ETSI TR 102 397-5-2 V1.1.1 (2005-12)

Technical Report

**Open Service Access (OSA);
Mapping of Parlay X Web Services to Parlay/OSA APIs;
Part 5: Multimedia Messaging Mapping;
Sub-part 2: Mapping to Multi-Media Messaging**

Reference

DTR/TISPAN-01021-05-02-OSA

Keywords

API, OSA, service

Contents

Intellectual Property Rights

Foreword

This Technical Report (TR) has been produced by ETSI Technical Committee Telecommunications and Internet converged Services and Protocols for Advanced Networking (TISPAN).

The present document is part 5, sub-part 2, of a multi-part deliverable providing an informative mapping of Parlay X Web Services to the Parlay Open Service Access (OSA) APIs and, where applicable, to IMS, as identified below:

Part 1: "Common Mapping";

Part 2: "Third Party Call Mapping";

Part 3: "Call Notification Mapping";

Part 4: "Short Messaging Mapping";

Part 5: "Multimedia Messaging Mapping";

Sub-part 1: "Mapping to User Interaction";

Sub-part 2: "Mapping to Multi-Media Messaging";

Part 6: "Payment Mapping";

Part 7: "Account Management Mapping";

Part 8: "Terminal Status Mapping";

Part 9: "Terminal Location Mapping";

Part 10: "Call Handling Mapping";

Part 11: "Audio Call Mapping";

Part 12: "Multimedia Conference Mapping";

Part 13: "Address list Management Mapping";

Part 14: "Presence Mapping".

The present document has been defined jointly between ETSI, The Parlay Group (http://www.parlay.org) and the 3GPP.

1 Scope

The Parlay X Web Services provide powerful yet simple, highly abstracted, imaginative, telecommunications functions that application developers and the IT community can both quickly comprehend and use to generate new, innovative applications.

The Open Service Access (OSA) specifications define an architecture that enables application developers to make use of network functionality through an open standardised interface, i.e. the Parlay/OSA APIs.

The present document is part 5, sub-part 2, of an informative mapping of Parlay X Web Services to Parlay/OSA APIs.

The present document specifies the mapping of the Parlay X Multimedia Messaging Web Service to the Parlay/OSA Multi-Media Messaging Service Capability Feature (SCF).

2 References

For the purposes of this Technical Report (TR) the following references apply:

[1] ETSI TR 102 397-1: "Open Service Access (OSA); Mapping of Parlay X Web Services to Parlay/OSA APIs; Part 1: Common Mapping".

[2] IETF RFC 2822: "Internet Message Format".

NOTE: Available at http://www.ietf.org/rfc/rfc2822.txt

[3] W3C Note (11 December 2000): "SOAP Messages with Attachments".

NOTE: Available at http://www.w3.org/TR/SOAP-attachments

[4] IETF RFC 2045: "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies".

NOTE: Available at http://www.ietf.org/rfc/rfc2045.txt

[5] IETF RFC 2046: "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types".

NOTE: Available at http://www.ietf.org/rfc/rfc2046.txt

[6] IETF RFC 2557: "MIME Encapsulation of Aggregate Documents, such as HTML (MHTML)".

NOTE: Available at http://www.ietf.org/rfc/rfc2557.txt

[7] ETSI TR 121 905: "Universal Mobile Telecommunications System (UMTS); Vocabulary for 3GPP Specifications (3GPP TR 21.905)".

[8] W3C Recommendation (2 May 2001): "XML Schema Part 2: Datatypes".

[9] ETSI TS 123 140: "Digital cellular telecommunications system (Phase 2+); Universal Mobile Telecommunications System (UMTS); Multimedia Messaging Service (MMS); Functional description; Stage 2 (3GPP TS 23.140)".

3 Definitions and abbreviations

3.1 Definitions

For the purposes of the present document, the terms and definitions given in TR 102 397-1 [1] and the following apply:

Shortcode: Short telephone number, usually 4 to 6 digits long. This is represented by the 'tel:' URI defined in TR 102 397-1 [1].

Whitespace: See definition for CFWS as defined in RFC 2822 [2].

3.2 Abbreviations

For the purposes of the present document, the abbreviations given in TR 102 397-1 [1] and the following apply:

MMS	Multimedia Messaging Service

4 Mapping Description

The Multimedia Messaging capability can be implemented with the Parlay/OSA Multi-Media Messaging SCF.

It is applicable to ETSI OSA 3.x, Parlay/OSA 5.x and 3GPP Release 6.x.

The following tables list the versions of the applicable mapping source documents, for both the ETSI/Parlay and the 3GPP specification sets.

Table 1: ETSI/Parlay Mapping Source Documents

Parlay X 2.1 Web Services	Parlay 3.5	Parlay 4.3	Parlay 5.1
Draft ES 202 391-5 V1.1.3	n/a	n/a	Draft ES 203 915-15 V0.0.4

Table 2: 3GPP Mapping Source Documents

Release 6 Parlay X Web Services	Release 4	Release 5	Release 6
TS 29.199-5 V6.3.0	n/a	n/a	TS 29.198-15 V6.2.1

5 Sequence Diagrams

5.1 Send Multimedia Message to One or More Addresses (Messaging Paradigm)

This describes where an application sends a multimedia message to one or more addresses.

1. The application requests the sending of a multimedia message to multiple addresses using the **sendMessage** operation. If the contents of the **sendMessageRequest** message are invalid for any reason, the appropriate service or policy exception is thrown. Otherwise processing continues as described below.

2. The web service creates a Multi-Media Messaging interface object for this application request (single-shot, page mode); no source or destination address information is provided in the method invocation. If the method invocation fails for any reason, the appropriate service or policy exception is thrown. Otherwise processing continues as described below.

3. A **sendMessageResponse** message is returned to the application containing a unique identifier for this message delivery request.

4. The web service invokes the `sendMessageReq` method on the Multi-Media Messaging interface object to send the message to each individual destination address.

5. The application can invoke the **getMessageDeliveryStatus** operation at any time after it receives the **sendMessageResponse** message and use the unique identifier it received in this message to obtain the current delivery status for each individual destination address. At this stage, the status returned for each address is either **MessageWaiting** or, in the event of an error, **DeliveryImpossible**.

6. The web service processes an invocation of the `sendMessageRes` method indicating that the message has been successfully sent to the destination address(es). However it does not indicate that the message was delivered or read.

7. The application can invoke the **getMessageDeliveryStatus** operation. At this stage, the status returned for each individual destination address is one of the following:

   - **DeliveryImpossible,** in the event an error occurred.

   - **DeliveryUncertain**, otherwise.

8. The web service processes one or more invocations of the `messageStatusReport` method, one for each destination address associated with the message, which contains the terminal delivery related status.

9. *[RESERVED FOR FUTURE USE] If the **receiptRequest** part of the associated, original **sendMessageRequest** message was present, and this capability is supported by the web service, then the web service invokes the **notifyMessageDeliveryReceipt** operation to notify the application of the final status of the message delivery to an individual destination address.*

10. The application can invoke the **getMessageDeliveryStatus** operation. At this stage, the status returned for an individual destination address depends on whether a `messageStatusReport` method has been invoked for that address. If the method has not been invoked, the delivery status is as described in step 7. Otherwise this method has been invoked and the delivery status is one of the following:

    - **Delivered**, if deliveryReportType parameter value = P_MESSAGE_REPORT_DELIVERED.

    - **DeliveryImpossible**, if deliveryReportType parameter value = P_MESSAGE_REPORT_ NOT_DELIVERABLE.

    - **DeliveryUncertain**, if deliveryReportType parameter value = P_MESSAGE_REPORT_ DELIVERY_UNDEFINED.

11. If the web service has not yet received all the requested terminal delivery related status reports, it may optionally invoke the `queryStatusReq` method to poll the network for this information.

12. The web service processes an invocation of the `queryStatusRes` method containing terminal delivery related status for all destination addresses associated with the message.

13. *[RESERVED FOR FUTURE USE] If the **receiptRequest** part of the associated, original **sendMessageRequest** message was present, and this capability is supported by the web service, then the web service invokes the **notifyMessageDeliveryReceipt** operation to notify the application of the final status of the message delivery to an individual destination address. (However if the delivery status is unchanged from the status previously reported to the application, then the web service does not need to invoke this operation.)*

14. The application can invoke the **getMessageDeliveryStatus** operation. At this stage, the status returned for all associated destination addresses reflects the results provided by the `queryStatusRes` method (step 12), i.e.:

    - **Delivered**, if deliveryReportType parameter value = P_MESSAGE_REPORT_DELIVERED.

    - **DeliveryImpossible**, if deliveryReportType parameter value = P_MESSAGE_REPORT_ NOT_DELIVERABLE.

    - **DeliveryUncertain**, if deliveryReportType parameter value = P_MESSAGE_REPORT_ DELIVERY_UNDEFINED.

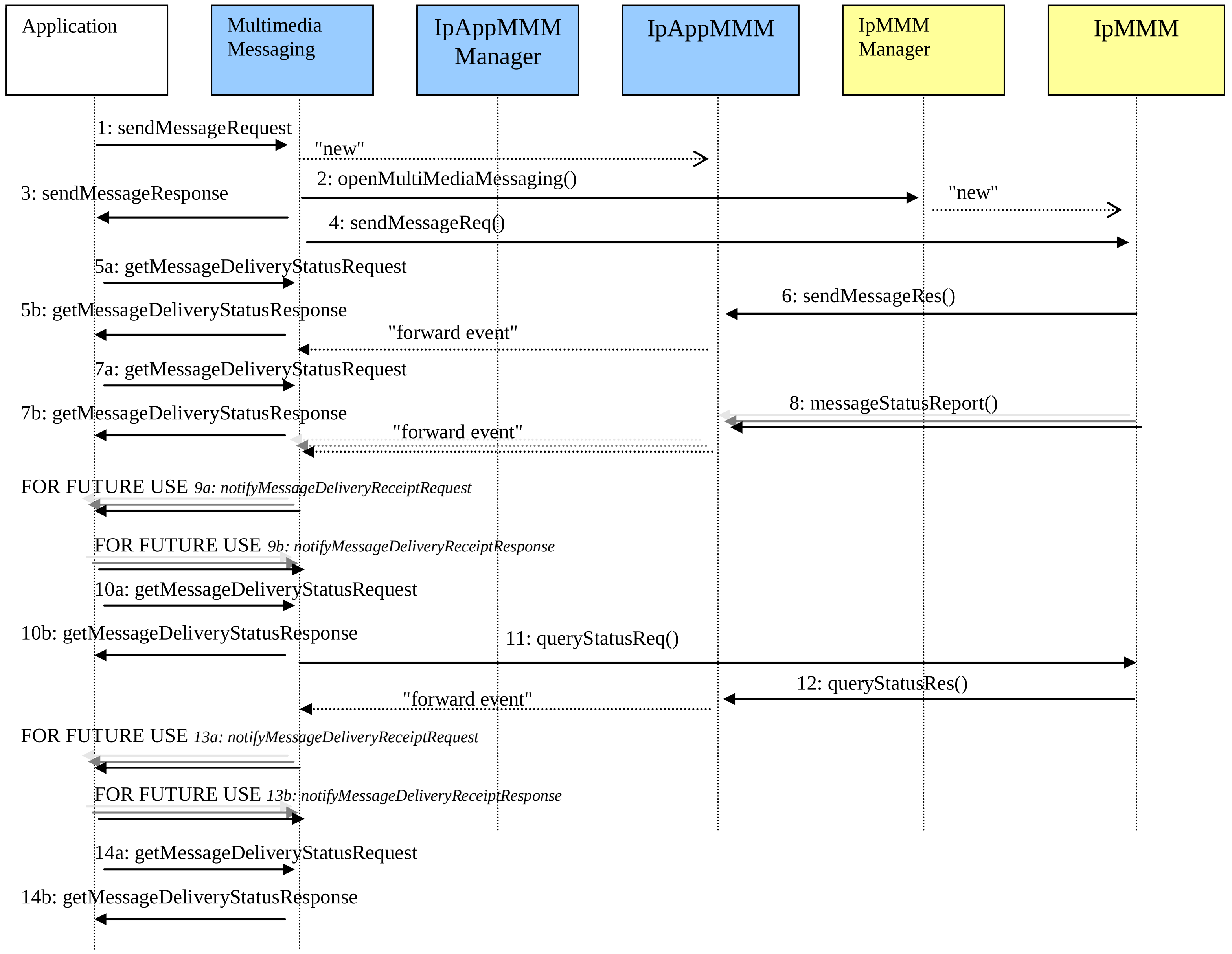
Application
Multimedia Messaging
IpAppMMM Manager
IpAppMMM
IpMMM Manager
IpMMM
1: sendMessageRequest
"new"
2: openMultiMediaMessaging()
3: sendMessageResponse
"new"
4: sendMessageReq()
5a: getMessageDeliveryStatusRequest
6: sendMessageRes()
5b: getMessageDeliveryStatusResponse
"forward event"
7a: getMessageDeliveryStatusRequest
8: messageStatusReport()
7b: getMessageDeliveryStatusResponse
"forward event"
FOR FUTURE USE 9a: notifyMessageDeliveryReceiptRequest
FOR FUTURE USE 9b: notifyMessageDeliveryReceiptResponse
10a: getMessageDeliveryStatusRequest
10b: getMessageDeliveryStatusResponse
11: queryStatusReq()
12: queryStatusRes()
"forward event"
FOR FUTURE USE 13a: notifyMessageDeliveryReceiptRequest
FOR FUTURE USE 13b: notifyMessageDeliveryReceiptResponse
14a: getMessageDeliveryStatusRequest
14b: getMessageDeliveryStatusResponse

Figure 1

5.2 Notification of Multimedia Message Reception and Retrieval (Messaging Paradigm)

1. The application registers for the reception of multimedia messages by invoking **startMessageNotification**. The request includes event criteria consisting of a value for the multimedia message destination address (the **messageServiceActivationNumber** part) and an optional text string for matching against the first word of the subject of the multimedia message or the first word in the text part of the multimedia message (the **criteria** part); also a URI for a Web Service implementing the **MessageNotification** interface on the client application side, and a correlation value for identifying this event registration request.

   - Note that the application may also register offline for the reception of multimedia messages: i.e. without using the Parlay X interface and the **startMessageNotification** operation. The registration request should at a minimum specify the message destination address. The request may also specify a URI for a Web Service implementing the **MessageNotification** interface on the client application side and/or the optional text string criteria. The registration request is assigned a unique registration identifier.

2. A check is made within the web service to see if a notification for the given multimedia message destination address is active. If no notification is active, then the Multimedia Messaging web service requests that a notification be created by the MMM SCS; note that the optional text string criteria (for matching against the first word in the message subject or body) is not sent to the MMM SCS. Otherwise a notification is already active and the request is not made.

3. The MMM SCS sends a `reportNotification` containing a set of one (or more) multimedia message(s) and related message information, where the destination address of each message is the same: i.e. equivalent to the value specified in the event criteria (steps 1 and 2).

4. For each multimedia message, the web service verifies the first word of the message body matches the value of the optional text string criteria associated with this destination address. If a message is verified, then the web service stores the message and notifies the application by invoking the **notifyMessageReception** operation on the corresponding, previously provisioned, application Web Service (i.e. as defined in the **reference** part of the **startMessageNotificationRequest** message). Note that if the multimedia message is pure ASCII text, then the whole message is delivered to the application Web Service. Otherwise, if a message cannot be verified, the web service discards it.

5. The application may invoke the **getReceivedMessages** operation to request a list of references to received multimedia messages matching a registration identifier associated with off-line provisioned notification criteria. The web service returns the list of any such multimedia message references. Note that for each multimedia message that is pure ASCII text, the web service delivers the whole message to the application and then deletes it.

6. The application retrieves the text portion of a multimedia message associated with one of the message references, and a list of URI file references for any message attachments, by invoking the **getMessageURIs** operation.

7. The application retrieves the whole multimedia message associated with one of the message references, by invoking the **getMessage** operation.

8. The application terminates an existing registration for the reception of multimedia messages by invoking the **stopMessageNotification** operation. The request includes the same correlation value previously specified in the earlier **startMessageNotification** operation (step 1).

   - Note that the application may also deregister offline for the reception of multimedia messages: i.e. without using the Parlay X interface and the **stopMessageNotification** operation. The deregistration request would specify the registration identifier associated with the original, offline registration operation (step 1).

9. A check is made within the web service to see if the registration identifier (correlation value) represents the last active notification for the corresponding destination address. If it is the last, then the web service requests that the notification be destroyed by the MMM SCS. Otherwise at least one other notification (i.e. associated with a different text string criteria value) remains active for this destination address and the request is not made.

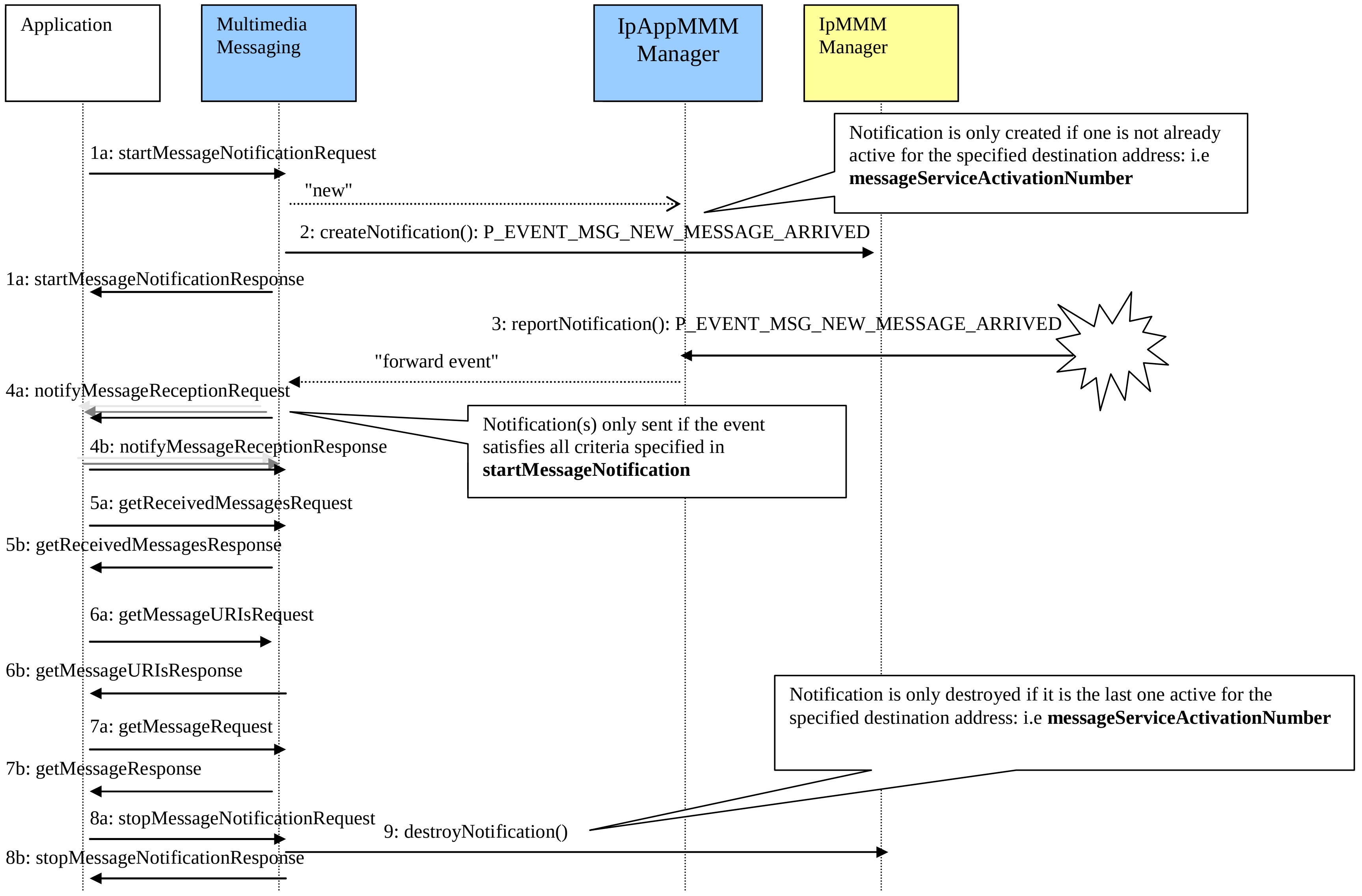
Application
Multimedia Messaging
IpAppMMM Manager
IpMMM Manager
1a: startMessageNotificationRequest
"new"
Notification is only created if one is not already active for the specified destination address: i.e **messageServiceActivationNumber**
2: createNotification(): P_EVENT_MSG_NEW_MESSAGE_ARRIVED
1a: startMessageNotificationResponse
3: reportNotification(): P_EVENT_MSG_NEW_MESSAGE_ARRIVED
"forward event"
4a: notifyMessageReceptionRequest
Notification(s) only sent if the event satisfies all criteria specified in **startMessageNotification**
4b: notifyMessageReceptionResponse
5a: getReceivedMessagesRequest
5b: getReceivedMessagesResponse
6a: getMessageURIsRequest
6b: getMessageURIsResponse
7a: getMessageRequest
7b: getMessageResponse
Notification is only destroyed if it is the last one active for the specified destination address: i.e **messageServiceActivationNumber**
8a: stopMessageNotificationRequest
9: destroyNotification()
8b: stopMessageNotificationResponse

Figure 2

5.3 Send Multimedia Message to One or More Addresses (Mailbox Paradigm)

This describes where an application sends a multimedia message to one or more addresses.

1. The application requests the sending of a multimedia message to multiple addresses using the **sendMessage** operation. If the contents of the **sendMessageRequest** message are invalid for any reason, the appropriate service or policy exception is thrown. Otherwise processing continues as described below.

2. If a mailbox for the requesting application is not already open, then the web service opens a Mailbox interface object. If the method invocation fails for any reason, the appropriate service or policy exception is thrown. Otherwise processing continues as described below.

3. A **sendMessageResponse** message is returned to the application containing a unique identifier for this message delivery request.

4. The web service invokes the `putMessageReq` method one or more times on the Mailbox interface object to place the message in an 'outbox' to be sent to each individual destination address. Note that, by invoking the method separately for each individual destination address, the web service receives a `messageId` for each destination that can be subsequently used to poll for delivery status on a per destination basis, e.g. in step 8.

5. The application can invoke the **getMessageDeliveryStatus** operation at any time after it receives the **sendMessageResponse** message and use the unique identifier it received in this message to obtain the current delivery status for each individual destination address. At this stage, the status returned for each address is either **MessageWaiting** or, in the event of an error, **DeliveryImpossible**.

6. The web service processes invocations of the `putMessageRes` method indicating that the message has been successfully sent to the destination address(es). However it does not indicate that the message was delivered or read.

7. The application can invoke the **getMessageDeliveryStatus** operation. At this stage, the status returned for each individual destination address is one of the following:

   - **DeliveryImpossible,** in the event an error occurred.

   - **DeliveryUncertain**, otherwise.

8. The web service invokes the `getMessageInfoPropertiesReq` method one or more times on the Mailbox interface object, one for each destination address associated with the message, to poll for message delivery status.

9. The web service processes invocations of the `getMessageInfoPropertiesRes` method containing message delivery status.

10. *[RESERVED FOR FUTURE USE] If the **receiptRequest** part of the associated, original **sendMessageRequest** message was present, and this capability is supported by the web service, then the web service invokes the **notifyMessageDeliveryReceipt** operation to notify the application of the final status of the message delivery to an individual destination address.*

11. The application can invoke the **getMessageDeliveryStatus** operation. At this stage, the status returned for an individual destination address depends on whether a `getMessageInfoPropertiesRes` method has been invoked for that address. If the method has not been invoked, the delivery status is as described in step 7. Otherwise this method has been invoked and the delivery status is one of the following:

    - **Delivered**, if MessageStatus parameter value = P_MMM_SENT_MSG_STATUS_DELIVERED, P_MMM_SENT_MSG_STATUS_READ or P_MMM_SENT_MSG_STATUS_DELETED_UNREAD.

    - **DeliveryImpossible**, if MessageStatus parameter value = P_MMM_SENT_MSG_STATUS_NOT_DELIVERABLE or P_MMM_SENT_MSG_STATUS_EXPIRED.

    - **DeliveryUncertain**, if MessageStatus parameter value = P_MMM_SENT_MSG_STATUS_SENT.

12. If the web service has not yet received a final message delivery status for all the destination addresses, it may optionally (re-)invoke the `getMessageInfoPropertiesReq` method one or more times on the Mailbox interface object to poll for message delivery status.

13. The web service processes invocations of the `getMessageInfoPropertiesRes` method containing message delivery status.

14. *[RESERVED FOR FUTURE USE] If the **receiptRequest** part of the associated, original **sendMessageRequest** message was present, and this capability is supported by the web service, then the web service invokes the **notifyMessageDeliveryReceipt** operation to notify the application of the final status of the message delivery to an individual destination address. (However if the delivery status is unchanged from the status previously reported to the application, then the web service does not need to invoke this operation.)*

15. The application can invoke the **getMessageDeliveryStatus** operation. At this stage, the status returned for all associated destination addresses reflects the results provided by the `getMessageInfoPropertiesRes` methods (steps 9 and 13), i.e.:

    - **Delivered**, if MessageStatus parameter value = P_MMM_SENT_MSG_STATUS_DELIVERED, P_MMM_SENT_MSG_STATUS_READ or P_MMM_SENT_MSG_STATUS_DELETED_UNREAD.

    - **DeliveryImpossible**, if MessageStatus parameter value = P_MMM_SENT_MSG_STATUS_NOT_DELIVERABLE or P_MMM_SENT_MSG_STATUS_EXPIRED.

    - **DeliveryUncertain**, if MessageStatus parameter value = P_MMM_SENT_MSG_STATUS_SENT.

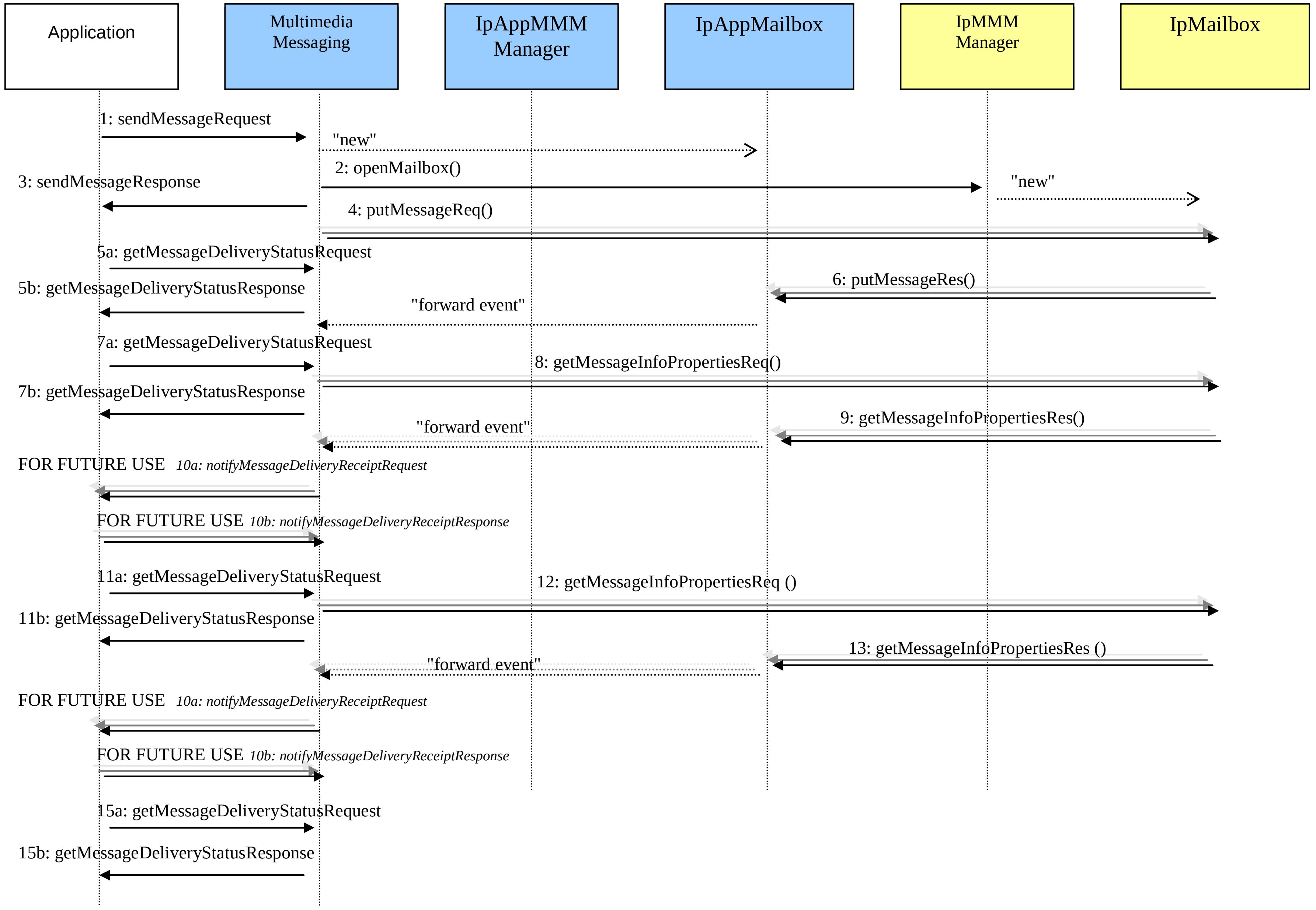
Application
Multimedia Messaging
IpAppMMM Manager
IpAppMailbox
IpMMM Manager
IpMailbox
1: sendMessageRequest
"new"
2: openMailbox()
3: sendMessageResponse
"new"
4: putMessageReq()
5a: getMessageDeliveryStatusRequest
5b: getMessageDeliveryStatusResponse
6: putMessageRes()
"forward event"
7a: getMessageDeliveryStatusRequest
8: getMessageInfoPropertiesReq()
7b: getMessageDeliveryStatusResponse
"forward event"
9: getMessageInfoPropertiesRes()
FOR FUTURE USE 10a: notifyMessageDeliveryReceiptRequest
FOR FUTURE USE 10b: notifyMessageDeliveryReceiptResponse
11a: getMessageDeliveryStatusRequest
12: getMessageInfoPropertiesReq ()
11b: getMessageDeliveryStatusResponse
13: getMessageInfoPropertiesRes ()
"forward event"
FOR FUTURE USE 10a: notifyMessageDeliveryReceiptRequest
FOR FUTURE USE 10b: notifyMessageDeliveryReceiptResponse
15a: getMessageDeliveryStatusRequest
15b: getMessageDeliveryStatusResponse

Figure 3

5.4 Notification of Multimedia Message Reception and Retrieval (Mailbox Paradigm)

1. The application registers for the reception of multimedia messages by invoking **startMessageNotification**. The request includes event criteria consisting of a value for the multimedia message destination address (the **messageServiceActivationNumber** part) and an optional text string for matching against the first word of the subject of the multimedia message or the first word in the text part of the multimedia message (the **criteria** part); also a URI for a Web Service implementing the **MessageNotification** interface on the client application side, and a correlation value for identifying this event registration request.

   - Note that the application may also register offline for the reception of multimedia messages: i.e. without using the Parlay X interface and the **startMessageNotification** operation. The registration request should at a minimum specify the message destination address. The request may also specify a URI for a Web Service implementing the **MessageNotification** interface on the client application side and/or the optional text string criteria. The registration request is assigned a unique registration identifier.

2. A check is made within the web service to see if a notification for the given multimedia message destination address is active. If no notification is active, then the Multimedia Messaging web service requests that a notification be created by the MMM SCS; note that the optional text string criteria (for matching against the first word in the message subject or body) is not sent to the MMM SCS. Otherwise a notification is already active and the request is not made.

3. The MMM SCS sends a `reportNotification` containing a set of one (or more) received message notification(s) and related message information, where the mailbox identifier of each message is the same: i.e. equivalent to the value specified in the event criteria (steps 1 and 2).

4. If one is not already open, the web service opens a Mailbox interface object associated with the mailbox identifier reported in the event notification.

5. The MMM SCS performs the following processing for each received message notification:

   - If the Subject field is present (i.e. the value of the `MessageDescription.Subject` element is non-null) but the first word does not match the value of any of the optional text string criteria associated with this destination address, then the web service discards the notification. The web service may invoke either the `deleteMessageReq` or `moveMessageReq` method on the Mailbox interface object to clean-up the mailbox and folder.

   - If the Subject field is not present (i.e. the value of the `MessageDescription.Subject` element is null), then the web service retrieves the message content in order to perform the criteria matching operation. The web service invokes the `getMessageContentReq` method.

   - If the Subject field is present (i.e. the value of the `MessageDescription.Subject` element is non-null) and the first word matches the value of an optional text string criteria associated with this destination address, then processing is as follows:

     - If the message is pure text [i.e. the value(s) of the `ExtendedHeaderInformation.MimeContent` element(s) denote text content], then the web service retrieves the message content by invoking the `getMessageContentReq` method.

     - If the message is not pure text, then the web service retrieves all parts of the multimedia message by invoking the `getMessageBodyPartsReq` method.

6. The web service notifies the application of a valid received message by invoking the **notifyMessageReception** operation on the corresponding, previously provisioned, application Web Service. Note that if the multimedia message is pure ASCII text, then the whole message is delivered to the application Web Service. This operation is invoked under the following circumstances:

   - The first word of the subject field (step 5) matches the optional text string criteria associated with the application Web Service.

   - The subject field (step 5) was absent, the message is pure text, and the first word of the message body matches the optional text string criteria associated with the application Web Service.

7. The web service performs one of the following method invocations on the Mailbox interface object:

   - `deleteMessageReq` or `moveMessageReq` to clean-up the mailbox and folder, if the subject field (step 5) associated with the message was absent and the first word of the message body does not match any optional text string criteria associated with the application Web Service;

   - `setMessageInfoPropertiesReq`, if it stores a retrieved message, in order to change the value of the `MessageStatus` element from `P_MMM_RECEIVED_MSG_STATUS_UNREAD` to `P_MMM_RECEIVED_MSG_STATUS_READ;`

   - `getMessageBodyPartsReq`, to retrieves all parts of a multimedia message, if the subject field (step 5) associated with the multimedia message was absent and the first word of the message body matches optional text string criteria associated with the application Web Service.

8. The web service processes invocations of the `getMessageBodyPartsRes` method. For each invocation, the web service stores the message and notifies the application of a valid received multimedia message by invoking the **notifyMessageReception** operation on the corresponding, previously provisioned, application Web Service.

9. The web service invokes `setMessageInfoPropertiesReq`, for each stored multimedia message, in order to change the value of the `MessageStatus` element from `P_MMM_RECEIVED_MSG_STATUS_UNREAD` to `P_MMM_RECEIVED_MSG_STATUS_READ`.

10. The application may invoke the **getReceivedMessages** operation to request a list of references to received multimedia messages matching a registration identifier associated with off-line provisioned notification criteria. The web service returns the list of any such multimedia message references. Note that for each multimedia message that is pure ASCII text, the web service delivers the whole message to the application and then deletes it.

11. The application retrieves the text portion of a multimedia message associated with one of the message references, and a list of URI file references for any message attachments, by invoking the **getMessageURIs** operation.

12. The application retrieves the whole multimedia message associated with one of the message references, by invoking the **getMessage** operation.

13. The application terminates an existing registration for the reception of multimedia messages by invoking the **stopMessageNotification** operation. The request includes the same correlation value previously specified in the earlier **startMessageNotification** operation (step 1).

    - Note that the application may also deregister offline for the reception of multimedia messages: i.e. without using the Parlay X interface and the **stopMessageNotification** operation. The deregistration request would specify the registration identifier associated with the original, offline registration operation (step 1).

14. A check is made within the web service to see if the registration identifier (correlation value) represents the last active notification for the corresponding destination address. If it is the last, then the web service requests that the notification be destroyed by the MMM SCS. Otherwise at least one other notification (i.e. associated with a different text string criteria value) remains active for this destination address and the request is not made.

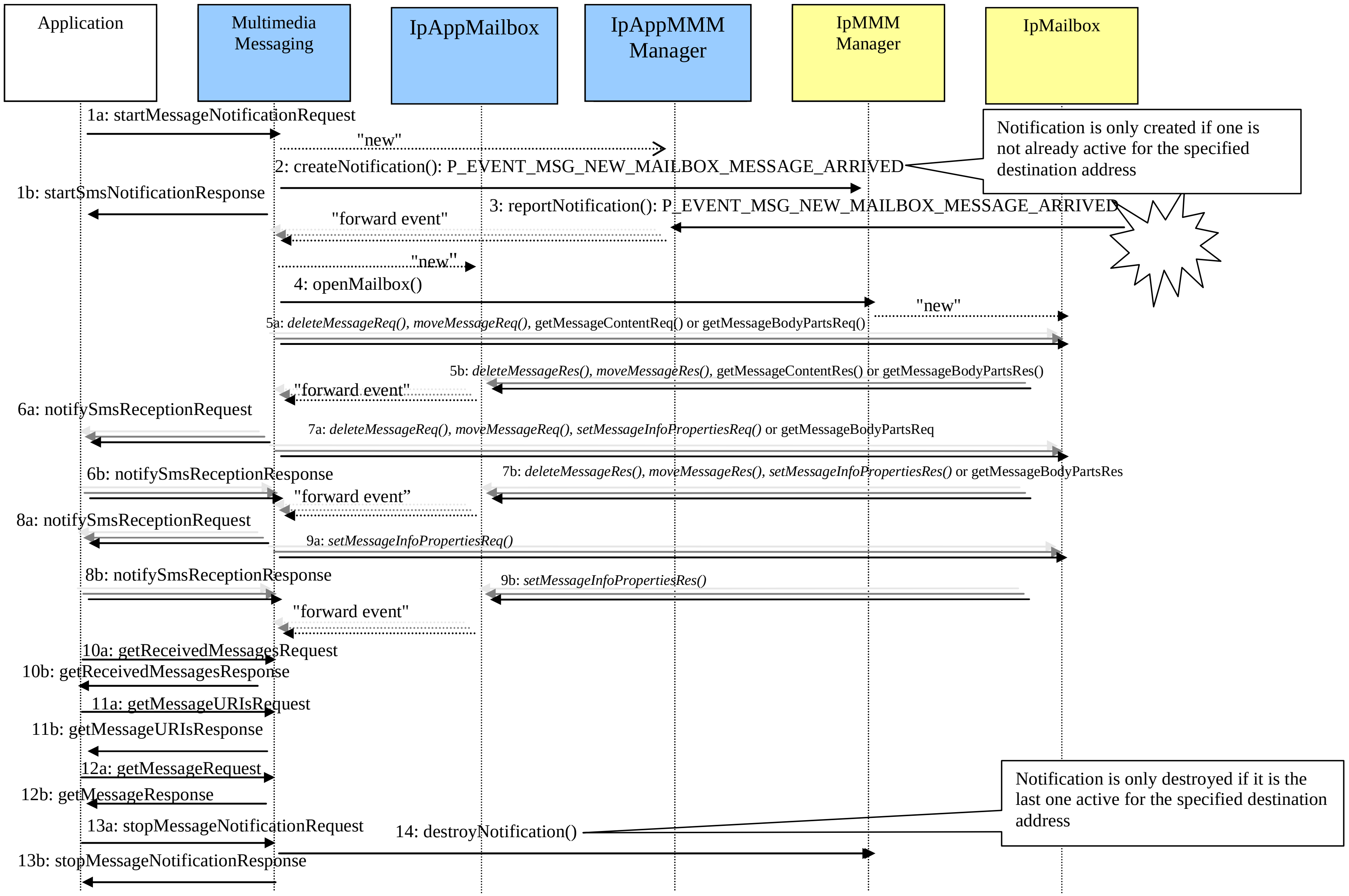
Application
Multimedia Messaging
IpAppMailbox
IpAppMMM Manager
IpMMM Manager
IpMailbox
Notification is only created if one is not already active for the specified destination address
1a: startMessageNotificationRequest
"new"
2: createNotification(): P_EVENT_MSG_NEW_MAILBOX_MESSAGE_ARRIVED
1b: startSmsNotificationResponse
3: reportNotification(): P_EVENT_MSG_NEW_MAILBOX_MESSAGE_ARRIVED
"forward event"
"new"
4: openMailbox()
"new"
5a: deleteMessageReq(), moveMessageReq(), getMessageContentReq() or getMessageBodyPartsReq()
5b: deleteMessageRes(), moveMessageRes(), getMessageContentRes() or getMessageBodyPartsRes()
"forward event"
6a: notifySmsReceptionRequest
7a: deleteMessageReq(), moveMessageReq(), setMessageInfoPropertiesReq() or getMessageBodyPartsReq
6b: notifySmsReceptionResponse
7b: deleteMessageRes(), moveMessageRes(), setMessageInfoPropertiesRes() or getMessageBodyPartsRes
"forward event"
8a: notifySmsReceptionRequest
9a: setMessageInfoPropertiesReq()
8b: notifySmsReceptionResponse
9b: setMessageInfoPropertiesRes()
"forward event"
10a: getReceivedMessagesRequest
10b: getReceivedMessagesResponse
11a: getMessageURIsRequest
11b: getMessageURIsResponse
12a: getMessageRequest
12b: getMessageResponse
13a: stopMessageNotificationRequest
14: destroyNotification()
Notification is only destroyed if it is the last one active for the specified destination address
13b: stopMessageNotificationResponse

Figure 4

6 Detailed Mapping Information

6.1 Operations (Messaging Paradigm)

6.1.1 sendMessage

The sequence diagram in clause 5.1 (figure 1) illustrates the flow for the **sendMessage** operation.

The **sendMessage** operation is synchronous from the Parlay X client's point of view. It is mapped to the following Parlay/OSA methods:

- `IpMMMManager.openMMM.`
- `IpMMM.sendMessageReq.`

6.1.1.1 Mapping to `IpMMMManager.openMMM`

The `IpMMMManager.openMMM` method is invoked with the following parameters:

Name	Type	Comment
defaultDestination AddressList	TpTerminating AddressList	Not mapped. [Optional parameter]
defaultSource Address	TpAddress	Not mapped. [Optional parameter]
appMMM	IpAppMMMRef	Reference to callback (internal)

The result from `IpMMMManager.openMMM` is of type `TpMMMIdentifier` and identifies the MMM interface object upon which future methods are invoked: e.g. `IpMMM.sendMessageReq`. It is also correlated with the value of the **requestIdentifier** part returned to the application in the **sendMessageResponse** message

Parlay exceptions thrown by `IpMMMManager.openMMM` are mapped to Parlay X exceptions as defined in clause 6.2.1.

6.1.1.2 Mapping to IpMMM.sendMessageReq

The `IpMMM.sendMessageReq` method is invoked with the following parameters:

Name	Type	Comment
sessionID	TpSessionID	Not mapped. [The value provided in the result from `IpMMMManager.openMMM`]
sourceAddress	TpAddress	The address used to represent the sender of the message. For alphanumeric message addresses - i.e. the optional **senderAddress** part of **sendMessageRequest** - the address plan `P_ADDRESS_PLAN_UNDEFINED` is used.
destination AddressList	TpTerminating AddressList	Specifies the addresses to which the message should be sent. It is constructed based on the URIs provided in the **addresses** part of **sendMessageRequest**, mapped as described in TR 102 397-1 [1]. Only the `ToAddressList` element of `TpTerminatingAddressList` is populated.
deliveryType	TpMessage DeliveryType	Set to the `P_MMM_MMS` value.
message Treatment	TpMessage TreatmentSet	Consists of the following elements: • a `DeliveryReport` element with value set to a value of "9", which represents a logical "OR" (and request for notification) of ONLY the following delivery states: `P_MESSAGE_REPORT_DELIVERY_UNDEFINED`; `P_MESSAGE_REPORT_DELIVERED;` and `P_MESSAGE_REPORT_NOT_DELIVERABLE.` • a `BillingID` element constructed from the **code** element of the optional **charging** part (if present); • a `DeliveryTime` element set to a value of `P_MMM_SEND_IMMEDIATE`; • a `ValidityTime` element set to a vendor-specific value.
message	TpOctetSet	The actual message that needs to be sent: i.e. the Attachment(s) to the **sendMessageRequest** message. The message and the headers are transferred to the Messaging service. The message will be taken as is. No checking is done on the message. If the web service knows the messaging system and understands the format for sending attachments, it can do so as an alternative, or in addition, to populating the `message` and `additionalHeaders` parameters.
additionalHeaders	TpMessage HeaderFieldSet	Consists of multiple elements mapped as follows: • the optional **subject** part maps to the `Subject` field • the optional **priority** part maps to the `Priority` field, as follows: • **Default**, **Normal** -> `P_MMM_MESSAGE_PRIORITY_UNDEFINED` • **Low** -> `P_MMM_MESSAGE_PRIORITY_LOW` • **High** -> `P_MMM_MESSAGE_PRIORITY_HIGH` • the Attachment(s) containing the message content map(s) to multiple `MimeXxx` fields. For each Attachment, values must be assigned to at least the following header fields (see note): • `MimeContent` (e.g. 'image/gif'), which has the same semantics as the `"Content-Type:"` field that is defined in RFC 2045 [4] and RFC 2046 [5]. • EITHER `MimeID` (e.g. 'abcd'), which has the same semantics as the `"Content-ID:"` field that is defined in RFC 2045 [4], OR `MimeExtensionField.FieldName` = '`Content-Location:`' (e.g. 'filename.ext'), which has semantics defined in RFC 2557 [6] • `MimeEncoding` (e.g. 'binary'), which has the same semantics as the `"Content-Transfer-Encoding:"` field that is defined in RFC 2045 [4].
NOTE: The content of a multimedia message is sent as an Attachment(s). However, since the Attachment(s) are sent transparently to the underlying network, there are additional considerations. The Parlay X Multimedia Messaging Web Service should check that all minimum required header fields are available; the set of minimum required header fields may vary based on the underlying network or as mandated by policies in the gateway.		

The result from `IpMMM.sendMessageReq` is of type `TpAssignmentID` and is used internally to correlate the callbacks. Specifically it is used to correlate with future invocations of the `IpMMM.queryStatusReq` method and of `IpAppMMM` callback interface methods.

Parlay exceptions thrown by `IpMMM.sendMessageReq` are not mapped to Parlay X exceptions. Instead they are reported to the application in a **getMessageDeliveryStatusResponse** message, with the following part values:

- the **deliveryStatus.address** element has an address value contained in the `ToAddressList` element of the `terminatingAddressList` parameter of the `IpMMM.sendMessageReq` method, mapped as described in TR 102 397-1 [1];
- the deliveryStatus.deliveryStatus element has the value: DeliveryImpossible.

6.1.2 getMessageDeliveryStatus

The sequence diagram in clause 5.1 (figure 1) illustrates the flow for the **getMessageDeliveryStatus** operation.

The **getMessageDeliveryStatus** operation is synchronous from the Parlay X client's point of view. It is mapped to the following Parlay/OSA methods:

- `IpAppMMM.sendMessageRes.`
- `IpAppMMM.sendMessageErr.`
- `IpAppMMM.messageStatusReport.`
- `IpMMM.queryStatusReq.`
- `IpAppMMM.queryStatusRes.`
- `IpAppMMM.queryStatusErr.`

The delivery status provided to the Parlay X client will depend on the timing of the **getMessageDeliveryStatus** operation invocation. If a message status report is received from the network as a result of an earlier **sendMessage**-related operation, then the delivery status information provided in the `IpAppMMM.messageStatusReport` callback is mapped. If such a report has not been received, then the `IpMMM.queryStatusReq` method is invoked.

6.1.2.1 Mapping from `IpAppMMM.sendMessageRes`

The `IpAppMMM.sendMessageRes` method is invoked with the following parameters:

Name	Type	Comment
sessionID	TpSessionID	Not mapped. [The value provided in the result from `IpMMMManager.openMMM`]
assignmentID	TpAssignmentID	Not mapped. [The value provided in the result from `IpMMM.sendMessageReq`]

In the absence of more recent delivery status information (i.e. as provided in an `IpAppMMM.messageStatusReport` or an `IpAppMMM.queryStatusRes` method), this method results in the assignment of the **DeliveryUncertain** value to the **deliveryStatus** element of each **DeliveryInformation** parameter of the **deliveryStatus** part of a **getMessageDeliveryStatusResponse** message.

6.1.2.2 Mapping from `IpAppMMM.sendMessageErr`

The `IpAppMMM.sendMessageErr` method is invoked with the following parameters:

Name	Type	Comment
sessionID	TpSessionID	Not mapped. [The value provided in the result from `IpMMMManager.openMMM`]
assignmentID	TpAssignmentID	Not mapped. [The value provided in the result from `IpMMM.sendMessageReq`]
error	TpMessaging Error	Maps to the **DeliveryImpossible** value of the **deliveryStatus** element of each **DeliveryInformation** parameter of the **deliveryStatus** part of a **getMessageDeliveryStatusResponse** message.
errorDetails	TpString	Not mapped.

6.1.2.3 Mapping from `IpAppMMM.messageStatusReport`

The `IpAppMMM.messageStatusReport` method is invoked with the following parameters:

Name	Type	Comment
sessionID	TpSessionID	Not mapped. [The value provided in the result from `IpMMMManager.openMMM`]
assignmentID	TpAssignmentID	Not mapped. [The value provided in the result from `IpMMM.sendMessageReq`]
destinationAddress	TpAddress	Maps to the **address** element of one **DeliveryInformation** parameter of the **deliveryStatus** part of **getMessageDeliveryStatusResponse.**
deliveryReportType	TpMessageDeliveryReportType	Maps to the **deliveryStatus** element of one **DeliveryInformation** parameter of the **deliveryStatus** part of **getMessageDeliveryStatusResponse**, as follows: • `P_MESSAGE_REPORT_ DELIVERY_UNDEFINED` maps to **DeliveryUncertain**. • `P_MESSAGE_REPORT_DELIVERED` maps to **Delivered**. • `P_MESSAGE_REPORT_NOT_DELIVERABLE` maps to **DeliveryImpossible**.
deliveryReportInfo	TpString	Not mapped.

6.1.2.4 Mapping to `IpMMM.queryStatusReq`

The `IpMMM.queryStatusReq` method is invoked with the following parameters:

Name	Type	Comment
sessionID	TpSessionID	Not mapped. [The value provided in the result from `IpMMMManager.openMMM`]
assignmentID	TpAssignmentID	Not mapped. [The value provided in the result from `IpMMM.sendMessageReq`]

Parlay exceptions thrown by `IpMMM.queryStatusReq` are not mapped to Parlay X exceptions.

6.1.2.5 Mapping from `IpAppMMM.queryStatusRes`

The `IpAppMMM.queryStatusRes` method is invoked with the following parameters:

Name	Type	Comment
sessionID	TpSessionID	Not mapped. [The value provided in the result from `IpMMMManager.openMMM`]
assignmentID	TpAssignmentID	Not mapped. [The value provided in the result from `IpMMM.sendMessageReq`]
result	TpQueryStatusReportSet	This is a set of tuples where each tuple contains a `DestinationAddress` of the message, together with the `ReportedStatus` for that address. Each tuple maps to the **address** and **deliveryStatus** elements of one **DeliveryInformation** parameter of the **deliveryStatus** part of the **getMessageDeliveryStatusResponse** message. The mapping to the **deliveryStatus** element is as follows: • `P_MESSAGE_REPORT_ DELIVERY_UNDEFINED` maps to **DeliveryUncertain**. • `P_MESSAGE_REPORT_DELIVERED` maps to **Delivered**. • `P_MESSAGE_REPORT_NOT_DELIVERABLE` maps to **DeliveryImpossible**. In the event that the messaging system provides additional delivery states to those requested in the `messageTreatment` parameter (clause 6.1.1.2), the mapping to the **deliveryStatus** element is as follows: • `P_MESSAGE_REPORT_READ` and `P_MESSAGE_REPORT_DELETED_UNREAD` map to **Delivered**. • `P_MESSAGE_REPORT_EXPIRED` maps to **DeliveryImpossible**.

6.1.2.6 Mapping from `IpAppMMM.queryStatusErr`

This failed attempt to poll for delivery status does not change the current **deliveryStatus** value (i.e. **Delivered**, **DeliveryImpossible**, or **DeliveryUncertain**) for any of the destination addresses.

6.1.3 startMessageNotification

The sequence diagram in clause 5.2 (figure 2) illustrates the flow for the **startMessageNotification** operation, which is mapped to the Parlay/OSA method: `IpMMMManager.createNotification`, provided there is no existing notification already established for the destination address contained in the **messageServiceActivationNumber** part.

6.1.3.1 Mapping to `IpMMMManager.createNotification`

The `IpMMMManager.createNotification` is invoked with the following parameters:

Name	Type	Comment
appMMM Manager	IpAppMMM ManagerRef	Not mapped. Reference to callback (internal).
eventCriteria	TpMessaging EventCriteriaSet	Contains a single element specifying the event notification criteria, for the messaging event: `P_EVENT_MSG_NEW_MESSAGE_ARRIVED`. The criteria consist of 3 fields: • The `SourceAddress` is not mapped. It is set to be valid for all senders. • The `DestinationAddress` is constructed based on the URI provided in the **messageServiceActivationNumber** part of the **startMessageNotificationRequest** message, mapped as described in TR 102 397-1 [1]. • The `CreateMultiMediaMessagingSession` element is not mapped. It is set to a value of 'FALSE': i.e. the SCF will not create a MMM session object when a new message arrives.

The result from `IpMMMManager.createNotification` is of type `TpAssignmentID` and is used internally to correlate the callbacks. Specifically it is correlated with the value of the **reference** part received from the application in the **startMessageNotificationRequest** message and the **correlator** part returned to the application in the **notifyMessageReceptionRequest** message.

Note that the **reference** part and the optional **criteria** part of a **startMessageNotificationRequest** message are not mapped to `IpMMMManager.createNotification`. Instead the web service uses all the text string criteria values associated with a specific destination address to parse any event reported for that address by the `IpAppMMMManager.reportNotification` method. The web service determines whether the event is valid, i.e. there is a match with a text string criteria value. If valid, the web service stores the message and selects the previously provisioned application callback web service to receive the **notifyMessageReceptionRequest** message. If invalid, the web service discards the event notification.

Parlay exceptions thrown by `IpMMMManager.createNotification` are mapped to Parlay X exceptions as defined in clause 6.2.1.

6.1.4 notifyMessageReception

The **notifyMessageReception** operation is mapped from the following Parlay/OSA methods:

- `IpAppMMMManager.reportNotification`, as illustrated in the sequence diagram in clause 5.2 (figure 2).
- `IpAppMMM.messageReceived`, which contains a message received for a remote party within the context of the conversation or session currently active. The message may be, but is not necessarily in reply to a message sent by the application using the `IpMMM.sendMessageReq` method (clause 6.1.1.2). Note that the reference information for the application web service, upon which the **notifyMessageReception** operation is invoked, must be provisioned offline, since online provisioning using the **MessageNotificationManager** interface is only applicable for messages which are received outside the context of the conversation or session.

6.1.4.1 Mapping from `IpAppMMMManager.reportNotification`

The `IpAppMMMManager.reportNotification` method is invoked with the following parameters:

Name	Type	Comment
assignmentID	TpAssignmentID	Not mapped. [The value provide in the result from `IpMMMManager.createNotification`]
eventInfo	TpMessaging EventInfoSet	Contains a set of multimedia messages with the same destination address and an event type = `EventNewMessageArrived`. The mapping of each message (type `TpNewMessageArrivedInfo`) to the **message** part of a **notifyMessageReceptionRequest** messages is described in clause 6.1.4.2.

The result from `IpAppMMMManager.reportNotification` is of type `IpAppMultiMediaMessagingRef`. It is set to null.

6.1.4.2 Mapping from `TpNewMessageArrivedInfo`

The mapping from `TpNewMessageArrivedInfo` to the **message** part of a **notifyMessageReceptionRequest** message is as follows:

Name	Type	Comment
SourceAddress	TpAddress	Maps to the **senderAddress** element of the **message** part. The data type mapping from `TpAddress` to **xsd:anyURI** is described in TR 102 397-1 [1].
DestinationAddress Set	TpAddressSet	Consists of a single destination address element, which maps to the **messageServiceActivationNumber** element of the **message** part. The data type mapping from `TpAddress` to **xsd:anyURI** is described in TR 102 397-1 [1].
Message	TpOctetSet	If the `Message` field is pure ASCII text, then this field maps to the **message** element of the **message** part. Otherwise, the `Message` field, together with content from the `Headers` field, is stored by the Multimedia Messaging web service and a reference to it is returned to the application in the **messageIdentifier** element.
Headers	TpMessage HeaderFieldSet	Consists of multiple elements mapped to elements of the **message** part as follows: • the `Subject` field maps to the **subject** element; • the `Priority` field maps to the **priority** element, as follows: • `P_MMM_MESSAGE_PRIORITY_UNDEFINED` -> **Default** or **Normal**; • `P_MMM_MESSAGE_PRIORITY_LOW` -> **Low**; • `P_MMM_MESSAGE_PRIORITY_HIGH` -> **High**; • other fields, if present, are not directly mapped.
MultiMedia MessagingIdentifier	TpMultiMedia MessagingIdentifier	Not applicable. This parameter is null, reflecting the criteria value included in the `IpMMMManager.createNotification` invocation.

Note that this mapping occurs if there is at least one active notification established for the value of the `eventInfo.DestinationAddress(Set)` element, an associated application callback web service, and one of the following conditions is satisfied:

- There is only one active notification that was defined without the optional text string criteria value.
- There is one active notification that was defined with the optional text string criteria value and that value matches the first word in the value of the `eventInfo.Message` element.
  - Note that the 'first word' in the message is defined as the initial characters after discarding any leading Whitespace and ending with a Whitespace or end of message. The matching shall be case-insensitive.

6.1.4.3 Mapping from `IpAppMMM.messageReceived`

The `IpAppMMM.messageReceived` method is invoked with the following parameters:

Name	Type	Comment
sessionID	TpSessionID	Not mapped. [The value provide in the result from `IpMMMManager.openMMM` - clause 6.1.1.1]
message	TpOctetSet	If the `message` parameter is pure ASCII text, then this field maps to the **message** element of the **message** part. Otherwise, the `message` parameter, together with content from the `headers` parameter, is stored by the Multimedia Messaging web service and a reference to it is returned to the application in the **messageIdentifier** element.
headers	TpMessage HeaderFieldSet	Consists of multiple elements mapped to elements of the **message** part as follows: • The optional `Sender` field maps to the **senderAddress** element. [The data type mapping from `TpAddress` to **xsd:anyURI** is described in TR 102 397-1 [1]. • The optional `Subject` field maps to the **subject** element. • The optional `Priority` field maps to the **priority** element, as follows: • `P_MMM_MESSAGE_PRIORITY_UNDEFINED` -> **Default** or **Normal**. • `P_MMM_MESSAGE_PRIORITY_LOW` -> **Low**. • `P_MMM_MESSAGE_PRIORITY_HIGH` -> **High**. other fields, if present, are not directly mapped.

The optional **senderAddress** part of the original **sendMessageRequest** message associated with this multimedia session, which was established as described in clause 6.1.1.1, is mapped to the **messageServiceActivationNumber** element of the **message** part of the **notifyMessageReceptionRequest** message.

As previously noted, the endpoint definition of the application web service to which the **notifyMessageReceptionRequest** message is sent, including the value of the **correlator** part, is provisioned offline.

6.1.5 getReceivedMessages

The sequence diagram in clause 5.2 (figure 2) illustrates the flow for the **getReceivedMessages** operation. It is not explicitly mapped to any Parlay/OSA method. Instead, the **getReceivedMessages** operation is a bulk retrieval capability for previously received multimedia messages matching criteria defined in an off-line provisioning step. This retrieval operation includes matching messages previously and individually reported to the application via the **notifyMessageReception** operation.

6.1.6 getMessageURIs

The sequence diagram in clause 5.2 (figure 2) illustrates the flow for the **getMessageURIs** operation. It is not explicitly mapped to any Parlay/OSA method. Instead, the **getMessageURIs** operation is a retrieval capability for a received multimedia message whose reference was previously obtained by the application via the **notifyMessageReception** or **getReceivedMessages** operations.

6.1.7 getMessage

The sequence diagram in clause 5.2 (figure 2) illustrates the flow for the **getMessage** operation. It is not explicitly mapped to any Parlay/OSA method. Instead, the **getMessage** operation is a retrieval capability for a received multimedia message whose reference was previously obtained by the application via the **notifyMessageReception** or **getReceivedMessages** operations.

6.1.8 stopMessageNotification

The sequence diagram in clause 5.2 (figure 2) illustrates the flow for the **stopMessageNotification** operation, which is mapped to the Parlay/OSA method: `IpMMMManager.destroyNotification`, provided that the referenced notification is the last active notification for the associated destination address. Otherwise at least one other notification (i.e. associated with a different text string criteria value) remains active for this destination address and the mapping is not performed.

6.1.8.1 Mapping to `IpMMMManager.destroyNotification`

The `IpMMMManager.destroyNotification` is invoked with the following parameters:

Name	Type	Comment
assignmentID	TpAssignmentID	Not mapped. [The value provide in the result from `IpMMMManager.createNotification` and correlated with the value of the **reference** part received from the application in the original **startMessageNotificationRequest** message and the value of the **correlator** part received from the application in the **stopMessageNotificationRequest** message]

Parlay exceptions thrown by `IpMMMManager.destroyNotification` are mapped to Parlay X exceptions as defined in clause 6.2.1.

6.2 Operations (Mailbox Paradigm)

6.2.1 sendMessage

The sequence diagram in clause 5.3 (figure 3) illustrates the flow for the **sendMessage** operation.

The **sendMessage** operation is synchronous from the Parlay X client's point of view. It is mapped to the following Parlay/OSA methods:

- `IpMMMManager.openMailbox`, if not already opened for the application.
- `IpMMM.putMessageReq.`

6.2.1.1 Mapping to `IpMMMManager.openMailbox`

The `IpMMMManager.openMailbox` method is invoked with the following parameters:

Name	Type	Comment
mailboxID	TpString	Not mapped. [Specifies the identity of the application's mailbox in the messaging system]
authenticationInfo	TpString	Not mapped. [Authentication information needed to open the application's mailbox, such as a key or password]
appMailbox	IpAppMailboxRef	Reference to callback (internal)

The result from `IpMMMManager.openMailbox` is of type `TpMailboxIdentifier` and identifies the Mailbox interface object upon which future methods are invoked: e.g. `IpMailbox.putMessageReq`. It is also correlated with the value of the **requestIdentifier** part returned to the application in the **sendMessageResponse** message.

Parlay exceptions thrown by `IpMMMManager.openMailbox` are mapped to Parlay X exceptions as defined in clause 6.3.

6.2.1.2 Mapping to `IpMailbox.putMessageReq`

The `IpMailbox.putMessageReq` method is invoked with the following parameters:

Name	Type	Comment
mailboxSessionID	TpSessionID	Not mapped. [The value provided in the result from `IpMMMManager.openMailbox`]
folderID	TpString	In order to send a message from the mailbox, the web service places the message in a designated folder, from which it will be sent. The folder to use is indicated by the service property P_PUT_MESSAGE_FOLDER_TO_SEND.
message	TpOctetSet	The actual message that needs to be sent. The message and the headers are transferred to the Messaging service. The message will be taken as is. No checking is done on the message. The web service may construct the content of the `message` parameter from the parts of the **sendMessageRequest** message by including the following information: • the 'To:' header field containing a single destination address, derived from the **addresses** part; • the 'From:' header field containing an individual destination address, derived from the **senderName**; • the 'Priority:' header field containing a value mapped from the **priority** part; • the 'Subject:' header field containing a value mapped from the **subject** part; • the message 'body' field containing the message content, which is derived from the Attachment(s) to the **sendMessageRequest** message. For each Attachment, values must be assigned to at least the following MIME Header fields (see note): ◦ '`Content-Type:`' (e.g. 'image/gif'), which has semantics defined in RFC 2045 [4] and RFC 2046 [5] ◦ EITHER '`Content-ID:`' (e.g. 'abcd'), as defined in RFC 2045 [4], OR '`Content-Location`' (e.g. 'filename.ext'), as defined in RFC 2557 [6] ◦ '`Content-Transfer-Encoding:`' (e.g. 'binary'), as defined in RFC 2045 [4] Notes: • The optional **charging** part is not mapped. • If the web service knows the messaging system and understands the format for sending attachments, it can do so as an alternative, or in addition, to populating the `message` parameter.
NOTE: The content of a multimedia message is sent as an Attachment(s). However, since the Attachment(s) are sent transparently to the underlying network, there are additional considerations. The Parlay X Multimedia Messaging Web Service should check that all minimum required header fields are available; the set of minimum required header fields may vary based on the underlying network or as mandated by policies in the gateway.		

The result from `IpMailbox.putMessageReq` is of type `TpAssignmentID` and is used internally to correlate the callback invocation of the `IpAppMailbox.getMessageRes/Err` method.

Parlay exceptions thrown by `IpMailbox.putMessageReq` are not mapped to Parlay X exceptions. Instead they are reported to the application in a **getMessageDeliveryStatusResponse** message, with the following part values:

- the **deliveryStatus.address** element contains the associated message destination address, originally derived from the **addresses** part;
- the **deliveryStatus.deliveryStatus** element has the value: **DeliveryImpossible.**

6.2.2 getMessageDeliveryStatus

The sequence diagram in clause 5.3 (figure 3) illustrates the flow for the **getMessageDeliveryStatus** operation.

The **getMessageDeliveryStatus** operation is synchronous from the Parlay X client's point of view. It is mapped to/from the following Parlay/OSA methods:

- `IpAppMailbox.putMessageRes.`
- `IpAppMailbox.putMessageErr.`
- `IpMailbox.getMessageInfoPropertiesReq.`

- `IpAppMailbox.getMessageInfoPropertiesRes.`
- `IpAppMailbox.getMessageInfoPropertiesErr.`

The delivery status provided to the Parlay X client will depend on the timing of the **getMessageDeliveryStatus** operation invocation. If the delivery status for some destination addresses is known, as a result of earlier invocations of the `IpMailbox.getMessageInfoPropertiesReq` method, then the delivery status information provided in the `IpAppMailbox.getMessageInfoPropertiesRes` callback methods is mapped. If such a report has not been received for some destination addresses, then the `IpMailbox.getMessageInfoPropertiesReq` method is invoked for each of those destination addresses.

6.2.2.1 Mapping from `IpAppMailbox.putMessageRes`

The `IpAppMailbox.putMessageRes` method is invoked with the following parameters:

Name	Type	Comment
mailboxSessionID	TpSessionID	Not mapped. [The value provided in the result from `IpMMMManager.openMailbox`]
requestID	TpAssignmentID	Not mapped. [The value provided in the result from `IpMailbox.putMessageReq`]
messageID	TpString	Not mapped. [The new ID of the message which has been placed in the folder, from which it will be sent, as requested]

In the absence of more recent delivery status information (i.e. as provided in an `IpAppMailbox.getMessageInfoPropertiesRes` method), this method results in the assignment of the following values to one **DeliveryInformation** parameter of the **deliveryStatus** part of a **getSmsDeliveryStatusResponse** message:

- the **address** element contains the associated message destination address;
- the **deliveryStatus** element has the value: **DeliveryUncertain.**

6.2.2.2 Mapping from `IpAppMailbox.putMessageErr`

The `IpAppMailbox.putMessageErr` method is invoked with the following parameters:

Name	Type	Comment
mailboxSessionID	TpSessionID	Not mapped. [The value provided in the result from `IpMMMManager.openMailbox`]
requestID	TpAssignmentID	Not mapped. [The value provided in the result from `IpMailbox.putMessageReq`]
error	TpMessaging Error	Results in the assignment of the following values to one **DeliveryInformation** parameter of the **deliveryStatus** part of a **getSmsDeliveryStatusResponse** message: • the **address** element contains the associated message destination address; • the **deliveryStatus** element has the value: **DeliveryImpossible**.
errorDetails	TpString	Not mapped.

6.2.2.3 Mapping to `IpMailbox.getMessageInfoPropertiesReq`

The `IpMailbox.getMessageInfoPropertiesReq` method is invoked with the following parameters:

Name	Type	Comment
mailboxSessionID	TpSessionID	Not mapped. [The value provided in the result from `IpMMMManager.openMailbox`]
messageID	TpString	Not mapped. [The value provided in the result from `IpAppMailbox.putMessageRes`]

The result from `IpMailbox.getMessageInfoPropertiesReq` is of type `TpAssignmentID` and is used internally to correlate the callback invocation of the `IpAppMailbox.getMessageInfoPropertiesRes/Err` method.

Parlay exceptions thrown by `IpMailbox.getMessageInfoPropertiesReq` are not mapped to Parlay X exceptions.

6.2.2.4 Mapping from `IpAppMailbox.getMessageInfoPropertiesRes`

The `IpAppMailbox.getMessageInfoPropertiesRes` method is invoked with the following parameters:

Name	Type	Comment
mailboxSessionID	TpSessionID	Not mapped. [The value provided in the result from `IpMMMManager.openMailbox`]
requestID	TpAssignmentID	Not mapped. [The value provided in the result from `IpMailbox.getMessageInfoPropertiesReq`]
messageID	TpString	Not mapped. [The value provided in the invocation of `IpMailbox.getMessageInfoPropertiesReq`]
returnedProperties	TpMessageInfo PropertySet	Provides various message properties (names and values). Of these, the value of a single element, `MessageStatus`, is mapped to the **deliveryStatus** element of one **DeliveryInformation** parameter of the **deliveryStatus** part of a **getSmsDeliveryStatusResponse** message, as follows: • **Delivered**, if `MessageStatus` parameter value = `P_MMM_SENT_MSG_STATUS_DELIVERED`, `P_MMM_SENT_MSG_STATUS_READ` or `P_MMM_SENT_MSG_STATUS_DELETED_UNREAD`. • **DeliveryImpossible**, if `MessageStatus` parameter value = `P_MMM_SENT_MSG_STATUS_NOT_DELIVERABLE` or `P_MMM_SENT_MSG_STATUS_EXPIRED`. • **DeliveryUncertain**, if `MessageStatus` parameter value = `P_MMM_SENT_MSG_STATUS_SENT`. [Note that the **address** element of the **DeliveryInformation** parameter contains the associated message destination address]

6.2.2.5 Mapping from `IpAppMailbox.getMessageInfoPropertiesErr`

This failed attempt to poll for delivery status does not change the current **deliveryStatus** value (i.e. **Delivered**, **DeliveryImpossible**, or **DeliveryUncertain**) for any of the destination addresses.

6.2.3 startMessageNotification

The sequence diagram in 0 illustrates the flow for the **startMessageNotification** operation, which is mapped to the Parlay/OSA method: `IpMMMManager.createNotification`, provided there is no existing notification already established for the destination address contained in the **messageServiceActivationNumber** part.

6.2.3.1 Mapping to `IpMMMManager.createNotification`

The `IpMMMManager.createNotification` is invoked with the following parameters:

Name	Type	Comment
appMMM Manager	IpAppMMM ManagerRef	Not mapped. Reference to callback (internal)
eventCriteria	TpMessaging EventCriteriaSet	Contains a single element specifying the event notification criteria, for the messaging event: `P_EVENT_MSG_NEW_MAILBOX_MESSAGE_ARRIVED`. The criteria consist of 2 fields: • `MailboxID`, which identifies a mailbox in the messaging system that is correlated with the short message destination address contained in the **messageServiceActivationNumber** part • `AuthenticationInfo`, which provides the authentication information needed to open the mailbox, such as a key or password

The result from `IpMMMManager.createNotification` is of type `TpAssignmentID` and is used internally to correlate the callbacks. Specifically it is correlated with the value of the **reference** part received from the application in the **startMessageNotificationRequest** message and the **correlator** part returned to the application in the **notifyMessageReceptionRequest** message.

Note that the **reference** part and the optional **criteria** part of a **startMessageNotificationRequest** message are not mapped to `IpMMMManager.createNotification`. Instead the web service uses all the text string criteria values associated with a specific destination address to parse any received message event reported for that address by the `IpAppMMMManager.reportNotification` method. The web service determines whether the event is valid - i.e. there is a match with a text string criteria value. If valid, the web service retrieves and stores the message and selects the previously provisioned application callback web service to receive the **notifyMessageReceptionRequest** message. If invalid, the web service discards the event notification.

Parlay exceptions thrown by `IpMMMManager.createNotification` are mapped to Parlay X exceptions as defined in clause 6.3.

6.2.4 notifyMessageReception

The sequence diagram in clause 5.4 (figure 4) illustrates the flow for the **notifyMessageReception** operation, which is mapped to/from the following Parlay/OSA methods:

- IpAppMMMManager.reportNotification.
- `IpMMMManager.openMailbox.`
- `IpMailbox.`getMessageContentReq.
- IpAppMailbox.getMessageContentRes.
- `IpMailbox.`getMessageBodyPartsReq.
- IpAppMailbox.getMessageBodyPartsRes.

6.2.4.1 Mapping from `IpAppMMMManager.reportNotification`

The `IpAppMMMManager.reportNotification` method is invoked with the following parameters:

Name	Type	Comment
assignmentID	TpAssignmentID	Not mapped. [The value provide in the result from `IpMMMManager.createNotification`]
eventInfo	TpMessaging EventInfoSet	Contains a set of one (or more) received message notification(s) and related message information. For each notification, the fields of the `EventNewMailboxMessageArrived` element are mapped as follows: • `MailboxID`: the mailbox identifier in each message notification is the same; i.e. it is equivalent to the value specified in the event criteria (clause 6.2.3.1). This field correlates with the message destination address returned in the **messageServiceActivationNumber** element of the **message** part. • `FolderID`: the folder identifier in each message notification specifies the identity of the folder in which the received message is stored • `MessageDescription` contains sub-fields, of which three are applicable for the mapping: • `MessageID`: the message identifier for the received message, which is used by the web service to retrieve the message. • `From`: the sender of the received message, which maps to the **senderAddress** element of the **message** part. The data type mapping from `TpAddress` to **xsd:anyURI** is described in TR 102 397-1 [1]. • `Subject`, which maps to the **subject** element of the **message** part. If the `Subject` is a non-null text string, then the first word is used for matching against optional text string criteria, where: • the 'first word' in the message is defined as the initial characters after discarding any leading Whitespace and ending with a Whitespace or end of message; • the matching is case-insensitive. • `ExtendedHeaderInformation` contains sub-fields, of which the following are applicable for the mapping: • `Priority`, which maps to the **priority** element of the **message** part, as follows: • `P_MMM_MESSAGE_PRIORITY_HIGH` -> **High**. • `P_MMM_MESSAGE_PRIORITY_LOW` -> **Low**. • `P_MMM_MESSAGE_PRIORITY_UNDEFINED` -> **Default** or **Normal**. • `MimeContent` sub-field(s), defining the content type for each attachment, if any. If none exist, or all the value(s) denote text formatting only, then the message is pure ASCII text.

The result from `IpAppMMMManager.reportNotification` is of type `IpAppMultiMediaMessagingRef`. It is set to null.

6.2.4.2 Mapping to `IpMMMManager.openMailbox`

The `IpMMMManager.openMailbox` method is invoked with the following parameters:

Name	Type	Comment
mailboxID	TpString	Specifies the identity of the application's mailbox in the messaging system: i.e. as specified in the `eventInfo` parameter of the `reportNotification` method (clause 6.2.4.1).
authenticationInfo	TpString	Specifies authentication information needed to open the application's mailbox, such as a key or password: i.e. as specified in the `AuthenticationInfo` field of the `eventCriteria` parameter of the `createNotification` method (clause 6.2.3.1).
appMailbox	IpAppMailboxRef	Reference to callback (internal)

The result from `IpMMMManager.openMailbox` is of type `TpMailboxIdentifier` and identifies the Mailbox interface object upon which future methods are invoked: e.g. `IpMailbox.getMessageContentReq, IpMailbox.getMessageBodyPartsReq`.

Parlay exceptions thrown by `IpMMMManager.openMailbox` are not mapped to Parlay X exceptions.

6.2.4.3 Mapping to `IpMailbox.getMessageContentReq`

The `IpMailbox.getMessageContentReq` method is invoked with the following parameters:

Name	Type	Comment
mailboxSessionID	TpSessionID	Not mapped. [The value provided in the result from `IpMMMManager.openMailbox`]
folderID	TpString	Not mapped. [The value provided in the `eventInfo` parameter of the `reportNotification` method (clause 6.2.4.1)]
messageID	TpString	Not mapped. [The value provided in the `MessageDescription.MessageID` field of the `eventInfo` parameter of the `reportNotification` method (clause 6.2.4.1)]

The result from `IpMailbox.getMessageContentReq` is of type `TpAssignmentID` and is used internally to correlate the callback invocation of the `IpAppMailbox.getMessageContentRes/Err` method.

Parlay exceptions thrown by `IpMailbox.getMessageContentReq` are not mapped to Parlay X exceptions.

6.2.4.4 Mapping from `IpAppMailbox.getMessageContentRes`

The `IpAppMailbox.getMessageContentRes` method is invoked with the following parameters:

Name	Type	Comment
mailboxSessionID	TpSessionID	Not mapped. [The value provide in the result from `IpMMMManager.openMailbox`]
requestID	TpAssignmentID	Not mapped. [The value provided in the result from `IpMailbox.getMessageContentReq`]
contentType	TpString	Not mapped.
contentTransfer Encoding	TpString	Not mapped.
content	TpOctetSet	Contains the body of the message. If the `MessageDescription.Subject` field of the `eventInfo` parameter of the `reportNotification` method (clause 6.2.4.1) was null, the first word of this `content` field is used for matching against optional text string criteria, where: • the 'first word' in the message is defined as the initial characters after discarding any leading Whitespace and ending with a Whitespace or end of message; • the matching is case-insensitive. If the message is pure ASCII text, as indicated in the `ExtendedHeaderInformation.MimeContent` field(s) of the `eventInfo` parameter of the `reportNotification` method (clause 6.2.4.1), then this `content` field is mapped to the **message** element of the **message** part. [Otherwise, the `content` field is discarded and all parts of the multimedia message are retrieved using the `Ip(App)Mailbox.getMessageBodyPartsReq/Res` methods].

6.2.4.5 Mapping to `IpMailbox.getMessageBodyPartsReq`

The `IpMailbox.getMessageBodyPartsReq` method is invoked with the following parameters:

Name	Type	Comment
mailboxSessionID	TpSessionID	Not mapped. [The value provided in the result from `IpMMMManager.openMailbox`]
folderID	TpString	Not mapped. [The value provided in the `eventInfo` parameter of the `reportNotification` method (clause 6.2.4.1]
messageID	TpString	Not mapped. [The value provided in the `MessageDescription.MessageID` field of the `eventInfo` parameter of the `reportNotification` method (clause 6.2.4.1)]
partIDs	TpStringList	Set to a null string value to indicate that all message parts are to be retrieved.

The result from `IpMailbox.getMessageBodyPartsReq` is of type `TpAssignmentID` and is used internally to correlate the callback invocation of the `IpAppMailbox.getMessageBodyPartsRes/Err` method.

Parlay exceptions thrown by `IpMailbox.getMessageBodyPartsReq` are not mapped to Parlay X exceptions.

6.2.4.6 Mapping from `IpAppMailbox.getMessageBodyPartsRes`

The `IpAppMailbox.getMessageBodyPartsRes` method is invoked with the following parameters:

Name	Type	Comment
mailboxSessionID	TpSessionID	Not mapped. [The value provide in the result from `IpMMMManager.openMailbox`]
requestID	TpAssignmentID	Not mapped. [The value provided in the result from `IpMailbox.getMessageBodyPartsReq`]
bodyParts	TpBodyPartList	Contains the details and content of each part of the multimedia message. The web service stores the retrieved information and returns a reference for the stored multipart message to the application in the **messageIdentifier** element of the **message** part.

6.2.5 getReceivedMessages

The sequence diagram in clause 5.4 (figure 4) illustrates the flow for the **getReceivedMessages** operation. It is not explicitly mapped to any Parlay/OSA method. Instead, the **getReceivedMessages** operation is a bulk retrieval capability for previously received multimedia messages matching criteria defined in an off-line provisioning step. This retrieval operation includes matching messages previously and individually reported to the application via the **notifyMessageReception** operation.

6.2.6 getMessageURIs

The sequence diagram in clause 5.4 (figure 4) illustrates the flow for the **getMessageURIs** operation. It is not explicitly mapped to any Parlay/OSA method. Instead, the **getMessageURIs** operation is a retrieval capability for a received multimedia message whose reference was previously obtained by the application via the **notifyMessageReception** or **getReceivedMessages** operations.

6.2.7 getMessage

The sequence diagram in clause 5.4 (figure 4) illustrates the flow for the **getMessage** operation. It is not explicitly mapped to any Parlay/OSA method. Instead, the **getMessage** operation is a retrieval capability for a received multimedia message whose reference was previously obtained by the application via the **notifyMessageReception** or **getReceivedMessages** operations.

6.2.8 stopMessageNotification

The sequence diagram in clause 5.4 (figure 4) illustrates the flow for the **stopMessageNotification** operation, which is mapped to the Parlay/OSA method: `IpMMMManager.destroyNotification`, provided that the referenced notification is the last active notification for the associated destination address. Otherwise at least one other notification (i.e. associated with a different text string criteria value) remains active for this destination address and the mapping is not performed.

6.2.8.1 Mapping to `IpMMMManager.destroyNotification`

The `IpMMMManager.destroyNotification` is invoked with the following parameters:

Name	Type	Comment
assignmentID	TpAssignmentID	Not mapped. [The value provide in the result from `IpMMMManager.createNotification` and correlated with the value of the **reference** part received from the application in the original **startMessageNotificationRequest** message and the value of the **correlator** part received from the application in the **stopMessageNotificationRequest** message]

Parlay exceptions thrown by `IpMMMManager.destroyNotification` are mapped to Parlay X exceptions as defined in clause 6.1.6.

6.3 Exceptions

In addition to the common mapping of Parlay/OSA API method exceptions to Parlay X Web Service exceptions, which is defined in TR 102 397-1 [1], there are the following service-specific exception mappings:

Parlay/OSA Exception	Service Exception	Notes
P_MMM_INVALID_MAILBOX	SVC0001	With error number
P_MMM_INVALID_AUTHENTICATION_ INFORMATION	SVC0001	With error number

7 Additional Notes

No additional notes are provided.

History

Document history		
V1.1.1	December 2005	Publication