White Paper on Implementation Guidelines for IMPS 1.3

Approved – 22 December 2009

Open Mobile Alliance
OMA-WP-IMPS-V1_3-20091222-A

Contents

Figures

Tables

1. Scope

The objective of this white paper is to ensure homogeneous user experience across different IMPS applications based on the CSP 1.3 protocol. It will provide informative best practice recommendations for the CSP 1.3 protocol, ensuring consistent and compatible end-user experience for an IM user while communicating on a device from one manufacturer with end-users on devices from other manufacturers, or when an end-user uses multiple clients from various manufacturers.

The Implementation Guidelines are written with a focus on applications written on top of CSP 1.3. Considerations for end-user experience when upgrading from older 1.1/1.2.1 clients to 1.3 clients(s) are also discussed. Considerations for SSP 1.3 are out of scope for this document.

2. References

[AppChar] "Application Characteristic for IMPS", OMA-TS-wA-Application-Characteristic-for-IMPSV1_0. Open Mobile Alliance™. URL: http://www.openmobilealliance.org

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

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

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

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

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

[CSP XMLS] "Client-Server Protocol XML Syntax Version 1.3", OMA-TS-IMPS-CSP-XMLS-V1_3. Open Mobile Alliance™. URL: http://www.openmobilealliance.org

[GSMAPH2] IM Phase 2 Service Definition, GSM Association Official Document SE.44, URL: http://www.gsmworld.com/documents/pim/im_phase_2_service_def.doc

[IANAPorts] "PORT NUMBERS", URL: http://www.iana.org/assignments/port-numbers

[IM] "Instant Messaging Application Category 1.0", URL: http://member.openmobilealliance.org/ftp/public_documents/imps/2003/oma-imps-2003-0168-support_of_appcategory.zip (embedded as an attachment within the document)

[MO] "OMA IMPS Management Object Version 1.0", OMA-TS-IMPS-MO-V1_0. Open Mobile Alliance™. URL: http://www.openmobilealliance.org

[OMADICT] "Dictionary for OMA Specifications, Version 2.6", Open Mobile Alliance™, OMA-ORG-Dictionary-V2_6, URL: http://www.openmobilealliance.org

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

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

[PEP] "Presence Enhanced Phonebook Application Category, 1.0", URL: http://member.openmobilealliance.org/ftp/public_documents/imps/2003/oma-imps-2003-0168-support_of_appcategory.zip (embedded as an attachment within the document)

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

[RFC2396] " Uniform Resource Identifiers (URI): Generic Syntax", URL: http://www.ietf.org/rfc/rfc2616.txt

[RFC2616] " Hypertext Transfer Protocol – HTTP/1.1", URL: http://www.ietf.org/rfc/rfc2616.txt

[RFC3986] "Uniform Resource Identifier (URI): Generic Syntax", URL: http://www.ietf.org/rfc/rfc3986.txt

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

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

[SSP XMLS] "Server-Server Protocol XML Syntax Document Version 1.3", OMA-TS-IMPS-SSP_XMLSV1_3. Open Mobile Alliance™. URL: http://www.openmobilealliance.org

3. Terminology and Conventions

3.1 Conventions

This is an informative document, which is not intended to provide testable requirements to implementations.

The key words "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

Each section of this document is organized into a main chapter, whose name is intended to describe the issue, a sub-section entitled "Rationale" justifying the inclusion of the topic in this document, a second optional sub-section with use cases describing the intended end user experience relevant to the section including examples of requests and responses (only transaction content will be shown in examples) and a third sub-section entitled "Recommendation" which describes the recommendations in terms of implementations for client or server vendors.

3.2 Definitions

Definitions already defined in IMPS 1.3 will not be repeated here.

3.3 Abbreviations

Abbreviations already defined in IMPS 1.3 will not be repeated here.

GSMA	GSM Association
GSM	Global System for Mobile communication

4. Introduction

The goal of this white paper is to ensure homogeneous user experience across different IMPS Applications.

Although the current IMPS 1.3 enabler package provides a complete architecture, reference points and interfaces, and protocol specifications, there have been some implementation issues at an application level whereas two or more applications that implement similar services using the same IMPS 1.3 technology do not behave in the same manner in the same situations. Additionally, due to the high level of optional information elements in the primitives of the IMPS protocol, it is not known what is the best and widely implemented way of executing a number of use cases that are not described in detail in the IMPS specifications - how transactions and options shall be combined and used to implement such use cases. This document is intended to be such guideline by describing:

- Rationale – in general this is described as a use case that is perceived by the end user
- Problem Statement describing what an issue(s) or a requirement(s) is when implementing the use case
- Recommendation that fulfills the use case and the requirement when using IMPS 1.3

Note that this document will not change or bend the existing IMPS 1.3 standard. The goal of this document is to support the use cases that are found as urgent in the market without breaking a compatibility with the implementations that do not necessarily follow this guideline but still use the IMPS 1.3 as base technology.

Also note this document is solely informative and thus there is NO conformance requirement or interoperability test consideration against this document.

5. Auto registration

5.1 Rationale

"Alice wants to auto-register to the service."

The auto-registration feature allows a server to create a new user account comprising a User-ID and a password, as described in [CSP], section 6.5. This is a new feature in the IMPS 1.3 specifications. The feature is intended to increase user take-up by facilitating first-time usage and registration of the service.

5.2 Use cases and examples

This section describes the use cases related to the auto-registration feature. Two main use cases are described that are distinguished based on where the User-ID is chosen/assigned. One, where the User-ID is chosen/assigned by the server [5.2.1] and a second, where the end-user is allowed to choose his/her own User-ID [5.2.2]. Section [5.2.3] elaborates on how to provide for the user to start using an already registered user account.

5.2.1 Server assigns User-ID and password

Actors	End user, Client, Server
Success Guarantees	End user is registered as a new user in the IM system
Preconditions	Client, Server
Trigger	Step 1
Main Success Scenario	1. End user attempts to log in without specifying a user name 2. Client sends a login request to the Server with an empty User-ID field 3. Server provisions the user choosing a User-ID and returns a login response indicating success with a Session-ID, User-ID and an auto-generated password
Extension Scenarios	**Server assigns User-ID based on MSISDN of the client** 3b Server discovers the MSISDN of the client, auto-provisions the user and returns a login response indicating success with a Session-ID, User-ID based on MSISDN and an auto-generated password
Variations	
Design Notes	

Table 1: Auto-registration - Server assigns User-ID and password

1. Client sends a login request to the Server with an empty User-ID field

```
<Login-Request>
  <User-ID/>
  <ClientID>client_id</ClientID>
  <SessionCookie>session_cookie</SessionCookie>
</Login-Request>
```

2. Server provisions the user choosing a User-ID and returns a login response indicating success with a Session-ID, User-ID and an auto-generated password

```
<Login-Response>
  <User-ID>wv:newuser@imps.com</User-ID>
  <Password>password</Password>
  <ClientID>client_id</ClientID>
  <Result>
```

```
      <Code>200</Code>
    </Result>
    <SessionID>session_id</SessionID>
    <KeepAliveTime>3600</KeepAliveTime>
    <CapabilityRequest>T</CapabilityRequest>
  </Login-Response>
```

5.2.2 User selects own User-ID

See figure [1] on the next page for an overview of the message flow in this use case. Note that steps 9 in Table 2 uses the word reserved which in this context mean either that the User-ID is currently in use, it does not pass the requirements for user-IDs as defined by the service (e.g., reserved words, bad language) or the User-ID has not passed a safe re-use period yet.

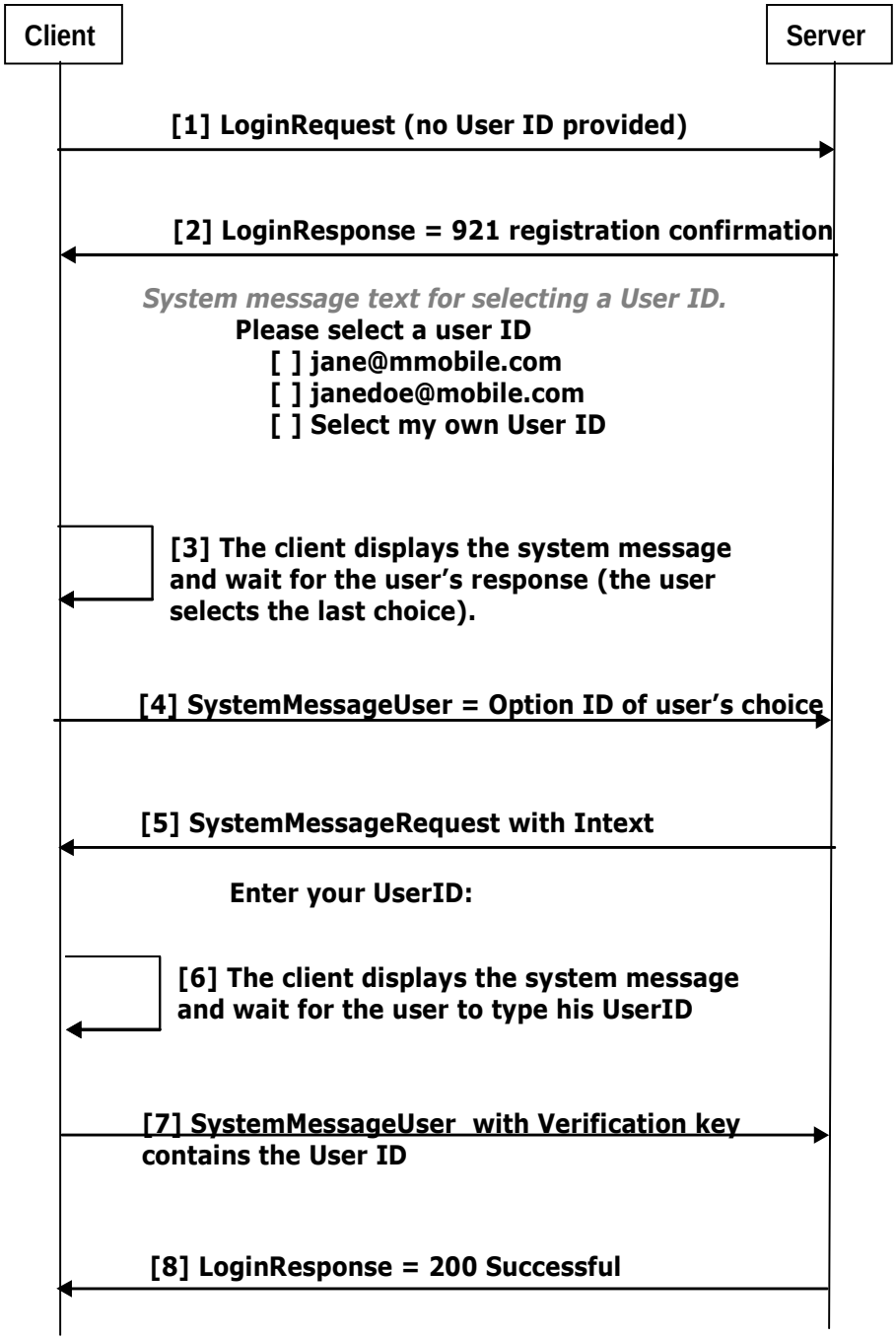

Figure 1: Auto registration - User selects own User-ID

Actors	End user, Client, Server
Success Guarantees	End user is provisioned in the IM system
Preconditions	The user has not registered before and does not have a User ID.
Trigger	The user launches his IMPS client to access the service.

Main success scenario Steps	1. End user attempts to log in without specifying a user name 2. Client sends a login request to the Server with an empty User-ID field 3. The server authenticates the user with network authentication information and returns a Login response primitive with the result code 921 “registration confirmation” and a system message with the following: a. Information about User-ID selection b. The RequiresResponse = TRUE c. The various User-ID options available to the user, where the last option allows the user to choose a custom User-ID. 4. The client returns the user’s selection in the ChosenOptionID element of the SystemMessageUser primitive. 5. The client returns the user’s selection ChosenOptionID in a SystemMessageUser, 6. The server sends a SystemMessageRequest with the InText field asking the user to enter a User-ID. 7. The client displays the system message and waits for the user to enter a User-ID. 8. The client sends the User-ID in the VerificationKey element of the SystemMessageUser primitive. 9. The server verifies that the chosen User-ID is valid and not reserved and the server returns a Login response with the User-ID selected by the user, a password selected by the server and the result code 200 “Successful”. Note: Future authentication will be done according to either CSP User-ID and password authentication or network authentication procedures.

Table 2: Auto-registration - User selects own User-ID

5.2.3 User is already registered and want to log in with a new device

To support scenarios where the user logs in with client not used before and where the user is already registered with the service, then servers SHOULD provide an “already registered” option for the end user to choose. When the “already registered” option is chosen by the user the steps 4 through 7 in section [5.2.1] must be applied. To make sure that the user actually owns the account that he/she claims, the steps in the 3 through 6 in section [9.2.1] must be applied.

If network authentication is in use on the service then the steps of the use case will be the same as in [5.2.1].

5.3 Recommendation

In order to facilitate an easy first-time setup, clients and server SHOULD implement the trigger and response mechanisms described in 5.2.1 and 5.2.2 and take advantage of them as they see fit. Servers that allow a user to choose their own User-IDs should facilitate the User-ID choice by taking advantage of the special use of the System Message feature as it is described in 5.2.2. Whenever a client receives a User-ID and password from the server in a LoginResponse primitive, it SHOULD store these newly received credentials on safe storage for the current network (domain).

6. Terms of use

6.1 Rationale

"Alice's operator wants to inform her of the terms of using the service."

Before using the IM service, operators or legislation often requires that the user agrees with the “terms of use” of the service.

6.2 Use cases and examples

This section describes how the terms of use are delivered to the end-user and how to get the user’s response prior to taking the service into use. Refer to figure 2 for a visualization of the message flow in this use case.

Due to parser size limitations in the client the terms of use text might be too large to process. In this case the server can choose to present the entire terms of use text as a URL pointing to a location where the terms of use text can be presented.

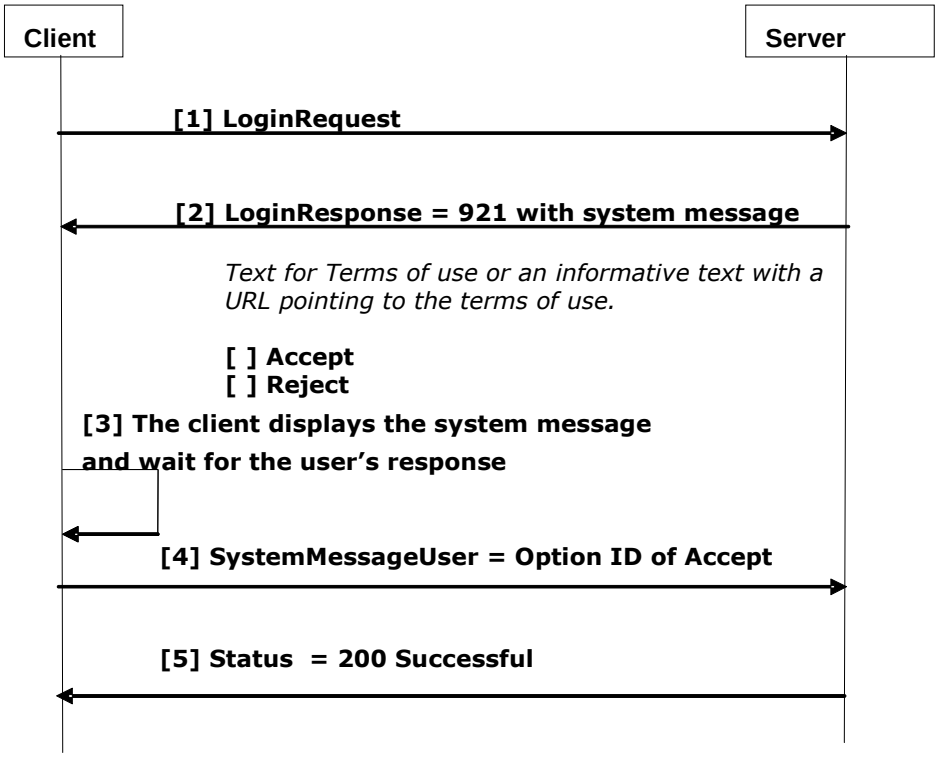

Figure 2: Terms of use message flows

Actors	End user, Client, Server
Success Guarantees	The terms of use are delivered to the user and the user's response is returned to the server.
Preconditions	The service provider deems it necessary to deliver the terms of use to the user and get the user's response.
Trigger	The server, issues a System Message when appropriate - based on service provider's policies - , during login phase.
Main success scenario Steps	1. The client sends a Login request primitive 2. The server responds with a Login response primitive with the result code 921 "registration confirmation" and a system message containing the terms of use text, the RequiresResponse = TRUE and the options available to the user. 3. The client displays the system message and waits for the user's selection. The user selects the 'Agree' option. 4. The client returns the user's selection in the ChosenOptionID element of the SystemMessageUser primitive, 5. The server returns a Login response with the result code 200 "Successful".
Extension scenarios	**Client chooses reject choice** 3b. The client displays the system message and waits for the user's selection. The user selects the 'reject' option. 4b. Same as for the successful flow. 5b. The server returns a Login response with the result code 921 "registration confirmation" and a system message with a text indicating that the user has to agree to the terms of use in order to use the IM service. The RequiresResponse = TRUE. Continue on step 2.

Table 3: Terms-of-use - main success use case description

6.3 Additional considerations

In addition to terms of use a service provider may also use other system message features like verification mechanisms, age verification and more.

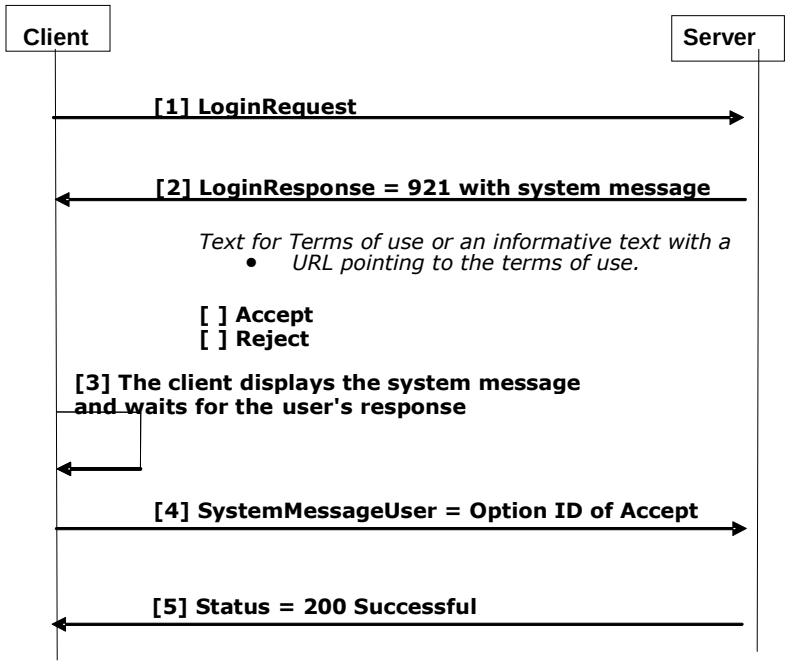

Figure 3: Terms of use message flows

7. Internationalization Support before Client Capability Negotiation

7.1 Rationale

"Alice wants to view terms and conditions during login in a language of her own preference."

In IMPS a client can negotiate preferred language for internationalized text by setting the DefaultLanguage in the ClientCapability-Request. Unfortunately this only applies to messages received from the server *after* client capability negotiation.

On auto registration, terms and conditions and similar System Messages, a user will be presented with text messages (e.g., "Choose your own User-ID", "By using this service…") where it would be important to present information using the native language.

7.2 Recommendation

The preferred way of indicating language preference is to provide the DefaultLanguage element in the CapabilityList element in the Login-Request (login with embedded client capabilities).

In some cases it is not desirable to embed client capabilities in the Login-Request due to limitations in the client or due to network bandwidth usage (the client capabilities must in some cases be presented multiple times on e.g. auto registration with system messages).To allow for a client to indicate to the server which language the user prefers when using HTTP as a bearer, clients can optionally utilize the Accept-Language HTTP header tag to indicate its preferred language on all requests HTTP requests until client capabilities has been negotiated.

The Accept-Language HTTP tags format is defined in section 3.10 of [RFC2616] (and RFC[1766]) and is different from the format defined on the DefaultLanguage element (three letter language code as defined in section 5.2 of [CSP DataType]).

```
da, en-gb;q=0.8, en;q=0.7
```

which means: "I prefer Danish, but will accept British English and other types of English".

Servers SHOULD ignore the Accept-Language HTTP header tag once client capabilities have been negotiated.

8. Finding out the MSISDN in use

8.1 Rationale

There are several use cases where the server requires the MSISDN of the client Examples of such use cases:

- if the user has several mobiles with different MSISDNs,
- if the server needs to send an WAP Push CIR message to the client,
- to verify the authenticity of a user,
- or to retrieve the credentials of a user.

8.2 Recommendation

Clients SHOULD support and use Standalone SMS Binding (81.4 of [CSP Trans]) – this allows the server to find out the correct MSISDN that the client is using. The server should use the received “HELO” message (and the Session-ID within) to find out the MSISDN used by the client – and store the current MSISDN for any further correspondence that might be necessary.

9. Logging in with username and not knowing the password

9.1 Rationale

"Alice wants to retrieve log in to the service when knowing only the username of her account."

The use-cases described in this chapter allows for the user to log onto the service without having to type in his password. This is considered crucial for end-user take up as end users increasingly do not want to relate to having another set of passwords and usernames and to remember these as he moves from one client to another or changes terminal.

9.2 Use cases and examples

9.2.1 User attempts to log in knowing his User-ID but not his password

Actors	End user, Client, Server
Success Guarantees	End user is logged onto the service
Preconditions	Client, Server, end user's account provisioned in the server. Network authentication is not used in this use case, so the password on the network cannot be empty for a normal login flow to work. If the server stores the password encrypted, then this use case will not work.
Trigger	Step 1
Main Success Scenario	End user launches IM client Client does a login request to the Server with an empty password Server returns a system message, asking the client where he wants his password to be sent, including a list of known MSISDN, e-mail addresses and other online IM clients of that user. Client chooses to get password on MSISDN number or email address Server sends out password on the desired channel Client manually makes another login request with password retrieved from the channel
Extension Scenarios	**Password retrieval through another logged in client** 4b. Client chooses to retrieve password through another online IM session 5b. Server sends a system message to the other client asking him to verify that another client is trying to log onto the service, with answer options "allow" and "deny" 6b. End user presses "allow" 7b. The server allows the user to log in by returning login response with the password of the user for storage in the client **Continuation of the "b" use case:** 6c. End user presses "deny" 7c. The server disallows the user to log in
Variations	
Design Notes	Note that this implies that the server cannot accept empty passwords for users.

Table 4: Credential Retrieval

9.2.1.1 Examples

1 client sends login request with an empty password

```
<Login-Request>
  <User-ID>user_id<User-ID/>
  <Password/>
  <ClientID>client_id</ClientID>
  <SessionCookie>session_cookie</SessionCookie>
</Login-Request>
```

2 Server returns a system message, asking the client where he wants his password to be sent, including a list of known MSISDN, e-mail addresses and other online IM clients of that user

```
<Login-Response>
  <ClientID>client_id</ClientID>
  <Result>
    <Code>436</Code>
    <SystemMessageList>
      <SystemMessage>
        <SystemMessageID>id#0</SystemMessageID>
        <RequiresResponse>T</RequiresResponse>
        <SystemMessageText Where would you like to send your password?</SystemMessageText>
        <AnswerOptions>
          <AnswerOption>
            <AnswerOptionID>1</AnswerOptionID>
            <AnswerOptionText>Telephone number 90000000</AnswerOptionText>
          </AnswerOption>
          <AnswerOption>
            <AnswerOptionID>2</AnswerOptionID>
            <AnswerOptionText>Telephone number 90909090</AnswerOptionText>
          </AnswerOption>
          <AnswerOption>
            <AnswerOptionID>3</AnswerOptionID>
            <AnswerOptionText>Email address me@mail.com</AnswerOptionText>
          </AnswerOption>
          <AnswerOption>
            <AnswerOptionID>4</AnswerOptionID>
            <AnswerOptionText> Email address minime@mail.com</AnswerOptionText>
          </AnswerOption>
          <AnswerOption>
            <AnswerOptionID>5</AnswerOptionID>
            <AnswerOptionText>Send it to all my online clients</AnswerOptionText>
          </AnswerOption>
        </AnswerOptions>
      </SystemMessage>
    </SystemMessageList>
  </Result>
</Login-Response>
```

3 Client chooses to get password on MSISDN number or email address

```
<SystemMessageResponseList>
  <SystemMessageResponse>
    <SystemMessageID>id#0</SystemMessageID>
    <ChosenOptionID >2</ChosenOptionID>
  </SystemMessageResponse>
</SystemMessageResponseList>
```

4 Server sends out password on the desired channel

5 Client manually makes another normal login request with password retrieved from the channel

9.2.1.2 Password retrieval through another logged in client

3. Client chooses to retrieve password through another online IM session

```
<SystemMessageResponseList>
  <SystemMessageResponse>
    <SystemMessageID>id#0</SystemMessageID>
```

```
      <ChosenOptionID>5</ChosenOptionID>
    </SystemMessageResponse>
  </SystemMessageResponseList>
```

4. Server sends a system message to the other client asking him to verify that another client is trying to log onto the service, with answer options "allow" and "deny"

```
<SystemMessageList>
  <SystemMessage>
    <SystemMessageID>id#0</SystemMessageID>
    <RequiresResponse>T</RequiresResponse>
    <SystemMessageText>
      Another client is trying to log onto the service with your username.
      Allow the client to log on?
    </SystemMessageText>
    <AnswerOptions>
      <AnswerOption>
        <AnswerOptionID>1</AnswerOptionID>
        <AnswerOptionText>Allow</AnswerOptionText>
      </AnswerOption>
      <AnswerOption>
        <AnswerOptionID>2</AnswerOptionID>
        <AnswerOptionText>Deny</AnswerOptionText>
      </AnswerOption>
    </AnswerOptions>
  </SystemMessage>
</SystemMessageList>
```

5. End user chooses to allow the log on

```
<SystemMessageResponseList>
  <SystemMessageResponse>
    <SystemMessageID>id#0</SystemMessageID>
    <ChosenOptionID>1</ChosenOptionID>
  </SystemMessageResponse>
</SystemMessageResponseList>
```

6. The client logs in with an empty password again, and succeeds this time. The password will be returned on the Login-Response for safe storage in the client.

```
 <Login-Response>
<User-ID>wv:newuser@imps.com</User-ID>
<Password>password</Password>
<ClientID>client_id</ClientID>
<Result>
  <Code>200</Code>
</Result>
<SessionID>session_id</SessionID>
<KeepAliveTime>3600</KeepAliveTime>
<CapabilityRequest>T</CapabilityRequest>
</Login-Response>
```

9.3 Recommendation

Clients and servers SHOULD support the use cases and use of system messages as described in this section.

10. Multi-sessions

10.1 Rationale

"Alice wants to have her PC and mobile client running at the same time."

The multi-session support is a feature to enable the user to be logged on with several clients at one time. The end user can then be logged on at e.g. his home computer, work computer and mobile phone(s) at the same time. Also, this ensures that if a user updates presence on one client, then the presence attributes are updated on all logged in clients.

Several lists and structures such as contact lists, authorization lists, block/grant list, groups and public profile are attached to the User-ID and no particular client. Since these lists and structures can be altered by any one of the currently logged on clients on a User-ID, the clients need to subscribe to changes for these structures.

10.2 Keeping Presence attributes up-to-date

10.2.1 Use cases and examples

10.2.1.1 End user updates presence attributes on one of his active clients and the change is reflected in other active clients

Actors	End user A, Client A1, Client A2, Server.
Success Guarantees	End user presence update is reflected on all active clients
Preconditions	End user A is logged onto the server with client A1 and A2.
Trigger	Step 1
Main Success Scenario	1. Client A1 and client A2 subscribes to all presence attributes of end user A 2. End user changes presence attribute on client A1 3. Server receives presence updates and generates notifications with the appropriate updates to all active clients 4. Presence update is reflected on client A2 5. Client A2 applies the notified presence changes locally, so that its local set reflects the latest changes.
Extension Scenarios	
Variations	

Table 5: Presence attributes synchronization

10.3 Recommendation

Clients SHOULD subscribe to the presence information of the own User-ID and at least all the user status presence attributes supported by the client. Note that a user can only have one instance of each user status presence attribute (as defined in 8.3 in [PA]) and multiple instances of each client presence attribute (one per client). Clients are RECOMMENDED to only update user status presence attributes when notified about changes on the logged in user.

10.4 Keeping various user-managed things up-to-date

Since presence authorization, block/grant lists, contact lists, groups and public profile can be altered by any one of the currently online clients of a particular User-ID, the clients need to subscribe for such change notifications.

10.4.1 Use cases and examples

10.4.1.1 End user adds a contact to one of his active clients and the change is reflected in other active clients

Actors	End user A, Client A1, Client A2, Server.
Success Guarantees	End user update is reflected on all active clients
Preconditions	End user A is logged onto the server with client A1 and A2.
Trigger	Step 1
Main Success Scenario	1. Client A2 subscribes to Contact-List-Changed notification 2. End user adds a contact on client A1 3. Server receives a contact list update request and generates notifications with the appropriate change to all active clients 4. Contact list update is reflected on client A2 5. The client applies the notified changes locally, so that its local set reflects the latest changes.
Extension Scenarios	
Variations	

Table 6: Contact list synchronization

10.4.2 Recommendation

All clients SHOULD subscribe to any of the following General Notification types which match the service tree of the client:

1. Authorization-Changed,
2. Block-List-Changed,
3. Block-List-UsageChange,
4. Contact-List-Created,
5. Contact-List-Changed,
6. Contact-List-Deleted,
7. Grant-List-Changed,
8. Grant-List-UsageChange,
9. Group-Created,
10. Group-Deleted,
11. Invitation-Accepted,
12. Invitation-Cancelled,
13. Invitation-Rejected,
14. OnlineETEMHandling-Updated,
15. PublicProfile-Updated,
16. Session-Priority-Adjusted
17. Upon receipt of a notification the client SHOULD fetch the updated list/structure on the server.

11. Client-ID

11.1 Rationale

According to IMPS 1.3 CSP [CSP], the Client-ID is a unique identifier of the IMPS client within the scope of a particular user and it must be a URI as defined in [RFC 2396]. Having a consistent way of representing unique Client-IDs simplifies the development of servers and clients. Client and server implementations should support the Client-ID structure defined in this section.

11.2 Recommendation

Clients should generate Client-Ids according to the following ABNF syntax:

Client-ID = "wv:" SW_NAME ":" SW_VERSION ":" OPERATOR_KEY ":" PHONE_VENDOR ":" PHONE_MODEL ":" UNIQUE_ID

SW_NAME = ALPHA *(ALPHA / DIGIT / "-")

SW_VERSION = DIGIT *(“.” DIGIT))

OPERATOR_KEY = ALPHA *(ALPHA / DIGIT / "-")

PHONE_VENDOR = ALPHA *(ALPHA / DIGIT / "-")

PHONE_MODEL = ALPHA *(ALPHA / DIGIT / "-")

UNIQUE_ID = ALPHA *(ALPHA / DIGIT / "-")

If a SW_NAME, SW_VERSION, OPERATOR_KEY or PHONE_MODEL of the Client-ID is not available (not set) then the component will be empty resulting in two consecutive ‘:’ signs.

The meaning of each Client-ID component is described in the following table:

Field	Role	Data type
Wv	IMPS URI scheme	Constant string
SWName	Short name of the client software	Alphanumeric string
SWVersion	Client software base version	Alphanumeric string
OperatorKey	Operator identifier	Alphanumeric string
PhoneVendor	Phone vendor identifier	Alphanumeric string
PhoneModel	Phone model identifier	Alphanumeric string
UniqueID	An identifier to uniquely identify a particular client for a particular user	Alphanumeric string

Table 7: Client-ID components

As an example, for the following Client-ID **wv:ZOMI2.0.1$NoWire@FLY.X95.384759,** the components are shown in the following Table:

wv:	SWName	SWVersion	$	OperatorKey	@	PhoneVendor	.	PhoneModel	.	UniqueID
wv:	ZOMI	2.0.1	$	NoWire	@	FLY	.	X95	.	384759

Table 8: Client-ID example

The generated Client-ID for a particular client within a particular user SHOULD be stored safely on the client device so that it can be re-used across sessions for the user.

In case a server detects that a client is trying to log in with an already registered Client-ID on another session, then servers are RECCOMENDED to “ping” the client owning the already existing session. In case the server detects that the client is not there any more then the attempted login can be granted with the Client-ID and the current session must be terminated. If the client owning the already existing session responds to the “ping” then the attempted login with the new client must be denied with a status indicating that the Client-ID is not unique. “Ping” here means to attempt notify the client with a CIR message. Care must be taken on clients to not accept any CIR message before CIR methods has been negotiated. This is to avoid a situation where the “pinged” client is actually the client logging in.

12. Contact lists usage

12.1 Rationale

"Alice wants to see the same contact list on different clients."

IMPS CSP [CSP] has opened for clients specifying their own contact lists names, authorization lists and subscriptions. The challenge with this is that when an end user switches from one client to another, his contact list might not appear in the same way if at all on the new client. In addition, some client manufacturers use contact lists as place holders for blocked users, whereas others do not and can interpret the blocked list as a separate friends list to be represented in the client.

12.2 Use cases and examples

12.2.1 User logs on with client manufactured by A and then with client manufactured by B

Actors	End user Alice, Client A, Client B, Server.
Success Guarantees	End user sees no difference between the two clients
Preconditions	
Trigger	Step 1
Main Success Scenario	1 Alice logs on to her brand new "A" terminal and retrieves her contact list 2 Alice loves new phones, buys terminal "B" and logs onto the service again, retrieving her contact list
Extension Scenarios	2b. Alice logs on to terminal A and B at the same time, seeing the same contact lists.
Variations	

Table 9: Contact list usage

12.3 Recommendation

Clients SHOULD use a contact list named `wv:userid/oma_allcontacts@domain` to store all non-blocked friends. Friends blocked for presence SHOULD be stored in a contact list named `wv:userid/oma_blockedcontacts@domain`.

Both presence authorization and presence subscription will be done on the `oma_allcontacts` contact list. Presence blocking will be done by moving the friend from the `oma_allcontacts` contact list to the `oma_blockedcontacts` and by assigning the empty authorization on the blocked friend.

12.4 Contact list names

All clients must keep all non-blocked friends in a list named `wv:userid/oma_allcontacts@domain`. It is up to the client if this list should be visible in the user interface or not. If the client supports multiple contact lists (e.g., friends, co-workers) then upon adding a friend to any other list must also result in adding the friend to the `oma_allcontacts` contact list. Blocking a friend for presence will result in moving the friend from the `oma_allcontacts` contact list to the blocked list. Servers can provision the contact lists upon provisioning of the user. Clients who choose to display the list in the user interface must choose a human readable name for the contact list. The contact lists DisplayName property is not to be used.

Presence blocked users must be kept in a list named `wv:userid/oma_blockedcontacts@domain`. Blocking a friend for presence will result in moving the friend from the `oma_allcontacts` contact list to the `oma_blockedcontacts`, and by assigning the empty user authorization to the friend. Unblocking a friend from presence blocking will result in removing the

empty user authorization on the friend and by moving the contact from the `oma_blockedcontacts` to the `oma_allcontacts` contact list.

12.5 Subscription

Presence subscription will be done on the `oma_allcontacts` contact list. Clients SHOULD subscribe to at least the following set of presence attributes:

1. UserAvailability
2. StatusText
3. CommCap
4. OnlineStatus
5. ClientType
6. ClientInfo

Clients SHOULD not subscribe for presence on other contact lists or on users directly.

12.5.1 Authorization

Clients SHOULD authorize for presence on the `oma_allcontacts` contact list. Blocked contacts will be moved from the `oma_allcontacts` contact list to the `oma_blockedcontacts` and clients SHOULD assign the empty presence authorization on users in the `oma_blockedcontacts`. Since presence authorizations are shared among clients on the same user, clients SHOULD fetch the current authorization on the `oma_allcontacts` contact list and only increase the authorization set if to include the required attributes by the client if needed. Clients SHOULD authorize at least the following the of presence attributes:

1. UserAvailability
2. StatusText
3. CommCap
4. OnlineStatus
5. ClientInfo

Clients SHOULD have the possibility to authorize for presence on other lists that the `oma_allcontacts` lists.

13. Presence attribute interpretation

13.1 Rationale

"Alice wants to log on as invisible to everyone else"

Having concise and consistent presence attributes ensures end users the possibility to portray their willingness to communicate to friends on his contact list. Slightly different presence attributes have been used by client manufacturers to signify an available IM client, and few manufacturers have defined an invisible attribute.

13.2 Background

From the Presence Enhanced Phone book Application Profile [PEP] and the Instant Message Application Profile [IM] the following are defined:

- AVAILABLE: (as defined in section 8.3.1 of [PA]) Publisher is available with the means available in his/her device
- DISCREET: (as defined in section 8.3.1 of [PA]) Publisher has selective availability to communication means or to contacting parties. By setting this value, the publisher is asking for consideration before a communication is initiated to him/her or when he/she doesn't respond to communication. The exact nature of the users communication status can be clarified using the status message. Some example use cases are:
  - The publisher prefers to receive text messages rather than voice call because he's in a meeting.
  - The publisher is busy and wishes to receive only urgent communication.
  - The publisher is selective about the communication parties to whose communication he responds.
- NOT_AVAILABLE: (as defined in section 8.3.1 of [PA]) Publisher is not immediately available with the communication means in his/her device. The contacting party should not expect an immediate response/reaction by the publisher. (as defined in section 8.3.1 of [PA])
- UNKNOWN: This value shows the publisher might not be logged onto the presence service and thus PEP is not able to provide any presence information about the publisher. When publisher does not have an active OMA Imps session the UserAvailability is replaced with this "UNKNOWN" indicator.

In addition, being connected to the IM server as invisible implies that the user appears as off-line everyone's contact list, but that the user himself receives presence updates and can send and receive IMs normally.

UserAvailability is a User attribute and is therefore shared across multiple clients on the same account. Since UserAvailabilty is the "steering" attribute the user can only be DISCREET or NOT_AVAILABLE on all clients.

13.3 Recommendation

Clients wishing to signalize their user availability SHOULD set the following presence attributes:

Availability Status	OnlineStatus		CommCap/IM		UserAvailability	
	Value	Qualifier	Value	Qualifier	Value	Qualifier
NOT_AVAILABLE	T	T	OPEN	T	NOT_AVAILABLE	T
DISCREET	T	T	OPEN	T	DISCREET	T
AVAILABLE	T	T	OPEN	T	AVAILABLE	T
INVISIBLE	Ignored	F	CLOSED	T	NOT_AVAILABLE	F

Table 10: Presence attributes interpretation

Note that all other combination of the OnlineStatus, CommCap/IM and UserAvailability attributes SHOULD be interpreted as offline.

Clients who want to appear as invisible SHOULD set the Qualifier of OnlineStatus to “F”. To support legacy devices adhering to older recommendations, servers SHOULD interpret a client setting CommCap/IM to “CLOSED” as also setting the Qualifier of OnlineStatus to “F”. Invisible means here that presence watchers will not be able to detect that the client is online.

In order to support invisibility directly when logging in, servers are RECOMMENDED to set the value of the Qualifier of OnlineStatus as “F” and set OnlineStatus to “F” when a client logs in, and change the qualifier and and OnlineStatus presence value to “T” once the client sends the first presence publishing primitive. If there was no presence support negotiated during service negotiation, then the server must set OnlineStatus to ‘T’.

Servers SHOULD not send presence notifications to watchers of a user that is currently set to invisible. When the user becomes visible again, then servers should send presence notifications to all watchers about the changed attributes. Servers can either remember the set of changed attributes and only send the delta of the updated attributes or send the entire set of presence attributes.

14. Extension Presence Attributes

14.1 Idle state

Idle state is about a clients attempt to indicate to watching users about the users activity on the client. On fixed line IM system it is very common for desktop clients to indicate idle state when the user has not used the client and/or the computer (desktop) for a period of time. This is to give a hint to the watchers that the user might not be around to receive messages. On mobile clients this can easily be translated to when the IM client is minimized or when the user locks the keypad of the mobile device (black screen). Idle state is important not only to watchers but also to servers since they can make routing decisions on IMs based on (among other things) the idle state of the clients.

If a client wants to indicate whether the user is idle, i.e. have not used the computer or device for a certain amount of time, then the client SHOULD use an extension to the PresenceSubList using the namespace `http://www.openmobilealliance.org/DTD/IMPS1.3-IG1.0`. The DTD for this namespace is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT IdleState (Qualifier?, PresenceValue?, ClientID?)>
<!ELEMENT IdleSince (Qualifier?, PresenceValue?, ClientID?)>
```

Defined information elements are:

Information element	IdleState
Data type	Boolean
Format	Following values: T – The user of the client is idle. F – The user of the client is not idle.
Description	The idle state of the user of the client
Range	

Table 11: IdleState information element

Information element	IdleSince
Data type	Date and Time, see 4.5 [CSP DataType]
Format	Defined in 4.5 [CSP DataType]
Description	The date and time from when the client went idle
Range	

Table 12: IdleSince information element

Clients SHOULD publish the IdleState presence value. The actual idle time from that the user is idle to when the IdleState is set to “T” is implementation specific. The server MAY also publish the IdleSince element, indicating with a time when the user went idle.

Clients that want to retrieve the idle state of other users SHOULD subscribe to the IdleState and IdleSince presence attributes. Absence of the Presence-Attribute-List in a SubscribePresence-Request or a GetPresence-Request (aka blank get/subscribe request) will not include these extension attributes. The extension attributes must be subscribed to/fetch explicitly.

The value of IdleSince SHALL be considered invalid when the IdleState Qualifier is set to “F”.

14.2 Examples

The client publishes the idle state of the user resulting in a notification sent out to watchers:

```
<PresenceSubList
    xmlns="http://www.openmobilealliance.org/DTD/IMPS-PA1.3"
    xmlns:ig="http://www.openmobilealliance.org/DTD/IMPS1.3-IG1.0">
  <ig:IdleState>
    <Qualifier>T</Qualifier>
    <PresenceValue>T</PresenceValue>
    <ClientID>foo</ClientID>
  </ig:IdleState>
  <ig:IdleSince>
    <Qualifier>T</Qualifier>
    <PresenceValue>2008-05-13T14:45:03Z</PresenceValue>
    <ClientID>foo</ClientID>
  </ig:IdleSince>
</PresenceSubList>
```

15. Discovery of other clients capabilities

15.1 Rationale

"Alice wants to know whether Bob can receive the picture she wants to send him."

End to end messaging enriches the messaging experience as it enables a client to be aware of the capabilities of the recipient client. Typically, if Alice is chatting with Bob, the clients would signalize the capabilities of the other client to the end user via icons, for instance greying out the "send picture" icon to indicate that the other client does not support pictures.

Clients SHOULD authorize and subscribe to the ClientInfo presence attribute in which servers will publish client capabilities (see chapter 8.1.1 in [PA]). Note this provides the ability for a user to online with clients with different capabilities.

15.2 Use cases and examples

Alice's client supports pictures and text and chats with Bob who is logged on with 2 clients having different capabilities.

Actors	End user Alice, end user Bob, client A, client B1, client B2, Server.
Success Guarantees	Messages Alice send are delivered to the device supporting the content of the IM
Preconditions	Alice is logged on to the service with client A. Bob is logged on to the service with clients B1and B2 respectively Client A supports text and pictures Client B1 supports text Client B2 supports text and pictures
Trigger	Step 1
Main Success Scenario	1. Alice starts a chat dialogue with Bob 2. Alice sees in her client that Bob can receive text and pictures 3. Alice send a text message to Bob 4. Client A sets Bob as the recipient (no specific client) 5. The server uses Bobs OnlineETEMHandling setting to route the message to both client B1 and B2 6. Bob receives the text message on client B1 and B2 7. Alice sends a picture message to Bob 8. Client A sets Bob on client B2 as the recipient 9. Bob receives the picture message on client B2
Extension Scenarios	
Variations	

Table 13: End-to-end messaging (text and multimedia)

15.3 Recommendations

Servers SHOULD publish clients supported capabilities (including content types) in the ClientInfo->ClientContentLimit-presence attribute field.

Clients SHOULD enable and disable functions (e.g., send picture) in the user interface according to what communication parties support.

16. Rich content IM

16.1 Rationale

"Alice wants to nudge Bob."

A complete messaging experience allows the use of rich content from client to client. This implies being able to send typing alerts and formatted text from client to client, regardless of who the manufacturer of the client or the server is.

Nudges are “intrusions” on a conversation party's instant message user interface (message dialogue). Examples of nudges are shake, bump, moo, fart and honk.

16.2 Formatted Plain Text Messages

To allow for in-text formatting of plain text instant messages as bold, italic and underline, clients SHOULD adhere to the following set of rendering rules for visual formatting of instant messages,

- Formatting rules only applies to instant messages sent as plain text (i.e., the MIME type text/plain),
- Text surrounded by the asterix character (*) SHOULD be displayed as **bold**,
- Text surrounded by the fore slash character (/) SHOULD be displayed as *italic*,
- Text surrounded by the underscore character (_) SHOULD be displayed as underlined,
- Text surrounded by any combination of the formatting characters should be displayed with the combined formatting (e.g., ‘*/bold-italic/*’ would be rendered as ‘***bold-italic***’)
- Any character except white space and line break can be placed within the formatting characters,
- Formatting characters SHOULD not be visible in the user interface on clients supporting formatting
- Two consecutive formatting characters SHOULD be rendered as the character itself in the clients user interface. This allows for effectively escaping the formatting interpretation of the character.

16.2.1 Formatting Plain Text Message Example

The following message will appear as “Do you **really** want to come tonight?” in Bobs client.

```
<SendMessage-Request>
    <DeliveryReport>F</DeliveryReport>
    <MessageInfo>
        <ContentType>text/plain</ContentType>
        <ContentSize>37</ContentSize>
        <Recipient>
            <User>
                <UserID>wv:bob@imps.com</UserID>
            </User>
        </Recipient>
        <Sender>
            <User>
                <UserID>alice@imps.com</UserID>
            </User>
        </Sender>
    </MessageInfo>
    <ContentData>Do you *really* want to come tonight?</ContentData>
</SendMessage-Request>
```

16.3 Typing alerts

Typing alerts are informational instant messages sent between clients involved in a conversation to indicate to the recipient party about if the sending party is typing or not. Especially in a mobile context typing alerts are valuable to the recipient user since it can be expected that typing a message will take somewhat longer time than if done on a computer.

Typing alert messages are transported in CSP as instant messages with a content type of `application/vnd.oma.imps.typing-alert` and with no content.

A recipient client SHOULD indicate that a sender is typing upon receiving a typing alert. If no new instant message of any content type is received from the sending client within 20 seconds, the recipient client SHOULD change the typing indication to indicate that the sending party has typed (or started to type) a message. Furthermore, if no new instant message (of any content) has been received after 60 seconds from when the last message was received, then the client SHOULD remove the typing indication.

Upon receipt of an instant message (of any content) from the sending client, the recipient client SHOULD remove the typing alert (regardless of state), if any such indication exists.

The sending client must send a typing alert whenever the end user starts to compose a new instant message to a recipient. If the user is still typing after 10 seconds, the sending client SHOULD send a new typing alert to the recipient party. Furthermore, if the end user chooses to erase all of the written content or close the composer window, then the recipient client will by the above rules first set the state of the sender as “has typed on a message” and then to no typing alert state. Note that simply erasing portions of the typed message is still considered as typing.

Clients SHOULD avoid sending a typing alert as the first message in a message dialog. In some cases this might be difficult (e.g., when responding to a message received some time ago), and clients who receive the very first typing alert from a sender (i.e., there exist no indication in the UI that the sender and the recipient has an ongoing dialog) SHOULD ignore the typing alert.

Typing alerts SHOULD only be sent to clients who indicate support for the typing alert content type and where the recipient is online and available as indicated by the OnlineStatus and UserAvailability presence attributes.

Clients must follow the messaging context rules as defined in [CSP] when sending typing alerts. In particular clients can send typing alerts into a group or when whispering within a group.

16.3.1 Typing Alert Example

Bob starts typing a message in his message composer, and a typing alert is sent to the recipient Alice:

```
<SendMessage-Request>
    <DeliveryReport>F</DeliveryReport>
    <MessageInfo>
        <ContentType>application/vnd.oma.imps.typing-alert</ContentType>
        <ContentSize>0</ContentSize>
        <Recipient>
            <User>
                <UserID>alice@imps.com</UserID>
            </User>
        </Recipient>
        <Sender>
            <User>
                <UserID>bob@imps.com</UserID>
            </User>
        </Sender>
    </MessageInfo>
    <ContentData> </ContentData>
</SendMessage-Request>
```

The terminal that Alice uses indicates that Bob is typing a message to Alice. Bob stops to type immediately after the first couple of words. This makes Alice’s terminal show the “has typed text” indicator. Bob continues to type on his message to Alice and chooses to send the IM to Alice. This will upon receipt in Alice’s terminal erase the typing indicator on Bob.

16.4 Nudges

A nudge is a message triggering visual, auditory and if possible tactile effects, designed to grab the attention of the receiving user.

Nudging allows a user to get the attention of another user with whom s/he is having a conversation. A nudge is an IMPS instant message with a specific MIME type. A nudge can be a vibration, a sound, an animation, and visual modifications of the UI or any combination of these. To get the attention of a contact, the client sending the nudge specifies the nudge MIME type with an optional contents body. The content body contains an optional nudge ID identifying a particular sound or action. When an IMPS client receives a nudge, it causes the client to do an action according to the nudge identifier.

The client sending a nudge SHOULD limit the number of nudges it sends per period of time to avoid annoying the recipient. The nudging frequency is an implementation choice. A client SHOULD allow a user to turn-off the nudging facility to avoid receiving nudges. When nudging is turned off, any instant message carrying a nudge SHOULD be ignored.

Message Content	Nudge type	Suggested sound	Suggested image/animation or visual modifications
1	Vibrate	Device vibrates or emits a vibrating sound	Chat window vibrates
2	Bark	Resembling the sound made by a dog	Picture/animation of a dog barking
3	Boom	Sound of an explosion	Picture or animation of a bomb going off or chat window exploding
4	Moo	Noise as uttered by a cow	Picture/animation of a cow mooing, or possibly walking across the chat window
5	Moan	A low, sustained, mournful cry of pleasure	Picture/animation of something pleasurely.
6	Vroom	Similar sound to that of a racing car passing by	Picture or animation of a racing car
7	Buzz	Low, continuous, humming or sibilant sound, like that made by bees with their wings (as in a annoying fashion)	Picture/animation of a bee flying in
8	Fart	Device emits a crackling or trumpeting sound (sound like the auditory pitch of a flatulence outburst)	Picture/animation of an embarrassed chatter
9	Ring Phone	Device sound the default ringing tone (just as if the user had a phone call).	
10	Ring SMS	Sound of the default SMS tone (just as if the user had an SMS).	
11	Ring IM	Sound of the default IM tone (just as if the user had an IM).	
12	Ring Mail	Sound of the default email tone (just as if the user had an email).	
13	Beep	Sound of a beep.	
14	Type	Sound as if the user was typing on the keypad or a typewriter.	
15	Whistle	Sound as a whistle.	

Table 14: OMA-IMPS nudge types

A client SHOULD send a nudge in a SendMessageRequest primitive; the content type SHOULD be "`application/vnd.oma.imps.nudge; vendor=oma-imps`". The `vendor=oma-imps` defines 15 well defined nudges. The content of the nudge is defined by the number 1 through 15 as defined in the table below:

The default nudge is the "`application/vnd.oma.imps.nudge; vendor=oma-imps`" with message content = 1.

Different values for the vendor CAN be supported. For instance the MIME type "`application/vnd.oma.imps.nudge; vendor=operator-a`" defines the set of nudges that are defined by the operator a.

Clients indicate what types of nudges they support during the content negotiations at login. If the recipient client does not support a certain type of nudges, then the server SHOULD emit the default nudge if the client supports this. For instance, for a client made by vendor Rec, the following nudges supported could be negotiated at login: "`application/vnd.oma.imps.nudge; vendor=oma-imps`, `application/vnd.oma.imps.nudge; vendor=rec`".

This indicates the client supports both the OMA standard defined nudges as defined in this document, and the nudges that vendor Rec has defined.

If the nudge mime type is specified without any vendor parameter it is to be considered as with the vendor parameter oma-imps.

17. Group usage

17.1 Rationale

"Alice wants to create chat rooms in the same manner Bob does."

"Alice wants to chat with Bob and Clare at the same time."

Previous client implementations of groups have been varied, with the result that groups do not always work when chatting with users having clients from different manufacturers.

In addition, ad-hoc groups have not been implemented by all client manufacturers.

17.2 Recommendation

By default, new groups created by clients, SHOULD be private, open and searchable, with private messaging set to 'T' (true). Clients can offer end-users the option of creating non-searchable groups or groups which does not allow private messaging.

The clients SHOULD support whisper, the server SHOULD support whispers.

Clients SHOULD support invitations to groups.

Client SHOULD support group change notifications.

18. SMS only users

18.1 Rationale

SMS only users are subscribers who have been automatically registered with the IM service. The registration occurs when an IM user adds or searches for a friend by MSISDN and the MSISDN has not been provisioned in the IM system yet. They can only receive and send/reply messages on SMS.

IM users can view SMS only users as any other IM service registered users. They can add SMS only users to contact lists, initiate IM dialogs with SMS only users, see SMS only users presence and block/unblock SMS only users.

Messages sent from an IM user to an SMS only user will be forwarded to the SMS only user on SMS. Replies from the SMS only user sent on SMS will be delivered as instant messages to the IM user.The SMSes are processed and received by the IM users as SMSes and IMs can be sent back to SMS only users that receive them as SMSes.

18.2 Recommendation

Servers SHOULD provide presence state of SMS only users in accordance with GSMA Phase 2 Service definition [GSMAPH2], also, since the Client-ID element is required for client presence attributes in OMA IMPS 1.3, it is RECOMMENDED that all client presence attributes on SMS only users must use a Client-ID containing the MSISDN (international format) of the SMS only users.

- OnlineStatus: Qualifier=T, PresenceValue=T, ClientID=MSISDN,
- ClientInfo: Qualifier=T, ClientType=CLI, ClientID=MSISDN,
- UserAvailability: Qualifier=T, PresenceValue=AVAILABLE,
- StatusText: Qualifier=T, PresenceValue=SMS-only,
- CommCap: Qualifier=T, Cap=SMS, Status=Open

Clients SHOULD visually display SMS only users with an indication that they are on SMS.

19. Extending clients with custom menu items

19.1 Rationale

"An Operator wants the end user to be able to perform self administration from the IMPS client."

In several operators' network operators wish to link the use of the service with service provider specific services. This could be for instance self-administration of the end user's subscription, or self-administration of the users' private profile containing more data about the user than what is contained in the IM client.

Self administration is here suggested solved by allow for the operator to provide a customized menu in the client where each item in the menu is a link to a WAP/WEB page.

19.2 Recommendation

IM clients SHOULD negotiate support for the OMA IMPS extended service content type. The content type to be negotiated is

```
application/vnd.oma.imps.extendedservice
```

Servers who understand this content type and who has enabled extended services SHOULD as soon as possible send an IM to the client with the extended service content. Clients who receive such content SHOULD be prepared to handle the IM differently than any other IM destined for the user. Note that the content type can only be negotiated if the device the client runs on a device that supports a WAP/WEB browser.

The IM sent to the client containing the extended service must be addressed to the particular user, the content type SHOULD be `application/vnd.oma.imps.extendedservice`, and the content SHOULD follow the following ABNF format:

```
EXTENDED_SERVICE_MESSAGE = “MENUITEM:” MENU_TEXT “: (URL

MENU_TEXT = *(ALPHA / DIGIT/ SP)

URL = see ABNF for URL in [RFC 3986]
```

MENU_TEXT is the actual text to display on the client UI and the URL is a link to a WAP/WEB page where the user can perform the extended service.

Clients SHOULD present the extended service IM as a menu item. The menu item should be rendered with the server proposed menu text. Clients SHOULD open the WAP/WEB browser with the menu items URL when the user selects the menu item.

Clients who do not have menu support SHOULD NOT negotiate support for the content type.

Example of a customized URL IM:

```
<NewMessage>
    <MessageInfo>
        <MessageID>0x0000f132</MessageID>
        <ContentType>application/vnd.oma.imps.extendedservice</ContentType>
        <ContentEncoding>None</ContentEncoding>
        <ContentSize>41</ContentSize>
        <Recipient>
            <User>
                <UserID>wv:someuser@domain</UserID>
            </User>
        </Recipient>
        <Sender>
            <User>
                <UserID>wv:system@domain</UserID>
            </User>
        </Sender>
        <DateTime>20010925T1340Z</DateTime>
        <Validity>600</Validity>
```

```
            </MessageInfo>
            <ContentData>
               MENUITEM:Manage My Subscriptions:URL:http://wap.someserver.com/subscription
            </ContentData>
         </NewMessage>
```

Clients SHOULD store the menu item between sessions to avoid resending of the menu item upon each login. When a client receives a new menu item while already having one stored, then the previous menu item should be replaced by the newly received menu item. Servers SHOULD register that a particular client defined by the clients Client-ID has received a menu message and servers SHOULD not re-send the menu message unless there are any changes to the menu.

Clients SHOULD in no circumstances make the IM visible to the end user.

20. Service provider network indication

20.1 Rationale

"Alice wants to know whether Bob is on her price plan or not."

Service providers increasingly have advantageous price plans for end users that all belong to the same service provider. In some cases, these involve free calls to all the people having this price plan. However, it is not easy for the end user to know who belongs to what price plan. This can be solved by using a separate presence field to indicate the service provider the user belongs to.

20.2 Recommendation

If the server makes use of the PLMN field, clients SHOULD use it to represent these contacts with a different icon than the others.

21. Optimized network usage

21.1 Rationale

"Alice wants to send a picture 33% faster than her old client does."

Most clients use BASE64 encoding for multi-media content on CSP requests. This leads to a 33% overhead on the network transmission of multi-media e.g. pictures, files and sound content. For plain XML there is no other way of transporting binary content, however, the WBXML specification allows for use of inline opaque data. This means that clients and servers can embed binary content “as is” without applying any transfer encoding.

The WBXML specification has no notion on required or optional features and the OMA IMPS specification does not mention this in particular. Also, existing clients on IMPS 1.1/1.2.1 have not used this feature. This implies that a server cannot use inline opaque data towards clients since a server cannot be sure that the client understands inline opaque data.

21.2 Recommendation

To indicate support for inline opaque data transfer clients and servers SHOULD negotiate the IDENTITY transfer encoding during client capability negotiation.

22. Efficient network usage when idle

22.1 Rationale

"Alice wants to use her phone for days and days without re-charging the battery."

The optimal usage of the CIR mechanisms leads to the ability of the client to be "always on" without consuming battery resources on the terminal when the client is not in active use.

22.2 Recommendation

Clients SHOULD use at least one IP based CIR method for intensive periods of usage of the client, and clients SHOULD in addition use at least one SMS based CIR method for periods between intensive periods of usage of the client. Also, clients SHOULD actively takes down the IP based CIR mechanism when it deems that the client is not in intensive use, in which case the server reverts to a SMS based CIR mechanism.

Provided the client supports more than one CIR mechanism at a time, then the client SHOULD prioritize between the different CIR mechanisms in the following order:

- STCP
- SUDP
- WAPUDP
- WAPSMS
- SSMS
- SHTTP

As given in the specification a server is to use CIR methods in the order and with availability as enlisted by the client.

23. Standalone SMS CIR compatibility

23.1 Rationale

"Alice wants to use a third-party IM application, which conflicts with the native IM application."

The registered port for the standalone SMS CIR channel is 3716. For a third party application it may be impossible to listen to incoming SMSs to this port, since a native application (or another third party application) is already using it. There is a need for specifying an alternative port to be used for standalone SMS CIR.

23.2 Recommendation

Clients SHOULD try to use the reserved port (3716) for standalone SMS CIR. However, if it's impossible to register to that port, then clients SHOULD negotiate another port using an extension block to the ClientCapability-Request.

The server SHOULD respond with the same port number in an extension block to the ClientCapability-Response, and send any standalone SMS CIR messages to the indicated port.

If the server does not support these implementation guidelines, the extension block will be ignored and the server will respond without the SMS port in the response. Clients SHOULD then renegotiate to other appropriate CIR methods.

The following is the DTD for the extension block:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT SMSPort (#PCDATA)>
```

Defined information elements are:

Information element	SMSPort
Data type	Integer
Format	An integer expressed in decimal format.
Description	The client may indicate that it supports other than the default port for the standalone SMS CIR method.
Range	49152 – 65535 (see [IANAPorts]

Table 15: SMSPort information element

23.3 Examples

Client sends the ClientCapability-Request indicating that it will listen to SSMS messages on port 50000:

```
<Transaction>
  <TransactionDescriptor>
    ...
  </TransactionDescriptor>
  <TransactionContent>
    <ClientCapability-Request>
      <CapabilityList>
        ...
        <SupportedCIRMethod>STCP</SupportedCIRMethod>
        <SupportedCIRMethod>SSMS</SupportedCIRMethod>
      </CapabilityList>
    </ClientCapability-Request>
  </TransactionContent>
  <ExtBlock xmlns="http://www.openmobilealliance.org/DTD/IMPS1.3-IG1.0">
    <SMSPort>50000</SMSPort>
  </ExtBlock>
</Transaction>
```

Server responds with the Client-Capability-Response:

```
<Transaction>
  <TransactionDescriptor>
    ...
  </TransactionDescriptor>
  <TransactionContent>
    <ClientCapability-Response>
      <AgreedCapabilityList>
        ...
        <SupportedCIRMethod>SSMS</SupportedCIRMethod>
      </AgreedCapabilityList>
    </ClientCapability-Response>
  </TransactionContent>
  <ExtBlock xmlns=”http://www.openmobilealliance.org/DTD/IMPS1.3-IG1.0”>
    <SMSPort>50000</SMSPort>
  </ExtBlock>
</Transaction>
```

24. Advice of charge

24.1 Rationale

Alice wants to send a large file to Bob, which is more expensive than a text IM, and the operator notifies her of the cost.

System Messages can be displayed to the user at any time during the session. One of the usages could be to advice the user of the cost of a certain action. For example when the user sends a file as an IM the operator could advice the user of the cost of this action.

This section intends to clarify how advice of charged can be used using System Messages.

The System Message can optionally include options to “always allow extra charge for this content type”, “always allow extra charge when sending a message to this user”, “always allow costs that are equal or less to this amount <enter amount below>”, “do not notify me until I reach this amount of additional cost <enter amount below>” , etc, to avoid spamming the user with advice of charge messages.

24.2 Use cases and examples

24.2.1 End user is advised of the cost of the action

Actors:	End user, Client, Server
Success Guarantees:	End user is advised of the cost of the action
Preconditions:	Client, Server
Trigger:	Step 1
Main Success Scenario:	1. End user sends a file as an IM 2. Client sends a SendMessageRequest to the server 3. Server responds with a Status 100 and includes a System Message asking the user to accepted the cost additional cost(s) associated with delivering the content 4. The client presents the System Message to the user 5. The user accepts the additional cost indicated in the System Message 6. The client responds to the server with a SystemMessageUser transaction conveying the user’s choice 7. The server responds with a SendMessageResponse indicating success
Extension Scenarios:	**User rejects the cost** 5b. The user rejects the additional cost indicated in the System Message 6b. The client responds to the server witha SystemMessageUser transaction, conveying the user’s choice 7b. The server responds with a SendMessageResponse indicating failure
Variations:	
Design Notes:	

Table 16: Advice of charge

24.2.2 User accepts cost example

Client sends a SendMessageRequest to the server

```
<SendMessage-Request>
  <DeliveryReport>T</DeliveryReport>
```

```
  <MessageInfo>
    <ContentType>image/jpeg</ContentType>
    <ContentEncoding>BASE64</ContentEncoding>
    <ContentSize>1048576</ContentSize>
    <Recipient><User><UserID>wv:bob@imps.com</UserID></User></Recipient>
    <Sender><User><UserID>wv:alice@imps.com</UserID></User></Sender>
    <Validity>600</Validity>
  </MessageInfo>
  <ContentData>...</ContentData>
</SendMessage-Request>
```

Server responds with a Status 100 and includes a System Message asking the user to accepted the cost of the message

```
<Status>
  <Result>
    <Code>100</Code>
    <SystemMessageList>
      <SystemMessage>
        <SystemMessageID>0x1234</SystemMessageID>
        <SystemMessageText>Sending this message will cost you $1.</SystemMessageText>
        <AnswerOptions>
          <AnswerOption>
            <AnswerOptionID>0</AnswerOptionID>
            <AnswerOptionText>Accept<AnswerOptionText>
          </AnswerOption>
          <AnswerOption>
            <AnswerOptionID>1</AnswerOptionID>
            <AnswerOptionText>Decline<AnswerOptionText>
          </AnswerOption>
        </AnswerOptions>
      </SystemMessage>
    </SystemMessageList>
  </Result>
</Status>
```

4. User accepts the System Message and the client sends a SystemMessageUser to the server

```
<SystemMessage-User>
  <SystemMessageResponseList>
    <SystemMessageResponse>
      <SystemMessageID>0x1234</SystemMessageID>
      <ChosenOptionID>0</ChosenOptionID>
    </SystemMessageResponse>
  </SystemMessageResponseList>
</SystemMessage-User>
```

5. Server responds with a SendMessageResponse indicating success

```
<SendMessage-Response>
  <Result>
    <Code>200</Code>
    <Description>Successfully completed.</Description>
  </Result>
  <MessageID>0x0000f132</MessageID>
</SendMessage-Response>
```

24.2.3 User rejects cost example

4b. User rejects the System Message and the client sends a SystemMessageUser to the server

```
<SystemMessage-User>
  <SystemMessageResponseList>
    <SystemMessageResponse>
      <SystemMessageID>0x1234</SystemMessageID>
      <ChosenOptionID>1</ChosenOptionID>
    </SystemMessageResponse>
  </SystemMessageResponseList>
</SystemMessage-User>
```

5b. Server responds with a SendMessageResponse indicating failure

```
<SendMessage-Response>
  <Result>
```

```
      <Code>540</Code>
      <Description>Cost not accepted</Description>
    </Result>
    <MessageID>0x0000f132</MessageID>
  </SendMessage-Response>
```

25. Large Contact Lists

25.1 Rationale

"Alice wants to see the same contacts on her phone as on her PC client."

Limited memory on mobile handsets can introduce a challenge in having long contact lists displayed in the client. The client needs to store presence state and possibly dialog state on each contact.

25.2 Recommendation

The client SHOULD inform the server of the maximum number of contacts it can handle. This SHOULD be done when the client does the ClientCapability request during login by using the MaxContacts element in the ExtBlock element.

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT MaxContacts (#PCDATA)>
```

Defined information elements are:

Information element	MaxContacts
Data type	Integer
Format	An integer expressed in decimal format.
Description	The client may indicate the maximum number of contacts it can handle
Range	Defined in section 4.2 in [DataTypes]

Table 17: MaxContacts information element

The server SHOULD not store the the MaxContacts value between sessions. Also, the setting only applies to the client who set the value. The value must not be disclosed to any other client of the user.

If a the user has more contacts in any contact list than allowed in the client, the server SHOULD prioritize the contacts in an optimal manner, and only deliver up to the maximum number of contacts to the client on a contact list get request (ListMange-Request). When a server finds it necessary to update the the set of contact enlisted in the client, then the server SHOULD inform the client about changes in the contact list by issuing a ContactList-Changed general notification to the client.

The MaxContacts property must be considered as an overall number of friends which the client can handle, and hence it is the limiting factor in a client and not the number of friends on a particular list.

The rules for generating the selection of contact on the server side can be based on actions taken by the client and events happening on any of the friends in the contact list set. Events which may affect the selection of contacts to present to the client can be any one of:

- Addition, removal of friends,
- Blocking/unblocking of friends,
- IM received from or sent to friends,
- Presence updates on friends (e.g, online/offline status changes)

Server implementation of the selection algorithm SHOULD use some kind of heuristic to choose the best selection of friends to present to the client (e.g, choose friends whom the user has often exchanged instant messages with).

25.2.1 Example

The user logs in with a client on that only support 30 contacts. The follow example shows how the client informs the server about this limit by setting the MaxContacts property on the ClientCapability-Request:

Example of a ClientCapability-Request with the MaxContacts element present:

```
<Transaction>
  <TransactionDescriptor>
    ...
  </TransactionDescriptor>
  <TransactionContent>
    <ClientCapability-Request>
      <CapabilityList>
        ...
      </CapabilityList>
    </ClientCapability-Request>
  </TransactionContent>
  <ExtBlock xmlns=”http://www.openmobilealliance.org/DTD/IMPS1.3-IG1.0”>
    <MaxContacts>30</MaxContacts>
  </ExtBlock>
</Transaction>
```

25.3 Content-types defined

Content type	Usage	Reference
application/vnd.oma.imps.typing-alert	Typing alert IMs	Section 16.3
application/vnd.oma.imps.nudge; vendor=oma-imps	Nudge IMs	Section 16.4
application/vnd.oma.imps.extendedservice	Extended service menu IMs	Section 19.1

25.4 DTDs

Extension to the PresenceSubList defined within the namespace `http://www.openmobilealliance.org/DTD/IMPS1.3-IG1.0` is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT IdleState (Qualifier?, PresenceValue?, ClientID?)>
<!ELEMENT IdleSince (Qualifier?, PresenceValue?, ClientID?)>
```

Extension to to the ExtBlock defined within the namespace `http://www.openmobilealliance.org/DTD/IMPS1.3-IG1.0` is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT SMSPort (#PCDATA)>
<!ELEMENT MaxContacts (#PCDATA)>
```

Appendix A. Change History (Informative)

Document Identifier	Date	Sections	Description
OMA-WP-IMPS_V1_3_Impl	22 Dec 2009	all	Approved by OMA TP OMA-TP-2009-0554- INP_IMPS_V1_3_IMPL_V1_0_RRP_for_Final_Approval