-Information | M | Structure of Meta-Information | The meta-information (see 8.1). |
| System-Message-Response-List | M | Structure | System Message Response(s) from user. |

**Table 81. Information elements in SystemMessageUser Primitive**

# 13.2 Status Code

## 13.2.1 "GeneralNotification" Transactions

- Invalid Notification Type (433)
- Not allowed notification type (440)

## 13.2.2 "GetPublicProfile" Transaction

- Unknown user ID (531)
- Missing mandatory profile field(s) of requesting user (904)
- Missing mandatory profile field(s)of requested user (905)
- Too many public profiles requested (201/906)

## 13.2.3 "UpdatePublicProfile" Transaction

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

## 13.2.4 "GeneralSearch" Transaction

- Unable to parse criteria. (Invalid Search-Element) (402)
- Initial search request was not sent (Invalid Search-ID) (424)
- Invalid Search-Index (out of range) (425)
- Search timeout (in case of continued search the subsequent request primitive is late) (535)
- Server search limit is exceeded (610)
- Too many hits (536)
- Too broad search criteria (537)
- Too many elements in advanced criteria (544)
- Too many levels of nesting in advanced criteria (545)

- Unsupported search-element was requested (560)
- Supported search-element (561)
- Unsupported search-element (562)
- Missing mandatory elements of requesting user (904)
- Missing mandatory elements of requested user (905)

### 13.2.5 “StopSearch” Transaction

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

### 13.2.6 “Invitation” Transaction

- Invalid invitation type (402)
- Unknown user (ID or screen-name) (531)
- Group does not exist (800)
- Delivery to recipient not available (410)
- Invalid invite-ID (423)
- Invalid User-ID (427)
- Invalid Client-ID (428)
- Missing Application-ID (450)
- Invalid Application-ID (451)
- Forbidden Application-ID (452)
- Delivery to recipient domain not available (516)
- Recipient unknown (Contact list) (700)
- Invalid or unsupported presence value (751)

### 13.2.7 “CancelInvitation” Transaction

- Invalid invitation type (402)
- Invalid invitation ID (423)
- Unknown user (ID or screen-name) (531)
- Delivery to recipient not available (410)
- Delivery to recipient domain not available (516)
- Recipient unknown (Contact list) (700)

### 13.2.8 “GetMap” Transaction

- Service Not Supported (405)
- Unknown user (531)
- Unable to find suitable content type (411)

- Response too large (432)
- Location Not Supported (517)

### 13.2.9 VerifyID" Transaction

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

### 13.2.10 "SystemMessageRequest" Transaction

- Verification Mechanism Not Supported (439)

### 13.2.11 "SystemMessageUser" Transaction

- Unknown System Message ID (437)
- Incorrect Verification Key (438)
- Too many non-conformant System Message replies (611)

# 14 Service Relay – Contact List Features

## 14.1 Overview

A "***contact list***" is a list created and maintained by a User so that the User may send messages to the "*contact list*" as a 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 the members of the list 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*" has 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 allows 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) each other's contact list(s).

There are three 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.

**DoNotNotify**: a Boolean value indicating whether the user added to a Contact List should be notified of such event. When the value is true ('T'), the added user will not be notified. When the value is false ('F'), the added user will be notified.

When the user creates his/her first contact list, the server 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 automatically set to "False" by the server.

Watchers list is a system defined contact list with the functionality limited to holding 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. The server shall maintain one Watcher List for each user.

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

The transactions below belong to the complementary service.

# 14.2 Transactions

## 14.2.1 The "CreateContactList" Transaction

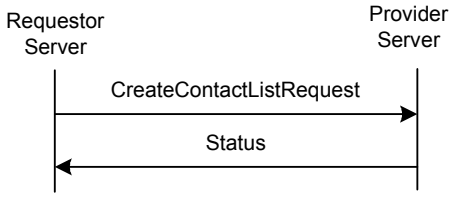


**Figure 43: The "CreateContactList" Transaction**

The IMPS system MAY support the "CreateContactList" transaction. The service tree node that allows negotiation of this transaction is 'Contact List Update'.

The requestor server sends a `CreateContactListRequest` to the provider server. The provider server MUST create the contact list and respond with a `Status` message to the requestor server.

A user MUST be able to create more than one contact list. There MAY be system specific limitations for the maximum number of lists per user. After a 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 82. Primitive Directions for CreateContactList Transaction**

### 14.2.1.1 Primitives

#### 14.2.1.1.1 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 | CreateContactListRequest | Message identifier |
| Meta-Information | M | Structure of Meta-Information | The meta-information (see 8.1). |
| Contact-List-ID | M | String | Identifies the contact list. |
| Contact-List-Props | O | Structure | The initial properties of the contact list. |
| User-List | O | Structure | Identifies the initial users to be added to the contact list (User-ID, Nickname). |

**Table 83. Information elements in CreateContactListRequest Primitive**

## 14.2.2 The "DeleteContactList" Transaction

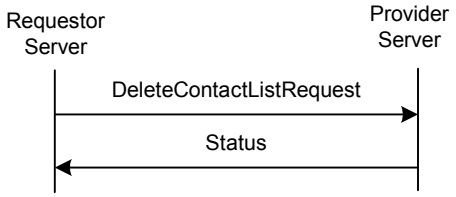


**Figure 44: The "DeleteContactList" Transaction**

The IMPS system MAY support the "DeleteContactList" transaction. The service tree node that allows negotiation of this transaction is 'Contact List Update'

The requestor server sends a `DeleteContactListRequest` to the provider server. The provider server MUST delete the contact lists(s) and respond with a `Status`. The server SHOULD not unsubscribe the members implicitly; if a contact list that 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 in one transaction.

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

**Table 84. Primitive Directions for DeleteContactList Transaction**

### 14.2.2.1 Primitives

#### 14.2.2.1.1 The "DeleteContactListRequest" Primitive

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

| Information Element | Req | Type | Description |
|---|---|---|---|
| Message-Type | M | DeleteContactListRequest | Message identifier |
| Meta-Information | M | Structure of Meta-Information | The meta-information (see 8.1). |
| Contact-List-ID-List | M | Structure | Identifies the contact list(s). |

**Table 85. Information elements in DeleteContactListRequest Primitive**

## 14.2.3 The "GetContactList" Transaction


**Figure 45: The "GetContactList" Transaction**

The IMPS system MAY support the "GetContactList" transaction. The service tree node that allows negotiation of this transaction is 'Contact List Get'

The "**GetContactList**" transaction allows 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 MUST return 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 86. Primitive Directions for GetContactList Transaction**

#### 14.2.3.1 Primitives

##### 14.2.3.1.1 The "GetContactListRequest" Primitive

The `GetContactListRequest` primitive allows 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 8.1). |

**Table 87. Information elements in GetContactListRequest Primitive**

##### 14.2.3.1.2 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 8.2). |
| 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 88. Information elements in GetContactListResponse Primitive**

### 14.2.4 The "GetListMember" Transaction

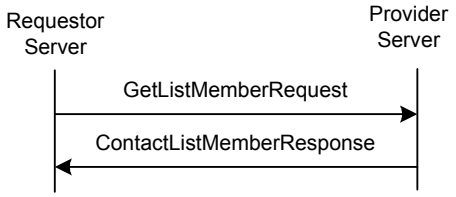


**Figure 46: The "GetListMember" Transaction**

The IMPS system MAY support the "GetListMember" transaction. The service tree node that allows negotiation of this transaction is 'Contact List Get'

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 MUST respond to the requestor server with a `ContactListMemberResponse` containing the list of all members of the contact list.

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

**Table 89. Primitive Directions for GetListMember Transaction**

#### 14.2.4.1 Primitives

##### 14.2.4.1.1 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 8.1). |
| Contact-List-ID | M | String | Identifies the contact list. |

**Table 90. Information elements in GetListMemberRequest Primitive**

##### 14.2.4.1.2 The "ContactListMemberResponse" Primitive

The `ContactListMemberResponse` primitive returns a list of all members in a contact list.

| Information Element | Req | Type | Description |
|---|---|---|---|
| Message-Type | M | ContactListMemberResponse | Message identifier |
| Status-Info | M | Structure of Status-Primitive | Status information (see 8.2). |
| Contact-List-ID | M | String | Identifies the contact list. |
| User-List | C | Structure | Identifies the contact list members (User-ID, Nickname). |

**Table 91. Information elements in ContactListMemberResponse Primitive**

### 14.2.5 The "AddListMember" Transaction

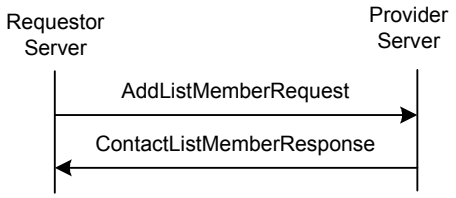


**Figure 47: The "AddListMember" Transaction**

The IMPS system MAY support the "AddListMember" transaction. The service tree node that allows negotiation of this transaction is 'Contact List Update'

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 to the requestor server with a `ContactListMemberResponse` containing the list of all members of the contact list.

The requestor server MAY also request authorizing OnlineStatus presence attribute to all users on the contact list and addition of Contact-List-ID to the grant list using the Authorize-And-Grant information element.

The Authorize-And-Grant is not applicable with value 'F', thus it MUST NOT be used. The provider server MUST ignore the Authorize-And-Grant element when its value is 'F'.

When the provider server receives the ListManageRequest primitive including Authorize-And-Grant information element and its value is 'T', the server MUST:

Add the Contact-List-ID to the grant list of the requested user. If the Contact-List-ID already exists in the grant list of the requesting user, the server MUST NOT add the Contact-List-ID again to the grant list.

Add the OnlineStatus presence attribute to the presence attribute list that is assigned to the requested contact list. When the contact list does not have a presence attribute list assigned, the server MUST create a presence attribute list containing only the OnlineStatus attribute, and assign it to the requested contact list.

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

**Table 92. Primitive Directions for AddListMember Transaction**

#### 14.2.5.1 Primitives

##### 14.2.5.1.1 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 8.1). |
| 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). |
| Authorize-And-Grant | O | Boolean | Indicates if the contact list ID is to be added to the grant list. It also indicates that the online status presence attribute is to be added to the presence attributes list assigned to the contact list. The default value is false. |

**Table 93. Information elements in AddListMemberRequest Primitive**

### 14.2.6 The "RemoveListMember" Transaction

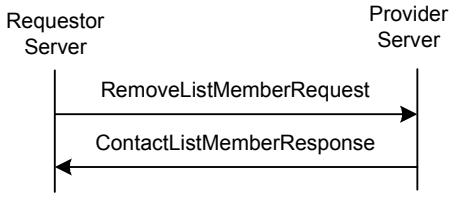


**Figure 48: The "RemoveListMember" Transaction**

The IMPS system MAY support the "RemoveListMember" transaction. The service tree node that allows negotiation of this transaction is 'Contact List Update'

The requestor server sends a `RemoveListMemberRequest` to the provider server. The provider server MUST delete the specified user(s) from the specified contact list, and return a list of all members of the contact list in the `ContactListMemberResponse`.

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

**Table 94. Primitive Directions for RemoveListMember Transaction**

#### 14.2.6.1 Primitives

##### 14.2.6.1.1 The "RemoveListMemberRequest" Primitive

The `RemoveListMemberRequest` primitive is used to remove 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 8.1). |
| 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 95. Information elements in RemoveListMemberRequest Primitive**

### 14.2.7 The "GetListProperties" Transaction


**Figure 49: The "GetListProperties" Transaction**

The IMPS system MAY support the "GetListProperties" transaction. The service tree node that allows negotiation of this transaction is 'Contact List Get'

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

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

**Table 96. Primitive Directions for GetListProperties Transaction**

#### 14.2.7.1 Primitives

##### 14.2.7.1.1 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- | The meta-information (see 8.1). |

| | | | |
|---|---|---|---|
| | | Information | |
| Contact-List-ID | M | String | Identifies the contact list. |

**Table 97. Information elements in GetListPropsRequest Primitive**

#### 14.2.7.1.2 The "ContactListPropsResponse" Primitive

The `ContactListPropsResponse` primitive returns a list of all members in a contact list.

| Information Element | Req | Type | Description |
|---|---|---|---|
| Message-Type | M | ContactListPropsResponse | Message identifier |
| Status-Info | M | Structure of Status-Primitive | Status information (see 8.2). |
| Contact-List-Props | M | Structure | The properties of the contact list. |

**Table 98. Information elements in ContactListPropsResponse Primitive**

## 14.2.8 The "SetListProperties" Transaction

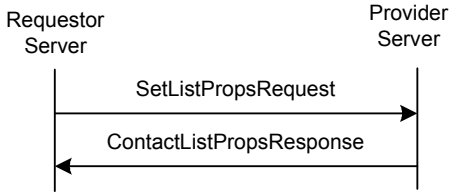


**Figure 50: The "SetListProperties" Transaction**

The IMPS system MAY support the "SetListProperties" transaction. The service tree node that allows negotiation of this transaction is 'Contact List Update'

The "**SetListProperties**" transaction is used to set the properties of a contact list (Display-Name, Default or DoNotNotify), 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 MUST respond with a `ContactListPropsResponse` to the requestor server containing the new properties.

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

**Table 99. Primitive Directions for SetListProperties Transaction**

### 14.2.8.1 Primitives

#### 14.2.8.1.1 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 8.1). |
| Contact-List-ID | M | String | Identifies the contact list. |
| Contact-List-Props | M | Structure | The properties to be set. |

**Table 100. Information elements in SetListPropsRequest Primitive**

## 14.2.9 The "CreateAttributeList" Transaction

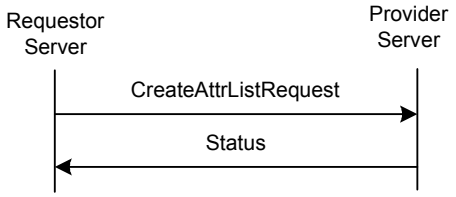


**Figure 51: The "CreateAttributeList" Transaction**

The IMPS system MUST support 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 MUST create an attribute list, and attach the attribute list to specified contact list(s) and / or user(s).

The provider server MUST persistently store:

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

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

An attribute list SHALL be overwritten by creating a new one for the same user or contact list. This way, an attribute list can be modified.

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

**Table 101. Primitive Directions for CreateAttributeList Transaction**

### 14.2.9.1 Primitives

#### 14.2.9.1.1 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 8.1). |
| Presence-Attribute-List | M | Structure | A list of presence attributes. |
| Default-List | M | Boolean | 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. |
| Default-Notify | M | Boolean | User wants to receive reactive authorization notification for a user not |

| | | | |
|---|---|---|---|
| | | | proactively authorized in any way. |
| User-Notify | O | Boolean | User wants to receive notification whenever reactive authorization is necessary for a user specified by User-ID-List. |
| Contact-List-Notify | O | Boolean | User wants to receive notification whenever reactive authorization is necessary for a user on a Contact List specified by Contact-List-ID-List. |

**Table 102. Information elements in CreateAttrListRequest Primitive**

## 14.2.10 The "DeleteAttrList" Transaction

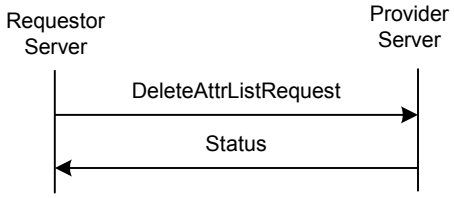


**Figure 52: The "DeleteAttrList" Transaction**

The IMPS system MUST support the "DeleteAttributeList" 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 MUST 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 MUST be cleared from the provider server (garbage collection).

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

**Table 103. Primitive Directions for DeleteAttrList Transaction**

### 14.2.10.1 Primitives

#### 14.2.10.1.1 The "DeleteAttrListRequest" Primitive

The `DeleteAttrListRequest` primitive is used to delete an 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 8.1). |
| Default-List | M | Boolean | 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 104. Information elements in DeleteAttrListRequest Primitive**

## 14.2.11 The "GetAttrList" Transaction

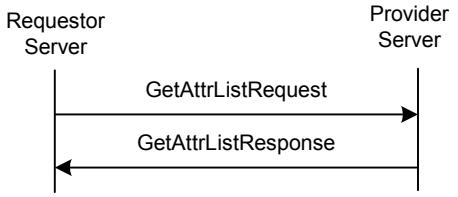


**Figure 53: The "GetAttrList" Transaction**

The IMPS system MUST support 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 MUST return the requested attributes. If the user(s) or contact list(s) are not specified in the request, the response MUST include all user-specific and contact list-specific attributes.

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

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

**Table 105. Primitive Directions for GetAttrList Transaction**

### 14.2.11.1 Primitives

#### 14.2.11.1.1 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).

| Information Element | Req | Type | Description |
|---|---|---|---|
| Message-Type | M | GetAttrListRequest | Message identifier |
| Meta-Information | M | Structure of Meta-Information | The meta-information (see 8.1). |
| Default-List | M | Boolean | Indicates if the default attribute list should be retrieved 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 106. Information elements in GetAttrListRequest Primitive**

#### 14.2.11.1.2 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 8.2). |
| 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 107. Information elements in GetAttrListResponse Primitive**

# 14.3 Status Code

## 14.3.1 Contact List Transactions

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

## 14.3.2 Attribute List Transactions

- Unknown user ID (531)
- Contact list does not exist (700)
- Unknown presence attribute (not defined in [PA]) (750)

# 15 Service Relay – Presence Features

## 15.1 Overview

This chapter focuses on the functional relay of Presence features. Because of the server interoperation nature, the SSP has its own requirements 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 have been defined in the following two sections.

In order to achieve minimum level of interoperability both the requestor and provider servers MUST support the following functionalities:

- The “Subscribe” Transaction
- The “Unsubscribe” Transaction
- The “PresenceNotification” Transaction
- The “GetPresence” Transaction
- The “UpdatePresence” Transaction
- The “Suspend” Transaction

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

Please refer to the [CSP] document to understand how to relay the Presence features from client-server interaction (CSP) to server-server interoperation (SSP).

## 15.2 Transactions

### 15.2.1 The “Subscribe” Transaction

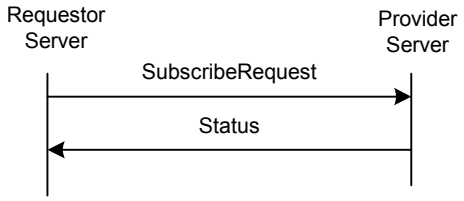


**Figure 54: The “Subscribe” Transaction**

The IMPS system MUST support the “Subscribe” transaction for a UserID list and MAY support the “Subscribe” transaction for a List object. The service tree node that allows negotiation of List addressing is ‘Contact List Addr’

The subscription for 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 MUST 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 MUST return a `Status` message indicating that the provider server has accepted and processed the request.

If the subscription succeeds, the requestor server MUST receive immediately the current presence information through a “**PresenceNotification**” transaction. 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 to 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 108. Primitive Directions for Subscribe Transaction**

#### 15.2.1.1 Primitives

##### 15.2.1.1.1 The "SubscribeRequest" Primitive

The `SubscribeRequest` primitive is used to create subscriptions to obtain notifications about changes of the PRESENCE INFORMATION and attributes of other PRINCIPALS.

| Information Element | Req | Type | Description |
|---|---|---|---|
| Message-Type | M | SubscribeRequest | Message identifier |
| Meta-Information | M | Structure of Meta-Information | The meta-information (see 8.1). |
| 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 109. Information elements in SubscribeRequest Primitive**

### 15.2.2 The "Unsubscribe" Transaction

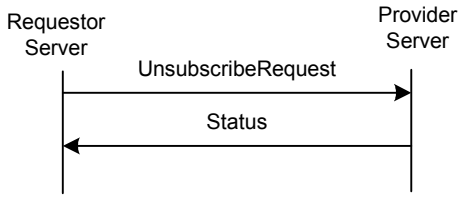


**Figure 55: The "Unsubscribe" Transaction**

The IMPS system MUST support the "Unsubscribe" transaction for a UserID list and MAY support the "Unsubscribe" transaction for a List object. The service tree node that allows negotiation of List addressing is 'Contact List Addr'

The cancellation of a 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 110. Primitive Directions for Unsubscribe Transaction**

#### 15.2.2.1 Primitives

##### 15.2.2.1.1 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 8.1). |
| User-ID-List | C | Structure | Identifies the users to be unsubscribed. |
| Contact-List-ID-List | C | Structure | Identifies the set of users. |

**Table 111. Information elements in UnsubscribeRequest Primitive**

### 15.2.3 The "PresenceNotification" Transaction

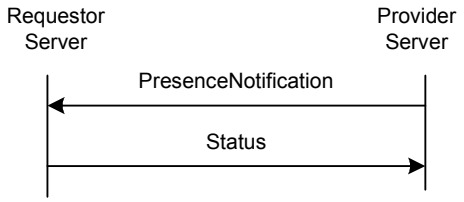


**Figure 56: The "PresenceNotification" Transaction**

The IMPS system MUST support the "PresenceNotification" transaction

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

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

**Table 112. Primitive Directions for PresenceNotification Transaction**

#### 15.2.3.1 Primitives

##### 15.2.3.1.1 The "PresenceNotification" Primitive

The `PresenceNotification` primitive allows 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 8.1). |
| User-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 113. Information elements in PresenceNotification Primitive**

## 15.2.4 The "GetWatcherList" Transaction

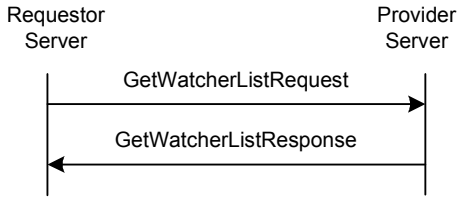


**Figure 57: The "GetWatcherList" Transaction**

The IMPS system MAY support the "GetWatcherList" transaction. The service tree node that allows negotiation of this transaction is 'Watcher List'

The purpose of the `GetWatcherList` transaction is to allow 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 |
| GetWatcherListResponse | Requestor Server ← Provider Server |

**Table 114. Primitive Directions for GetWatcherList Transaction**

### 15.2.4.1 Primitives

#### 15.2.4.1.1 The "GetWatcherListRequest" Primitive

The `GetWatcherListRequest` primitive allows 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 8.1). |
| History-Period | C | Integer | Indicates the time period in seconds on the longest possible history of the watcher (from the time of the request) that SHOULD be returned. In case of absence, it indicates the user request the watcher list at the time of the request only. The value 0 MUST NOT be used. |
| MaxWatcherList | C | Integer | Indicates the maximum number of Watcher elements in GetWatcherResponse. |

**Table 115. Information elements in GetWatcherRequest Primitive**

#### 15.2.4.1.2 The "GetWatcherListResponse" Primitive

The `GetWatcherListResponse` primitive allows the provider server to return the subscriber list to the requestor server.

| Information Element | Req | Type | Description |
|---|---|---|---|
| Message-Type | M | GetWatcherListResponse | Message identifier |
| Status-Info | M | Structure of Status-Primitive | Status information (see 8.2). |
| History-Period | C | Integer | Indicates the time period in seconds in which the watcher history has been accumulated. This value MUST NOT be larger than the requested period. Absence indicates that the Watcher information element (see below) returns only the current subscribers at the time of the request even if the history log is requested. The value 0 MUST NOT be used. |
| WatcherCount | C | Integer | Indicates the total number of watchers. WatcherCount is only returned if the number of entries is bigger than MaxWatcherList in the GetWatcherListRequest primitive. |
| Watcher | C | Structure | Identifies the watchers and their status from the history period. If this element is not present at all within the response, it indicates that the server does not give any watcher information at this time. The number of this element in GetWatcherListResponse MUST NOT be larger than Max Watcher List value in the corresponding GetWatcherListRequest. |

**Table 116. Information elements in GetWatcherListResponse Primitive**

### 15.2.5 The "GetPresence" Transaction

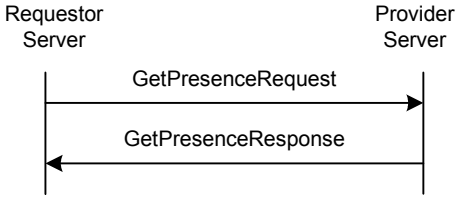


**Figure 58: The "GetPresence" Transaction**

The IMPS system MUST support the "GetPresence" transaction for a UserID list and MAY support the "GetPresence" transaction for a List object. The service tree node that allows negotiation of List addressing is 'Contact List Addr'

The purpose of the `GetPresence` transaction is to allow 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 it will relay the requested PRESENCE INFORMATION.

| Primitive | Direction |
|---|---|
| GetPresenceRequest | Requestor Server → Provider Server |

| | |
|---|---|
| GetPresenceResponse | Requestor Server ← Provider Server |

**Table 117. Primitive Directions for GetPresence Transaction**

#### 15.2.5.1 Primitives

##### 15.2.5.1.1 The "GetPresenceRequest" Primitive

The `GetPresenceRequest` primitive allows 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 8.1). |
| 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 118. Information elements in GetPresenceRequest Primitive**

##### 15.2.5.1.2 The "GetPresenceResponse" Primitive

The `GetPresenceResponse` primitive allows 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 8.2). |
| Presence-Value-List | O | Structure | List of User IDs and corresponding presence values. |

**Table 119. Information elements in GetPresenceResponse Primitive**

## 15.2.6 The "UpdatePresence" Transaction

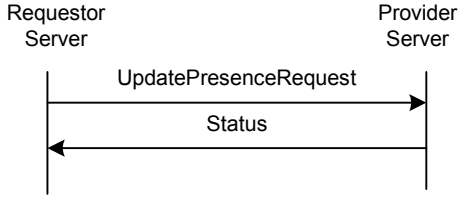


**Figure 59: The "UpdatePresence" Transaction**

The IMPS system MUST support the "UpdatePresence" transaction for a UserID list and MAY support the "UpdatePresence" transaction for a List object. The service tree node that allows negotiation of List addressing is 'Contact List Addr'

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 120. Primitive Directions for UpdatePresence Transaction**

#### 15.2.6.1 Primitives

##### 15.2.6.1.1 The "UpdatePresenceRequest" Primitive

The `UpdatePresenceRequest` primitive allows 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 8.1). |
| Presence-Value-List | M | Structure | A list of presence values to update. |

**Table 121. Information elements in UpdatePresenceRequest Primitive**

### 15.2.7 The "Suspend" Transaction

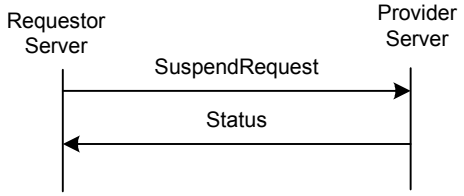


**Figure 60: The "Suspend" Transaction**

The IMPS system MUST support the "Suspend" transaction for a UserID list and MAY support the "Suspend" transaction for a List object. The service tree node that allows negotiation of List addressing is 'Contact List Addr'

The suspension of presence notification to current subscription is accomplished through a Suspend transaction. The notifications are delivered again when a new Subscribed is preformed. The difference of Suspend and Unsubscribe is that the user remains in the watcher list when a suspend is requested but is removed with an Unsubscribe. The provider server MUST return a "Status" message indicating that the provider server has accepted and processed the request.

| Primitive | Direction |
|---|---|
| SuspendRequest | Requestor Server → Provider Server |
| Status | Requestor Server ← Provider Server |

**Table 122. Primitive Directions for Suspend Transaction**

#### 15.2.7.1 Primitives

##### 15.2.7.1.1 The "SuspendRequest" Primitive

The "*SuspendRequest*" primitive is used to suspend presence notifications.

| Information Element | Req | Type | Description |
|---|---|---|---|
| Message-Type | M | SuspendRequest | Message identifier |
| Meta-Information | M | Structure of Meta-Information | The meta-information (see 8.1). |
| User-ID-List | C | Structure | Identifies the IM users whose presence notifications to be suspended. |
| Contact-List-ID-List | C | Structure | Identifies the set of users. |

**Table 123. Information elements in SuspendRequest Primitive**

# 15.3 Status Code

## 15.3.1 "GetPresence" Transaction

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

## 15.3.2 "UpdatePresence" Transaction

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

## 15.3.3 Other Presence Transactions

- Unknown user ID (531)
- Unknown contact list (700)
- Unknown presence attribute (not defined in [PA]) (750)
- Unknown presence value (not defined on the [PA]) (751)
- The maximum number of users in watcher list has been reached for the user (758)

# 16 Service Relay – Instant Messaging Features

## 16.1 Overview

This chapter focuses on the functional relay of IM features. Because of server interoperation, the SSP has its own requirements 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 have been defined in the following two sections.

In order to achieve minimum level of interoperability both the requestor and provider servers MUST support the following functionalities:

- The "SendMessage" Transaction
- The "NotifyDeliveryStatusReport" Transaction

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

Please refer to the CSP document to understand how to relay the IM features from client-server interaction (CSP) to server-server interoperation (SSP) , address routing (client and / or user), and for information on the Message-Info structure.

## 16.2 Transactions

### 16.2.1 The "SendMessage" Transaction


**Figure 61: The "SendMessage" Transaction**

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

Servers MUST support the "SendMessage" transaction. Servers MUST support recpient addressed by User-ID. Servers MAY support recipient addressed by Contact List ID. Servers MAY also support recipient addressed by Group-ID and addressing by screen name. The service tree node that allows negotiation of Group and Contacts addressing is 'Send Message' and sub nodes 'Groups' and 'Contacts'.

The Instant Messaging Service Element depends on the Group Service Element for sending to a group. The GetJoinedMembers (Group features) transaction MUST be used by the IM service element to get the list of the joined members of the group to route a message addressed to a group in case of complementary service.

The requestor server sends a `SendMessageRequest` message to the provider server. The provider server returns a `SendMessageResponse` response containing the result and the message ID.

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

**Table 124. Primitive Directions for SendMessage Transaction**

#### 16.2.1.1 Primitives

##### 16.2.1.1.1 The "SendMessageRequest" Primitive

The `SendMessageRequest` primitive allows 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 8.1). |
| Delivery-Report-Request | M | Boolean | Indicates if the user wants delivery report. |
| Message-Info | M | Structure | Message information data. See [CSP]. |
| Content | C | String or Binary data | The content of the instant message. |

**Table 125. Information elements in SendMessageRequest Primitive**

##### 16.2.1.1.2 The "SendMessageResponse" Primitive

The `SendMessageResponse` primitive allows 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 8.2). |
| Message-ID | C | String | Server generated message id for this message. |

**Table 126. Information elements in SendMessageResponse Primitive**

### 16.2.2 The "ForwardMessage" Transaction

The purpose of "**ForwardMessage**" transaction is to allow the requestor server to forward instant messages through the provider server.

Servers MAY support the "ForwardMessage" transaction. Servers MAY support recipient addressed by User-ID. Servers MAY support recipient addressed by Contact List ID. Servers MAY also support recipient addressed by Group-ID and addressing by screen name. The service tree node that allows negotiation of Group, screen name and Contacts addressing is 'Send Message' and sub nodes 'Groups' and 'Contacts'.

The Instant Messaging Service Element depends on the Group Service Element for forwarding to a group. The GetJoinedMembers (Group features) transaction MUST be used by the IM service element to get the list of the joined members of the group to route a message addressed to a group in case of complementary service.

The requestor server sends a `ForwardMessageRequest` message to the provider server. The provider server returns a `Status` response containing the result.

This transaction belongs to the complementary service.

**Figure 62: The "ForwardMessage" Transaction**

| Primitive | Direction |
|---|---|
| ForwardMessageRequest | Requestor Server → Provider Server |
| ForwardMessageResponse | Requestor Server ← Provider Server |

**Table 127. Primitive Directions for ForwardMessage Transaction**

### 16.2.2.1 Primitives

#### 16.2.2.1.1 The "ForwardMessageRequest" Primitive

The `ForwardMessageRequest` primitive allows 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 8.1). |
| Message-ID | M | String | Identifies the message (either Message-ID or Message-URI). |
| Delivery-Report-Request | M | Boolean | Indicates if the forwarding users wants delivery report. |
| Recipients | M | Structure | Identifies the users to whom the message is forwarded (User-ID-List (with optional Client-ID for each User-ID), Contact-List-ID-List, Screen-Name-List and Group-ID-List) |

**Table 128. Information elements in ForwardMessageRequest Primitive**

| Information Element | Req | Type | Description |
|---|---|---|---|
| Message-Type | M | ForwardMessageResponse | Message identifier |
| Status-Info | M | Structure of Status-Primitive | Status information (see 8.2). |
| Message-ID | C | String | Server generated message id for this message. |

**Table 129. Information elements in ForwardMessageResponse Primitive**

## 16.2.3 The "PushMessage" Transaction

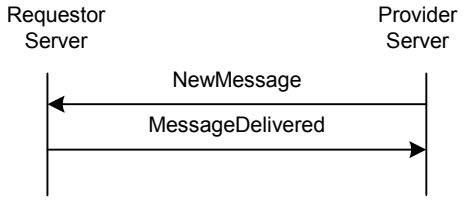


**Figure 63: The "PushMessage" Transaction**

The purpose of "**PushMessage**" transaction is to allow the provider server to deliver the instant messages to users through the requestor server. Servers MAY support the "PushMessage" transaction. The service tree node that allows negotiation of this transaction is 'Push Msg'.

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 130. Primitive Directions for PushMessage Transaction**

### 16.2.3.1 Primitives

#### 16.2.3.1.1 The "NewMessage" Primitive

The `NewMessage` primitive allows 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 8.1). Present if in PushMessage transaction. |
| Status-Info | C | Structure of Status-Primitive | Status information (see 8.2). Present if in GetMessage transaction. |
| Recipient-User-List | M | Structure | Identifies the recipients with a list of User-IDs and optionally Client-IDs. |
| Message-Info | M | Structure | Message information data. See [CSP]. |
| Content | M | String or Binary data | Message data. |

**Table 131. Information elements in NewMessage Primitive**

#### 16.2.3.1.2 The "MessageDelivered" Primitive

The `MessageDelivered` primitive allows 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 8.1). Present if in GetMessage transaction. |
| Status-Info | C | Structure of Status-Primitive | Status information (see 8.2). Present if in PushMessage transaction. |
| Message-ID | M | String | ID of message that has been delivered |

**Table 132. Information elements in MessageDelivered Primitive**

## 16.2.4 The "MessageNotification" Transaction

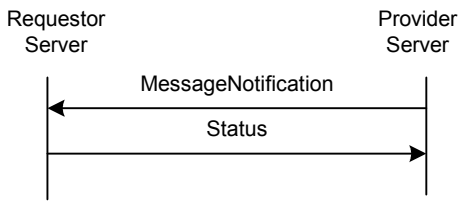


**Figure 64: The "MessageNotification" Transaction**

The purpose of "**MessageNotification**" transaction is to allow the provider server to notify the users of new instant messages through the requestor server. Servers MAY support the "MessageNotification" transaction. The service tree node that allows negotiation of this transaction is 'Notify Msg'.

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 133. Primitive Directions for MessageNotification Transaction**

#### 16.2.4.1 Primitives

##### 16.2.4.1.1 The "MessageNotification" Primitive

The `MessageNotification` primitive allows 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 8.1). |
| Recipient-User-List | M | Structure | Identifies the recipients with a list of User-IDs and optionally Client-IDs. |
| Message-Info | M | Structure | Message information data. See [CSP]. |

**Table 134. Information elements in MessageNotification Primitive**

### 16.2.5 The "GetMessage" Transaction

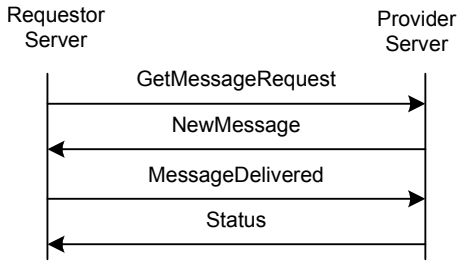


**Figure 65: The "GetMessage" Transaction**

The purpose of the "**GetMessage**" transaction is to allow the requestor server to retrieve a new instant message from the provider server. Servers MAY support the "GetMessage" transaction. The service tree node that allows negotiation of this transaction is 'Get Msg'.

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 135. Primitive Directions for GetMessage Transaction**

#### 16.2.5.1 Primitives

##### 16.2.5.1.1 The "GetMessageRequest" Primitive

The `GetMessageRequest` primitive allows 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 8.1). |
| Message-ID | M | String | ID of message to retrieve |

**Table 136. Information elements in GetMessageRequest Primitive**

### 16.2.6 The "SetMessageDeliveryMethod" Transaction

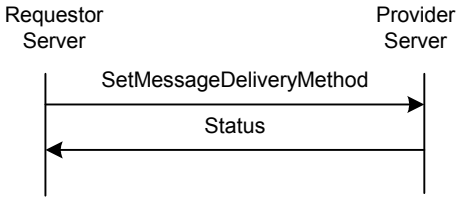


**Figure 66: The "SetMessageDeliveryMethod" Transaction**

The purpose of the "**SetMessageDeliveryMethod**" transaction is to allow the user in the requestor server to set the appropriate message delivery method from the provider server. Servers MAY support the "SetMessageDeliveryMethod" transaction. The service tree node that allows negotiation of this transaction is 'Delivery Mtd'.

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 137. Primitive Directions for SetMessageDeliveryMethod Transaction**

#### 16.2.6.1 Primitives

##### 16.2.6.1.1 The "SetMessageDeliveryMethod" Primitive

The `SetMessageDeliveryMethod` primitive allows user in the requestor server to set the instant message delivery method.

| Information Element | Req | Type | Description |
|---|---|---|---|
| Message-Type | M | SetMessageDelivery Method | Message identifier |
| Meta-Information | M | Structure of Meta-Information | The meta-information (see 8.1). |
| 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. |
| Group-ID | O | String | Group ID if Delivery method refers to a group. |
| Group-Content-Limit | O | Integer | Maximum size of message that can be sent from the specified group. It MUST NOT be indicated when Group-ID is missing. |

**Table 138. Information elements in SetMessageDeliveryMethod Primitive**

## 16.2.7 The "GetMessageList" Transaction


**Figure 67: The "GetMessageList" Transaction**

The purpose of the "**GetMessageList**" transaction is to allow 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. Servers MAY support the "GetMessageList" transaction for either Group history requests or for undelivered instant messages. The service tree node that allows negotiation of this transaction without group functionality is 'Get msg list'. The service tree node that allows negotiation of undelivered instant messages is 'Msg List'.

Servers MAY support Group History caching. If Group History caching is supported then the Server MUST support it as a part of the IM service. The service tree node that allows negotiation of chat history functionality is 'Group History'.

The GetJoinedMembers (Group features) transaction MUST be used by the IM service element to check the validity of the GetMessageList requester user to receive the history in case of complementary service.

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 139. Primitive Directions for GetMessageList Transaction**

### 16.2.7.1 Primitives

#### 16.2.7.1.1 The "GetMessageListRequest" Primitive

If the provider server offers a space where messages are stored, the user MAY retrieve the undelivered message list or group history list. The `GetMessageListRequest` primitive allows the requestor server to get the stored Message-ID's or Message-URI's so that they MAY be used in GetMessage or RejectMessage transactions. If "Group-ID" is present, the user MUST have the group history list. Otherwise, the user MUST 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 8.1). |

| | | | |
|---|---|---|---|
| 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 140. Information elements in GetMessageListRequest Primitive**

#### 16.2.7.1.2 The "GetMessageListResponse" Primitive

The `GetMessageListResponse` primitive allows the provider server to return a list of message information.

| Information Element | Req | Type | Description |
|---|---|---|---|
| Message-Type | M | GetMessageListResponse | Message identifier. |
| Status-Info | M | Structure of Status-Primitive | Status information (see 8.2). |
| Message-Info-List | M | Structure | Message information data for each message. See [CSP]. |
| Message-Total-Count | O | Integer | When the Message-Count was in the request but there are more instant messages waiting than the amount requested in Message-Count, this value MUST indicate the total number of instant messages. |

**Table 141. Information elements in GetMessageListResponse Primitive**

## 16.2.8 The "RejectMessage" Transaction

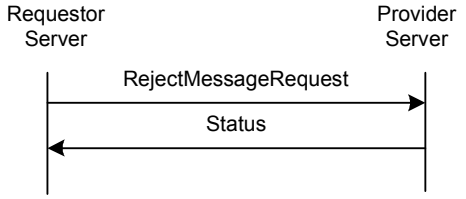


**Figure 68: The "RejectMessage" Transaction**

This transaction belongs to the complementary service. Servers MAY support the "RejectMessage" transaction. The service tree node that allows negotiation of this transaction is 'Reject msg'.

| Primitive | Direction |
|---|---|
| RejectMessageRequest | Requestor Server → Provider Server |
| Status | Requestor Server ← Provider Server |

**Table 142. Primitive Directions for RejectMessage Transaction**

### 16.2.8.1 Primitives

#### 16.2.8.1.1 The "RejectMessageRequest" Primitive

The `RejectMessageRequest` primitive allows 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 8.1). |

| | | | |
|---|---|---|---|
| Message-ID-List | M | Structure | Identifies the messages to be removed using a list of Message-ID(s). |

**Table 143. Information elements in RejectMessageRequest Primitive**

## 16.2.9 The "NotifyDeliveryStatusReport" Transaction

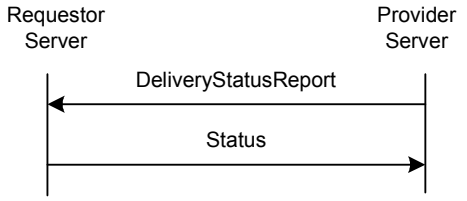


**Figure 69: The "NotifyDeliveryStatusReport" Transaction**

Servers MUST support the "NotifyDeliveryStatusReport" transaction.

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

**Table 144. Primitive Directions for NotifyDeliveryStatusReport Transaction**

### 16.2.9.1 Primitives

#### 16.2.9.1.1 The "DeliveryStatusReport" Primitive

The `DeliveryStatusReport` primitive allows the provider server to give the sender the instant message delivery status report. The delivery report MAY 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 8.1). |
| Delivery-Result | M | Structure of Status-Primitive | The delivery result shares the same structure as Status (see 8.2). |
| Delivery-Time | O | DateTime | Date and time of delivery |
| Message-Info | M | Structure | Message information data. See [CSP]. |

**Table 145. Information elements in DeliveryStatusReport Primitive**

## 16.2.10 The "BlockEntity" Transaction

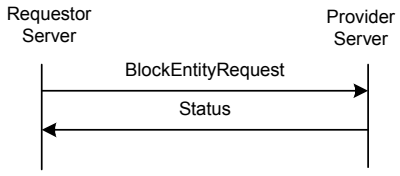


**Figure 70: The "BlockUEntity" Transaction**

A user may block/ grant messsage or invitation delivery from certain entities. These entities MAY be:

- user(s) specified by User-ID(s) or Screen-Name(s),
- group(s) specified by Group-ID(s),
- contact list(s) specified by ContactList-ID(s),
- application(s) specified by Application-ID(s).

Servers MAY support the "BlockEntity" transaction. The service tree node that allows negotiation of this transaction is 'Blocking'.

Including/removing a contact list in/from the grant or block list affects all the users of the contact list(s) in the grant or block list. The IM Service Element MUST use up to date contents of the contact list(s) included in the grant or block list when it executes the grant/block decision procedure.

The requestor server sends a `BlockEntityRequest` request to the provider server containing the list of entities to be blocked / granted . The provider server returns a `Status` response.

This transaction belongs to the complementary service.

| Primitive | Direction |
|---|---|
| BlockEntityRequest | Requestor Server → Provider Server |
| Status | Requestor Server ← Provider Server |

**Table 146. Primitive Directions for BlocEntity Transaction**

## 16.2.10.1 Primitives

### 16.2.10.1.1 The "BlockEntityRequest" Primitive

The `BlockEntityRequest` primitive allows the blocking of groups, contact lists or users. None of the message or invitations from the blocked entity will be delivered to the blocking user.

| Information Element | Req | Type | Description |
|---|---|---|---|
| Message-Type | M | BlockEntityRequest | Message identifier |
| Meta-Information | M | Structure of Meta-Information | The meta-information (see 8.1). |
| Block-Entity-List | O | Structure | A list of entities to be added to the block list. |
| Unblock-Entity-List | O | Structure | A list of entities to be removed from the block list. |
| Block-List-Status | M | Boolean | Indicates if the list of blocked entities is currently in use (active). |
| Grant-Entity-List | O | Structure | The list of entities to be added to the grant list. |
| Ungrant-Entity-List | O | Structure | The list of entities to be removed from the grant list. |
| Grant-List-Status | M | Boolean | Indicates if the list of granted entities is currently in use (active). |

**Table 147. Information elements in BlockUserRequest Primitive**

## 16.2.11 The "GetBlockedList" Transaction

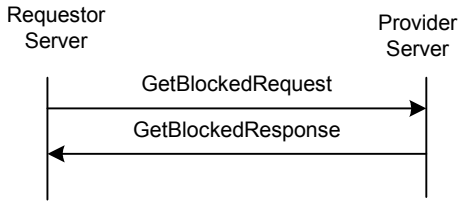


**Figure 71: The "GetBlockedList" Transaction**

A user may get its own list of blocked users, groups, contact lists or applications at any suitable time. The purpose of the "**GetBlockedList**" transaction is to allow the blocking user in the requestor server to get its own list of blocked entities and granted entities. Servers MAY support the "GetBlockedList" transaction. The service tree node that allows negotiation of this transaction is 'Blocking'.

The requestor server sends a `GetBlockedRequest` request to the provider server. The provider server returns a `GetBlockedResponse` response containing the list of blocked entities.

This transaction belongs to the complementary service.

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

**Table 148. Primitive Directions for GetBlockedList Transaction**

### 16.2.11.1 Primitives

#### 16.2.11.1.1 The "GetBlockedRequest" Primitive

The `GetBlockedRequest` primitive allows the blocking user in the requestor server to get its own list of blocked and granted entities, 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 8.1). |

**Table 149. Information elements in GetBlockedRequest Primitive**

#### 16.2.11.1.2 The "GetBlockedResponse" Primitive

The `GetBlockedResponse` primitive allows the provider server to return a list of blocked entities 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 8.2). |
| Block-Entity-List | C | Structure | The list of currently blocked entities. |
| Block-List-Status | M | Boolean | If the list of blocked entities is currently in use (active). |
| Grant-Entity-List | C | Structure | The list of currently granted entities. |
| Grant-List-Status | M | Boolean | If the list of granted entities is currently in use (active). |

**Table 150. Information elements in GetBlockedResponse Primitive**

## 16.2.12 The "IsUserBlocked" Transaction

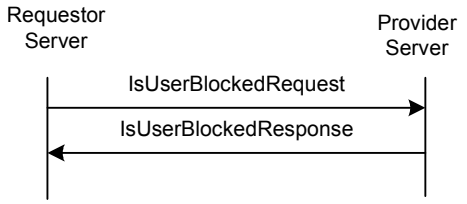


**Figure 72: The "IsUserBlocked" Transaction**

The purpose of the "**IsUserBlocked**" transaction is to allow the requestor server to verify whether a user is blocking a specific user from sending IMs/Invitations. Servers MAY support the "IsUserBlocked" transaction. The service tree node that allows negotiation of this transaction is 'IsBlocked'.

The requestor server sends an `IsUserBlockedRequest` request to the provider server containing the UserID of "possibly" blocking user and the UserID of the "possibly" blocked user. The provider server returns a `IsUserBlockedResponse` response containing a Boolean indicating the blocked state.

| Primitive | Direction |
|---|---|
| IsUserBlockedRequest | Requestor Server → Provider Server |
| IsUserBlockedResponse | Requestor Server ← Provider Server |

**Table 151. Primitive Directions for IsUserBlocked Transaction**

### 16.2.12.1 Primitives

#### 16.2.12.1.1 The "IsUserBlockedRequest" Primitive

The `IsUserBlockedRequest` primitive allows the requestor server to verify if a certain user on the providing server has blocked a specific user.

| Information Element | Req | Type | Description |
|---|---|---|---|
| Message-Type | M | IsUserBlockedRequest | Message identifier |
| Meta-Information | M | Structure of Meta-Information | The meta-information (see 8.1). |
| Blocking-User | M | String | The User-ID of the user who is possibly blocking the user |
| Blocked-User | M | String | The User-ID of the user who is possibly blocked by the Blocking-User |

**Table 152. Information elements in IsUserBlockedRequest Primitive**

#### 16.2.12.1.2 The "IsUserBlockedResponse" Primitive

The `IsUserBlockedResponse` primitive allows the provider server to return the result of the IsUserBlockedRequest. The Response contains a Boolean value. If the value is "T", the Blocking-User has blocked the Blocked-User. If the value is "F", the Blocking-User has not blocked the Blocked-User

| Information Element | Req | Type | Description |
|---|---|---|---|
| Message-Type | M | IsUserBlockedResponse | Message identifier. |
| Status-Info | M | Structure of Status- | The status information (see 8.2). |

| | | | |
|---|---|---|---|
| | | Primitive | |
| Blocked-State | M | Boolean | Boolean value indicating the blocked state |

**Table 153. Information elements in IsUserBlockedResponse Primitive**

## 16.2.13 The "Extend one-to-one IM conversation" Transaction

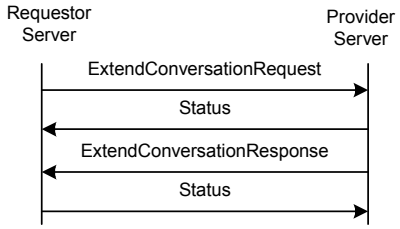


**Figure 73: The "ExtendConversation" Transaction**

The purpose of "ExtendConversation" transaction is to allow the requestor server to invite other users to an IM conversation through the provider server. Servers MUST support the "ExtendConveration" transaction.

The Instant Messaging Service Element depends on the Group Service Element for extending an IM conversation. The CreateGroup (Group features) transaction MUST be used by the IM service element to create a public open group.

The requestor server sends an ExtendConversationRequest message to the provider server. The provider server returns a Status response containing the result. The ExtendConversationRequest MUST contain a list of User IDs of the contacts that are to be invited to the ongoing conversation. The ExtendConversationRequest also includes the ExtendConversation-User-ID, which identifies the user that the originating client is already having a conversation with.

Following the proceedures specified in the CSP Sessions and Transasctions Document [CSP] the provider server will send a ExtendConversationResponse notification to the requestor server. The ExtendConversationResponse MUST include the Group ID of the created group. The requestor server MUST return a Status response.

Please refer to CSP Sessions and Transactions Document [CSP] for the "ExtendConversation" transaction details.

| Primitive | Direction |
|---|---|
| ExtendConversationRequest | Requestor Server → Provider Server |
| Status | Requestor Server ← Provider Server |
| ExtendConversationResponse | Requestor Server ← Provider Server |
| Status | Requestor Server → Provider Server |

**Table 154. Primitive Directions for ExtendConversation Transaction**

### 16.2.13.1 Primitives

#### 16.2.13.1.1 The "ExtendConversationRequest" Primitive

The `ExtendConversationRequest` primitive allows the requestor server to invite other users to an IM conversation through the provider server.

| Information Element | Req | Type | Description |
|---|---|---|---|
| Message-Type | M | ExtendConversationRequest | Message identifier |
| Meta-Information | M | Structure of Meta-Information | The meta-information (see 8.1). |
| ExtendConversation-ID | M | String | Identifier for the Extended |

| | | | |
|---|---|---|---|
| | | | Conversation |
| User-ID-List | M | Structure | A list of User-Ids, which identifies the new users who are to be invited to the existing conversation |
| ExtendConversation-User | M | Structure | Identifies the user who is already in the conversation. |
| Subscribe-Notif | M | Boolean | A flag indicating that the client wants to activate the group change notifications while joining the group. |
| WelcomeNote | O | String | A welcome message for the group. |
| Own-Screen-Name | M | Structure | The user's screen name from whom the Extended Conversation request originates. |

**Table 155. Information elements in ExtendConversationRequest Primitive**

#### 16.2.13.1.2 The "ExtendConversationResponse" Primitive

The `ExtendConversationResponse` primitive allows the provider server to notify the requestor server about the status of the IM conversation extension.

| Information Element | Req | Type | Description |
|---|---|---|---|
| Message-Type | M | ExtendConversationResponse | Message identifier |
| Meta-Information | M | Structure of Meta-Information | The meta-information (see 8.1). |
| ExtendConversation-ID | M | String | Identifier for the Extended Conversation |
| Group-ID | M | String | Identifies the group that has been created. |
| Response-Note | C | String | Indicates the reason why Extended Conversation has been rejected or accepted by the other party |
| Result | M | Structure | Result of the request. |

**Table 156. Information elements in ExtendConversationResponse Primitive**

# 16.3 Status Code

## 16.3.1 "SendMessage" Transaction

- `Unknown content-type (415)`
- `Invalid User-ID (427)`
- `Invalid Client-ID (428)`
- `Invalid Application-ID (451)`
- `Forbidden Application-ID (452)`
- `Message queue full (507)`
- `Unsupported message context (508)`
- `Related client capabilities are missing (510)`
- `Recipient user does not exist (531)`

- Recipient user blocked the sender (532)
- Recipient user is not logged in (533)
- Message has been rejected due to limitations (540)
- Contact list does not exist (700)
- Contact list is empty (703)
- Recipient group does not exist (800)
- Sender has not joined the group (or kicked) (808)
- Private messaging is disabled in the group (812)
- Private messaging is disabled for the recipient (813)
- Domain not supported (516)

### 16.3.2 “SetMessageDeliveryMethod” Transaction

- Group does not exist (800)

### 16.3.3 “GetMessageList” Transaction

- Group does not exist (800)
- Group is not joined (808)
- History is not supported (821)
- Related services are missing (509)
- Message-Count exceeded (539)
- User has been rejected (809)
- Not a group member (810)
- There are no instant messages (908)

### 16.3.4 “RejectMessage” Transaction

- Invalid Message-ID (426)

### 16.3.5 “NewMessage” Transaction

- Invalid Message-ID (426)
- Client will not accept the message delivery (410)
- Client does not support the content type (415)

### 16.3.6 “GetMessage” Transaction

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

### 16.3.7 “NotifyDeliveryStatusReport” Transaction

- Unsupported content-type (415)
- Invalid User-ID (427)

- Invalid Client-ID (428)
- Invalid Application-ID (451)
- Forbidden Application-ID (452)
- Message queue full (507)
- Unsupported message context (508)
- Related client capabilities are missing (510)
- Domain not supported (516)
- Recipient user does not exist (531)
- Recipient user blocked the sender (532)
- Recipient user is not logged in (533)
- Contact list does not exist (700)
- Contact list is empty (703)
- Recipient group does not exist (800)
- Sender has not joined the group (or kicked) (808)
- Private messaging is disabled in the group (812)
- Private messaging is disabled for the recipient (813)

### 16.3.8 “ForwardMessage” Transaction

- Message queue full (507)
- Recipient user does not exist (531)
- Recipient user blocked the sender (532)
- Recipient user is not logged in (533)
- Contact list does not exist (700)
- Contact list is empty (703)
- Recipient group does not exist (800)
- Sender has not joined the group (or kicked) (808)
- Private messaging is disabled in the group (812)
- Private messaging is disabled for the recipient (813)
- Invalid Message-ID (426)
- Unsupported content-type (415)
- Domain not supported (516)

### 16.3.9 Block Transactions

- Unknown user ID (531)
- Unknown group-ID (800)
- Wildcard characters not allowed (547)

- Wildcard expression is too complicated (548)
- The maximum number of users in grant list has been reached for the user (756)
- The maximum number of users in block list has been reached for the user (757)

### 16.3.10 "IsUserBlocked" Transaction

- Unknown user ID (531)

### 16.3.11 "ExtendConversation" Transaction

- The maximum number of groups has been reached (server-limit) (815)
- Delivery to recipient not available (410)
- Extend conversation rejected (825)
- Recipient does not support the requested functionality (909)

# 17 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.

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

- The “JoinGroup” Transaction
- The “LeaveGroup” Transaction
- The “ServerInitiatedLeaveGroup” Transaction
- The “GetGroupProps” Transaction
- The “SetGroupProps” Transaction
- The “SubscribeGroupChange” Transaction
- The “UnsubscribeGroupChange” Transaction
- The “GetGroupSubStatus” Transaction
- The “NotifyGroupChange” Transaction

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

The Group Service Element MUST use the IM Service Element to send/receive and store messages to chat groups.

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).

## 17.1 Transactions

### 17.1.1 The “CreateGroup” Transaction

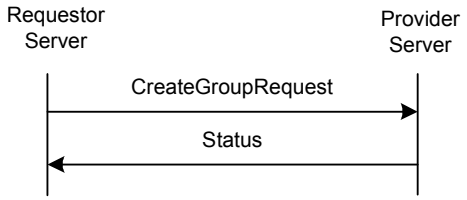


**Figure 74: The “CreateGroup” Transaction**

A user may create its own private group at any suitable time. The purpose of “**CreateGroup**” transaction is to allow the user in the requestor server to create the user’s own private group. Servers MAY support the “CreateGroup” transaction. The service tree node that allows negotiation of this transaction is ‘Group Mgmt’.

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 157. Primitive Directions for CreateGroup Transaction**

#### 17.1.1.1 Primitives

##### 17.1.1.1.1 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, the initial properties of the group, the user's intention of joining to the created group, getting the group change notifications and optionally to define the screen name as well. 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 8.1). |
| Group-ID | M | String | Identifies the group |
| Group-Props | M | Structure | The properties of the group. |
| Own-Props | M | Structure | The list of the users' own group properties with the corresponding values. |
| Join-Group | M | Boolean | A flag indicating that the user creating the group joins the group at the same time. |
| Screen-Name | O | Structure | Screen name of the user in the group. |
| Subscribe-Notif | M | Boolean | A flag indicating that the user wants to activate the group change notifications while joining the group. |

**Table 158. Information elements in CreateGroupRequest Primitive**

### 17.1.2 The "DeleteGroup" Transaction

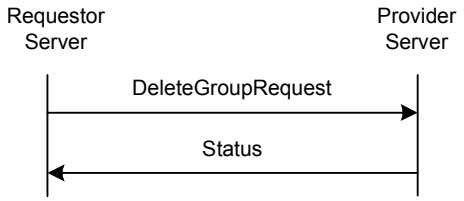


**Figure 75: The "DeleteGroup" Transaction**

A user with sufficient access rights may delete a private user group at any suitable time. Servers MAY support the "DeleteGroup" transaction. Servers MAY support the "CreateGroup" transaction. The service tree node that allows negotiation of this transaction is 'Group Mgmt'.

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 159. Primitive Directions for DeleteGroup Transaction**

#### 17.1.2.1 Primitives

##### 17.1.2.1.1 The "DeleteGroupRequest" Primitive

The `DeleteGroupRequest` primitive allows 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 8.1). |
| Group-ID | M | String | Identifies the group |

**Table 160. Information elements in DeleteGroupRequest Primitive**

### 17.1.3 The "JoinGroup" Transaction

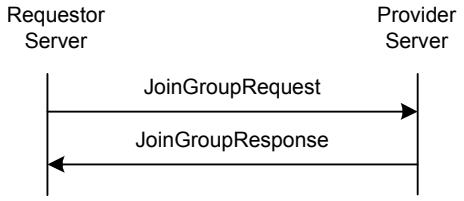


**Figure 76: The "JoinGroup" Transaction**

A user may join a discussion group at any suitable time. Servers MUST support the "JoinGroup" transaction.

The requestor server sends a `JoinGroupRequest` request to the provider server with the Group-ID, its screen name shown during the discussion, the user's age according to his public profile 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.

If the JoinGroupRequest primitive contains the ScreenName for the user, the server MUST apply the requested ScreenName or, if the ScreenName is not unique in the relevant group, generate a unique ScreenName. If the ScreenName was missing from the request, the server MAY generate a screen name. The algorithm to generate a ScreenName is an implementation issue, however the newly generated ScreenName MUST be unique within the group. The user MUST be informed about the ScreenName if created by the server.

If the group's MinimumAge property is higher than zero, the provider server MUST verify this value versus the joining user's age, and:

- when the age field is missing from the request, the provider server MUST allow the user to join – providing that the rest of the conditions are met.
- when the age field is present in request, and:
  - this age is smaller than the MinimumAge value, the provider server MUST NOT allow the user to join the group.

- this age is equal or higher than the MinimumAge value, the provider server MUST allow the user to join the group– providing that the rest of the conditions are met.

After a user successfully joins 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 161. Primitive Directions for JoinGroup Transaction**

## 17.1.3.1 Primitives

### 17.1.3.1.1 The "JoinGroupRequest" Primitive

The `JoinGroupRequest` primitive allows 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, the joined users' list request and the user's intention of getting the group change notifications.

| Information Element | Req | Type | Description |
|---|---|---|---|
| Message-Type | M | JoinGroupRequest | Message identifier |
| Meta-Information | M | Structure of Meta-Information | The meta-information (see 8.1). |
| Group-ID | M | String | Identifies the group |
| Joined-Request | M | Boolean | Indicates if the user wants the list of currently joined users or not. |
| Screen-Name | O | String | Screen name of the user in the group. |
| Subscribe-Notif | M | Boolean | A flag indicating that the user wants to activate the group change notifications while joining the group. |
| Own-Prop-List | O | Structure | The list of the user's properties in that group. |
| Joined-Users-Age | O | Integer | The age of the joined user according to his public profile. |

**Table 162. Information elements in JoinGroupResquest Primitive**

### 17.1.3.1.2 The "JoinGroupResponse" Primitive

The `JoinGroupResponse` primitive allows 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 8.2). |
| Joined-User-Screen-Name-List | C | Structure | The list of currently joined users identified by their Screen-Name's. Present if it was requested. |
| Screen-Name | C | String | A unique screen name created by the server. Present if the Screen Name sent in the JoinGroupRequest was not unique. |
| Welcome-Text | O | Structure | A short text to be shown to the user when he/she has joined the group. |
| Joined-Blocked-Users- | O | Structure | The list of the users who are currently |

| | | | |
|---|---|---|---|
| List | | | joined in the group and who are blocked by the requesting user (Screen Name, User-ID). |

**Table 163. Information elements in JoinGroupResponse Primitive**

## 17.1.4 The "LeaveGroup" Transaction

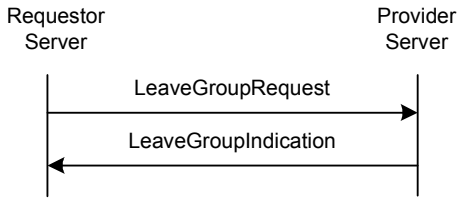


**Figure 77: The "LeaveGroup" Transaction**

A user may leave a discussion group at any suitable time. Servers MUST support the "LeaveGroup" transaction.

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 164. Primitive Directions for LeaveGroup Transaction**

### 17.1.4.1 Primitives

#### 17.1.4.1.1 The "LeaveGroupRequest" Primitive

The `LeaveGroupRequest` primitive allows 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 8.1). |
| Group-ID | M | String | Identifies the group |

**Table 165. Information elements in LeaveGroupRequest Primitive**

#### 17.1.4.1.2 The "LeaveGroupIndication" Primitive

The `LeaveGroupIndication` primitive allows the provider server to return the group leaving result requested from the requestor server. The `LeaveGroupIndication` 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 8.1). Present if in ServerInitiatedLeaveGroup transaction |
| Status-Info | C | Structure of Status-Primitive | Status information (see 8.2). 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 166. Information elements in LeaveGroupIndication Primitive**

## 17.1.5 The "ServerInitiatedLeaveGroup" Transaction

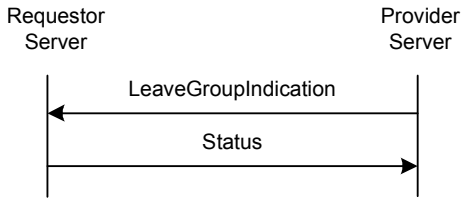


**Figure 78: The "ServerInitiatedLeaveGroup" Transaction**

A server may initiate a group leaving due to user kickout, group deletion etc. Servers MUST support the "ServerInitiatedLeaveGroup" transaction.

The provider server sends a `LeaveGroupIndication` request to the requestor server.

| Primitive | Direction |
|---|---|
| LeaveGroupIndication | Requestor Server ← Provider Server |
| Status | Requestor Server → Provider Server |

**Table 167. Primitive Directions for ServerInitiatedLeaveGroup Transaction**

### 17.1.5.1 Primitives

See Section 17.1.4.1.2 for a description of the LeaveGroupIndication primitive.

## 17.1.6 The "GetGroupMember" Transaction

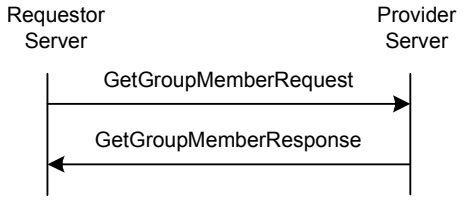


**Figure 79: The "GetGroupMember" Transaction**

A user with sufficient access rights may retrieve the member list of a group. Servers MAY support the "GetGroupMember". Servers MAY support the "CreateGroup" transaction. The service tree node that allows negotiation of this transaction is 'Member mgmt'.

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 168. Primitive Directions for GetGroupMember Transaction**

#### 17.1.6.1 Primitives

##### 17.1.6.1.1 The "GetGroupMemberRequest" Primitive

The `GetGroupMemberRequest` primitive allows 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 8.1). |
| Group-ID | M | String | Identifies the group |

**Table 169. Information elements in GetGroupMemberRequest Primitive**

##### 17.1.6.1.2 The "GetGroupMemberResponse" Primitive

The `GetGroupMemberResponse` primitive allows 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 8.2). |
| 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 170. Information elements in GetGroupMemberResponse Primitive**

### 17.1.7 The "AddGroupMember" Transaction

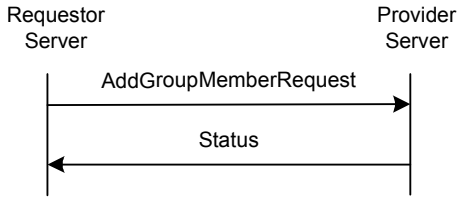


**Figure 80: The "AddGroupMember" Transaction**

A user with sufficient access rights may add user(s) to the member list of a group. Servers MAY support the "AddGroupMember" transaction. Servers MAY support the "CreateGroup" transaction. The service tree node that allows negotiation of this transaction is 'Member mgmt'.

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 171. Primitive Directions for AddGroupMember Transaction**

#### 17.1.7.1 Primitives

##### 17.1.7.1.1 The "AddGroupMemberRequest" Primitive

The `AddGroupMemberRequest` primitive allows 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 8.1). |
| Group-ID | M | String | Identifies the group |
| User-ID-List | O | Structure | The list of users to be added. |

**Table 172. Information elements in AddGroupMemberRequest Primitive**

### 17.1.8 The "RemoveGroupMember" Transaction

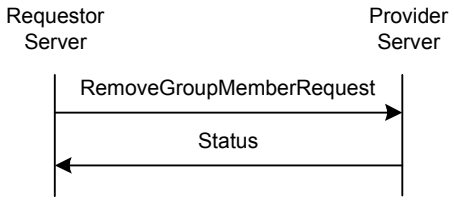


**Figure 81: The "RemoveGroupMember" Transaction**

A user with sufficient access rights may remove user(s) from the member list of a group. Servers MAY support the "RemoveGroupMember" transaction. Servers MAY support the "CreateGroup" transaction. The service tree node that allows negotiation of this transaction is 'Member mgmt'.

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 173. Primitive Directions for RemoveGroupMember Transaction**

#### 17.1.8.1 Primitives

##### 17.1.8.1.1 The "RemoveGroupMemberRequest" Primitive

The `RemoveGroupMemberRequest` primitive allows the user with sufficient access rights in the requestor server to remove 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 (8.1). |
| Group-ID | M | String | Identifies the group |
| User-ID-List | M | Structure | A list of removed users. |

**Table 174. Information elements in RemoveGroupMemberRequest Primitive**

## 17.1.9 The "MemberAccess" Transaction

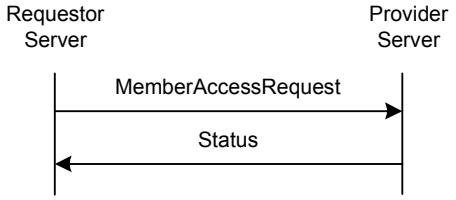


**Figure 82: The "MemberAccess" Transaction**

This transaction belongs to the complementary service. Servers MAY support the "MemberAccess" transaction. Servers MAY support the "CreateGroup" transaction. The service tree node that allows negotiation of this transaction is 'Member mgmt'.

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

**Table 175. Primitive Directions for MemberAccess Transaction**

### 17.1.9.1 Primitives

#### 17.1.9.1.1 The "MemberAccessRequest" Primitive

The `MemberAccessRequest` primitive allows 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 (8.1). |
| 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 176. Information elements in MemberAccessRequest Primitive**

## 17.1.10 The "GetJoinedUsers" Transaction

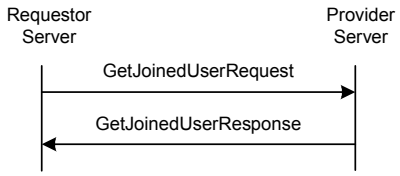


**Figure 83: The "GetJoinedUsers" Transaction**

This transaction belongs to the complementary service. Servers MAY support the "GetJoinedUsers" transaction. The service tree node that allows negotiation of this transaction is 'Member mgmt'.

Please refer to CSP Sessions and Transactions Document [CSP] for the get joined user transaction details.

| Primitive | Direction |
|---|---|
| GetJoinedUsersRequest | Requestor Server → Provider Server |
| GetJoinedUsersResponse | Requestor Server ← Provider Server |

**Table 177. Primitive Directions for GetJoinedUsers Transaction**

### 17.1.10.1 Primitives

#### 17.1.10.1.1 The "GetJoinedUsersRequest" Primitive

The `GetJoinedUsersRequest` primitive allows the requestor server to retrieve the joined user list of a group.

| Information Element | Req | Type | Description |
|---|---|---|---|
| Message-Type | M | GetJoinedUsersRequest | Message identifier |
| Meta-Information | M | Structure of Meta-Information | The meta-information (see 8.1). |
| Group-ID | M | String | Identifies the group |

**Table 178. Information elements in GetJoinedUsersRequest Primitive**

#### 17.1.10.1.2 The "GetJoinedUsersResponse" Primitive

The `GetJoinedUsersResponse` primitive allows the provider server to return the result with a list of joined group users.

| Information Element | Req | Type | Description |
|---|---|---|---|
| Message-Type | M | GetJoinedUsersResponse | Message identifier |
| Status-Info | M | Structure of Status-Primitive | Status information (see 8.2). |
| Admin-Map-List | C | Structure | Administrators and moderators receive this list. |
| User-Map-List | C | Structure | Ordinary users receive this list. |
| Joined-Blocked-Users-List | O | Structure | The list of the users who are currently joined in the group and who are blocked by the requesting user (Screen Name, User-ID). |

**Table 179. Information elements in GetJoinedUsersResponse Primitive**

## 17.1.11 The "GetGroupProps" Transaction

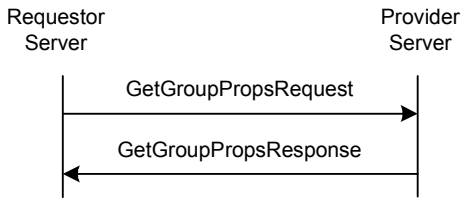


**Figure 84: The "GetGroupProps" Transaction**

A user with sufficient access rights may retrieve the properties of a group, and it's the user's own properties in that particular group. Servers MUST support the "GetGroupProps" transaction.

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 the user's own properties for the specified group.

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

**Table 180. Primitive Directions for GetGroupProps Transaction**

### 17.1.11.1 Primitives

#### 17.1.11.1.1 The "GetGroupPropsRequest" Primitive

The `GetGropPropsRequest` primitive allows 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 `GetGropPropsRequest` 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 8.1). |
| Group-ID | M | String | Identifies the group |

**Table 181. Information elements in GetGroupPropsRequest Primitive**

#### 17.1.11.1.2 The "GetGroupPropsResponse" Primitive

The `GetGroupPropsResponse` primitive allows the provider server to return the result with a list of group properties and its 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 8.2). |
| Group-Prop-List | M | Structure | The list of group properties. |
| Own-Prop-List | M | Structure | The list of the user's properties in that group. |

**Table 182. Information elements in GetGroupPropsResponse Primitive**

## 17.1.12 The "SetGroupProps" Transaction

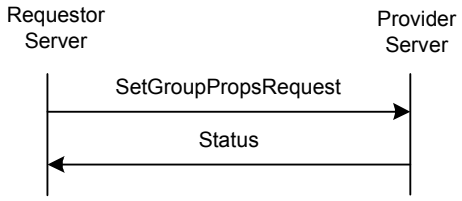


**Figure 85: The "SetGroupProps" Transaction**

A user with sufficient access rights may update the properties of a group, and/or it's the user's own properties in that particular group. Servers MUST support the "SetGroupProps" transaction.

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 183. Primitive Directions for SetGroupProps Transaction**

### 17.1.12.1 Primitives

#### 17.1.12.1.1 The "SetGroupPropsRequest" Primitive

The `SetGroupPropsRequest` primitive allows 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-Information | The meta-information (see 8.1). |
| 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 184. Information elements in SetGroupPropsRequest Primitive**

## 17.1.13 The "RejectList" Transaction

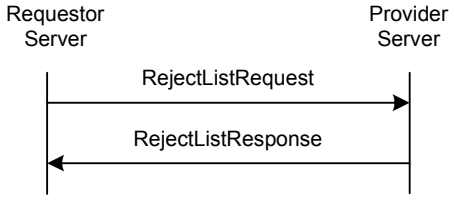


**Figure 86: The "RejectList" Transaction**

This transaction belongs to the complementary service. Servers MAY support the "RejectList" transaction. The service tree node that allows negotiation of this transaction is 'Reject list'.

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

**Table 185. Primitive Directions for RejectList Transaction**

### 17.1.13.1 Primitives

#### 17.1.13.1.1 The "RejectListRequest" Primitive

The `RejectListRequest` primitive allows 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 8.1). |
| 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 186. Information elements in RejectListRequest Primitive**

### 17.1.13.2 The "RejectListResponse" Primitive

The `RejectListResponse` primitive allows 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 8.2). |
| Reject-User-ID-List | O | Structure | A list of users in the reject list. |

**Table 187. Information elements in RejectListResponse Primitive**

## 17.1.14 The "SubscribeGroupChange" Transaction

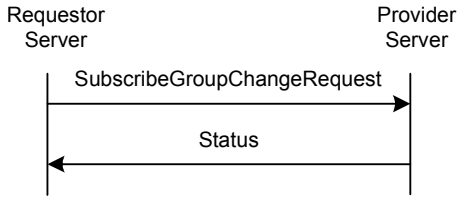


**Figure 87: The "SubscribeGroupChange" Transaction**

A user may subscribe to a group change notice whenever another user leaves or joins the group, or the group properties have been changed. Servers MUST support the "SubscribeGroupChange" transaction.

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 188. Primitive Directions for SubscribeGroupChange Transaction**

### 17.1.14.1 Primitives

#### 17.1.14.1.1 The "SubscribeGroupChangeRequest" Primitive

The `SubscribeGroupChangeRequest` primitive allows the user in the requestor server to subscribe to a group change notice whenever another 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 8.1). |
| Group-ID | M | String | Identifies the group |

**Table 189. Information elements in SubscribeGroupChangeRequest Primitive**

## 17.1.15 The "UnsubscribeGroupChange" Transaction

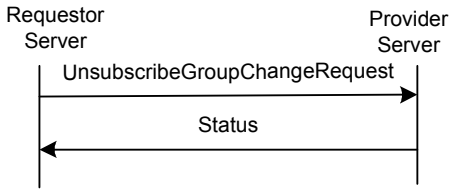


**Figure 88: The "UnsubscribeGroupChange" Transaction**

A user may cancel the subscription to the group change notice. Servers MUST support the "UnsubscribeGroupChange" transaction.

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 190. Primitive Directions for UnsubscribeGroupChange Transaction**

### 17.1.15.1 Primitives

#### 17.1.15.1.1 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 8.1). |
| Group-ID | M | String | Identifies the group |

**Table 191. Information elements in UnsubscribeGroupChangeRequest Primitive**

## 17.1.16 The "GetGroupSubStatus" Transaction

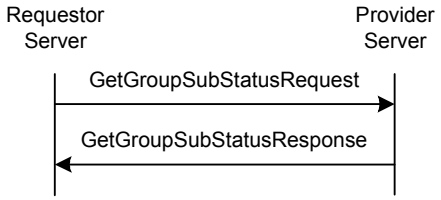


**Figure 89: The "GetGroupSubStatus" Transaction**

A user may retrieve its subscription status to a group change notice. Servers MUST support the "GetGroupSubStatus" transaction.

The requestor server sends a `GetGroupSubStatusRequest` request to the provider server with the Group-ID. The provider server returns a `GetGroupSubStatusResponse` response with the user's current subscription status to a group change notice.

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

**Table 192. Primitive Directions for GetGroupSubStatus Transaction**

### 17.1.16.1 Primitives

#### 17.1.16.1.1 The "GetGroupSubStatusRequest" Primitive

The `GetGroupSubStatusRequest` primitive allows 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 8.1). |
| Group-ID | M | String | Identifies the group |

**Table 193. Information elements in GetGroupSubStatusRequest Primitive**

#### 17.1.16.1.2 The "GetGroupSubStatusResponse" Primitive

The `GetGroupSubStatusResponse` primitive allows 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 8.2). |
| Group-ID | M | String | Identifies the group |
| Subscription-Status | M | 'S' \| 'U' | Indicates the subscription status – subscribed ('S') or not ('U'). |

**Table 194. Information elements in GetGroupSubStatusResponse Primitive**

## 17.1.17 The "NotifyGroupChange" Transaction

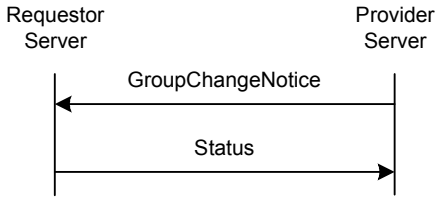


**Figure 90: The "NotifyGroupChange" Transaction**

The server MUST send group change notification(s) to the subscribed users whenever a user leaves or joins the group, or the group properties have been changed. The server MUST send the GroupChangeNotice primitive to the users (whose group change subscription is active) containing a list of users, identified by their screen names and User-IDs of the recently joined or left users, or the new properties of the group. The server MUST also include a list of the users who have been blocked by the receiving user and who have recently joined or left the group. The server MUST include users in the Joined-Blocked-Users-List or Left-Blocked-Users-List also in the Joined-Users-List or Left-Users-List.

Servers MUST support the "NotifyGroupChange" transaction.

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 195. Primitive Directions for NotifyGroupChange Transaction**

### 17.1.17.1 Primitives

#### 17.1.17.1.1 The "GroupChangeNotice" Primitive

The `GroupChangeNotice` primitive allows the provider server to send notifications to the subscribed users whenever users leave or join 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 8.1). |
| Subscribing-User-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. |
| Joined-Blocked-Users-List | O | Structure | A list of the blocked users that recently joined the group (Screen Name, User-ID),. |
| Left-Blocked-Users-List | O | Structure | A list of the blocked users that |

| | | | |
|---|---|---|---|
| | | | recently left the group (Screen Name, User-ID). |
| Group-Prop-List | O | Structure | The new properties of the group. |
| Own-Props | O | Structure | The new properties of the user in the group. |

**Table 196. Information elements in GroupChangeNotice Primitive**

# 17.2 Status Code

## 17.2.1 "CreateGroup" Transaction

- 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)
- Cannot have searchable group without name or topic (822)

## 17.2.2 "DeleteGroup" Transaction

- Group does not exist (800)
- Group is public (804)
- Insufficient group privileges (816)

## 17.2.3 "JoinGroup" Transaction

- Group does not exist (800)
- Invalid/unsupported group properties (806)
- User already joined (807)
- Cannot join: "rejected" (809)
- Insufficient group privileges (816)
- The maximum number of allowed users has been reached (817)
- Minimum age requirement not fulfilled (818)

## 17.2.4 "LeaveGroup" Transaction

- Group was not joined before transaction (808)

## 17.2.5 Group Membership Transactions

- Unknown user (531)
- Group does not exist (800)
- Insufficient group privileges (816)
- Group was not joined before transaction (808)

### 17.2.6 Group Properties Transactions

- Group does not exist (800)
- Invalid group attribute(s) (806)
- Insufficient group privileges (816)
- Cannot have searchable group without name or topic (822)

### 17.2.7 “RejectList” Transaction

- User unknown (531)
- Group does not exist (800)
- Insufficient group privileges (816)

### 17.2.8 Group Change Transactions

- Group does not exist (800)
- Group was not joined before transaction (808)

### 17.2.9 “GetJoinedMember” Transaction

- Group does not exist (800)

# 18 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.

## 18.1 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 complete a transaction.

### 18.1.1 100 – 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.

### 18.1.2 101 – 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.

### 18.1.3 102 – 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.

### 18.1.4 104 – 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.

## 18.2 2xx – Successful

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

### 18.2.1 200 – Successful

This is used to indicate that the request succeeded.

### 18.2.2 201 – 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.

### 18.2.3 203 – Extension block ignored

The client/server requested a transaction that carries an extension block however the extension block was ignored on the terminating end – in an otherwise successful transaction. The originator of the request MUST NOT perform the behavior described in the proprietary solution as the requested proprietary functionality was ignored on the terminating end.

# 18.3 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.

## 18.3.1 400 – Bad Request

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

## 18.3.2 401 – Unauthorized

When an authorization request is expected, the provider server will respond with this status code. Properties will contain details of available authorization schemes.

## 18.3.3 402 – Bad Parameter

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

## 18.3.4 403 – Forbidden

The server understood the request, but the principal settings denied access to some of the presence, contact information, group or IMPS service. Authorization will not help and the request SHOULD NOT be repeated. This type of response is also returned if user not logged into the network.

## 18.3.5 404 - Not Found

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

## 18.3.6 405 – Service Not Supported

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

## 18.3.7 409 – Invalid password

The password provided by the client was incorrect; it does not match with the given User-ID.

## 18.3.8 410 – 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.

## 18.3.9 411 – Unable to find suitable content type

The provider server cannot deliver the response because the client does not support any suitable content type

## 18.3.10 415 – Unsupported Media Type

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

## 18.3.11 420 – Invalid Transaction-ID

The server encountered an invalid Transaction-ID.

### 18.3.12 422 – User-ID and Client-ID Does Not Match

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

### 18.3.13 423 – Invalid Invitation-ID

The server encountered an invalid invitation ID.

### 18.3.14 424 – Invalid Search-ID

The server encountered an invalid search ID.

### 18.3.15 425 – Invalid Search-Index

The server encountered an invalid search index.

### 18.3.16 426 – Invalid Message-ID

The server encountered an invalid Message-ID.

### 18.3.17 427 – Invalid User-ID

The server encountered an invalid User-ID.

### 18.3.18 428 – Invalid Client-ID

The server encountered an invalid Client-ID.

### 18.3.19 429 – Missing Group-ID

The server encountered a request without Group-ID while the Group-ID is necessary to fulfill the request.

### 18.3.20 431 – Unauthorized Group Membership

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

### 18.3.21 433 – Invalid notification type

The server encountered an invalid notification type.

### 18.3.22 436 – System Message Response required

The provider server has sent a System Message notification to the requestor server requiring response from the end-user.

### 18.3.23 437 – Unknown System Message ID

The System Message response contains an unknown system message id.

### 18.3.24 438 – Incorrect Verification Key

The System Message response contains an incorrect verification key.

### 18.3.25 439 – Verification Mechanism Not Supported

The client does not support the verification mechanism required by the provider server in the System Message notification.

### 18.3.26 440 – Not allowed notification type

The client subscribed to an event of functionality not agreed during service negotiation.

### 18.3.27 441 – Number of characters exceeds the maximum number of characters

The client submitted a value where the number of characters is more than the allowed maximum numbers of characters for this value.

### 18.3.28 442 – Wrong value type

The type of the value submitted by the client does not match the required value types. E.g. the Age field MUST be an Integer

### 18.3.29 450 – Missing Application-ID

The requestor server attempted to accept an invitation without providing an Application-ID, thus the provider server rejected the request. The requestor server MUST NOT repeat the request without modification.

### 18.3.30 451 – Invalid Application-ID

The client attempted to use an Application-ID that was not registered to it during login. The client MUST NOT repeat the request without modification during the active session.

### 18.3.31 452 – Forbidden Application-ID

The requesting user is blocking the requested Application-ID while the BlockList is in use, or the Application-ID is not permitted while the GrantList is in use. The client MUST NOT repeat the request without applying modifications to his/her access control rules.

## 18.4 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.

### 18.4.1 500 – Internal Server Error

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

### 18.4.2 501 – 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.

### 18.4.3 502 – Session could not be recovered

The providing server was not able to recover the session requested by the requesting server.

### 18.4.4 503 – Service Unavailable

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

### 18.4.5 504 – Invalid Timeout

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

### 18.4.6 505 – Version Not Supported

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

### 18.4.7 506 – Service Not Agreed

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

### 18.4.8 507 – Message Queue is Full

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

### 18.4.9 508 – Unsupported message context

The server is unable to understand the message context that the client has used. The client MUST NOT repeat the request without modification.

### 18.4.10 509 – Related services are missing

The server cannot perform the request, as some services that are indirectly related to the requested transaction have not been agreed during service negotiation. The client MUST NOT repeat the request until the indirectly required services have been agreed using a service negotiation.

### 18.4.11 510 – Related client capabilities are missing

The server cannot perform the request, as some client capabilities that are indirectly related to the requested transaction have not been agreed during client capability negotiation. The client MUST NOT repeat the request until the indirectly required capabilities have been agreed using a client capability negotiation.

### 18.4.12 515 – Application is forbidden

The client attempted to assign an Application-ID for the session during login that the service provider does not allow for some reason – such reasons might be that the *application* has been identified by service provider to be un-secure, non-compliant, spy-ware, etc –, thus the server rejected the login request. The client MUST NOT repeat the request without modification. The client MAY repeat the request without Application-ID, or using another Application-ID.

### 18.4.13 516 – Domain Not Supported

The server does not support forwarding to different a domain space.

### 18.4.14 517 – Location Not Supported

The provider server is unable to generate a map for the requested location for some reason

### 18.4.15 531 – Unknown User

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

### 18.4.16 532 –Recipient Blocked the Sender

The recipient of the message or invitation blocked the sender.

### 18.4.17 533 – Message Recipient Not Logged in

The recipient of the message is not logged in.

### 18.4.18 534 – Message Recipient Unauthorized

The recipient of the message is not authorized.

### 18.4.19 535 – Search Timed Out

The server has invalidated the requested search-request.

### 18.4.20 536 – Too many hits.

The query returned too many hits. The client needs to narrow the query.

### 18.4.21 537 – Too broad search criteria

The query cannot be processed since it is too broad.

### 18.4.22 539 – Message-Count exceeded

The list of messages on the server exceeds the maximum number of Message-Info structures to be returned as specified in Message-Count.

### 18.4.23 540 – Message has been rejected due to limitations

The providing server has rejected the instant message because the instant message content is not allowed either because the content type is not supported, or because it is too large. The requesting server MUST NOT repeat the transaction without modification.

### 18.4.24 543 – No matching digest scheme supported

The provider server does not support any of the digest schemas that the requestor server has requested.

### 18.4.25 544 – Too many elements in advanced criteria

The server did not perform the search – it has received an advanced search request, which includes advanced criteria with too many elements. The client MUST NOT repeat the request without modification.

### 18.4.26 545 – Too many levels of nesting in advanced criteria

The server did not perform the search – it has received an advanced search request, which includes advanced criteria with too many levels of nesting. The client MUST NOT repeat the request without modification.

### 18.4.27 546 – Message cannot be delivered due to limitations

The requesting server attempted to retrieve an instant message where the content is not allowed either because the content type is not supported or it is too large. The requesting server MUST NOT repeat the transaction without modification.

### 18.4.28 547 – Wildcard characters not allowed

The requesting server attempted to use wildcard characters, however the providing server does not allow wildcard characters to be used. The requesting server MUST NOT repeat the request without modification.

### 18.4.29 548 – Wildcard expression is too complicated

The requesting server attempted to use wildcard characters, however the providing server finds the expression too complicated. The requesting server MUST NOT repeat the request without modification.

### 18.4.30 560 – Unsupported search-element was requested

The requesting server requested a search using a search-element that is not supported by the providing server. The requesting server MUST NOT repeat the request without modification.

### 18.4.31 561 – Supported search-element

The providing server supports searching using the search-element marked with this error code. The requesting server MAY attempt to search using this element.

### 18.4.32 562 – Unsupported search-element

The providing server does not support searching using the search-element marked with this error code. The requesting server SHOULD NOT attempt to search using this element.

## 18.5 6xx – Session

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

### 18.5.1 600 – Session Expired

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

### 18.5.2 601 – Forced Logout

The provider server has disconnected the requestor server.

### 18.5.3 604 – Invalid Session / Not Logged In

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

### 18.5.4 606 – Invalid Service-ID

Unknown Service-ID.

### 18.5.5 607 – Redirection Refused

The redirected connection is refused.

### 18.5.6 608 – 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.

### 18.5.7 609 – Connection Expired

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

### 18.5.8 610 – Server Search Limit is Exceeded

The search limit exceeds the server limit.

### 18.5.9 611 – Too many non-conformant System Message replies

The provider server will not accept new requests from the client for a period of time (it is implementation-specific), because the client already attempted to respond a System Message with non-conformant replies too many times. The provider server cannot verify whether the user is making mistakes, or the client is not OMA IMPS compliant, but this error code allows the

provider server to protect itself against undesired attempts. The provider server is NOT REQUIRED to validate any requests while this protection is active – it MAY respond any request without validation with a Status primitive using the same error code. The requestor server MUST NOT repeat the request on behalf of the client until the protection time indicated in the status details expires

### 18.5.10 620 – 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.

### 18.5.11 630 – User Session Expired

The user session was disconnected because time-to-live parameter of user session has expired.

### 18.5.12 631 – User Session - Forced Logout

The provider server has disconnected the user session for some reason.

### 18.5.13 635 – New value not accepted.

The provider server does not accept the new timeout value requested for the new user session, the old value MUST be used.

### 18.5.14 636 – Some services are not available

The provider server does not accept the user session re-establishment request because some of the services that have been agreed during the terminated session are not available.

### 18.5.15 637 – Too many non-conformant System Message replies

The server will not accept new requests from the client for a period of time (it is implementation-specific), because the client already attempted to respond a System Message with non-conformant replies too many times. The server cannot verify whether the user is making mistakes, or the client is not OMA IMPS compliant, but this error code allows the server to protect itself against undesired attempts. The server is NOT REQUIRED to validate any requests while this protection is active – it MAY respond any request without validation with a Status primitive using the same error code. The client MUST NOT repeat the request until the protection time indicated in the status details expires.

### 18.5.16 638 – Client-ID is not unique

The server did not accept the new user session establishment, because the Client-ID is already in use.

### 18.5.17 639 – User session limitation reached

The user has already reached the maximum number of concurrent sessions that are allowed across all SAPs.

## 18.6 7xx – Presence and contact list

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

### 18.6.1 700 – Contact List Does Not Exist

The contact list specified in the request does not exist.

### 18.6.2 701 – Contact List Already Exists

The contact list specified in the request already exists.

### 18.6.3 702 – Invalid or Unsupported User Properties

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

### 18.6.4 703 – Contact List is empty

The client attempted to use a contact list that is empty.

### 18.6.5 750 – Invalid or Unsupported Presence Attributes

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

### 18.6.6 751 – Invalid or Unsupported Presence Value

The presence value(s) specified in the request are invalid or not supported. The client SHOULD NOT repeat the request without modification.

### 18.6.7 752 – Invalid or Unsupported Contact List Property

One or more contact list properties specified in the request are invalid or not supported. The client SHOULD NOT repeat the request without modification.

### 18.6.8 756 – The maximum number of Users in grant list has been reached for the user

The server limits the maximum number of users in grant list per user. The limit has been reached; so the user cannot add new entities to the grant list. The client SHOULD NOT repeat the request until the user has removed entities from grant list.

### 18.6.9 757 – The maximum number of Users in block list has been reached for the user

The server limits the maximum number of users in block list per user. The limit has been reached; so the user cannot add new entities to the block list. The client SHOULD NOT repeat the request until the user has removed entities from block list.

### 18.6.10 758 – The maximum number of users in watcher list has been reached for the user

The server limits the maximum number of users in watcher list per user. The limit has been reached; so other users cannot subscribe the presence attributes of the user.

## 18.7 8xx – Groups

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

### 18.7.1 800 – Group Does Not Exist

The group specified in the request does not exist.

### 18.7.2 801 – Group Already Exists

The group specified in the request already exists.

### 18.7.3 802 – Group is Open

The group specified in the request is an open group.

### 18.7.4 803 – Group is Restricted

The group specified in the request is a restricted group.

### 18.7.5 804 – Group is Public

The group specified in the request is public.

### 18.7.6 805 – Group Private

The group specified in the request is private.

### 18.7.7 806 – Invalid / Unsupported Group Properties

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

### 18.7.8 807 – 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 to the particular group.

### 18.7.9 808 – Group is Not Joined

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

### 18.7.10 809 – Rejected

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

### 18.7.11 810 – Not a Group Member

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

### 18.7.12 812 – Private Messaging is Disabled for Group

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

### 18.7.13 813 – Private Messaging is Disabled for User

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

### 18.7.14 814 – The Maximum Number of Groups Has Been Reached for the User

The server limits the maximum number of groups per user. The limit has been reached; additional groups cannot be created. The client SHOULD NOT repeat the request until a group that belongs to the particular user has been deleted.

### 18.7.15 815 – The Maximum Number of Groups Has Been Reached for the Server

The maximum number of groups is limited on the server. The server limit has been reached; additional groups cannot be created. The client MAY repeat the request.

### 18.7.16 816 – 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.

### 18.7.17 817 – The Maximum Number of Joined Users Has Been Reached

The maximum number of joined users has been reached in the requested group. The client MAY repeat the request.

### 18.7.18 818 – Minimum age requirement not fulfilled

The group has an active age restriction limitation, and the requesting user does not fulfill the requirements needed to perform this transaction. The client SHOULD NOT repeat the request within a reasonable time period.

### 18.7.19 821 – History is Not Supported

The server does not support group message history caching.

### 18.7.20 822 - Cannot have searchable group without name or topic.

The server cannot perform group search without group name or group topic. Either group name or group topic or both must be non-empty to support group search.

### 18.7.21 825 – Extend conversation rejected

The invitee rejected the invitation to the extended conversation.

## 18.8 9xx – General errors

The 9xx class indicates status codes too general to fit into other classes.

### 18.8.1 900 – Multiple errors

No part of the transaction was successfully processed for several reasons, thus not only one other status code can indicate the errors. The details of the error cases are indicated in the response.

### 18.8.2 901 – General Address Error

The general address is not supported. No specific error is given due to security or privacy reason.

### 18.8.3 902 – MSISDN error

The client attempted to use an MSISDN that is not used by the device, thus the server rejected the request. The client MUST NOT repeat the request without modification.

### 18.8.4 903 – Registration confirmation

This status code indicates that the registration is successful but extra registration information is required before the user can use the service and the client MUST NOT continue the login.

### 18.8.5 904 – Missing mandatory field(s) of requesting user

The requesting user did not fill in the mandatory fields of his/her public profile. The client MUST NOT repeat the request, and MAY receive a system message – see system message in 13.1.11.

### 18.8.6 905 – Missing mandatory field(s) of requested user

The requested user did not fill in the mandatory fields of his/her public profile. The client SHOULD NOT repeat the request within a reasonable time period.

### 18.8.7 906 – Too many public profiles requested

The client requested too many public profiles in a request. The server has successfully delivered as much public profiles as its implementation allows within a single transactions, however some public profiles have not been delivered due the limitations on server side. The client MAY retrieve the rest of the profiles, however it MUST retrieve the excess of public profiles in a separate transaction.

### 18.8.8 907 – Service provider agreement missing

The client attempted to perform an operation that involves another service provider, however the agreement between the related service providers prevents the server from performing the requested operation. The client SHOULD NOT repeat the request without modification.

### 18.8.9 908 – There are no instant messages

The client attempted to retrieve the list of instant message on the server, however there are no instant messages.

### 18.8.10 909 – Recipient does not support the requested functionality

The recipient client does not support the requested functionality.

### 18.8.11 920 – Not enough credit to complete requested operation

The server cannot perform the requested operation since the user has not enough credit.

### 18.8.12 921 – Operation requires a higher class of service

The server cannot perform the requested operation since it requires a higher class of service. A class of service is a designation assigned by the service provider to describe the service treatment and privileges given to a particular user (e.g., premium, gold).

# Appendix A. Change History (Informative)

## A.1 Approved Version History

| Reference | Date | Description |
|---|---|---|
| OMA-TS-IMPS_SSP-V1_3 | 23 Jan 2007 | Status changed to Approved by TP<br>TP Doc ref# OMA-TP-2006-0453R02 |

# Appendix B. Static Conformance Requirements

The notation used in this appendix is specified in [IOPPROC].

## B.1 IMPS SSP Service Feature requirement

| Item | Function | Reference | Status | Requirement |
|---|---|---|---|---|
| IMPS-SSP-SERV-S-001 | Support the Service Access Point Features | section 10.3 | M | |
| IMPS-SSP-SERV-S-002 | Support the Common IMPS Features | section 10.4 | O | IMPS-SSP-CMSE-S-001 AND IMPS-SSP-CMSE-S-003 AND IMPS-SSP-CMSE-S-005 AND IMPS-SSP-CMSE-S-007 AND IMPS-SSP-CMSE-S-010 AND IMPS-SSP-CMSE-S-011 AND IMPS-SSP-CMSE-S-012 AND IMPS-SSP-CMSE-S-013 |
| IMPS-SSP-SERV-S-003 | Support the Instant Messaging Features | section 10.6 | O | IMPS-SSP-IMSE-S-001 AND IMPS-SSP-IMSE-S-002 AND IMPS-SSP-IMSE-S-017 AND IMPS-SSP-IMSE-S-019 AND IMPS-SSP-IMSE-S-025 AND IMPS-SSP-IMSE-S-026 |
| IMPS-SSP-SERV-S-004 | Support the Presence Service Features | section 10.5 | O | IMPS-SSP-PRSE-S-003 AND IMPS-SSP-PRSE-S-005 AND IMPS-SSP-PRSE-S-007 AND IMPS-SSP-PRSE-S-009 AND IMPS-SSP-PRSE-S-011 |
| IMPS-SSP-SERV-S-005 | Support the Group Service Features | section 10.7 | O | IMPS-SSP-GRSE-S-002 AND IMPS-SSP-GRSE-S-005 AND IMPS-SSP-GRSE-S-007 AND IMPS-SSP-GRSE-S-008 AND IMPS-SSP-GRSE-S-009 AND IMPS-SSP-GRSE-S-010 AND IMPS-SSP-GRSE-S-011 |
| IMPS-SSP-SERV-S-006 | Support for service forwarding feature | section 6.6.7 | O | |
| IMPS-SSP-SERV-S-007 | Support Remote User Session Management | section 12 | O | |

## B.2 Adressing requirement

| Item | Function | Reference | Status | Requirement |
|---|---|---|---|---|
| IMPS-SSP-ADDR-S-001 | Support the conversion from local address of the object(s) to global address when service is relayed. | section 6.3 | M | |
| IMPS-SSP-ADDR-S-002 | Support the conversion from local address of the own users into global address when service is relayed | section 6.3 | M | |

## B.3 Data Type requirement

| Item | Function | Reference | Status | Requirement |
|---|---|---|---|---|
| IMPS-SSP-DATA-S-001 | Support “Char” data type | section 6.4.1 | M | |

| Item | Function | Reference | Status | Requirement |
|---|---|---|---|---|
| IMPS-SSP-DATA-S-002 | Support “Integer” data type | section 6.4.2 | M | |
| IMPS-SSP-DATA-S-003 | Support “String” data type | section 6.4.3 | M | |
| IMPS-SSP-DATA-S-004 | Support “Boolean” data type | section 6.4.4 | M | |
| IMPS-SSP-DATA-S-005 | Support “DateTime” data type | section 6.4.6 | M | |
| IMPS-SSP-DATA-S-006 | Support “Enum” data types with defined value sets | section 6.4.5 | M | |
| IMPS-SSP-DATA-S-007 | Support “Structure” data types with defined combination of other data types | section 6.4.7 | M | |

## B.4 Infrastructure requirement

| Item | Function | Reference | Status | Requirement |
|---|---|---|---|---|
| IMPS-SSP-INFR-S-001 | Support the “Host-ID” address resolution | section 6.5.1 | M | |

## B.5 Session Management requirement

| Item | Function | Reference | Status | Requirement |
|---|---|---|---|---|
| IMPS-SSP-SESSION-S-001 | Support session pair between two domains. | section 6.1.1 | M | |
| IMPS-SSP-SESSION-S-002 | Support redirect connection pairs of the same session | section 9.1.1 | O | IMPS-SSP-SAP-S-011 |

## B.6 Transaction Management requirement

| Item | Function | Reference | Status | Requirement |
|---|---|---|---|---|
| IMPS-SSP-TRANS-S-001 | Support one-way transaction | Section 8 | M | |
| IMPS-SSP-TRANS-S-002 | Support two-way transaction | Section 8 | M | |
| IMPS-SSP-TRANS-S-003 | Support multiple-way transaction | Section 8 | M | |
| IMPS-SSP-TRANS-S-004 | Support asynchronous transactions | Section 8.3 | M | |

## B.7 Service Access Point Features requirement

| Item | Function | Reference | Status | Requirement |
|---|---|---|---|---|
| IMPS-SSP-SAP-S-001 | Support service relay between the Home Domains through direct SSP connection(s) | section 5.2 | M | |
| IMPS-SSP-SAP-S-002 | Support service relay between the Home Domain and its complementary service | section 5.2 | O | IMPS-SSP-SAP-S-005 |

| Item | Function | Reference | Status | Requirement |
|---|---|---|---|---|
| IMPS-SSP-SAP-S-003 | Support service relay – every message is sent directly to that Home Domain, which is addressed by the network entity (user, group, contactlist) in the request (message). | section 5.2 | M | |
| IMPS-SSP-SAP-S-004 | Support service relay - routing for SAP-1:<br>Every message is sent directly to that Home Domain, which is addressed by the network entity (user, group, contactlist) in the request (message). | section 5.2 | M | |
| IMPS-SSP-SAP-S-005 | Support service relay – routing for the SAP-2 | section 5.2 | O | IMPS-SSP-SAP-S-005 |
| IMPS-SSP-SAP-S-006 | Support the conversion from local object to global object(s) when service is relayed | section 6.3.1 | M | |
| IMPS-SSP-SAP-S-007 | Support Meta-Information primitive | section 8.1 | M | |
| IMPS-SSP-SAP-S-008 | Support Status primitive | section 8.2 | M | |
| IMPS-SSP-SAP-S-009 | Support CALLBACK session establishment and its steps | section 6.6.2 | M | |
| IMPS-SSP-SAP-S-010 | Support session establishment through Login transaction | section 9.3.1 | M | |
| IMPS-SSP-SAP-S-011 | Support redirect connection pairs through Login transaction | section 9.3.1 | O | IMPS-SSP-SESSION-002 |
| IMPS-SSP-SAP-S-012 | Support Logout transaction | section 9.3.2 | M | |
| IMPS-SSP-SAP-S-013 | Support Disconnect transaction | section 9.3.2 | M | |
| IMPS-SSP-SAP-S-014 | Support KeepAlive transaction | section 9.2.4 | M | |
| IMPS-SSP-SAP-S-015 | Support GetAvailableService transaction | section 10.8.1 | O | |
| IMPS-SSP-SAP-S-016 | Support ServiceIndication transaction | section 10.8.2 | O | |
| IMPS-SSP-SAP-S-017 | Support SetServiceAgreement transaction | section 10.8.3 | O | |
| IMPS-SSP-SAP-S-018 | Support GetUserProfile transaction | section 11.2.1 | O | |
| IMPS-SSP-SAP-S-019 | Support UpdateUserProfile transaction | section 11.2.2 | O | |
| IMPS-SSP-SAP-S-020 | Support GetAcceptedContentTypes transaction | section 10.8.4 | O | |

# B.8 Common IMPS Features requirement

| Item | Function | Reference | Status | Requirement |
|---|---|---|---|---|
| IMPS-SSP-CMSE-S-001 | Support GeneralSearch transaction | section 13.1.4 | O | IMPS-SSP-SERV-S-002 |
| IMPS-SSP-CMSE-S-002 | Support advanced search | section 13.1.4 | O | |

| Item | Function | Reference | Status | Requirement |
|---|---|---|---|---|
| IMPS-SSP-CMSE-S-003 | Support StopSearch transaction | section 13.1.6 | O | IMPS-SSP-SERV-S-002 |
| IMPS-SSP-CMSE-S-004 | Support VerifyWVID transaction | section 13.1.10 | O | |
| IMPS-SSP-CMSE-S-005 | Support Basic Invitation transaction | section 13.1.7.1 | O | IMPS-SSP-SERV-S-002 |
| IMPS-SSP-CMSE-S-006 | Support Complementary Invitation transaction | section 13.1.8.2 | O | IMPS-SSP-CMSE-S-008 |
| IMPS-SSP-CMSE-S-007 | Support Basic CancelInvitation transaction | section 13.1.8 | O | IMPS-SSP-SERV-S-002 |
| IMPS-SSP-CMSE-S-008 | Support Complementary CancelInvitation transaction | section 13.1.8.2 | O | IMPS-SSP-CMSE-S-006 |
| IMPS-SSP-CMSE-S-009 | Support GetMap transaction | Section 13.1.9 | O | |
| IMPS-SSP-CMSE-S-010 | Support for System Message transactions | section 13.1.11 and section 13.1.12 | O | IMPS-SSP-SERV-S-002 |
| IMPS-SSP-CMSE-S-011 | Support for GetPublicProfile transaction | section 13.1.2 | O | IMPS-SSP-SERV-S-002 |
| IMPS-SSP-CMSE-S-012 | Support for UpdatePublicProfile transaction | section 13.1.3 | O | IMPS-SSP-SERV-S-002 |
| IMPS-SSP-CMSE-S-013 | Support General Notification transactions | section 13.1.1 | O | IMPS-SSP-SERV-S-002 |

# B.9 Presence Features requirement

| Item | Function | Reference | Status | Requirement |
|---|---|---|---|---|
| IMPS-SSP-PRSE-S-001 | Support Contact List transactions | section 14 | O | |
| IMPS-SSP-PRSE-S-002 | Support AttributeList transactions | section 14.2.9, 14.2.10 & 14.2.11 | O | |
| IMPS-SSP-PRSE-S-003 | Support subscribe to presence for User-ID List | section 15.2.1 | O | IMPS-SSP-SERV-S-004 |
| IMPS-SSP-PRSE-S-004 | Support subscribe to presence for ContactList object | section 15.2.1 | O | |
| IMPS-SSP-PRSE-S-005 | Support GetPresence for User-ID List | section 15.2.5 | O | IMPS-SSP-SERV-S-004 |
| IMPS-SSP-PRSE-S-006 | Support GetPresence for ContactList object | section 15.2.5 | O | |
| IMPS-SSP-PRSE-S-007 | Support UpdatePresence for User-ID List | section 15.2.6 | O | IMPS-SSP-SERV-S-004 |
| IMPS-SSP-PRSE-S-008 | Support UpdatePresence for ContactList object | section 15.2.6 | O | |
| IMPS-SSP-PRSE-S-009 | Support for Presence Notifications | section 15.2.3 | O | IMPS-SSP-SERV-S-004 |
| IMPS-SSP-PRSE-S-010 | Support GetWatcherList transaction | section 15.2.4 | O | |

| Item | Function | Reference | Status | Requirement |
|---|---|---|---|---|
| IMPS-SSP-PRSE-S-011 | Support Suspend transaction for User-ID List | section 15.2.7 | O | IMPS-SSP-SERV-S-004 |
| IMPS-SSP-PRSE-S-012 | Support Suspend transaction for ContactList object | section 15.2.7 | O | |

# B.10 Instant Messaging Features requirement

| Item | Function | Reference | Status | Requirement |
|---|---|---|---|---|
| IMPS-SSP-IMSE-S-001 | Support SendMessage transaction | section 16.2.1 | O | IMPS-SSP-SERV-S-003 |
| IMPS-SSP-IMSE-S-002 | Support recipient addressed by User-ID in SendMessage transaction | section 16.2.1 | O | IMPS-SSP-SERV-S-003 |
| IMPS-SSP-IMSE-S-003 | Support recipients listed by Contact List ID in SendMessage transation | section 16.2.1 | O | |
| IMPS-SSP-IMSE-S-004 | Support recipient as Group-ID and addressing by screen name in SendMessage transaction | section 16.2.1 | O | |
| IMPS-SSP-IMSE-S-005 | Support ForwardMessage transaction | section 16.2.2 | O | IMPS-SSP-IMSE-S-006 OR IMPS-SSP-IMSE-S-007 OR IMPS-SSP-IMSE-S-008 |
| IMPS-SSP-IMSE-S-006 | Support recipient addressed by User-ID in ForwardMessage transaction | section 16.2.2 | O | IMPS-SSP-IMSE-S-005 |
| IMPS-SSP-IMSE-S-007 | Support recipients listed by Contact List ID in ForwardMessage transation | section 16.2.2 | O | IMPS-SSP-IMSE-S-005 |
| IMPS-SSP-IMSE-S-008 | Support recipient as Group-ID and addressing by screen name in ForwardMessage transaction | section 16.2.2 | O | IMPS-SSP-IMSE-S-005 |
| IMPS-SSP-IMSE-S-009 | Support PushMessage or MessageNotification transaction | Section 16.2.3 and 16.2.4 | O | IMPS-SSP-IMSE-S-010 OR IMPS-SSP-IMSE-S-011 |
| IMPS-SSP-IMSE-S-010 | Support PushMessage transaction | section 16.2.3 | O | |
| IMPS-SSP-IMSE-S-011 | Support MessageNotification transaction | section 16.2.4 | O | IMPS-SSP-IMSE-S-012 |
| IMPS-SSP-IMSE-S-012 | Support GetMessage transaction | section 16.2.5 | O | IMPS-SSP-IMSE-S-011 |
| IMPS-SSP-IMSE-S-013 | Support SetMessageDeliveryMethod transaction | section 16.2.6 | O | |
| IMPS-SSP-IMSE-S-014 | Support GetMessageList transaction for Group history requests | section 16.2.7 | O | IMPS-SSP-IMSE-S-019 |
| IMPS-SSP-IMSE-S-015 | Support GetMessageList transaction for Undelivered messages | section 16.2.7 | O | |
| IMPS-SSP-IMSE-S-016 | Support RejectMessage transaction | section 16.2.8 | O | |
| IMPS-SSP-IMSE-S-017 | Support NotifyDeliveryStatusReport transaction | section 16.2.9 | O | IMPS-SSP-SERV-S-003 |
| IMPS-SSP-IMSE-S-018 | Support of group history caching | section 16.2.7 | O | IMPS-SSP-IMSE-S-019 |
| IMPS-SSP-IMSE-S-019 | The save history is part of the IM service if the group history caching is supported. | section 16.2.7 | O | IMPS-SSP-IMSE-S-018 AND IMPS-SSP-SERV-S-003 |

| Item | Function | Reference | Status | Requirement |
|---|---|---|---|---|
| IMPS-SSP-IMSE-S-020 | Support Block functions: BlockUser/GetBlockList transactions | section 16.2.10& section 16.2.11 | O | |
| IMPS-SSP-IMSE-S-021 | Support the IsUserBlocked transaction | section 16.2.12 | O | |
| IMPS-SSP-IMSE-S-022 | Support for Extend one-to-one to private group conversation | section 16.2.13 | O | IMPS-SSP-GRSE-S-007 |
| IMPS-SSP-IMSE-S-023 | Support semantics mapping between SSP primitive and CSP primitive of the above transactions if the transactions are supported | | O | IMPS-SSP-SERV-S-003 |
| IMPS-SSP-IMSE-S-024 | Support syntax mapping between SSP primitive and CSP primitive of the above transactions if the transactions are supported | | O | IMPS-SSP-SERV-S-003 |

# B.11 Group Service Features requirement

| Item | Function | Reference | Status | Requirement |
|---|---|---|---|---|
| IMPS-SSP-GRSE-S-001 | Support group management functions: CreateGroup/DeleteGroup transactions | section 17.1.1& section 17.1.2 | O | |
| IMPS-SSP-GRSE-S-002 | Support Join/Leave/ServerInitiatedLeaveGroup transactions | section 17.1.3& section 17.1.4& section 17.1.5 | O | IMPS-SSP-SERV-S-005 |
| IMPS-SSP-GRSE-S-003 | Support group member management functions: Get/Add/RemoveGroupMember, MemberAccess transactions | section 17.1.6& section 17.1.7& section 17.1.8& section 17.1.9 | O | |
| IMPS-SSP-GRSE-S-004 | Support Get/SetGroupProps transactions | section 17.1.10 & section 17.1.12 | O | IMPS-SSP-SERV-S-005 |
| IMPS-SSP-GRSE-S-005 | Support RejectList transaction – reject members | section 17.1.13 | O | |
| IMPS-SSP-GRSE-S-006 | Support Subscribe/UnsubscribeGroupChange and GetGroupSubStatus transactions | section 17.1.14& section 17.1.15& section 17.1.16 | O | IMPS-SSP-SERV-S-005 |
| IMPS-SSP-GRSE-S-007 | Support NotifyGroupChange transaction | section 17.1.17 | O | IMPS-SSP-SERV-S-005 |

| Item | Function | Reference | Status | Requirement |
|---|---|---|---|---|
| IMPS-SSP-GRSE-S-008 | The Group service uses IM service to send/receive and store messages to chat groups | | O | IMPS-SSP-SERV-S-005 |
| IMPS-SSP-GRSE-S-009 | Support semantics mapping between SSP primitive and CSP primitive of the above transactions if the transactions are supported | | O | IMPS-SSP-SERV-S-005 |
| IMPS-SSP-GRSE-S-010 | Support syntax mapping between SSP primitive and CSP primitive of the above transactions if the transactions are supported | | O | IMPS-SSP-SERV-S-005 |