Messaging:1 Service

For UPnP Version 1.0
Status: Standardized DCP (SDCP)
Date: March 22, 2011
Document Version: 1.0
Service Template Version: 2.00

Authors[1]	Company
Mayuresh Patil (editor)	Samsung Electronics
Mahfuzur Rahman	Samsung Electronics
Jooyeol Lee	Samsung Electronics
Jeyoung Maeng	Samsung Electronics
Enrico Grosso (editor)	Telecom Italia
Davide Moreo	Telecom Italia
Massimo Messore (editor)	Telecom Italia
Alessandro De Vincentis	Telecom Italia
Yoshiki Nishikawa	NTT
Yu Zhu	Huawei

[1] Note: The UPnP Forum in no way guarantees the accuracy or completeness of this author list and in no way implies any rights for or support from those members listed. This list is not the specifications' contributor list that is kept on the UPnP Forum's website.

Contents

List of Tables

List of Figures

1 Overview and Scope

This service definition is compliant with the UPnP Device Architecture version 1.0. It defines a service type referred to herein as *Messaging* service.

1.1 Introduction

The *Messaging* service is a UPnP service that allows control points to use the messaging features provided by a Telephony Server (TS) or a Telephony Client (TC).

It provides in the UPnP network the overall set of messaging capabilities of a phone (e.g., smartphone, IP phone, VoIP gateway, etc.), as the role of a TS. Additionally, or as an alternative, a device with the role of a TC can provide the *Messaging* service via UPnP *Telephony*.

This service provides the control points with the following functionalities:

- Page Mode Messaging (e.g., e-Mail, SMS, MMS, etc.): notification of incoming messages, reading messages, sending a message, deleting a message.
- Session Mode Messaging (e.g., Instant Messaging, SMS, etc.): notification of incoming messages, retrieving and sending the messages within a session, creating, modifying or closing a messaging session.
- File transfer session, to transfer the files in near real time.

This service does not provide the following functionalities:

- Connection to a remote server for sending or receiving the messages (e.g., WAN side);
- Rendering of message notifications and messages to the end user;
- Configuring the *Messaging* service with account for connecting to remote servers;

1.2 Notation

- In this document, features are described as Required, Recommended, or Optional as follows:

  The key words “MUST,” “MUST NOT,” “REQUIRED,” “SHALL,” “SHALL NOT,” “SHOULD,” “SHOULD NOT,” “RECOMMENDED,” “MAY,” and “OPTIONAL” in this specification are to be interpreted as described in [RFC 2119].

  In addition, the following keywords are used in this specification:

  PROHIBITED – The definition or behavior is an absolute prohibition of this specification. Opposite of REQUIRED.

  CONDITIONALLY REQUIRED – The definition or behavior depends on a condition. If the specified condition is met, then the definition or behavior is REQUIRED, otherwise it is PROHIBITED.

  CONDITIONALLY OPTIONAL – The definition or behavior depends on a condition. If the specified condition is met, then the definition or behavior is OPTIONAL, otherwise it is PROHIBITED.

  These keywords are thus capitalized when used to unambiguously specify requirements over protocol and application features and behavior that affect the interoperability and security of implementations. When these words are not capitalized, they are meant in their natural-language sense.

- Strings that are to be taken literally are enclosed in “double quotes”.
- Words that are emphasized are printed in *italic*.
- Keywords that are defined by the UPnP Working Committee are printed using the *forum* character style.

- Keywords that are defined by the UPnP Device Architecture are printed using the **arch** character style.
- A double colon delimiter, "::", signifies a hierarchical parent-child (parent::child) relationship between the two objects separated by the double colon. This delimiter is used in multiple contexts, for example: Service::Action(), Action()::Argument, parentProperty::childProperty.

1.2.1 Data Types

This specification uses data type definitions from two different sources. The UPnP Device Architecture defined data types are used to define state variable and action argument data types [DEVICE]. The XML Schema namespace is used to define property data types [XML SCHEMA-2].

For UPnP Device Architecture defined Boolean data types, it is strongly RECOMMENDED to use the value "**0**" for false, and the value "**1**" for true. The values "**true**", "**yes**", "**false**", or "**no**" MAY also be used but are NOT RECOMMENDED. The values "**yes**" and "**no**" are deprecated and MUST NOT be sent out by devices but MUST be accepted on input.

For XML Schema defined Boolean data types, it is strongly RECOMMENDED to use the value "*0*" for false, and the value "*1*" for true. The values "*true*", "*yes*", "*false*", or "*no*" MAY also be used but are NOT RECOMMENDED. The values "*yes*" and "*no*" are deprecated and MUST NOT be sent out by devices but MUST be accepted on input.

1.3 Vendor-defined Extensions

Whenever vendors create additional vendor-defined state variables, actions or properties, their assigned names and XML representation MUST follow the naming conventions and XML rules as specified in [DEVICE], Section 2.5, "Description: Non-standard vendor extensions".

1.4 References

1.4.1 Normative References

This section lists the normative references used in this specification and includes the tag inside square brackets that is used for each such reference:

[DEVICE] – UPnP Device Architecture, version 1.0, UPnP Forum, June 13, 2000.
Available at: http://www.upnp.org/specs/architecture/UPnP-DeviceArchitecture-v1.0-20000613.htm.
Latest version available at: http://www.upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.0.pdf.

[ISO 8601] – Data elements and interchange formats – Information interchange -- Representation of dates and times, International Standards Organization, December 21, 2000.
Available at: ISO 8601:2000.

[RFC 2046] – IETF RFC 2046, Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types, N. Freed, Innosoft, N. Borenstein, First Virtual, November 1996.
Available at: http://www.ietf.org/rfc/rfc2046.txt

[RFC 2119] – IETF RFC 2119, Key words for use in RFCs to Indicate Requirement Levels, S. Bradner, 1997.
Available at: http://www.faqs.org/rfcs/rfc2119.html.

[RFC 2396] – IETF RFC 2396, Uniform Resource Identifiers (URI): Generic Syntax, T. Berners-Lee, MIT/LCS, R. Fielding, U.C. Irvine, L. Masinter, Xerox Corporation, August 1998.
Available at: http://www.ietf.org/rfc/rfc2396.txt

[RFC 3339] – IETF RFC 3339, Date and Time on the Internet: Timestamps, G. Klyne, Clearswift Corporation, C. Newman, Sun Microsystems, July 2002.
Available at: http://www.ietf.org/rfc/rfc3339.txt.

[XML] – Extensible Markup Language (XML) 1.0 (Third Edition), François Yergeau, Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, eds., W3C Recommendation, February 4, 2004.
Available at: http://www.w3.org/TR/2004/REC-xml-20040204.

[XML SCHEMA-2] – XML Schema Part 2: Data Types, Second Edition, Paul V. Biron, Ashok Malhotra, W3C Recommendation, 28 October 2004.
Available at: http://www.w3.org/TR/2004/REC-xmlschema-2-20041028.

1.4.2 Informative References

This section lists the informative references that are provided as information in helping understand this specification:

[PHONEARCH] – TelephonyArchitecture:1, UPnP Forum, March 22, 2011.
Available at: http://www.upnp.org/specs/phone/UPnP-phone-TelephonyArchitecture-v1-20110322.pdf.
Latest version available at: http://www.upnp.org/specs/phone/UPnP-phone-TelephonyArchitecture-v1.pdf.

2 Service Modeling Definitions (Normative)

2.1 Service Type

The following service type identifies a service that is compliant with this specification:

urn:schemas-upnp-org:service:*Messaging:1*

Messaging service is used herein to refer to this service type.

2.2 Terms and Abbreviations

2.2.1 Abbreviations

Table 2-1: Abbreviations

Abbreviation	Description
GUI	Graphical User Interface
ID	Identifier
IM	Instant Messaging
ISDN	Integrated Services Digital Network
MMS	Multimedia Messaging Service
PLMN	Public Land Mobile Network
PSTN	Public Switched Telephone Network
SMS	Short Message Service
TC	Telephony Client
TelCP	Telephony Control Point
TS	Telephony Server
VoIP	Voice over IP
WAN	Wide Area Network

2.2.2 Terms

The following terms are defined and then used in this specification document.

2.2.2.1 Message

A Message is an information unit (e.g., e-Mail, SMS, MMS, Instant Message, etc.) that is exchanged between two or more entities (e.g., UPnP TelCP and TS/TC).

2.2.2.2 Messaging Session

A Messaging Session is a logical connection established between two or more recipients before starting to exchange Messages. The Messages are then orderly managed within the Messaging Session.

The Messaging Sessions are also referred to as "Sessions" in this specification.

2.2.2.3 Page Mode Messaging

The Page Mode Messaging refers to operations of sending and receiving messages from any recipients without establishing a session between the recipients.

The typical Messages exchanged under the Page Mode Messaging are e-Mail, SMS, and MMS.

2.2.2.4 Session Mode Messaging

The Session Mode Messaging refers to operations of sending and receiving Messages within a Messaging Session.

The typical Messages exchanged under the Session Mode Messaging are Instant Messages (IM) or chat. SMS, MMS, etc. can also be managed under the concept of Session Mode Messaging and can be used to logically group the Messages into an one conversation. It is also possible to have a Messaging Session with different Message classes.

2.2.2.5 Peer

A Peer refers to a recipient for a Message or a participant in a Messaging Session.

2.3 *Messaging* Service Architecture

The *Messaging* service provides in the UPnP network the overall set of messaging capabilities of a phone (e.g., smartphone, IP phone, VoIP gateway, etc.), as the role of a TS.

According to the *Telephony* architecture, this *Messaging* service can be included both in the TS or TC devices. More than one *Messaging* services can coexist in the same UPnP network, it's up to a TelCP to manage multiple *Messaging* services.

The *Messaging* service provides the TelCP with the following features:

- Sending, retrieving and deleting of Messages;
- Receiving notification of Messages.
- Managing messaging and file transfer Sessions.

Both Page Mode and Session Mode Messaging are supported by the *Messaging* service.

The architecture for the *Messaging* service is shown in the following figure.

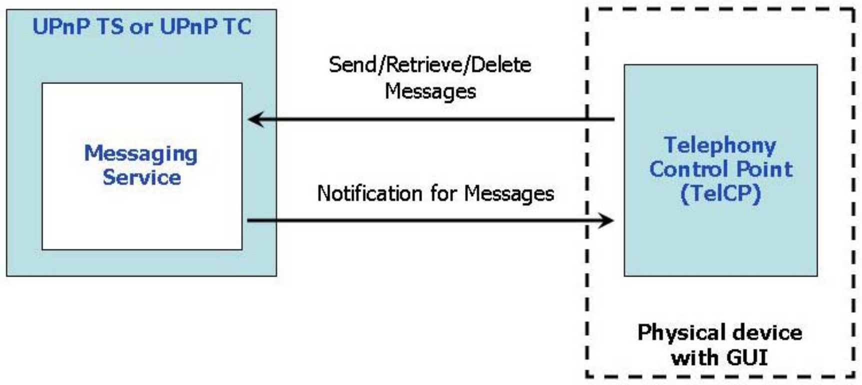

UPnP TS or UPnP TC
Messaging Service
Send/Retrieve/Delete Messages
Notification for Messages
Telephony Control Point (TelCP)
Physical device with GUI

Figure 2-1: Architecture of the *Messaging* Service

It is assumed that the TelCP features are implemented in a physical device that provides a Graphical User Interface (GUI), for a user to manage locally the messaging functionalities. These are however out of the scope of the *Messaging* service design.

The UPnP device that implements the *Messaging* service is capable of exposing messaging features to the peers through some telecommunication means (e.g., PSTN/ISDN, PLMN, VoIP, etc.). The implementation of these messaging features with peer networks are out of the scope.

It is possible to have a local user interface for the *Messaging* service in UPnP server. The local management of Messages by the end user should be taken into consideration when implementing the *Messaging* service.

2.4 State Variables

Note: For first-time reader, it may be more insightful to read the theory of operations first and then the action definitions before reading the state variable definitions.

2.4.1 State Variable Overview

Table 2-2: State Variables

Variable Name	R/O[1]	Data Type	Reference
A_ARG_TYPE_MessagingCapabilities	*R*	**string** (XML fragment)	See Section 2.4.2
NewMessages	*R*	**string** (XML fragment)	See Section 2.4.3
SessionUpdates	*O*	**string** (XML fragment)	See Section 2.4.4
A_ARG_TYPE_TelephonyServerIdentity	*R*	**string**	See Section 2.4.5
A_ARG_TYPE_MessageID	*R*	**string**	See Section 2.4.6
A_ARG_TYPE_MessageClass	*O*	**string**	See Section 2.4.7
A_ARG_TYPE_MessageFolder	*O*	**string**	See Section 2.4.8
A_ARG_TYPE_MessageStatus	*O*	**string**	See Section 2.4.9
A_ARG_TYPE_Message	*R*	**string** (XML fragment)	See Section 2.4.10
A_ARG_TYPE_MessageList	*O*	**string** (XML fragment)	See Section 2.4.11
A_ARG_TYPE_SessionID	*O*	**string**	See Section 2.4.12
A_ARG_TYPE_SessionClass	*O*	**string**	See Section 2.4.13
A_ARG_TYPE_RecipientsList	*O*	**string** (XML fragment)	See Section 2.4.14
A_ARG_TYPE_SessionInfo	*O*	**string** (XML fragment)	See Section 2.4.15
A_ARG_TYPE_SessionsList	*O*	**string** (XML fragment)	See Section 2.4.16
A_ARG_TYPE_SupportedContentType	*O*	**string**	See Section 2.4.17
A_ARG_TYPE_Subject	*O*	**string**	See Section 2.4.18
A_ARG_TYPE_SessionStatus	*O*	**string**	See Section 2.4.19
A_ARG_TYPE_FileInfoList	*O*	**string** (XML fragment)	See Section 2.4.20

[1] *R* = REQUIRED, *O* = OPTIONAL, *CR* = CONDITIONALLY REQUIRED, *CO* = CONDITIONALLY OPTIONAL, *X* = Non-standard, add *-D* when deprecated (e.g., *R-D*, *O-D*).

2.4.2 A_ARG_TYPE_MessagingCapabilities

This state variable defines an XML fragment that contains the features supported by the *Messaging* service, regarding:

- The Message classes supported for Page Mode Messaging;
- The Session classes supported for Session Mode Messaging;
- The Message folders supported.

2.4.2.1 XML Schema Definition

This is a string containing an XML fragment. The XML fragment in this argument MUST validate against the XML schema for *MessagingCapabilities* in the XML namespace "`urn:schemas-upnp-org:phone:messaging`" which is located at "`http://www.upnp.org/schemas/phone/messaging-v1.xsd`".

2.4.2.2 Description of fields in the *MessagingCapabilities* structure

```
<?xml version="1.0" encoding="UTF-8"?>
<messaging:messagingCapabilities
   xsi:schemaLocation="urn:schemas-upnp-org:phone:messaging
   http://www.upnp.org/schemas/phone/messaging-v1.xsd"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:messaging="urn:schemas-upnp-org:phone:messaging">
   <supportedMessageClasses>
      <messageClass>Message class</messageClass>
      <!-- Any other messageClass(if any) go here.-->
   </supportedMessageClasses>
   <supportedSessionClasses>
      <sessionClass>Session class</sessionClass>
      <!-- Any other sessionClass(if any) go here.-->
   </supportedSessionClasses>
   <supportedMessageFolders>
      <messageFolder>Message Folder</messageFolder>
      <!-- Any other messageFolder(if any) go here.-->
   </supportedMessageFolders>
</messaging:messagingCapabilities>
```

`<xml>`
REQUIRED. Case Sensitive.

`<messagingCapabilities>`
REQUIRED. MUST include a namespace declaration for the *Messaging* service Schema ("urn:schemas-upnp-org:phone:messaging"). This element includes the following sub elements:

`<supportedMessageClasses>`
REQUIRED. Indicates the list of supported Message classes.

`<messageClass>`
REQUIRED. Indicates the Message class as defined for the *A_ARG_TYPE_MessageClass* state variable. The empty value is not allowed. This element can appear one or more times.

`<supportedSessionClasses>`
REQUIRED. Indicates the list the supported Session classes.

`<sessionClass>`
OPTIONAL. Indicates the Session class as defined for the *A_ARG_TYPE_SessionClass* state variable. The empty value is not allowed. This element can appear one or more times.

`<supportedMessageFolders>`
REQUIRED. Indicates the list of the supported folders for the Messages.

`<messageFolder>`
REQUIRED. Indicates the folder name as defined for the *A_ARG_TYPE_ MessageFolder* state variable. The empty value is not allowed. This element can appear one or more times.

2.4.3 *NewMessages*

This state variable notifies incoming Messages (for both Page Mode and Session Mode Messaging), to the TelCP.

This state variable includes a list of *MessageIDs*. The TelCP can retrieve the details for each Message using the *MessageID*. It can also contain an overview of the Message.

2.4.3.1 XML Schema Definition

This is a string containing an XML fragment. The XML fragment in this argument MUST validate against the XML schema for *NewMessages* in the XML namespace "`urn:schemas-upnp-org:phone:messaging`" which is located at "`http://www.upnp.org/schemas/phone/messaging-v1.xsd`".

2.4.3.2 Description of fields in the *NewMessages* structure

```
<?xml version="1.0" encoding="UTF-8"?>
<messaging:newMessages
   xsi:schemaLocation="urn:schemas-upnp-org:phone:messaging
   http://www.upnp.org/schemas/phone/messaging-v1.xsd"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:messaging="urn:schemas-upnp-org:phone:messaging">
  <newMessage>
    <messageID>ID of the Message</messageID>
    <messageOverview>
      Preview of the Message (i.e.,SMS from +390112288046:Hallo,how…)
    </messageOverview>
  </newMessage>
  <!-- Any other newMessage(if any) go here.-->
</messaging:newMessages>
```

`<xml>`
REQUIRED. Case Sensitive.

`<newMessages>`
REQUIRED. MUST include a namespace declaration for the *Messaging* service Schema (“urn:schemas-upnp-org:phone:messaging”). This element includes the following sub elements:

`<newMessage>`
OPTIONAL. Includes a MessageID and Message overview. This element can appear zero or more times.

`<messageID>`
REQUIRED. Includes the unique identifier of the Message.

`<messageOverview>`
OPTIONAL. xsd:string, Includes the overview of the Message.

2.4.4 *SessionUpdates*

This state variable notifies any changes (e.g., session requested, session accepted, session closed, session modified, and user typing etc.) to the Messaging Sessions.

The notification message includes the Session ID, the Session status and any changes to the Session.

These information are contained in an XML data structure.

2.4.4.1 XML Schema Definition

This is a string containing an XML fragment. The XML fragment in this argument MUST validate against the XML schema for *SessionUpdates* in the XML namespace "`urn:schemas-upnp-org:phone:messaging`" which is located at "`http://www.upnp.org/schemas/phone/messaging-v1.xsd`".

2.4.4.2 Description of fields in the *SessionUpdates* structure

```
<?xml version="1.0" encoding="UTF-8"?>
<messaging:sessionUpdates
   xsi:schemaLocation="urn:schemas-upnp-org:phone:messaging
   http://www.upnp.org/schemas/phone/messaging-v1.xsd"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:messaging="urn:schemas-upnp-org:phone:messaging">
   <sessionUpdate>
      <sessionID>ID of the Session</sessionID>
      <sessionEvent>Session event description</sessionEvent>
      <!-- Any other sessionEvent (if any) go here.-->
      <sessionStatus>Status of the session</sessionStatus>
   </sessionUpdate>
   <!-- Any other sessionUpdate (if any) go here.-->
</messaging:sessionUpdates>
```

`<xml>`

REQUIRED. Case Sensitive.

`<sessionUpdates>`

REQUIRED. MUST include a namespace declaration for the *Messaging* service Schema ("urn:schemas-upnp-org:phone:messaging"). This element includes the following sub elements:

`<sessionUpdate>`

OPTIONAL. Includes sessionID, Session events to capture the Session changes and Session status. This element can appear zero or more times.

`<sessionID>`

REQUIRED. Includes the unique identifier of the Session.

`<sessionEvent>`

REQUIRED. xsd:string, Includes the descriptions of the events occurred. This element can appear one or more times. The following table shows the possible values of SessionEvent element and the resulting Session status:

Table 2-3: Allowed values for sessionEvent and the corresponding values of the sessionStatus

sessionEvent	sessionStatus	Description
"session requested by: "<peer:name>	"*Pending*"	An incoming Session has been requested by a peer
"session accepted by: "<peer:name>	"*Running*"	A Session has been accepted by a peer
"session closed"	"*Closed*"	The Session has been closed or rejected
"session modified"	"*Running*"	The Session has been modified
<peer:name>" is typing"	"*Running*"	One of the Session participants is typing a Message
"one TelCP has left"	"*Running*"/ "*Parked*"	One of the TelCP has left the Session. The Session status MUST be "*Parked*" if the last TelCP has left.
"one TelCP has joined"	"*Running*"	A "*Parked*" Session has been joined by a TelCP
<peer:name> " has left"	"*Running*"	One of the peers has left the Session.
<peer:name> " has joined"	"*Running*"	One of the peers has joined the Session.
"file transfer started"	"*Running*"	A file transfer has started within the Session
"file transfer completed"	"*Running*"	A file transfer has been completed within a Session

sessionEvent	sessionStatus	Description
“file transfer cancelled”	“*Running*”	A file transfer has been cancelled
“file transfer progress: ”<*nBytes*>“ Bytes”	“*Running*”	A file transfer is in progress within the Session and *nBytes* Bytes have been transferred since the transfer has started.
“file download failed”	“*Running*”	The file can not be retrieved by the TS/TC.
TBD	*TBD*	*(Specifed by UPnP vendors.)*

`<sessionStatus>`
REQUIRED. Includes the current status of the Session.

2.4.5 *A_ARG_TYPE_TelephonyServerIdentity*

This state variable contains the unique identifier of a Telephony Server. The identity of a Telephony Server is expressed using the standard URI (Unified Resource Identifier) scheme as specified in [RFC 2396]. In case of SIP, the identity of the Telephony Server identity contains the SIP URI. In case of a generic resource identified by a telephone number, the Telephony Server identity contains the TEL URI [RFC 3966].

2.4.6 *A_ARG_TYPE_MessageID*

This state variable contains an identifier that uniquely identifies a Message.

The format of this state variable is as follows:

- *MessageID*
  - to be used for Page Mode Messages;
- *SessionID*“.”*MessageID*
  - to be used for Session Mode Messages.

where:

- The *MessageID* is an unique alphanumerical identifier that MUST contain only characters from the sets: “0..9”, “A..Z”, “a..z”, generated by the UPnP device hosting the *Messaging* service.
- The *SessionID* is an unique alphanumerical identifier of the Messaging Session, as defined in the section § 2.4.12. It’s applicable for Session Mode Messaging.

Example:

Some examples of valid values for this variable are:

- Valid examples for Page Mode Messaging are: 123456, qwerty, M000123, XXY123.
- Valid examples for Session Mode Messaging are: abcdef.789012, ses001.123455.

2.4.7 *A_ARG_TYPE_MessageClass*

This state variable contains the value for the Message class. The *Messaging* service MUST support at least one of the Message classes. The possible values of this state variable are shown in the table below.

Table 2-4: allowedValueList for the *A_ARG_TYPE_MessageClass* state variable

Value	R/O[1]
"*e-Mail*"	*O*
"*SMS*" (Default)	*O*
"*MMS*"	*O*
"*Instant Message*"	*O*

where:

- "*e-Mail*" refers to an e-Mail Message;
- "*SMS*" refers to an SMS Message;
- "*MMS*" refers to an MMS Message;
- "*Instant Message*" refers to an instant Message (e.g., within a chat Session);

Vendor specific extensions are allowed for the values of this argument.

2.4.8 *A_ARG_TYPE_MessageFolder*

This state variable contains the possible values for the folders where Messages are kept and organized by the *Messaging* service.

This state variable is introduced to provide type information for various action arguments that refers to folders of Messages.

The possible values of this state variable are shown in the table below.

Table 2-5: allowedValueList for the *A_ARG_TYPE_MessageFolder* state variable

Value	R/O[2]
"*Received*" (DEFAULT)	*R*
"*Outgoing*"	*R*
"*Sent*"	*R*
"*Deleted*"	*O*
"*ClosedSession*"	*O*

where:

- The "*Received*" folder contains the Messages that have been received by the *Messaging* service;
- The "*Outgoing*" folder contains the Messages that are in the process of being sent by the *Messaging* service;
- The "*Sent*" folder contains the Messages that have been sent by the *Messaging* service;
- The "*Deleted*" folder contains the Messages that have been deleted from all other folders;
- The "*ClosedSession*" folder contains information related to all the closed and terminated Sessions. The Session informatiation includes Session ID, Session Class, and Messages exchanged during the Session;

[1] *R* = REQUIRED, *O* = OPTIONAL, *X* = Non-standard.
[2] *R* = REQUIRED, *O* = OPTIONAL, *X* = Non-standard.

2.4.9 A_ARG_TYPE_MessageStatus

This state variable contains the current status of a Message. The status includes whether a Message has been read or yet to be read. This state variable is used to provide type information for arguments of various actions (i.e., GetMessages()). The possible values of this state variable are shown in the table below.

Table 2-6: allowedValueList for the A_ARG_TYPE_MessageStatus state variable

Value	R/O[1]
"Read"	R
"Unread"	R

where:

- "Read" means the Message has been read by a TelCP by invocation of the ReadMessage() action, or by using the local user interface of the device.
- "Unread" means the Message has not been read yet.

2.4.10 A_ARG_TYPE_Message

This state variable defines an XML fragment that contains the details of a Message which includes the ID of the Message, the ID of the Session, the ID of the reply Message, information about the recipients, the subject, the text for the Message, and any attachment if exist, etc.

2.4.10.1 XML Schema Definition

This is a string containing an XML fragment. The XML fragment in this argument MUST validate against the XML schema for Message in the XML namespace "`urn:schemas-upnp-org:phone:messaging`" which is located at "`http://www.upnp.org/schemas/phone/messaging-v1.xsd`".

The imported XML namespace "`urn:schemas-upnp-org:phone:peer`" is here referenced for the Complex Type *peerType*.

2.4.10.2 Description of fields in the Message structure

```
<?xml version="1.0" encoding="UTF-8"?>
<messaging:message
   xsi:schemaLocation="urn:schemas-upnp-org:phone:messaging
   http://www.upnp.org/schemas/phone/messaging-v1.xsd"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:messaging="urn:schemas-upnp-org:phone:messaging"
   xmlns:peer="urn:schemas-upnp-org:phone:peer">
   <messageID>ID of the Message(i.e., abcdef.123456)</messageID>
   <sessionID>ID of the Session</sessionID>
   <messageClass>Class of the Message</messageClass>
   <messageFolder>Folder where the Message is stored</messageFolder>
   <messageStatus>Status of the Message</messageStatus>
   <replyMessageID>ID of the replied Message</replyMessageID>
   <recipientsList>
      <recipientTo>
         Recipient information (i.e.,
         <peer:id>jane.doe@acme.com</peer:id>
         <peer:contactInstanceId>12</peer:contactInstanceId>)
      </recipientTo>
      <!-- Any other recipientTo (if any) go here.-->
      <recipientCc>
```

[1] R = REQUIRED, O = OPTIONAL, X = Non-standard.

```
         Cc Recipient information (i.e.,
         <peer:id>Bull.Dog@dogs.com</peer:id>)
      </recipientCc>
      <!-- Any other recipientCc (if any) go here.-->
      <recipientBcc>
         Bcc Recipient information (i.e.,
         <peer:id>Fox.Terrier@dogs.com</peer:id>
         <peer:name>Fox Terrier</peer:name>
         <peer:contactInstanceId>4</peer:contactInstanceId>)
      </recipientBcc>
      <!-- Any other recipientBcc (if any) go here.-->
      <recipientFrom>
         Sender information (i.e.,
         <peer:id>The.Sender@fromhere.com</peer:id>
         <peer:name>Bob the Sender</peer:name>)
      </recipientFrom>
   </recipientsList>
   <subject>Message title</subject>
   <text>Message text</text>
   <attachments>
      <attachment>
         <uri>
         URI of the attachment(i.e.,http://192.168.1.3/file1.pdf)
         </uri>
         <size>Size of the attachment</size>
         <mimeType>MIME type of the attachment</mimeType>
      </attachment>
      <!-- Any other attachment (if any) go here.-->
   </attachments>
   <dateSent>Date when the Message is sent</dateSent>
   <dateReceived>Date when the Message is received</dateReceived>
</messaging:message>
```

`<xml>`
REQUIRED. Case Sensitive.

`<message>`
REQUIRED. MUST include a namespace declaration for the Complex Type <peerType> ("urn:schemas-upnp-org:phone:peer" as from the Appendix A) and a namespace declaration for the *Messaging* service Schema ("urn:schemas-upnp-org:phone:messaging"). This element includes the following sub elements:

`<messageID>`
REQUIRED. Indicates the unique identifier of the Message (see section 2.4.6).

`<sessionID>`
OPTIONAL, Indicates the unique identifier of the Session which the Message belongs to (see section 2.4.12). This element MUST exist if the case of Session Mode Messaging.

`<messageClass>`
REQUIRED. Indicates the class of the Message (see section 2.4.7)

`<messageFolder>`
REQUIRED. Indicates the folder in which the Message is currently stored (see section 2.4.8).

`<messageStatus>`
REQUIRED. Indicates the current status of the Message (see section 2.4.9).

`<replyMessageID>`
OPTIONAL. Indicates the id of the Message which is being replied.

`<recipientsList>`
REQUIRED. Indicates the list of recipients for a Message as defined in section 2.4.14. This element SHOULD reflect the current list of participants in a Session Mode Messaging.

`<recipientTo>`
OPTIONAL. peer:peerType, Indicates the recipient to whom the Message is sent. This element MUST exist if the value of the Message class is either SMS or MMS or Instant Message. In the case of e-Mail messages, at least one of the following elements MUST exist: `<recipientTo>`, `<recipientCc>` or `<recipientBcc>`. This element can appear zero or more times.

`<recipientCc>`
OPTIONAL. peer:peerType, Indicates the recipient to whom the Message is copied (CC); This element can appear zero or more times.

`<recipientBcc>`
OPTIONAL. peer:peerType, Indicates the recipient to whom the Message is blind copied (BCC). This element can appear zero or more times.

`<recipientFrom>`
REQUIRED. peer:peerType, Indicates the sender of the Message.

`<subject>`
OPTIONAL. xsd:string, Indicates the title of the Message.

`<text>`
OPTIONAL. xsd:string, Includes the text content of the Message.

`<attachments>`
OPTIONAL. Includes a list of attachments. This element MUST exist if the value of the message class is MMS. It has the following sub elements.

`<attachment>`
REQUIRED. Includes the information of the file attached to the Message. Thbis element can appear one or more times. It has the following sub elements.

`<uri>`
REQUIRED. xsd:anyURI, Indicates the identifier for the resource as defined by RFC 2396.

`<size>`
OPTIONAL. xsd:nonNegativeInteger, Indicates the size in Bytes of the resource.

`<mimeType>`
OPTIONAL. xsd:string, Indicates the MIME type of the resource as defined by RFC 2046.

`<dateSent>`
OPTIONAL. xsd:dateTime, Indicates the date when the Message is sent by the *Messaging* service as specified by RFC 3339. This element MUST exist in the sent Message.

`<dateReceived>`
OPTIONAL. xsd:dateTime, Indicates the date when the Message is received by the *Messaging* service as specified by RFC 3339. This element MUST exist in the received Message.

2.4.11 *A_ARG_TYPE_MessageList*

This state variable defines an XML fragment that contains a list of Messages, each one with some or all of its details (including the ID of the Message).

This state variable is introduced to provide type information for various action arguments that return a list of Messages.

2.4.11.1 XML Schema Definition

This is a string containing an XML fragment. The XML fragment in this argument MUST validate against the XML schema for *MessageList* in the XML namespace "`urn:schemas-upnp-org:phone:messaging`" which is located at "`http://www.upnp.org/schemas/phone/messaging-v1.xsd`".

The imported XML namespace “`urn:schemas-upnp-org:phone:peer`” is here referenced for the Complex Type *peerType*.

2.4.11.2 Description of fields in the *MessageList* structure

```
<?xml version="1.0" encoding="UTF-8"?>
<messaging:messageList
   xsi:schemaLocation="urn:schemas-upnp-org:phone:messaging
   http://www.upnp.org/schemas/phone/messaging-v1.xsd"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:messaging="urn:schemas-upnp-org:phone:messaging"
   xmlns:peer="urn:schemas-upnp-org:phone:peer">
   <message>
      <messageID>ID of the Message</messageID>
      <messageClass>Class of the Message</messageClass>
      <messageFolder>Message Folder</messageFolder>
      <messageStatus>Status of the Message</messageStatus>
      <recipientsList>
         <recipientTo>
            Recipient information (i.e.,
            <peer:id>jane.doe@acme.com</peer:id>
            <peer:contactInstanceId>12</peer:contactInstanceId>)
         </recipientTo>
         <!-- Any other recipientTo (if any) go here.-->
         <recipientFrom>
            Sender information (i.e.,
            <peer:id>The.Sender@fromhere.com</peer:id>
            <peer:name>Bob the Sender</peer:name>)
         </recipientFrom>
      </recipientsList>
      <subject>Message title</subject>
      <!-- Can have additional details -->
   </message>
   <!-- Any other message (if any) go here.-->
</messaging:messageList>
```

`<xml>`
REQUIRED. Case Sensitive.

`<messageList>`
REQUIRED. MUST include a namespace declaration for the Complex Type <peerType> ("urn:schemas-upnp-org:phone:peer" as from the Appendix A) and a namespace declaration for the *Messaging* service Schema ("urn:schemas-upnp-org:phone:messaging"). This element includes the following sub elements:

`<message>`
OPTIONAL. Contains the details of a Message as specified in the section 2.4.10. This element can appear zero or more times.

2.4.12 *A_ARG_TYPE_SessionID*

This state variable contains an identifier that uniquely identifies a Messaging Session.

The Session identifier (*SessionID*) is a unique alphanumerical identifier that MUST contain only characters from the sets: "0..9", "A..Z", "a..z", generated by the UPnP device hosting the *Messaging* service.

Example:

Some examples of valid values for this variable are:

- 654321
- abcdef
- ses0123
- XYZ123

2.4.13 *A_ARG_TYPE_SessionClass*

This state variable contains the value for the Session class. The possible values for this state variable are listed in the Table 2-7.

Table 2-7: allowedValueList for the *A_ARG_TYPE_SessionClass* state variable

Value	R/O[1]
"*e-Mail*"	O
"*SMS*"	O
"*MMS*"	O
"*Chat*" (DEFAULT)	O
"*File-Transfer*"	O
"*Mixed*"	O

where:

- "*e-Mail*" refers to an e-Mail Messaging Session;
- "*SMS*" refers to an SMS Messaging Session;
- "*MMS*" refers to an MMS Messaging Session;
- "*Chat*" refers to an instant Messaging Session, or IM Session;
- "*File-Transfer*" refers to a Session for transferring files;
- "*Mixed*" refers to a Messaging Session that can contain different classes of Messages including file transfer;

Vendor specific extensions are allowed for the values of this argument.

2.4.14 *A_ARG_TYPE_RecipientsList*

This state variable defines an XML fragment that contains the list of recipients for a Message or a Messaging Session.

A recipient can either be a contact or a group of contacts, and is identified by its address. It can also include a name and identification from the Address Book of the *Phone Data Model*, if implemented.

2.4.14.1 XML Schema Definition

This is a string containing an XML fragment. The XML fragment in this argument MUST validate against the XML schema for *RecipientsList* in the XML namespace "`urn:schemas-upnp-org:phone:messaging`" which is located at "`http://www.upnp.org/schemas/phone/messaging-v1.xsd`".

The imported XML namespace "`urn:schemas-upnp-org:phone:peer`" is here referenced for the Complex Type `<peerType>`.

2.4.14.2 Description of fields in the *RecipientsList* structure

```
<?xml version="1.0" encoding="UTF-8"?>
<messaging:recipientsList
   xsi:schemaLocation="urn:schemas-upnp-org:phone:messaging
   http://www.upnp.org/schemas/phone/messaging-v1.xsd"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:messaging="urn:schemas-upnp-org:phone:messaging"
   xmlns:peer="urn:schemas-upnp-org:phone:peer">
   <recipientTo>
      Recipient information (i.e.,
      <peer:id>MyGroupURI</peer:id>
      <peer:name>MyGroup</peer:name>
```

[1] R = REQUIRED, O = OPTIONAL, X = Non-standard.

```
      <peer:contactGroupId>37</peer:contactgroupId>)
   </recipientTo>
   <!-- Any other recipientTo (if any) go here.-->
   <recipientCc>
      Cc Recipient information (i.e.,
      <peer:id>Bull.Dog@dogs.com</peer:id>)
   </recipientCc>
   <!-- Any other recipientCc (if any) go here.-->
   <recipientBcc>
      Bcc Recipient information (i.e.,
      <peer:id>Fox.Terrier@dogs.com</peer:id>
      <peer:name>Fox Terrier</peer:name>
      <peer:contactInstanceId>4</peer:contactInstanceId>)
   </recipientBcc>
   <!-- Any other recipientBcc (if any) go here.-->
   <recipientFrom>
      Sender information (i.e.,
      <peer:id>The.Sender@fromhere.com</peer:id>
      <peer:name>Bob the Sender</peer:name>)
   </recipientFrom>
</messaging:recipientsList>
```

`<xml>`
REQUIRED. Case Sensitive.

`<recipientsList>`
REQUIRED. MUST include a namespace declaration for the Complex Type <peerType> ("urn:schemas-upnp-org:phone:peer" as from the Appendix A) and a namespace declaration for the *Messaging* service Schema ("urn:schemas-upnp-org:phone:messaging"). This element includes the following sub elements:

`<recipientTo>`
OPTIONAL. peer:peerType, Indicates the recipient to whom the Message is sent. This element can appear zero or more times.

`<recipientCc>`
OPTIONAL. peer:peerType, Indicates the recipient to whom the Message is copied (CC); This element can appear zero or more times.

`<recipientBcc>`
OPTIONAL. peer:peerType, Indicates the recipient to whom the Message is blind copied (BCC); This element can appear zero or more times.

`<recipientFrom>`
REQUIRED. peer:peerType, Indicates the sender of the Message or the creator of the Messaging Session.

2.4.15 *A_ARG_TYPE_SessionInfo*

This state variable defines an XML fragment that contains the details of a Messaging Session which includes the ID of the Session, the Session class, the information about the recipients, the subject, the status of the Session, and the list of Messages, etc.

2.4.15.1 XML Schema Definition

This is a string containing an XML fragment. The XML fragment in this argument MUST validate against the XML schema for *SessionInfo* in the XML namespace "`urn:schemas-upnp-org:phone:messaging`" which is located at "`http://www.upnp.org/schemas/phone/messaging-v1.xsd`".

The imported XML namespace "`urn:schemas-upnp-org:phone:peer`" is here referenced for the Complex Type *peerType*.

2.4.15.2 Description of fields in the *SessionInfo* structure

```
<?xml version="1.0" encoding="UTF-8"?>
<messaging:sessionInfo
```

```
   xsi:schemaLocation="urn:schemas-upnp-org:phone:messaging
   http://www.upnp.org/schemas/phone/messaging-v1.xsd"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:messaging="urn:schemas-upnp-org:phone:messaging"
   xmlns:peer="urn:schemas-upnp-org:phone:peer">
   <sessionID>ID of the Session (i.e., ses000372)</sessionID>
   <sessionClass>Class of the Session</sessionClass>
   <subject>Title of the Session</subject>
   <sessionStatus>Status of the Session</sessionStatus>
   <supportedContentType>
      Supported content type list (i.e., image/jpeg,video/mpeg)
   </supportedContentType>
   <recipientsList>
      <recipientTo>
         Recipient information (i.e.,
         <peer:id>john.doe@acme.com</peer:id>
      </recipientTo>
      <!-- Any other recipientTo (if any) go here.-->
      <recipientCc>
         Cc Recipient information (i.e.,
         <peer:id>diego.armando@biloxi.com</peer:id>)
      </recipientCc>
      <!-- Any other recipientCc (if any) go here.-->
      <recipientBcc>
         Bcc Recipient information (i.e.,
         <peer:id>Bob.Cat@cats.com</peer:id>
         <peer:name>Bob</peer:name>
         <peer:contactInstanceId>4</peer:contactInstanceId>)
      </recipientBcc>
      <!-- Any other recipientBcc (if any) go here.-->
      <recipientFrom>
         Session creator information (i.e.,
         <peer:id>Bull.Dog@dogs.com</peer:id>)
      </recipientFrom>
   </recipientsList>
   <listOfMessages>
      <messageID>ID of a Message</messageID>
      <!-- Any other messageID (if any) go here.-->
   </listOfMessages>
   <dateStarted>Date when the Session started</dateStarted>
   <dateLastMessage>Date of the last Message</dateLastMessage>
   <dateEnded>Date when the Session terminated</dateEnded>
</messaging:sessionInfo>
```

`<xml>`
REQUIRED. Case Sensitive.

`<sessionInfo>`
REQUIRED. MUST include a namespace declaration for the Complex Type <peerType> ("urn:schemas-upnp-org:phone:peer" as from the Appendix A) and a namespace declaration for the *Messaging* service Schema ("urn:schemas-upnp-org:phone:messaging"). This element contains the following elements:

`<sessionID>`
REQUIRED. Indicates the identifier that uniquely identifies the Messaging Session (see section 2.4.12).

`<sessionClass>`
REQUIRED. Indicates the class of Messaging Session (see section 2.4.13).

`<subject>`
OPTIONAL. xsd:string, Indicates the current title of the Messaging Session.

`<sessionStatus>`
REQUIRED. Indicates the current status of the Session (see section 2.4.19).

`<supportedContentType>`
OPTIONAL. Includes a list of supported contents that can be shared among the participants in the Session (see section 2.4.17).

`<recipientsList>`
REQUIRED. Indicates the list of recipients for the Messaging Session (see section 2.4.14).

`<recipientTo>`
OPTIONAL. peer:peerType, Indicates the recipient to whom the Messages are sent. This element MUST exist if the value of the Session class is either SMS or MMS or Instant Message. In the case of e-Mail Sessions, at least one of the following elements MUST exist: `<recipientTo>`, `<recipientCc>` or `<recipientBcc>`. This element can appear zero or more times.

`<recipientCc>`
OPTIONAL. peer:peerType, Indicates the recipient to whom the Messages are copied (CC); This element can appear zero or more times.

`<recipientBcc>`
OPTIONAL. peer:peerType, Indicates the recipient to whom the Messages are blind copied (BCC); This element can appear zero or more times.

`<recipientFrom>`
REQUIRED. peer:peerType, Indicates the creator of the Messaging Session.

`<listOfMessages>`
OPTIONAL. Includes the list of Message IDs sorted in a chronological order. It has the following sub elements:

`<messageID>`
OPTIONAL. Indicates the unique identifier of the Message. It MUST be in the format SessionID.MessageID (see section 2.4.6). This element can appear zero or more times.

`<dateStarted>`
OPTIONAL. xsd:dateTime, Indicates the date when the Session was started.

`<dateLastMessage>`
OPTIONAL. xsd:dateTime, Indicates the date when a Message was last exchanged in the Session.

`<dateEnded>`
OPTIONAL. xsd:dateTime, Indicates the date when the Session was terminated.

2.4.16 *A_ARG_TYPE_SessionsList*

This state variable defines an XML fragment that contains a list of Sessions, each one with some or all of its details (including the ID of the Session).

This state variable is introduced to provide type information for various action arguments that return a list of Sessions.

2.4.16.1 XML Schema Definition

This is a string containing an XML fragment. The XML fragment in this argument MUST validate against the XML schema for *SessionsList* in the XML namespace "`urn:schemas-upnp-org:phone:messaging`" which is located at "`http://www.upnp.org/schemas/phone/messaging-v1.xsd`".

The imported XML namespace “`urn:schemas-upnp-org:phone:peer`” is here referenced for the Complex Type *peerType*.

2.4.16.2 Description of fields in the *SessionsList* structure

```
<?xml version="1.0" encoding="UTF-8"?>
<messaging:sessionsList
   xsi:schemaLocation="urn:schemas-upnp-org:phone:messaging
   http://www.upnp.org/schemas/phone/messaging-v1.xsd"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:messaging="urn:schemas-upnp-org:phone:messaging"
   xmlns:peer="urn:schemas-upnp-org:phone:peer">
   <sessionInfo>
      <sessionID>ID of the Session</sessionID>
      <sessionClass>Class of the Session</sessionClass>
      <sessionStatus>Status of the Session</sessionStatus>
      <recipientsList>
      <recipientTo>
         Recipient information
      </recipientTo>
      <!-- Any other recipientTo (if any) go here.-->
      <recipientCc>
         Cc Recipient information
      </recipientCc>
      <!-- Any other recipientCc (if any) go here.-->
      <recipientBcc>
         Bcc Recipient information
      </recipientBcc>
      <!-- Any other recipientBcc (if any) go here.-->
      <recipientFrom>
         Session creator information
      </recipientFrom>
   </recipientsList>
   </sessionInfo>
   <!-- Any other sessionInfo (if any) go here.-->
</messaging:sessionsList>
```

`<xml>`
REQUIRED. Case Sensitive.

`<sessionsList>`
REQUIRED. MUST include a namespace declaration for the Complex Type <peerType> ("urn:schemas-upnp-org:phone:peer" as from the Appendix A) and a namespace declaration for the *Messaging* service Schema ("urn:schemas-upnp-org:phone:messaging"). This element contains the following elements:

`<sessionInfo>`
OPTIONAL. Includes the details of the Session as defined in the section 2.4.15. This element can appear zero or more times.

2.4.17 *A_ARG_TYPE_SupportedContentType*

This state variable contains a list of content types supported by the Messaging Session.

The format of this state variable is a comma separated values of MIME types [RFC 2046].

Example:

Some example of valid values for this state variable are:

- image/jpeg
- image/png,video/avi,video/mpeg

2.4.18 *A_ARG_TYPE_Subject*

This state variable contains the title of a Session.

2.4.19 *A_ARG_TYPE_SessionStatus*

This state variable contains the current status of a Messaging Session. The possible values of this state variable are shown in the table below.

Table 2-8: allowedValueList for the *A_ARG_TYPE_SessionStatus* state variable

Value	R/O[1]
"*Pending*"	*R*
"*Running*"	*R*

[1] *R* = REQUIRED, *O* = OPTIONAL, *X* = Non-standard.

Value	R/O[1]
"*Parked*"	*R*
"*Closed*"	*O*

where:

- "*Pending*" refers to the state of a Session which has been initiated, but has not been accepted yet by a TelCP or a remote peer;
- "*Running*" refers to the state of a Session which is active and at least one TelCP is participating in the Session.
- "*Parked*" refers to the state of a Session which has no TelCP participating in the Session.
- "*Closed*" refers to the state of a Session which is currently inactive. This is due to the fact that the Session has been closed by a local TelCP or by a remote peer.

2.4.20 *A_ARG_TYPE_FileInfoList*

This state variable defines an XML fragment that contains information of a file transfer Session.

2.4.20.1 XML Schema Definition

This is a string containing an XML fragment. The XML fragment in this argument MUST validate against the XML schema for *FileInfoList* in the XML namespace "`urn:schemas-upnp-org:phone:messaging`" which is located at "`http://www.upnp.org/schemas/phone/messaging-v1.xsd`".

2.4.20.2 Description of fields in the *FileInfoList* structure

```
<?xml version="1.0" encoding="UTF-8"?>
<messaging:fileInfoList
   xsi:schemaLocation="urn:schemas-upnp-org:phone:messaging
   http://www.upnp.org/schemas/phone/messaging-v1.xsd"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:messaging="urn:schemas-upnp-org:phone:messaging">
   <sessionID>ID of the Session</sessionID>
   <fileInfo>
      <fileType>Type of the file</fileType>
      <fileSize>Size of the file</fileSize>
      <fileLink>
         Link to the file (i.e., http://tcp.com/song.mp3)
      </fileLink>
      <progressNotif time=notification period>
         Progress notification switch
      </progressNotif>
   </fileInfo>
   <!-- Any other fileInfo (if any) go here.-->
</messaging:fileInfoList>
```

`<xml>`
REQUIRED. Case Sensitive.

`<fileInfoList>`
REQUIRED. MUST include a namespace declaration for the *Messaging* service Schema ("urn:schemas-upnp-org:phone:messaging"). This element contains the following elements and attributes:

`<sessionID>`
REQUIRED. Indicates the identifier that uniquely identifies the File Transfer Session (see section 2.4.12).

`<fileInfo>`
REQUIRED. Includes the metadata of the file. It can contain the following sub elements and can have one or more instances.

`<fileType>`
REQUIRED. xsd:string, Indicates the type of the file (e.g., Audio-file or content type of the file).

`<fileSize>`
REQUIRED. xsd:nonNegativeInteger, indicates the size of the file in bytes.

`<fileLink>`
REQUIRED. xsd:anyURI, indicates the URL for the file.

`<progressNotif>`
REQUIRED. xsd:boolean, Indicates the type of notification required (e.g., periodic notification or notification after completion of the file transfer). If set to 1, the notification period can be specified by the following attribute:

`time`
OPTIONAL. xsd:positiveInteger, Indicates the notification period of the file transfer which is in progress. This attribute MUST exist if the `<ProgressNotif>` element is set to 1.

2.5 Eventing and Moderation

Table 2-9: Eventing and Moderation

		Moderation	
Variable Name	Evented	Moderated[1]	Criteria
NewMessages	*YES*	*YES*	Minimum 1 second between events
SessionUpdates	*YES*	*YES*	Minimum 1 second between events

[1] *YES* = The state variable MUST be moderated with the criteria

2.5.1 Eventing of *NewMessages*

The moderation of this state variable takes into consideration for both Page and Session Mode Messaging.

This state variable can be evented when a new incoming Message arrives. However, this state variable MUST NOT be evented more than once in a second. If multiple Messages arrives within a second, then all the events will be accumulated into a single event Message and the event Message will be sent after the expiration of the second.

In order to have a security aware *Messaging* service implementation the state variable *NewMessages* MAY event only the *messageID* when a new incoming message arrives while it MAY return full set of information only to authorized TelCPs in response to *GetNewMessages()* action invoked via TLS tunnel.

2.5.2 Eventing of *SessionUpdates*

This state variable can be evented when one of the following events occurrs within a Session:

- A new Session request has arrived;
- A “Pending” Session has been accepted;
- A Session has been closed;
- A Session has been modified;
- A participant is typing a Message (if supported);
- A TelCP or a participant has left the Session;
- A new TelCP or a new participant has joined the Session;
- A file transfer has been started or cancelled;
- A file transfer progress notification is available;

- A download of a file to transfer has failed.

However, this state variable MUST NOT be evented more than once in a second. If multiple events occurr within a second, then all the events will be accumulated into a single event Message and the event Message will be sent after the expiration of the second.

In order to have a security aware *Messaging* service implementation the state *variableSessionUpdates* MAY only event *sessionID* when the status of the Messaging Session has changed while it MAY return full set of information only to authorized TelCPs in response to *GetSessionUpdates()* action invoked via TLS tunnel.

2.6 Actions

The following table lists the actions of the *Messaging* service.

The *Messaging* service requires the implementation of all the actions necessary for delivering the complete set of functions for Page Mode Messaging, while Session Mode Messaging (and corresponding specific and dedicated actions) is left optional.

Table 2-10: Actions

Name	Device R/O[1]	Control Point R/O[2]
GetTelephonyIdentity()	*R*	*R*
GetMessagingCapabilities()	*R*	*R*
GetNewMessages()	*O*	*O*
SearchMessages()	*O*	*O*
ReadMessage()	*R*	*R*
SendMessage()	*R*	*R*
DeleteMessage()	*R*	*R*
CreateSession()	*O*	*O*
ModifySession()	*O*	*O*
AcceptSession()	*O*	*O*
GetSessionUpdates()	*O*	*O*
GetSessions()	*O*	*O*
JoinSession()	*O*	*O*
LeaveSession()	*O*	*O*
CloseSession()	*O*	*O*
StartFileTransfer()	*O*	*O*
CancelFileTransfer()	*O*	*O*
GetFileTransferSession()	*O*	*O*

[1] For a device this column indicates whether the action MUST be implemented or not, where *R* = REQUIRED, *O* = OPTIONAL, *CR* = CONDITIONALLY REQUIRED, *CO* = CONDITIONALLY OPTIONAL, *X* = Non-standard, add *-D* when deprecated (e.g., *R-D*, *O-D*)..

[2] For a control pont this column indicates whether a control point MUST be capable of invoking this action, where *R* = REQUIRED, *O* = OPTIONAL, *CR* = CONDITIONALLY REQUIRED, *CO* = CONDITIONALLY OPTIONAL, *X* = Non-standard, add *-D* when deprecated (e.g., *R-D*, *O-D*)..

2.6.1 *GetTelephonyIdentity()*

This action returns the identity of the Telephony Server. If the action succeeds then the output argument *TelephonyIdentity* contains the unique identity of the Telephony Server. If the identity of the Telephony Server has not been assigned by the time this action is invoked then this action will fail with an error code.

2.6.1.1 Arguments

Table 2-11: Arguments for *GetTelephonyIdentity()*

Argument	Direction	relatedStateVariable
TelephonyIdentity	*OUT*	*A_ARG_TYPE_TelephonyServerIdentity*

2.6.1.2 Argument Descriptions

The value of the output argument *TelephonyIdentity* MUST contain a valid URI as specified in [RFC 2396]. The value follows the standard URI definitions. In case of SIP, the SIP URI [RFC 3261] is used. In case of a generic resource identified by a telephone number, the TEL URI [RFC 3966] is used.

The examples of SIP URI as the value of the output argument *TelephonyIdentity* are as follows:

```
sip:alice@atlanta.com
sip:alice:secretword@atlanta.com;transport=tcp
sips:alice@atlanta.com?subject=project%20x&priority=urgent
sip:+1-212-555-1212:1234@gateway.com;user=phone
sips:1212@gateway.com
sip:alice@192.0.2.4
```

The IETF RFC 3261 [http://tools.ietf.org/html/rfc3261] defines the `sip:` (`sips`:, for resources to be contacted securely) URI scheme whereas the general form, is: sip:user:password@host:port;uri-parameters?headers

The examples of TEL URI as the value of the output argument *TelephonyIdentity* are as follows:

Tel:+1-201-555-0123 (This URI points to a phone number in the United States. The hyphens are included to make the number more human readable; they separate country, area code and subscriber number.)

`tel:7042;phone-context=example.com` (This URI describes a local phone number valid within the context "example.com".)

`tel:863-1234;phone-context=+1-914-555` (This URI describes a local phone number that is valid within a particular phone prefix.)

The IETF RFC 3966 [http://tools.ietf.org/html/rfc3966] defines the URI scheme `tel`, which describes any resources identified by telephone numbers. A telephone number is a string of decimal digits that uniquely indicates the network termination point. The number contains the information necessary to route the call to this point.

The section 19.1.6 Relating SIP URIs and tel URLs of RFC 3261 describes how to deal with conversion and compatibility rules among SIP/SIPS URI and TEL URI. Implementers MUST be aware of these recommendations.

2.6.1.3 Service Requirements

None.

2.6.1.4 Control Point Requirements When Calling The Action

None.

2.6.1.5 Dependency on Device State

None.

2.6.1.6 Effect on Device State

None.

2.6.1.7 Errors

Table 2-12: Error Codes for *GetTelephonyIdentity()*

ErrorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
606	Action not Authorized	The CP does not have privileges to invoke this action.
714	Identity does not exist	The identity of the TS has not been assigned yet by the telephony service in the WAN side.

2.6.2 *GetMessagingCapabilities()*

This action allows a TelCP to discover the set of capabilities and features that are supported by the *Messaging* service.

The *Messaging* service can support a subset of the messaging features. For example, the *Messaging* service can support only a subset of the Message classes or a subset of Session classes for a Session Mode Messaging. Therefore, a TelCP can discover the features supported by the *Messaging* service by invoking the *GetMessagingCapabilities()* action.

2.6.2.1 Arguments

Table 2-13: Arguments for *GetMessagingCapabilities()*

Argument	Direction	relatedStateVariable
SupportedCapabilities	*OUT*	*A_ARG_TYPE_MessagingCapabilities*

2.6.2.2 Argument Descriptions

The output argument *SupportedCapabilitites* contains the features supported by the *Messaging* service. The argument follows the XML structure as defined in the section 2.4.2 *A_ARG_TYPE_MessagingCapabilities*.

2.6.2.3 Service Requirements

None.

2.6.2.4 Control Point Requirements When Calling The Action

None.

2.6.2.5 Dependency on Device State

None.

2.6.2.6 Effect on Device State

None.

2.6.2.7 Errors

Table 2-14: Error Codes for *GetMessagingCapabilities()*

ErrorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
600-699	TBD	See UPnP Device Architecture section on Control.

2.6.3 *GetNewMessages()*

This action allows a TelCP to read the value of the *NewMessages* state variable.This action may be used by a TelCP which does not subscribe to the *Messaging* service for the event notifications.

2.6.3.1 Arguments

Table 2-15: Arguments for *GetNewMessages()*

Argument	Direction	relatedStateVariable
NewMessages	*OUT*	*NewMessages*

2.6.3.2 Argument Descriptions

The output argument *NewMessages* contains the value of the *NewMessages* state variable. The value of the *NewMessages* state variable includes information about the incoming Messages that have not been read yet.

If there are no new Messages then the output argument will include an XML fragment of type *NewMessages* without any NewMessage element.

2.6.3.3 Service Requirements

None.

2.6.3.4 Control Point Requirements When Calling The Action

None.

2.6.3.5 Dependency on Device State

None.

2.6.3.6 Effect on Device State

None.

2.6.3.7 Errors

Table 2-16: Error Codes for *GetNewMessages*

ErrorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.

ErrorCode	errorDescription	Description
606	Action not Autohrized	The CP does not have privileges to invoke this action.

2.6.4 *SearchMessages()*

This action allows a TelCP to search for a list of Messages that satisfy the search criteria specified by the input arguments of this action. Based on the search criteria specified in the input arguments, this action can perform the following functions:

- Retrieve a complete list of Messages for a given Message class (e.g., E-Mail, SMS, MMS, Instant Message, etc.) and in a specific Message folder (e.g., Received, Outgoing, Sent, etc.).
- Retrieve a complete list of unread (or read) Messages of a given Message class (e.g., E-Mail, SMS, MMS, Instant Message, etc.), and in a specific Message folder (e.g., Received, Outgoing, Sent, etc.).
- Retrieve a complete list of Messages exchanged during a particular Session.

2.6.4.1 Arguments

Table 2-17: Arguments for *SearchMessages()*

Argument	Direction	relatedStateVariable
MessageClass	*IN*	*A_ARG_TYPE_MessageClass*
MessageFolder	*IN*	*A_ARG_TYPE_MessageFolder*
MessageStatus	*IN*	*A_ARG_TYPE_MessageStatus*
SessionID	*IN*	*A_ARG_TYPE_SessionID*
MessageList	*OUT*	*A_ARG_TYPE_MessageList*

2.6.4.2 Argument Descriptions

The input argument *MessageClass* specifies the class of Messages to be searched. If the value of this argument is the empty string (“”), then the *Messaging* service will search for all the classes of Messages.

The input argument *MessageFolder* specifies the folder to search for Messages (e.g., Received folder, Sent folder, any folder, etc.). If the value of this argument is the empty string (“”), then the *Messaging* service will search in all the folders of Messages.

The input argument *MessageStatus* specifies whether to search for read or unread Messages. If the value of this argument is the empty string (“”), then the *Messaging* service will search for both read and unread Messages.

The input argument *SessionID* identifies the Session. If the value of this argument is not an empty string the *Messaging* service will only search for Messages that belongs to the specific Session.

The output argument *MessageList* is an XML fragment as specified in the section 2.4.11 which will only contain the Messages that satisfy the criteria specified in the input arguments.

2.6.4.3 Service Requirements

The *Messaging* Service will respond to this action with the list of Messages in MessageList structure, the Messaging Service MUST include only required elements of the Message in the MessageList structure. The required elements are specified in the section 2.4.11.

2.6.4.4 Control Point Requirements When Calling The Action

Since the service only return the required elements for the Messages, if a TelCP requires the complete details of the Messages, then it can use the *ReadMessage()*action with the Message ID in the input argument to retrieve the details of the Message.

2.6.4.5 Dependency on Device State

None.

2.6.4.6 Effect on Device State

None.

2.6.4.7 Errors

Table 2-18: Error Codes for *SearchMessages()*

ErrorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
606	Action not Autohrized	The CP does not have privileges to invoke this action.
702	Invalid Class	The Message class is invalid.
705	Invalid Message status	The Message status is invalid.
706	Invalid Session id	The Session with the given id does not exist.
707	Invalid folder	The folder is invalid.

2.6.5 *ReadMessage()*

This action allows a TelCP to retrieve a Message as identified by MessageID. This action also allows to retrieve a specific Message from a stored Session as indentified by the Session ID.

If the requested Message does not exists in the *Messaging* service, then Messaging service responds with an error.

2.6.5.1 Arguments

Table 2-19: Arguments for *ReadMessage()*

Argument	Direction	relatedStateVariable
MessageID	*IN*	*A_ARG_TYPE_MessageID*
MessageRequested	*OUT*	*A_ARG_TYPE_Message*

2.6.5.2 Argument Descriptions

The input argument *MessageID* uniquely identifies the Message. The MessageID also includes the SessionID in the case of Session Mode Messaging. This allows a TelCP to retrieve a specific Message from a Session identified by the SessionID.

The output argument *MessageRequested* is an XML fragment as specified in the section 2.4.10 which contains all the details of the requested Message.

2.6.5.3 Service Requirements

None.

2.6.5.4 Control Point Requirements When Calling The Action

None.

2.6.5.5 Dependency on Device State

None.

2.6.5.6 Effect on Device State

When a *ReadMessage()* action is successful in the case of unread Messages, the value of the status of the Message changes to “Read”.

2.6.5.7 Errors

Table 2-20: Error Codes for *ReadMessage()*

ErrorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
606	Action not Autohrized	The CP does not have privileges to invoke this action.
701	Invalid Message ID	The Message with the given ID does not exist.

2.6.6 *SendMessage()*

This action allows a TelCP to send a Message. This action applies to both page and Session mode Messaging. This action returns immediately once invoked without waiting for the Messaging service to successfully sends the Message to the remote user as specified by the TelCP.

2.6.6.1 Arguments

Table 2-21: Arguments for *SendMessage()*

Argument	Direction	relatedStateVariable
MessageToSend	*IN*	*A_ARG_TYPE_Message*
MessageID	*OUT*	*A_ARG_TYPE_MessageID*

2.6.6.2 Argument Descriptions

The input argument *MessageToSend* includes all the details of the Message including the recipients, the subject of the Message, the body of the Message, the attachments, etc. as specified in the section 2.4.10.

The output argument *MessageID* includes the unique identifier for the Message as created by the *Messaging* service. This unique identifier can be used to reference the Message.

2.6.6.3 Service Requirements

Once this action is invoked, then the *Messaging* service stores the submitted Message into the *Outgoing* folder. If the Messaging service successfully sends the Message to the remote party, then the

Message is moved from the *Outgoing* folder to the *Sent* folder. The Message will remain in the *Outgoing* folder until it is successfully sent to the remote party.

The Messaging service MUST generate a unique Message ID for the Message if the action is successful.

If the class element of the Message is set to the empty string, then the *Messaging* service MUST decide whether to send the Message as a SMS, or MMS, or email or Instant Message by examining the recipients list, attachments and length of the Message, etc.

The *Messaging* service sets the value of the status of the Message as "Unread" in the *Outgoing* folder. After the Message is sent successfully to the remote party the Message is moved to the *Sent* folder and the status of the Message is set as "Read" in the *Sent* folder.

2.6.6.4 Control Point Requirements When Calling The Action

The *MessageToSend* input argument MUST include all the mandatory details for the requested Message class as defined in the section 2.4.10.

The Message ID, the Message Folder, and the Message Status elements are managed by the *Messaging* service so the TelCP MUST provide an empty string in the Message structure of the Message.

In the case of Session Mode Messaging, the *MessageClass* element in the Message structure of the *MessageToSend* input argument MUST match with the current Session class as identified by the SessionID. If the TelCP wants to send the Message with a different Message Class, then the one specified Session class, then the TelCP first SHOULD change the Session class to "Mixed" by the *ModifySession()* action before invoking this action.

2.6.6.5 Dependency on Device State

None.

2.6.6.6 Effect on Device State

None.

2.6.6.7 Errors

Table 2-22: Error Codes for *SendMessage()*

ErrorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
606	Action not Autohrized	The CP does not have privileges to invoke this action.
703	Invalid Message	The input Message is invalid.

2.6.7 *DeleteMessage()*

This action allows a TelCP to delete a Message as identified by Message ID.

If the *Deleted* folder is supported by the Messaging service, then the deleted Message is moved to the *Deleted* folder, otherwise the Message will be permanently removed. The request for deleting a Message from the *Deleted* folder will also remove the Message permanently.

2.6.7.1 Arguments

Table 2-23: Arguments for *DeleteMessage()*

Argument	Direction	relatedStateVariable
MessageID	*IN*	*A_ARG_TYPE_MessageID*

2.6.7.2 Argument Descriptions

The input argument *MessageID* uniquely identifies the Message to be deleted.

2.6.7.3 Service Requirements

According to the Message lifecycle state diagram defined in section 2.7.1.1, the execution of this action has the following effects.

- If the *Deleted* folder is supported by the Messaging service, then the deleted Message is moved to the *Deleted* folder, otherwise the Message will be permanently removed.
- The request for deleting the Message from the *Deleted* folder will also remove the Message permanently.
- A deletion of the Message does not change the status of the Message.

2.6.7.4 Control Point Requirements When Calling The Action

None.

2.6.7.5 Dependency on Device State

None.

2.6.7.6 Effect on Device State

None.

2.6.7.7 Errors

Table 2-24: Error Codes for *DeleteMessage()*

ErrorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
606	Action not Autohrized	The CP does not have privileges to invoke this action.
701	Invalid Message ID	The Message with the given ID does not exist.

2.6.8 *CreateSession()*

This action allows a TelCP to create a new Messaging Session with a specific Session Class and a set of contacts.

If successful the *Messaging* service returns a unique identifier for the Messaging Session.

2.6.8.1 Arguments

Table 2-25: Arguments for *CreateSession()*

Argument	Direction	relatedStateVariable
SessionClass	*IN*	*A_ARG_TYPE_SessionClass*
SessionRecipients	*IN*	*A_ARG_TYPE_RecipientsList*
Subject	*IN*	*A_ARG_TYPE_Subject*
SupportedContentType	*IN*	*A_ARG_TYPE_SupportedContentType*
SessionID	*OUT*	*A_ARG_TYPE_SessionID*

2.6.8.2 Argument Descriptions

The input argument *SessionClass* allows a TelCP to specify the class of the Session (e.g., SMS, MMS, Chat, etc.).

The input argument *SessionRecipients* includes the list of recipients for the Messaging Session.

The input arguments *Subject* and *SupportedContentType* allows a TelCP to specify the title of the Session and the intended content types for the Session. These arguments are only used for the Session Mode Messaging.

The output argument *SessionID* uniquely identifies the Messaging Session as assigned by the *Messaging* service.

2.6.8.3 Service Requirements

In the case of a valid request with SessionClass as *Chat/FileTransfer*, the *Messaging* service will create a new SessionID and send the Session invitations to the recipients. The *Messaging* service keeps the association between the local Session ID (Session ID of the LAN Messaing Session) and the global Session ID (the Session identifier on the WAN side) in order to correctly route the incoming and outgoing Messages.

If the Session Class value is the empty string then *Messaging* service MUST decide the Session class based on the first Message exchange within the Messaging Session.

2.6.8.4 Control Point Requirements When Calling The Action

None.

2.6.8.5 Dependency on Device State

None.

2.6.8.6 Effect on Device State

None.

2.6.8.7 Errors

Table 2-26: Error Codes for *CreateSession()*

ErrorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.

ErrorCode	errorDescription	Description
606	Action not Autohrized	The CP does not have privileges to invoke this action.
704	Invalid recipients	One or more recipients are invalid.
710	Invalid Session class	The Session class is invalid.

2.6.9 *ModifySession()*

This action allows a TelCP to modify a Messaging Session which includes adding or removing contacts, or changing the subject, or the class, or the supported content types of the Session.

2.6.9.1 Arguments

Table 2-27: Arguments for *ModifySession()*

Argument	Direction	relatedStateVariable
SessionID	*IN*	*A_ARG_TYPE_SessionID*
SessionRecipientsToAdd	*IN*	*A_ARG_TYPE_RecipientsList*
SessionRecipientsToRemove	*IN*	*A_ARG_TYPE_RecipientsList*
Subject	*IN*	*A_ARG_TYPE_Subject*
SupportedContentType	*IN*	*A_ARG_TYPE_SupportedContentType*
SessionClass	*IN*	*A_ARG_TYPE_SessionClass*

2.6.9.2 Argument Descriptions

The input argument *SessionID* includes the unique identifier of the Session to modify.

The input arguments *SessionRecipientsToAdd* and *SessionRecipientsToRemove* contain the list of recipients to add or remove within the current Session.

The input arguments *Subject* and *SupportedContentType* allow the TelCP to modify the main topic and the types of contents that can be shared among the participants within the Session.

The input argument *SessionClass* allows the TelCP to modify the class of the Session.

2.6.9.3 Service Requirements

If the input arguments contain valid values, the *Messaging* service MUST modify only the changed parameter of the Session.

A Session can only change its initial class to "*Mixed*" since the Session will contain different classes of Messages (e.g., if an "*SMS*" Session is modified by a TelCP into an "*MMS*" Session, the resulting status of that Session is turned into "*Mixed*").

The class of a "*Mixed*" Session can not be modified.

2.6.9.4 Control Point Requirements When Calling The Action

A TelCP needs to fill the input parameters it doesn't want to modify with the empty string.

2.6.9.5 Dependency on Device State

None.

2.6.9.6 Effect on Device State

None.

2.6.9.7 Errors

Table 2-28: Error Codes for *ModifySession()*

ErrorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
606	Action not Autohrized	The CP does not have privileges to invoke this action.
704	Invalid recipients	One or more recipients are invalid.
706	Invalid Session ID	The Session with the given ID does not exist.

2.6.10 *AcceptSession()*

This action allows a TelCP to accept a request for a new Messaging Session. When a TelCP receives the event for an incoming Session request, the TelCP can accept the incoming Session by invoking this action.

2.6.10.1 Arguments

Table 2-29: Arguments for *AcceptSession()*

Argument	Direction	relatedStateVariable
SessionID	*IN*	*A_ARG_TYPE_SessionID*

2.6.10.2 Argument Descriptions

The input argument *SessionID* includes the unique identifier of the Session that the TelCP wants to accept.

2.6.10.3 Service Requirements

If the *SessionID* is valid, the *Messaging* service will accept the Session.

Only a Session that is currently "*Pending*" can be accepted by a TelCP. If the request is successfully accepted, the Session status MUST be changed to "*Running*".

2.6.10.4 Control Point Requirements When Calling The Action

None.

2.6.10.5 Dependency on Device State

None.

2.6.10.6 Effect on Device State

None.

2.6.10.7 Errors

Table 2-30: Error Codes for *AcceptSession()*

ErrorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
606	Action not Autohrized	The CP does not have privileges to invoke this action.
706	Invalid Session ID	The Session with the given ID does not exist.

2.6.11 *GetSessionUpdates()*

This action allows a TelCP to read the value of the *SessionUpdates* state variable, that is used to provide notifications of changes to the status of the *Messaging* Sessions.

This action can be used by a TelCP that doesn't subscribe for notifications from the *Messaging* service.

2.6.11.1 Arguments

Table 2-31: Arguments for *GetSessionUpdates()*

Argument	Direction	relatedStateVariable
SessionUpdates	*OUT*	*SessionUpdates*

2.6.11.2 Argument Descriptions

The output argument *SessionUpdates* contains the value of the *SessionUpdates* state variable. The value of the *SessionUpdates* state variable includes information about active Sessions.

2.6.11.3 Service Requirements

None.

2.6.11.4 Control Point Requirements When Calling The Action

None.

2.6.11.5 Dependency on Device State

None.

2.6.11.6 Effect on Device State

None.

2.6.11.7 Errors

Table 2-32: Error Codes for *GetSessionUpdates()*

ErrorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.

ErrorCode	errorDescription	Description
606	Action not Autohrized	The CP does not have privileges to invoke this action.

2.6.12 *GetSessions()*

This action allows a TelCP to retrieve the Messaging Sessions available on the *Messaging* service either identified by the Session ID or filtered by the *SessionClass* and by the *SessionStatus* input arguments.

2.6.12.1 Arguments

Table 2-33: Arguments for *GetSessions()*

Argument	Direction	relatedStateVariable
SessionID	*IN*	*A_ARG_TYPE_SessionID*
SessionClass	*IN*	*A_ARG_TYPE_SessionClass*
SessionStatus	*IN*	*A_ARG_TYPE_SessionStatus*
SessionsList	*OUT*	*A_ARG_TYPE_SessionsList*

2.6.12.2 Argument Descriptions

The input argument *SessionID* includes the unique identifier of the Session that the TelCP wants to retrieve. If the value of this argument is the empty string (“”), then the *Messaging* service will retrieve all the Sessions filtered by *SessionClass* and *SessionStatus*. If a valid Session id is specified the other input arguments are irrelevant.

The input argument *SessionClass* specifies the class of Sessions to be retrieved. If the value of this argument is the empty string (“”), then the *Messaging* service will retrieve information about Sessions for all classes.

The input argument *SessionStatus* specifies the value of the status of the Sessions to be retrieved. If the value of this argument is the empty string (“”), then the *Messaging* service will retrieve information about all the Sessions regardless of their status.

The Sessions that match the specified criteria are listed in the *SessionsList* output argument.

2.6.12.3 Service Requirements

When responding to this action, in the resulting list of Sessions the *Messaging* service will provide at least those details of Sessions that are defined as required XML elements in the related state variable section. However, additional optional elements can be included by the *Messaging* service in order to add further details.

2.6.12.4 Control Point Requirements When Calling The Action

None.

2.6.12.5 Dependency on Device State

None.

2.6.12.6 Effect on Device State

None.

2.6.12.7 Errors

Table 2-34: Error Codes for *GetSessions()*

ErrorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
606	Action not Autohrized	The CP does not have privileges to invoke this action.
706	Invalid Session ID	The Session with the given ID does not exist.
710	Invalid Session class	The Session class is invalid.
711	Invalid Session status	The Session status is invalid.

2.6.13 *JoinSession()*

This action allows a TelCP to join a "*Parked*" or a "*Running*" Session.

2.6.13.1 Arguments

Table 2-35: Arguments for *JoinSession()*

Argument	Direction	relatedStateVariable
SessionID	*IN*	*A_ARG_TYPE_SessionID*

2.6.13.2 Argument Descriptions

The input argument *SessionID* includes the unique identifier of the Session that the TelCP wants to join.

2.6.13.3 Service Requirements

This action can be successfully invoked only on "*Parked*" or "*Running*" Sessions. When invoked on a Session with a status other than "*Parked*" or "*Running*" the action will fail with an appropriate error code.

If the request is successfully accepted by the *Messaging* service on a "*Parked*" Session, the status of the Session MUST change to "*Running*".

2.6.13.4 Control Point Requirements When Calling The Action

None.

2.6.13.5 Dependency on Device State

None.

2.6.13.6 Effect on Device State

None.

2.6.13.7 Errors

Table 2-36: Error Codes for *JoinSession()*

ErrorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
606	Action not Autohrized	The CP does not have privileges to invoke this action.
706	Invalid Session ID	The Session with the given ID does not exist.
708	Invalid action	The action is invalid in the current context.

2.6.14 *LeaveSession()*

This action allows a TelCP to leave a running Session without closing the Session.

2.6.14.1 Arguments

Table 2-37: Arguments for *LeaveSession()*

Argument	Direction	relatedStateVariable
SessionID	*IN*	*A_ARG_TYPE_SessionID*

2.6.14.2 Argument Descriptions

The input argument *SessionID* includes the unique identifier of the Session that the TelCP wants to leave.

2.6.14.3 Service Requirements

Upon receiving this request, the *Messaging* service keeps locally the Session context, without leaving the corresponding remote Session. This allows another TelCP, or the same TelCP, to join the Session later.

This action can be successfully invoked only on "*Running*" Sessions. When invoked on a Session with a different status this action will fail with an appropriate error code.

Once the TelCP leaves the Session the status of the Session is changed to "*Parked*" if no other TelCPs are active in the Session.

2.6.14.4 Control Point Requirements When Calling The Action

None.

2.6.14.5 Dependency on Device State

None.

2.6.14.6 Effect on Device State

None.

2.6.14.7 Errors

Table 2-38: Error Codes for *LeaveSession()*

ErrorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
606	Action not Autohrized	The CP does not have privileges to invoke this action.
706	Invalid Session ID	The Session with the given ID does not exist.
708	Invalid action	The action is invalid in the current context.

2.6.15 *CloseSession()*

This action allows a TelCP to close a Messaging Session. If this action is invoked on a "*Pending*" Session then the Session request is refused and the Session is closed.

When the Session is closed the Session is moved to the "*ClosedSession*" folder. It is left to the implementation of the service how long the closed Sessions are kept in the "*ClosedSession*" folder.

2.6.15.1 Arguments

Table 2-39: Arguments for *CloseSession()*

Argument	Direction	relatedStateVariable
SessionID	*IN*	*A_ARG_TYPE_SessionID*

2.6.15.2 Argument Descriptions

The input argument *SessionID* includes the unique identifier of the Session that the TelCP wants to close.

2.6.15.3 Service Requirements

If the action is successful then the value of the Session status MUST is set to "Closed".

2.6.15.4 Control Point Requirements When Calling The Action

None.

2.6.15.5 Dependency on Device State

None.

2.6.15.6 Effect on Device State

None.

2.6.15.7 Errors

Table 2-40: Error Codes for *CloseSession()*

ErrorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture section on Control.

ErrorCode	errorDescription	Description
500-599	TBD	See UPnP Device Architecture section on Control.
606	Action not Autohrized	The CP does not have privileges to invoke this action.
706	Invalid Session ID	The Session with the given ID does not exist.

2.6.16 *StartFileTransfer()*

This action allows a TelCP to initiate a file transfer Session with a WAN side user.

2.6.16.1 Arguments

Table 2-41: Arguments for *StartFileTransfer()*

Argument	Direction	relatedStateVariable
FileInfoList	*IN*	*A_ARG_TYPE_FileInfoList*

2.6.16.2 Argument Descriptions

The input argument *FileInfoList* includes the details of the files to transfer and also Session related information for initiating the file transfer Session with a WAN user. This input argument includes the information of the file such as the type, size, and link of the file.

The *FileInfo* argument also includes the notification type. Sometimes TelCP may require periodic notification about the progress of the transfer, or the completion of the file transfer. If the value of the XML element *ProgressNotif* is set to “1” then the TelCP will get continuous notification of the progress of the transfer. If the value of the XML element *ProgressNotif* is set to “0” then the TelCP will get the notification after the file transfer is completed.

The element *SessionID* uniquely identifies the file transfer Session.

2.6.16.3 Service Requirements

Once the *Messaging* service receives this action with valid arguments, then it will send the appropriate response to this action and also retrieve the file from the TelCP using the link provided in the *FileInfo* argument.

The *Messaging* service can retrieve the file or a chunk of the file and then it will establish the WAN file transfer Session, and also notify the TelCPs about the progress of the file transfer Session as requested in the initial request.

When the file is not available for fetching from the link then, the *Messaging* service will notify the TelCP with the status “file can not be downloaded”, with SessionUpdates state variable.

Once the file transfer Session is successfully established, the *Messaging* service will notify the file transfer progress as requested by the TelCP in the *StartFileTransfer* action.

2.6.16.4 Control Point Requirements When Calling The Action

None.

2.6.16.5 Dependency on Device State

None.

2.6.16.6 Effect on Device State

None.

2.6.16.7 Errors

Table 2-42: Error Codes for *StartFileTransfer()*

ErrorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
606	Action not Autohrized	The CP does not have privileges to invoke this action.
706	Invalid Session ID	The Session with the given ID does not exist.

2.6.17 *CancelFileTransfer()*

This action is used to cancel a file transfer Session. The TelCP can use this action to close an ongoing file transfer Session and to remove all the Session related information.

2.6.17.1 Arguments

Table 2-43: Arguments for *CancelFileTransfer()*

Argument	Direction	relatedStateVariable
SessionID	*IN*	*A_ARG_TYPE_SessionID*

2.6.17.2 Argument Descriptions

This input argument *SessionID* contains the unique identifier of the ongoing file transfer Session.

2.6.17.3 Service Requirements

If successful, the *Messaging* service also cancels the ongoing file transfer Session in the WAN side and also notifies the file transfer cancellation by sending an event with the *SessionStatus* state variable.

2.6.17.4 Control Point Requirements When Calling The Action

This action SHOULD be invoked only on a "*File-Transfer*" or a "*Mixed*" Session.

2.6.17.5 Dependency on Device State

None.

2.6.17.6 Effect on Device State

None.

2.6.17.7 Errors

Table 2-44: Error Codes for *CancelFileTransfer()*

ErrorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
606	Action not Autohrized	The CP does not have privileges to invoke this action.

ErrorCode	errorDescription	Description
706	Invalid Session ID	The Session with the given ID does not exist.
709	No active file transfer	No active file transfer is ongoing within the Session.

2.6.18 *GetFileTransferSession()*

This action is used to retrieve the metadata of closed file transfer Session stored in the *ClosedSession* folder.

2.6.18.1 Arguments

Table 2-45: Arguments for *GetFileTransferSession()*

Argument	Direction	relatedStateVariable
SessionID	*IN*	*A_ARG_TYPE_SessionID*
FileInfoList	*OUT*	*A_ARG_TYPE_FileInfoList*

2.6.18.2 Argument Descriptions

The input argument *SessionID* contains the unique identifier of the file transfer Session to be retrieved.

The output argument *FileInfoList* contains the metadata of the retrieved file transfer Session.

2.6.18.3 Service Requirements

If successful, the *Messaging* service retrieves the file transfer Session metadata from the *ClosedSession* folder.

2.6.18.4 Control Point Requirements When Calling The Action

This action SHOULD be invoked only on a "*File-Transfer*" Sessions that have been closed.

2.6.18.5 Dependency on Device State

None.

2.6.18.6 Effect on Device State

None.

2.6.18.7 Errors

Table 2-46: Error Codes for *GetFileTransferSession()*

ErrorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
606	Action not Autohrized	The CP does not have privileges to invoke this action.
706	Invalid Session ID	The Session with the given ID does not exist.
710	Invalid Session class	The Session class is invalid.

2.6.19 Error Code Summary

The following table lists error codes common to actions for this service type. If an action results in multiple errors, the most specific error should be returned.

Table 2-47: Error Code Summary

ErrorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture section on Control.
500-599	TBD	See UPnP Device Architecture section on Control.
606	Action not Autohrized	The CP does not have privileges to invoke this action.
700		Reserved for future extensions.
701	Invalid Message ID	The Message with the given ID does not exist.
702	Invalid Message class	The Message class is invalid.
703	Invalid Message	The input Message is invalid.
704	Invalid recipients	One or more recipients are invalid.
705	Invalid Message status	The Message status is invalid.
706	Invalid Session ID	The Session with the given ID does not exist.
707	Invalid folder	The folder is invalid.
708	Invalid action	The action is invalid in the current context.
709	No active file transfer	No active file transfer is ongoing within the Session.
710	Invalid Session class	The Session class is invalid.
711	Invalid Session status	The Session status is invalid.
714	Identity does not exist	The identity of the TS has not been assigned yet by the telephony service in the WAN side.

Note: 800-899 Error Codes are not permitted for standard actions. See UPnP Device Architecture section on Control for more details.

2.7 Service Behavioral Model

2.7.1 State Diagrams

2.7.1.1 Message Lifecycle

The following figure describes the lifecycle of a Message. The Message is moved among the folders depending on invoked actions or some events.

1. When a new Message is received, the Messaging service stores the Message in the "*Received*" folder.

2. A Message stored in the "*Received*" folder can be deleted by a TelCP by invoking the DeleteMessage() action with the ID of the Message. After this action succeeded, the Message is moved into the "*Deleted*" folder.
3. A Message stored in the "*Deleted*" folder can be permanently removed by invoking the DeleteMessage() action with the ID of the Message.
4. A Message can be sent by invoking the SendMessage() with the details of the Message to send. After this action succeeded, the Message is moved into the "*Outgoing*" folder.
5. A Message stored in the "*Outgoing*" folder is moved into the "*Sent*" folder after the Message is sent successfully to the remote party.
6. A Message stored in the "*Outgoing*" or in the "*Sent*" folder can be deleted by a TelCP by invoking the DeleteMessage() action with the ID of the Message. After this action succeeded the Message is moved into the "*Deleted*" folder.

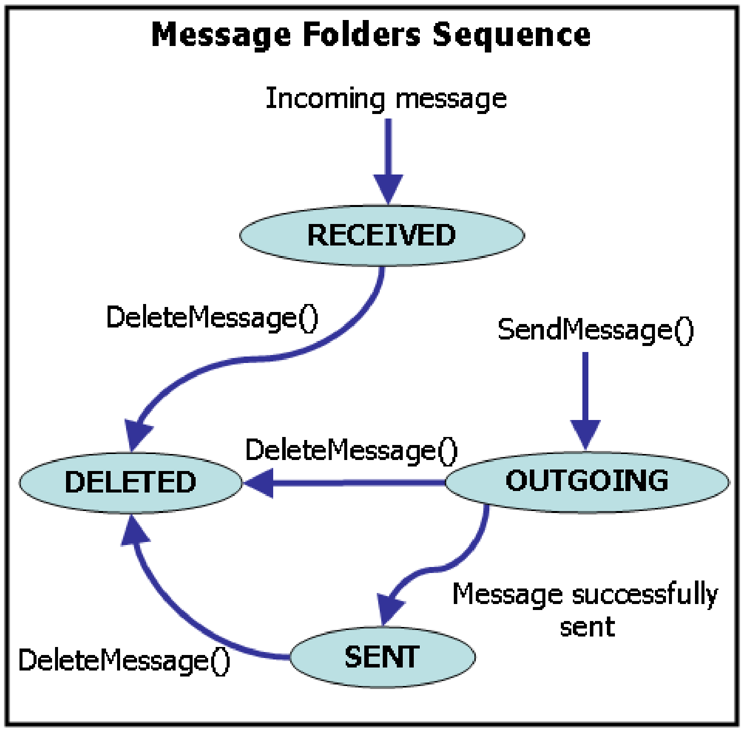


Figure 2-2: Lifecycle of Messages

2.7.1.2 Session Lifecycle

The following figure describes the lifecycle of a Session.

1. A TelCP creates a Session by invoking the CreateSession() action. After this action is successful, the new Session is created and its status is initialized as "*Pending*".
2. When the Messaging service receives the Session request from the remote party, the *Messaging* service creates a new Session and its status is initialized as "*Pending*".
3. A TelCP accepts a Session by invoking the AcceptSession() action. After this action is successful, the status of the Session is updated to "*Running*".
4. A TelCP joins a Session ("*Parked*" or "*Running*") by invoking the JoinSession() action. After this action is successful, the status of the Session is updated to "*Running*".

5. A TelCP leaves a "*Running*" Session by invoking the LeaveSession() action. After this action is successful, the status of the Session is updated to "*Parked*" if no other TelCPs are participating in the Session.
6. A TelCP closes a Session by invoking the CloseSession() action. After this action is successful, the status of the Session is updated to "*Closed*".

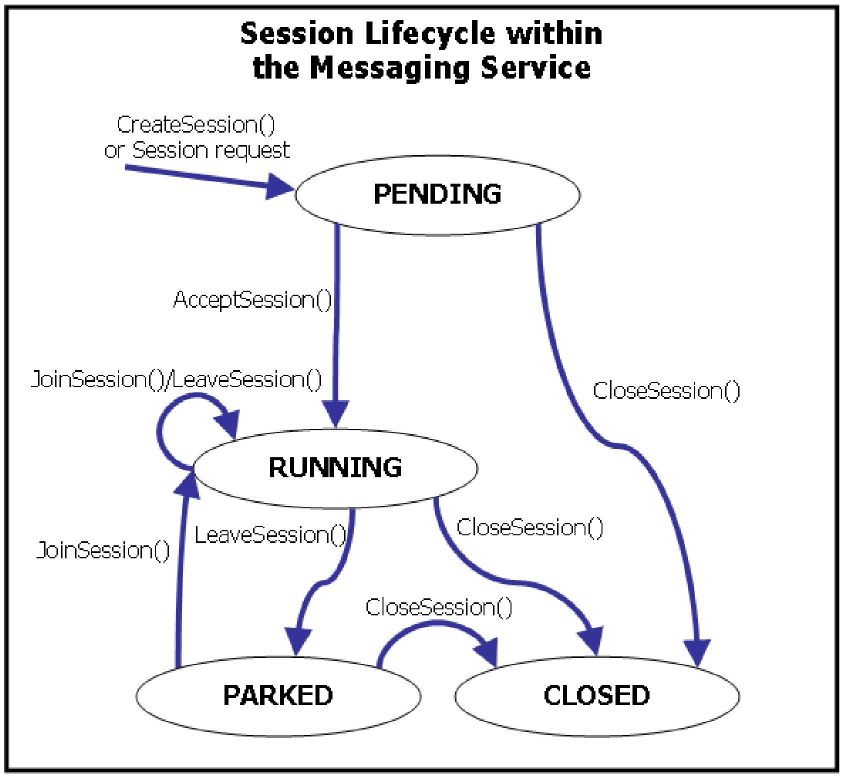


Figure 2-3: Lifecycle of Sessions

3 Theory of Operation (Informative)

3.1 Sending and Receiving Message

The following sequence describes how to send and receive Messages both in page mode and Session mode messaging.

1. When the TS receives an incoming Message from the WAN side, the Messaging service sends a NewMessages event to the TelCP. This event includes the ID of the incoming Message and optionally an overview of the Message.
2. After getting the event the TelCP invokes the ReadMessage() action on the TS with the ID of the incoming Message. The response to this action includes the details of the incoming Message.
3. The TelCP deletes a Message by invoking the DeleteMessage() action with the ID of the Message to delete.
4. The TelCP creates the Message with all the required details for the Message state variable and invokes the SendMessage() action no the TS. The TS returns the ID of the Message in response.
5. The TS keeps the Message in the Outgoing folder until it is successfully delivered to the WAN side. Once it is delivered to the WAN side the Message is moved to the Sent folder.
6. The TelCP can check if the Message has been successfully sent to the WAN side by invoking the ReadMessage() action

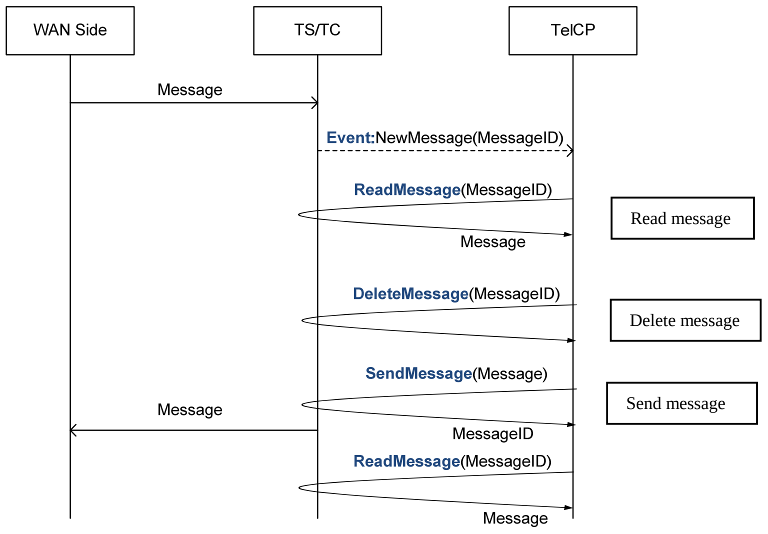


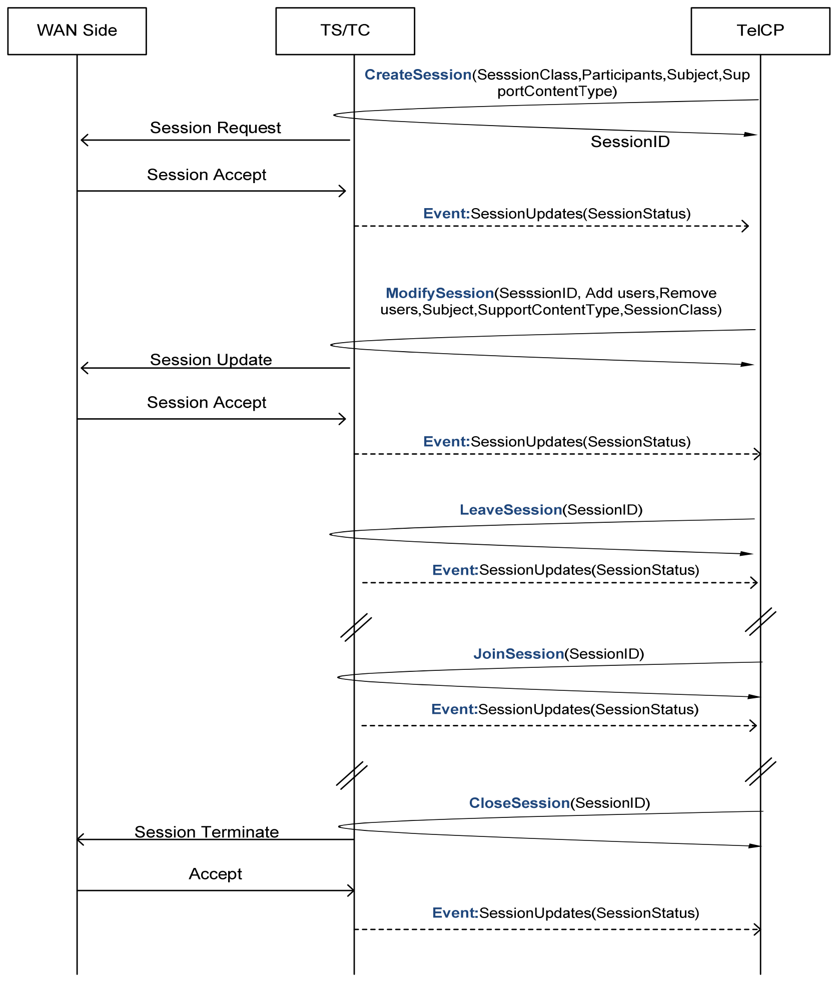
Figure 3-1: Message handling

3.2 Managing Session

The following sequence describes how to create, modify, join or close a messaging Session.

1. The TelCP creates the messaging Session by invoking the CreateSession() action on the TS with the detail information of the Session. The TS returns the ID of the Session in the response, and try to initiate the Session with the WAN side.

2. When the Session is successfully initiated in the WAN side, the Messaging service sends a SessionUpdates event to the TelCP. This event includes the status of the Session as "*Running*".
3. The TelCP modifies the messaging Session by invoking the ModifySession() action on the TS with the new information for the Session. The TS try to update the existing Session in the WAN side.
4. When the Session update is successful in the WAN side, the Messaging service sends the SessionUpdates event to the TelCP. This event includes "session modified" notification in the <sessionEvent> element.
5. The TelCP leaves the messaging Session by invoking the LeaveSession() action on the TS with the ID of the Session.
6. When the TS successfully removed the TelCP from the existing messaging Session, the Messaging service sends a SessionUpdates event to the TelCP. This event includes the status of the Session (e.g, "*Parked*" if no TelCP is active in the Session).
7. The TelCP joins the messaging Session in the TS by invoking the JoinSession() action with ID of the Session.
8. When the TS successfully added the TelCP in the messaging Session, the Messaging service sends a SessionUpdates event to the TelCP. This event includes the status of the Session as "*Running*"
9. The TelCP close the messaging Session by invoking the CloseSession() action with ID of the Session.
10. When the TS successfully terminated the messaging Session, the Messaging service sends a SessionUpdates event to the TelCP. This event includes the status of the Session as "*Closed*".



WAN Side
TS/TC
TelCP
CreateSession(SesssionClass,Participants,Subject,SupportContentType)
Session Request
SessionID
Session Accept
Event:SessionUpdates(SessionStatus)
ModifySession(SesssionID, Add users,Remove users,Subject,SupportContentType,SessionClass)
Session Update
Session Accept
Event:SessionUpdates(SessionStatus)
LeaveSession(SessionID)
Event:SessionUpdates(SessionStatus)
JoinSession(SessionID)
Event:SessionUpdates(SessionStatus)
CloseSession(SessionID)
Session Terminate
Accept
Event:SessionUpdates(SessionStatus)

Figure 3-2: Session handling in the Session Mode Messaging

The following sequence describes how to accept an incoming the messaging Session.

1. When the TS receives an incoming messaging Session request from the WAN side, the Messaging service sends the SessionUpdates event to the TelCP. This event includes the ID of the Session, and status of the Session as "*Pending*".
2. After getting the event the TelCP invokes the AcceptSession() action on the TS with the ID of the Session.
3. When the Session is successfully initiated, the Messaging service sends a SessionUpdates event to the TelCP. This event includes the status of the Session as "*Running*".

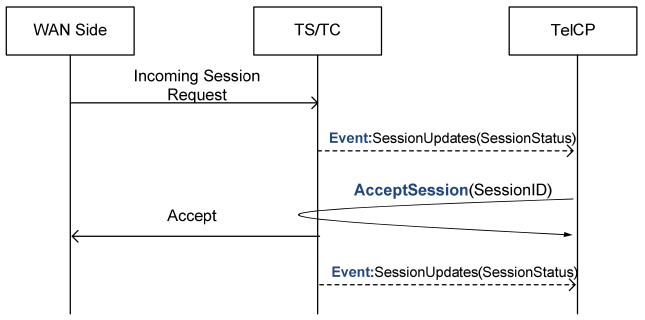


Figure 3-3: Incoming Session handling in the Session Mode Messaging

3.3 Managing File transfer

The following sequence describes how to transfer a file.

1. The TelCP creates the file transfer Session by invoking the CreateSession() action on the TS with the detail information of the Session and the class of the Session as File-Transfer. The TS returns the ID of the Session in the response.
2. After successfully creating the file transfer Session, the TelCP invokes the StartFileTransfer() action with the detail information of the files.
3. After receiving the StartFileTransfer() action, the Messaging service try to initiate a File transfer Session with the WAN side. Once the file transfer request is successful, then the Messaging service sends a SessionUpdates event to the TelCP. The event includes the status of the file transfer Session as "*Running*" and "File Transfer Started" in the <sessionEvent> element. The TS fetch the entire file or the chunks of the file and send the file or chunks of the file to the WAN side.
4. Once the file transfer is completely transferred to the WAN side, the Messaging service will sends the SessionUpdates event to the TelCP. This event includes "File Transfer completed" notification in the <sessionEvent> element.
5. After getting the event the TelCP invokes the CloseSession() action on the TS with the ID of the file transfer Session to close the file transfer Session.
6. After receiving the CloseSession() action, The Messaging service terminates the file transfer Session and sends the SessionUpdates event to the TelCP. The event includes the status of the file transfer Session as "*Closed*".
7. The TelCP can cancel the ongoing file transfer by invoking the CancelFileTransfer() action on the TS with the ID of the file transfer Session.
8. After receiving the CancelFileTransfer() action, the Messaging service terminates the ongoing file transfer and sends the SessionUpdates event to the TelCP. The event includes "File Transfer cancelled" notification in the <sessionEvent> element.

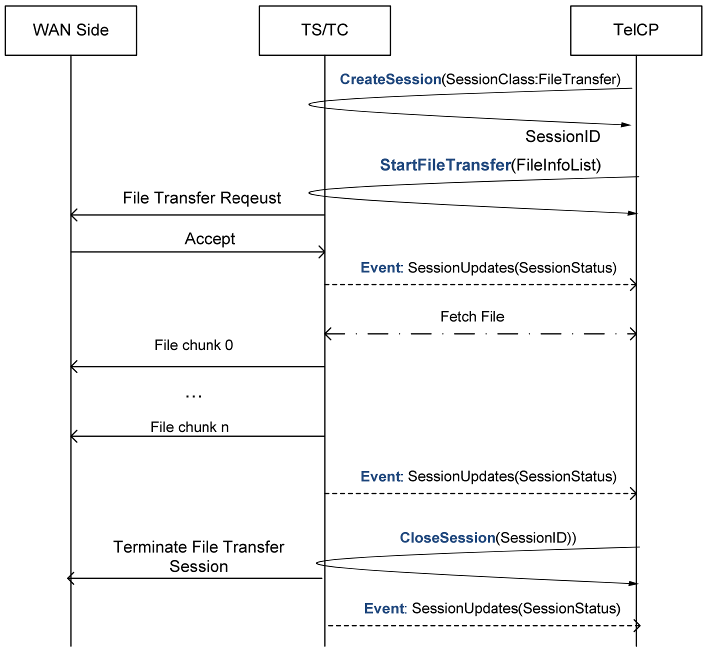

WAN Side
TS/TC
TelCP
CreateSession(SessionClass:FileTransfer)
SessionID
StartFileTransfer(FileInfoList)
File Transfer Reqeust
Accept
Event: SessionUpdates(SessionStatus)
Fetch File
File chunk 0
…
File chunk n
Event: SessionUpdates(SessionStatus)
CloseSession(SessionID))
Terminate File Transfer Session
Event: SessionUpdates(SessionStatus)

Figure 3-4: File Transfer Session handling

4 XML Service Description

```
<?xml version="1.0"?>
<scpd xmlns="urn:schemas-upnp-org:service-1-0">

   <specVersion>
      <major>1</major>
      <minor>0</minor>
   </specVersion>

      <action>
         <name>GetTelephonyIdentity</name>
         <argumentList>
            <argument>
               <name>TelephonyIdentity</name>
               <direction>out</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_TelephonyServerIdentity
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>

      <action>
         <name>GetMessagingCapabilities</name>
         <argumentList>
            <argument>
               <name>SupportedCapabilities</name>
               <direction>out</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_MessagingCapabilities
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>

      <action>
         <name>GetNewMessages</name>
         <argumentList>
            <argument>
               <name>NewMessages</name>
               <direction>out</direction>
               <relatedStateVariable>
                  NewMessages
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>

      <action>
         <name>SearchMessages</name>
         <argumentList>
            <argument>
               <name>MessageClass</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_MessageClass
               </relatedStateVariable>
            </argument>
            <argument>
               <name>MessageFolder</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_MessageFolder
               </relatedStateVariable>
            </argument>
            <argument>
               <name>MessageStatus</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_MessageStatus
               </relatedStateVariable>
            </argument>
            <argument>
               <name>SessionID</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_SessionID
               </relatedStateVariable>
            </argument>
            <argument>
               <name>MessageList</name>
               <direction>out</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_MessageList
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>

      <action>
         <name>ReadMessage</name>
         <argumentList>
            <argument>
               <name>MessageID</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_MessageID
               </relatedStateVariable>
            </argument>
            <argument>
               <name>MessageRequested</name>
               <direction>out</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_Message
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>

      <action>
         <name>SendMessage</name>
         <argumentList>
            <argument>
               <name>MessageToSend</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_Message
               </relatedStateVariable>
            </argument>
            <argument>
```

```
                  <name>MessageID</name>
                  <direction>out</direction>
                  <relatedStateVariable>
                     A_ARG_TYPE_MessageID
                  </relatedStateVariable>
               </argument>
            </argumentList>
         </action>

         <action>
            <name>DeleteMessage</name>
            <argumentList>
               <argument>
                  <name>MessageID</name>
                  <direction>in</direction>
                  <relatedStateVariable>
                     A_ARG_TYPE_MessageID
                  </relatedStateVariable>
               </argument>
            </argumentList>
         </action>

         <action>
            <name>CreateSession</name>
            <argumentList>
               <argument>
                  <name>SessionClass</name>
                  <direction>in</direction>
                  <relatedStateVariable>
                     A_ARG_TYPE_SessionClass
                  </relatedStateVariable>
               </argument>
               <argument>
                  <name>SessionRecipients</name>
                  <direction>in</direction>
                  <relatedStateVariable>
                     A_ARG_TYPE_RecipientsList
                  </relatedStateVariable>
               </argument>
               <argument>
                  <name>Subject</name>
                  <direction>in</direction>
                  <relatedStateVariable>
                     A_ARG_TYPE_String
                  </relatedStateVariable>
               </argument>
               <argument>
                  <name>SupportedContentType</name>
                  <direction>in</direction>
                  <relatedStateVariable>
                     A_ARG_TYPE_SupportedContentType
                  </relatedStateVariable>
               </argument>
               <argument>
                  <name>SessionID</name>
                  <direction>out</direction>
                  <relatedStateVariable>
                     A_ARG_TYPE_SessionID
                  </relatedStateVariable>
               </argument>
            </argumentList>
```

```
      </action>

      <action>
         <name>ModifySession</name>
         <argumentList>
            <argument>
               <name>SessionID</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_SessionID
               </relatedStateVariable>
            </argument>
            <argument>
               <name>SessionRecipientsToAdd</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_RecipientsList
               </relatedStateVariable>
            </argument>
            <argument>
               <name>SessionRecipientsToRemove</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_RecipientsList
               </relatedStateVariable>
            </argument>
            <argument>
               <name>Subject</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_String
               </relatedStateVariable>
            </argument>
            <argument>
               <name>SupportedContentType</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_SupportedContentType
               </relatedStateVariable>
            </argument>
            <argument>
               <name>SessionClass</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_SessionClass
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>

      <action>
         <name>AcceptSession</name>
         <argumentList>
            <argument>
               <name>SessionID</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_SessionID
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>

      <action>
         <name>GetSessionUpdates</name>
         <argumentList>
            <argument>
               <name>SessionUpdates</name>
               <direction>out</direction>
               <relatedStateVariable>
                  SessionUpdates
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>

      <action>
         <name>GetSessions</name>
         <argumentList>
            <argument>
               <name>SessionID</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_SessionID
               </relatedStateVariable>
            </argument>
            <argument>
               <name>SessionClass</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_SessionClass
               </relatedStateVariable>
            </argument>
            <argument>
               <name>SessionStatus</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_SessionStatus
               </relatedStateVariable>
            </argument>
            <argument>
               <name>SessionsList</name>
               <direction>out</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_SessionsList
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>

      <action>
         <name>JoinSession</name>
         <argumentList>
            <argument>
               <name>SessionID</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_SessionID
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>
```

```
<action>
   <name>LeaveSession</name>
   <argumentList>
      <argument>
         <name>SessionID</name>
         <direction>in</direction>
         <relatedStateVariable>
            A_ARG_TYPE_SessionID
         </relatedStateVariable>
      </argument>
   </argumentList>
</action>

<action>
   <name>CloseSession</name>
   <argumentList>
      <argument>
         <name>SessionID</name>
         <direction>in</direction>
         <relatedStateVariable>
            A_ARG_TYPE_SessionID
         </relatedStateVariable>
      </argument>
   </argumentList>
</action>

<action>
   <name>StartFileTransfer</name>
   <argumentList>
      <argument>
         <name>FileInfoList</name>
         <direction>in</direction>
         <relatedStateVariable>
            A_ARG_TYPE_FileInfoList
         </relatedStateVariable>
      </argument>
   </argumentList>
</action>

<action>
   <name>CancelFileTransfer</name>
   <argumentList>
      <argument>
         <name>SessionID</name>
         <direction>in</direction>
         <relatedStateVariable>
            A_ARG_TYPE_SessionID
         </relatedStateVariable>
      </argument>
   </argumentList>
</action>

<action>
   <name>GetFileTransferSession</name>
   <argumentList>
      <argument>
         <name>SessionID</name>
         <direction>in</direction>
         <relatedStateVariable>
            A_ARG_TYPE_SessionID
            </relatedStateVariable>
         </argument>
         <argument>
            <name>FileInfoList</name>
            <direction>out</direction>
            <relatedStateVariable>
               A_ARG_TYPE_FileInfoList
            </relatedStateVariable>
         </argument>
      </argumentList>
   </action>

</actionList>

<serviceStateTable>

   <stateVariable sendEvents="no">
      <name>A_ARG_TYPE_MessagingCapabilities</name>
      <dataType>string</dataType>
   </stateVariable>

   <stateVariable sendEvents="yes">
      <name>NewMessages</name>
      <dataType>string</dataType>
   </stateVariable>

   <stateVariable sendEvents="yes">
      <name>SessionUpdates</name>
      <dataType>string</dataType>
   </stateVariable>

   <stateVariable sendEvents="no">
      <name>A_ARG_TYPE_TelephonyServerIdentity</name>
      <dataType>string</dataType>
   </stateVariable>

   <stateVariable sendEvents="no">
      <name>A_ARG_TYPE_MessageID</name>
      <dataType>string</dataType>
   </stateVariable>

   <stateVariable sendEvents="no">
      <name>A_ARG_TYPE_MessageClass</name>
      <dataType>string</dataType>
      <allowedValueList>
         <allowedValue>e-Mail</allowedValue>
         <allowedValue>SMS</allowedValue>
         <allowedValue>MMS</allowedValue>
         <allowedValue>Instant Message</allowedValue>
         <allowedValue/>
      </allowedValueList>
   </stateVariable>

   <stateVariable sendEvents="no">
      <name>A_ARG_TYPE_MessageFolder</name>
      <dataType>string</dataType>
      <allowedValueList>
         <allowedValue>Received</allowedValue>
         <allowedValue>Outgoing</allowedValue>
         <allowedValue>Sent</allowedValue>
         <allowedValue>Deleted</allowedValue>
         <allowedValue>ClosedSession</allowedValue>
         <allowedValue/>
      </allowedValueList>
      <defaultValue>Received</defaultValue>
   </stateVariable>

   <stateVariable sendEvents="no">
      <name>A_ARG_TYPE_MessageStatus</name>
      <dataType>string</dataType>
      <allowedValueList>
         <allowedValue>Read</allowedValue>
         <allowedValue>Unread</allowedValue>
         <allowedValue/>
      </allowedValueList>
   </stateVariable>

   <stateVariable sendEvents="no">
      <name>A_ARG_TYPE_Message</name>
      <dataType>string</dataType>
   </stateVariable>

   <stateVariable sendEvents="no">
      <name>A_ARG_TYPE_MessageList</name>
      <dataType>string</dataType>
   </stateVariable>

   <stateVariable sendEvents="no">
      <name>A_ARG_TYPE_SessionID</name>
      <dataType>string</dataType>
   </stateVariable>

   <stateVariable sendEvents="no">
      <name>A_ARG_TYPE_SessionClass</name>
      <dataType>string</dataType>
      <allowedValueList>
         <allowedValue>e-Mail</allowedValue>
         <allowedValue>SMS</allowedValue>
         <allowedValue>MMS</allowedValue>
         <allowedValue>Chat</allowedValue>
         <allowedValue>File-Transfer</allowedValue>
         <allowedValue>Mixed</allowedValue>
         <allowedValue/>
      </allowedValueList>
      <defaultValue>Chat</defaultValue>
   </stateVariable>

   <stateVariable sendEvents="no">
      <name>A_ARG_TYPE_RecipientsList</name>
      <dataType>string</dataType>
   </stateVariable>

   <stateVariable sendEvents="no">
      <name>A_ARG_TYPE_SessionInfo</name>
      <dataType>string</dataType>
   </stateVariable>

   <stateVariable sendEvents="no">
      <name>A_ARG_TYPE_SessionsList</name>
      <dataType>string</dataType>
   </stateVariable>
```

```
      <stateVariable sendEvents="no">
         <name>A_ARG_TYPE_SupportedContentType</name>
         <dataType>string</dataType>
      </stateVariable>

      <stateVariable sendEvents="no">
         <name>A_ARG_TYPE_String</name>
         <dataType>string</dataType>
      </stateVariable>

      <stateVariable sendEvents="no">
         <name>A_ARG_TYPE_SessionStatus</name>
         <dataType>string</dataType>
         <allowedValueList>
            <allowedValue>Pending</allowedValue>
            <allowedValue>Running</allowedValue>
            <allowedValue>Parked</allowedValue>
            <allowedValue>Closed</allowedValue>
            <allowedValue/>
         </allowedValueList>
      </stateVariable>

      <stateVariable sendEvents="no">
         <name>A_ARG_TYPE_FileInfoList</name>
         <dataType>string</dataType>
      </stateVariable>

   </serviceStateTable>
</scpd>
```

Appendix A. XML complex type *peerType* (Normative)

A communication means the exchange of an information between two or more end entities. These end entities are herein referred as Peers. The Peer can be a caller of a phone call, recipient of an email message, or group of participants in a communication session, or a contact in an Address book.

In order to have a uniform representation of a *Peer* across all the services in the UPnP Telephony, the XML complex type `peerType` is defined. The same XML complex type can be reused by other UPnP Telephony services.

The complex type peerType contains the information to properly identify a contact and its communication address for e.g. a phone call needs a telephone number, an email message needs an email address etc. If TS supports the PhoneManagement profile, then the correspondence between the Peer element and either a contact or a group of contacts in the Address book is also included in the complex peerType element.

A.1 Using the *peerType* within XML Schemas

The complex type `peerType` can be used in the XML schemas by including the following statement:

```
<import
   namespace="urn:schemas-upnp-org:phone:peer"
   schemaLocation="http://www.upnp.org/schemas/phone/peer-v1.xsd"/>
```

where the `schemaLocation` refers to the last updated schema file for the *Peer*.

A.2 Description of fields of a `peerType` complex type

This section gives a description of the elements defined in the peerType complex type.

```
<?xml version="1.0" encoding="UTF-8"?>
<peer:peer
   xsi:schemaLocation="urn:schemas-upnp-org:phone:peer
   http://www.upnp.org/schemas/phone/peer-v1.xsd"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:peer="urn:schemas-upnp-org:phone:peer">
   <peer:id>
      The identifier of the Peer (e.g., phone number, etc.)
   </peer:id>
   <peer:name>A user friendly name for the Peer</peer:name>
   <peer:contactInstanceId>
      The instance identifier for a contact referred by the Peer
   </peer:contactInstanceId>
   <peer:groupInstanceId>
      The instance identifier for a group referred by the Peer
   </peer:groupInstanceId>
</peer:peer>
```

id

REQUIRED, xsd:string. Indicates the communication address or the identifier for the Peer (e.g., a telephone number, an e-mail address, an identifier of a group of contacts, etc).

name

OPTIONAL, xsd:string. Indicates a user friendly name for the Peer.

contactInstanceId

OPTIONAL, xsd:unsignedInt. Is the instance identifier of the contact present in the *Phone Data Model's* Address Book for the referenced *Peer*. The value of the contactInstanceId is an unsigned integer. If there is no Instance in the *Address Book* for the referenced *Peer*, then the contactInstanceId value MUST be 0 (no match with the list of contacts in the Address Book). If the PhoneManagement profile is not supported or the relationship between the *Address Book* and the *Peer* is not used by the service, then this element MUST NOT be used. The contactInstanceId and groupInstanceId are mutually exclusive elements.

groupInstanceId

OPTIONAL, xsd:unsignedInt. Is the instance identifier of a group present in the *Phone Data Model's Address Book* for the referenced Peer. The value of groupInstanceId is an unsigned integer. If there is no Instance in the *Address Book* for this referenced *Peer*, then the groupInstanceId value MUST be 0 (no match with the list of groups in the *Address* Book). If the PhoneManagement profile is not supported or the relationship between the *Address Book* and the *Peer* is not used by the service, then this element MUST NOT be used. The contactInstanceId and groupInstanceId are mutually exclusive elements.

any

OPTIONAL. Attachment point for custom extensions.

A.3 peerType Schema

The following XML schema defines the peerType complex type.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema" xmlns:peer="urn:schemas-upnp-
org:phone:peer" targetNamespace="urn:schemas-upnp-org:phone:peer"
elementFormDefault="qualified" attributeFormDefault="qualified" version="1">
   <complexType name="peerType">
      <sequence>
         <element name="id" nillable="0">
            <annotation>
               <documentation>Id of the peer. The content depends on the context. For
example it can be a phone number, ad e-mail address and so on.</documentation>
            </annotation>
            <complexType>
               <simpleContent>
                  <extension base="string"/>
               </simpleContent>
            </complexType>
         </element>
         <element name="name" type="string" nillable="0" minOccurs="0">
            <annotation>
               <documentation>Textual name of the peer. In case the Phone Data Model
is supported, this  element MUST be the FormattedName in the address
book.</documentation>
            </annotation>
         </element>
         <choice minOccurs="0">
            <element name="contactInstanceId" nillable="0">
               <annotation>
                  <documentation>The Instance Identifier of a Contact in the PDM
address book.</documentation>
               </annotation>
               <complexType>
                  <simpleContent>
                     <extension base="unsignedInt"/>
                  </simpleContent>
               </complexType>
            </element>
            <element name="groupInstanceId">
               <annotation>
                  <documentation>The Instance Identifier of a Group in the PDM address
book.</documentation>
               </annotation>
            </element>
         </choice>
         <any namespace="##other" minOccurs="0">
            <annotation>
               <documentation>Vendor defined extensions attachment
point.</documentation>
            </annotation>
         </any>
      </sequence>
   </complexType>
</schema>
```

Appendix B.   XML Schema

This appendix provides the global XML Schema for syntactical validation of all the XML fragments used in the *Messaging* service.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns:messaging="urn:schemas-upnp-org:phone:messaging"
xmlns="http://www.w3.org/2001/XMLSchema" xmlns:peer="urn:schemas-upnp-org:phone:peer"
targetNamespace="urn:schemas-upnp-org:phone:messaging"
elementFormDefault="unqualified" attributeFormDefault="unqualified">
  <import namespace="urn:schemas-upnp-org:phone:peer"
schemaLocation="http://www.upnp.org/schemas/phone/peer-v1.xsd"/>
  <simpleType name="messageID.type">
    <annotation>
      <documentation>Corresponding state variable: A_ARG_TYPE_MessageID. Text format
is: [SessionID.]MessageID</documentation>
    </annotation>
    <restriction base="string">
      <pattern value="[a-zA-Z0-9]*(\.[a-zA-Z0-9]+)?"/>
    </restriction>
  </simpleType>
  <simpleType name="messageClass.type">
    <annotation>
      <documentation>Corresponding state variable:
A_ARG_TYPE_MessageClass</documentation>
    </annotation>
    <restriction base="string"/>
  </simpleType>
  <simpleType name="messageFolder.type">
    <annotation>
      <documentation>Corresponding state variable:
A_ARG_TYPE_MessageFolder</documentation>
    </annotation>
    <restriction base="string"/>
  </simpleType>
  <simpleType name="messageStatus.type">
    <annotation>
      <documentation>Corresponding state variable:
A_ARG_TYPE_MessageStatus</documentation>
    </annotation>
    <restriction base="string"/>
  </simpleType>
  <simpleType name="sessionID.type">
    <annotation>
      <documentation>Corresponding state variable:
A_ARG_TYPE_SessionID</documentation>
    </annotation>
    <restriction base="string">
      <pattern value="[a-zA-Z0-9]*"/>
    </restriction>
  </simpleType>
  <simpleType name="sessionClass.type">
    <annotation>
      <documentation>Corresponding state variable:
A_ARG_TYPE_SessionClass</documentation>
    </annotation>
    <restriction base="string"/>
  </simpleType>
  <simpleType name="sessionStatus.type">
    <annotation>
      <documentation>Corresponding state variable:
A_ARG_TYPE_SessionStatus</documentation>
    </annotation>
    <restriction base="string"/>
  </simpleType>
  <simpleType name="supportedContentType.type">
    <annotation>
      <documentation>Corresponding state variable:
A_ARG_TYPE_SupportedContentType</documentation>
    </annotation>
    <restriction base="string"/>
  </simpleType>
  <complexType name="recipientsList.type">
    <annotation>
      <documentation>Complex type defining a list of recipients</documentation>
    </annotation>
    <sequence>
      <element name="recipientTo" type="peer:peerType" minOccurs="0"
maxOccurs="unbounded"/>
```

```
      <element name="recipientCc" type="peer:peerType" minOccurs="0"
maxOccurs="unbounded"/>
      <element name="recipientBcc" type="peer:peerType" minOccurs="0"
maxOccurs="unbounded"/>
      <element name="recipientFrom" type="peer:peerType"/>
    </sequence>
  </complexType>
  <complexType name="attachment.type">
    <annotation>
      <documentation>Complext type defining an attachment to a message</documentation>
    </annotation>
    <sequence>
      <element name="uRI" type="anyURI"/>
      <element name="size" type="nonNegativeInteger" minOccurs="0"/>
      <element name="mimeType" type="string" minOccurs="0"/>
    </sequence>
  </complexType>
  <complexType name="message.type">
    <annotation>
      <documentation>Complex type defining the possible properties of a message of any
Class.</documentation>
    </annotation>
    <sequence>
      <element name="messageID" type="messaging:messageID.type"/>
      <element name="sessionID" type="messaging:sessionID.type" minOccurs="0"/>
      <element name="messageClass" type="messaging:messageClass.type"/>
      <element name="messageFolder" type="messaging:messageFolder.type"/>
      <element name="messageStatus" type="messaging:messageStatus.type"/>
      <element name="replyMessageID" type="messaging:messageID.type" minOccurs="0"/>
      <element name="recipientsList" type="messaging:recipientsList.type"/>
      <element name="subject" type="string" minOccurs="0"/>
      <element name="text" type="string" minOccurs="0"/>
      <element name="attachments" minOccurs="0">
        <complexType>
          <sequence maxOccurs="unbounded">
            <element name="attachment" type="messaging:attachment.type"/>
          </sequence>
        </complexType>
      </element>
      <element name="dateSent" type="dateTime" minOccurs="0"/>
      <element name="dateReceived" type="dateTime" minOccurs="0"/>
    </sequence>
  </complexType>
  <complexType name="newMessage.type">
    <annotation>
      <documentation>Complex type defining the possible proprieties of a new incoming
message</documentation>
    </annotation>
    <sequence>
      <element name="messageID" type="messaging:messageID.type"/>
      <element name="messageOverview" type="string" minOccurs="0"/>
    </sequence>
  </complexType>
  <complexType name="session.type">
    <annotation>
      <documentation>Complex type defining the possible properties of a messaging
session of any Class.</documentation>
    </annotation>
    <sequence>
      <element name="sessionID" type="messaging:sessionID.type"/>
      <element name="sessionClass" type="messaging:sessionClass.type"/>
      <element name="subject" type="string" minOccurs="0"/>
      <element name="sessionStatus" type="messaging:sessionStatus.type"/>
      <element name="supportedContentType" type="messaging:supportedContentType.type"
minOccurs="0"/>
      <element name="recipientsList" type="messaging:recipientsList.type"/>
      <element name="listOfMessages" minOccurs="0">
        <complexType>
          <sequence minOccurs="0" maxOccurs="unbounded">
            <element name="messageID" type="messaging:messageID.type"/>
          </sequence>
        </complexType>
      </element>
      <element name="dateStarted" type="dateTime" minOccurs="0"/>
      <element name="dateLastMessage" type="dateTime" minOccurs="0"/>
      <element name="dateEnded" type="dateTime" minOccurs="0"/>
    </sequence>
  </complexType>
  <complexType name="fileInfo.type">
    <annotation>
```

```
      <documentation>Complex type defining the information of the file to
transfer</documentation>
    </annotation>
    <sequence>
      <element name="fileType" type="string"/>
      <element name="fileSize" type="nonNegativeInteger"/>
      <element name="fileLink" type="anyURI"/>
      <element name="progressNotif">
        <complexType>
          <simpleContent>
            <extension base="boolean">
              <attribute name="time" type="positiveInteger"/>
            </extension>
          </simpleContent>
        </complexType>
      </element>
    </sequence>
  </complexType>
  <complexType name="fileInfoList.type">
    <annotation>
      <documentation>Complex type defining the information associated to a file
transfer session</documentation>
    </annotation>
    <sequence>
      <element name="sessionID" type="messaging:sessionID.type"/>
      <element name="fileInfo" type="messaging:fileInfo.type" maxOccurs="unbounded"/>
    </sequence>
  </complexType>
  <element name="messagingCapabilities">
    <annotation>
      <documentation>Corresponding state variable:
A_ARG_TYPE_MessagingCapabilities</documentation>
    </annotation>
    <complexType>
      <sequence>
        <element name="supportedMessageClasses">
          <complexType>
            <sequence maxOccurs="unbounded">
              <element name="messageClass" type="messaging:messageClass.type"/>
            </sequence>
          </complexType>
        </element>
        <element name="supportedSessionClasses">
          <complexType>
            <sequence minOccurs="0" maxOccurs="unbounded">
              <element name="sessionClass" type="messaging:sessionClass.type"/>
            </sequence>
          </complexType>
        </element>
        <element name="supportedMessageFolders">
          <complexType>
            <sequence maxOccurs="unbounded">
              <element name="messageFolder" type="messaging:messageFolder.type"/>
            </sequence>
          </complexType>
        </element>
      </sequence>
    </complexType>
  </element>
  <element name="newMessages">
    <annotation>
      <documentation>Corresponding state variable: NewMessages</documentation>
    </annotation>
    <complexType>
      <sequence minOccurs="0" maxOccurs="unbounded">
        <annotation>
          <documentation>No new messages is a possible actual value</documentation>
        </annotation>
        <element name="newMessage" type="messaging:newMessage.type"/>
      </sequence>
    </complexType>
  </element>
  <element name="sessionUpdates">
    <annotation>
      <documentation>Corresponding state variable: SessionUpdates</documentation>
    </annotation>
    <complexType>
      <sequence minOccurs="0" maxOccurs="unbounded">
        <element name="sessionUpdate">
          <complexType>
            <sequence>
```

```
              <element name="sessionID" type="messaging:sessionID.type"/>
              <element name="sessionEvent" type="string" maxOccurs="unbounded"/>
              <element name="sessionStatus" type="messaging:sessionStatus.type"/>
            </sequence>
          </complexType>
        </element>
      </sequence>
    </complexType>
  </element>
  <element name="message" type="messaging:message.type">
    <annotation>
      <documentation>Corresponding state variable: A_ARG_TYPE_Message</documentation>
    </annotation>
  </element>
  <element name="messageList">
    <annotation>
      <documentation>Corresponding state variable:
A_ARG_TYPE_MessageList</documentation>
    </annotation>
    <complexType>
      <sequence minOccurs="0" maxOccurs="unbounded">
        <element name="message" type="messaging:message.type"/>
      </sequence>
    </complexType>
  </element>
  <element name="sessionInfo" type="messaging:session.type">
    <annotation>
      <documentation>Corresponding state variable:
A_ARG_TYPE_SessionInfo</documentation>
    </annotation>
  </element>
  <element name="sessionsList">
    <annotation>
      <documentation>Corresponding state variable:
A_ARG_TYPE_SessionsList</documentation>
    </annotation>
    <complexType>
      <sequence minOccurs="0" maxOccurs="unbounded">
        <element name="sessionInfo" type="messaging:session.type"/>
      </sequence>
    </complexType>
  </element>
  <element name="recipientsList" type="messaging:recipientsList.type">
    <annotation>
      <documentation>Corresponding state variable:
A_ARG_TYPE_RecipientsList</documentation>
    </annotation>
  </element>
  <element name="fileInfoList" type="messaging:fileInfoList.type">
    <annotation>
      <documentation>Corresponding state variable:
A_ARG_TYPE_FileInfoList</documentation>
    </annotation>
  </element>
</schema>
```