3GPP TS 29.199 V1.0.1 (2004-01)

Technical Specification

3rd Generation Partnership Project; Technical Specification Group Core Network; Open Service Access (OSA); Parlay X Web Services (Release 6)

Keywords
API, OSA

3GPP

Postal address

3GPP support office address
650 Route des Lucioles - Sophia Antipolis
Valbonne - FRANCE
Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Internet
http://www.3gpp.org

Contents

Foreword

This Technical Specification has been produced by the 3rd Generation Partnership Project (3GPP).

3GPP acknowledges the contribution of the Parlay X Web Services specification from The Parlay Group. The Parlay Group is pleased to see 3GPP acknowledge and publish this specification, and the Parlay Group looks forward to working with the 3GPP community to improve future versions of this specification.

The contents of the present document are subject to continuing work within the TSG and may change following formal TSG approval. Should the TSG modify the contents of the present document, it will be re-released by the TSG with an identifying change of release date and an increase in version number as follows:

Version x.y.z

where:

- x the first digit:
  - 1 presented to TSG for information;
  - 2 presented to TSG for approval;
  - 3 or greater indicates TSG approved document under change control.
- y the second digit is incremented for all changes of substance, i.e. technical enhancements, corrections, updates, etc.
- z the third digit is incremented when editorial only changes have been incorporated in the document.

1 Scope

The present document specifies an initial set of Parlay X Web Services. For each selected Web Service, this document describes the motivation for its inclusion, the commercial and technical rationale, and an illustrative usage scenario(s). This document also specifies the message(s) exchanged during invocations of each Web Service, by defining the semantics in English and the syntax using W3C WSDL.

The OSA APIs are designed to enable creation of telephony applications as well as to "telecom-enable" IT applications. IT developers, who develop and deploy applications outside the traditional telecommunications network space and business model, are viewed as crucial for creating a dramatic whole-market growth in next generation applications, services and networks.

The Parlay X Web Services are intended to stimulate the development of next generation network applications by developers in the IT community who are not necessarily experts in telephony or telecommunications. The selection of Web Services should be driven by commercial utility and not necessarily by technical elegance. The goal is to define a set of powerful yet simple, highly abstracted, imaginative, telecommunications capabilities that developers in the IT community can both quickly comprehend and use to generate new, innovative applications.

2 References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication, edition number, version number, etc.) or non-specific.
- For a specific reference, subsequent revisions do not apply.
- For a non-specific reference, the latest version applies. In the case of a reference to a 3GPP document (including a GSM document), a non-specific reference implicitly refers to the latest version of that document *in the same Release as the present document*.

[1] 3GPP TR 21.905: "Vocabulary for 3GPP Specifications".

[2] 3GPP TS 22.101: "Service aspects; Service principles".

[3] 3GPP TS 23.040: "Technical realization of Short Message Service (SMS)".

[4] RFC 2396: "Uniform Resource Identifiers (URI): Generic Syntax".

[5] RFC 2732: "Format for Literal IPv6 Addresses in URL's".

[6] RFC 2806: "URLs for Telephone Calls".

[7] RFC 3261: "SIP: Session Initiation Protocol".

[8] RFC 2848: "The PINT Service Protocol: Extensions to SIP and SDP for IP Access to Telephone Call Services".

3 Definitions and abbreviations

3.1 Definitions

For the purposes of the present document, the terms and definitions given in 3GPP TS 22.101 [2] and the following apply.

(Parlay X) Application: unless otherwise specified, the document will be using the term (Parlay X) application to refer to software that invokes or can invoke a Parlay X Web Service.

Parlay X Gateway: used to describe the implementation of a set of Parlay X Web Services. In telecommunications parlance an implementation of a Parlay X Web Service on a Parlay X Gateway would also be referred to as a service.

3.2 Abbreviations

For the purposes of the present document, the abbreviations given in 3GPP TR 21.905 [1] and the following apply.

A/IN	(Advanced) Intelligent Network
AAA	Authorization, Authentication, and Accounting
AM	Account Management
API	Application Programming Interface
APP	Application
CBC	Content Based Charging
CC	Call Control
DIME	Direct Internet Message Encapsulation
EMS	Enhanced Messaging Service
FW	(OSA/Parlay) Framework
GIF	Graphics Interchange Format
GPRS	General Packet Radio System
IM	Instant Messaging
IP	Internet Protocol
ICQ	???
IT	Information Technology
IVR	Interactive Voice Response
JAIN	Integrated Network APIs for the Java™ platform
JCP	Java™ Community Process℠
JPay	Payment API for the Java™ platform
MIME	Multipurpose Internet Mail Extensions
MM7	Communication protocol between MMS Relay/Server and MMS Application Server
MMS	Multimedia Message Service
MMS-C	Multimedia Message Service Center
MPS	Mobile Positioning System
MS	Mobile Station Why not UE User Equipment ???? - see 3GPP TR 21.905 [1]
MSC	Mobile Switching Center
MSISDN	Mobile Station ISDN Number
OASIS	Organization for the Advancement of Structured Information Standards
PIN	Personal Identification Number
PINT	PSTN and Internet inter-networking
PSTN	Public Switched Telephone Network
SAML	Security Assertion Markup Language: i.e. an XML-based security standard for exchanging authentication and authorization information, developed by SSTC)
SCF	Service Capability Feature
SCS	Service Capability Server
SIP	Session Initiation Protocol
SLA	Service Level Agreement
SMS	Short Message Service
SMS-C	Short Message Service Center
SOAP	Simple Object Access Protocol
SSTC	Security Services Technical Committee (of OASIS)
UCP	Universal Computer Protocol
URI	Uniform Resource Identifier
VASP	Virtual Application Service Provider
W3C	World Wide Web Consortium
WAP	Wireless Application Protocol
WG	Working Group
WGS 84	World Geodetic System 1984
WS	Web Service
WSDL	Web Service Definition Language
WS-I	Web Services-Interoperability Organization
XML	Extensible Markup Language

4 Parlay X Web Services

4.1 Selection Criteria

Each Parlay X Web Service should be abstracted from the set of telecommunications capabilities exposed by the Parlay APIs, but may also expose related capabilities that are not currently supported in the Parlay APIs where there are compelling reasons. These tiered levels of abstraction, and the Parlay – Parlay X relationship, are illustrated in Figure 1.

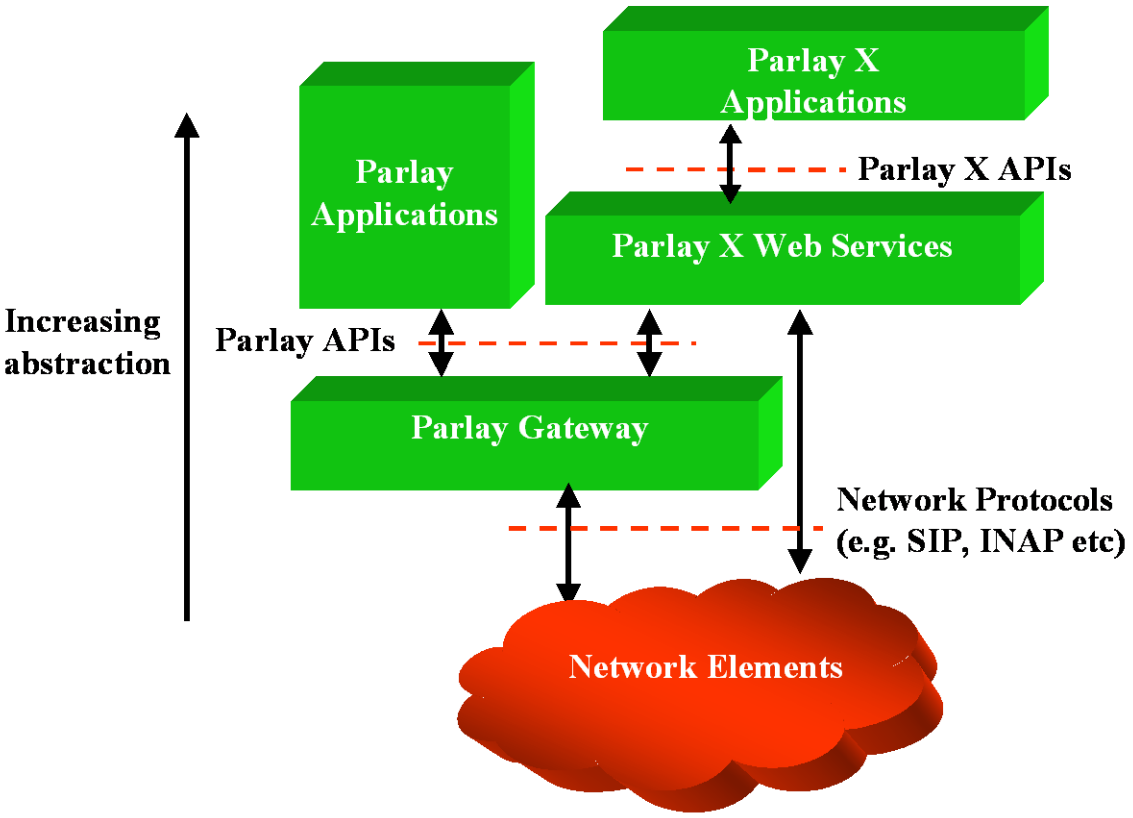

Figure 1: Parlay X Web Services, Parlay X APIs and Parlay APIs

The selection criteria for the Parlay X Web Services are as follows:

- The capabilities offered by a Parlay X Web Service may be either homogeneous (e.g. call control only) or heterogeneous (e.g. terminal location and user status).
- The interaction between an application incorporating a Parlay X Web Service and the server implementing the Parlay X Web Service will be done with an XML-based message exchange. The message exchange should follow the synchronous request/response model and be initiated by the application; the 'response' from the Parlay X Web Service is optional. However asynchronous messages from the Parlay X Web Service implementation (on a Parlay X Gateway) to the application may be defined where there are compelling reasons; e.g. to implement a notification type web service. In the latter case, the message exchange would invoke an application web service using a similar, synchronous request/response model.
- Parlay X Web Service invocations should not be correlated and the Web Service itself should be stateless from the perspective of the application, unless there are compelling reasons. In particular, in the case of asynchronous notifications from a Parlay X Web Service implementation (on a Parlay X Gateway) to an application, NO application-initiated invocations to provision (or de-provision) notification-related criteria in the Parlay X Web Service should be implemented.

- Parlay X Web Services follow simple application semantics, allowing the developer to focus on access to the telecom capability using common Web Services programming techniques.
- Parlay X Web Services are not network equipment specific, and not network specific where a capability is relevant to more than one type of network.
- Parlay X Web Services should be based on the reference Web Service technology, as it is defined in the IT world and according to the Parlay Web Services WG recommendations; more specifically, at the moment, WSDL is the chosen standard to invoke and describe Parlay X Web Services.
- The Parlay X APIs should be extendible; integration of third party provided interfaces must be supported using proven, reliable, and standard Web Service technology.
- Parlay X Web Services are application interfaces and do not provide an implementation of AAA (Authorization, Authentication, and Accounting), service level agreements or other environment-specific capabilities. Rather, they shall rely on proven and reliable solutions provided by the Web Services infrastructure.

4.2 Implementation Considerations

4.2.1 Versioning

The namespace will contain a version number as a suffix for the service name. The suffix will be a separate path element consisting of: the letter 'v' followed by a major version number followed by an underscore and a minor version number.

- The major version number will reflect the version of Parlay X for the last change to the API.
- The minor version number will reflect maintenance of the API between Parlay X versions.

EXAMPLE: www.csapi.org/wsdl/parlayx/account_management/v1_0 represents the initial public version of the Parlay X Account Management Web Service.

The rationale for this version number scheme is as follows:

- Prefixing the path element with "v" and using an underscore minimizes the possibility of tool issues. Using namespace elements that are only digits (or that contain characters other than letters, digits and underscores) may result in issues with different tools - e.g. period is a package delimiter
- Adding the version path element after the API element, instead of after the "parlayx" element, means that as Parlay X changes, that APIs that do not change will not require changes to running services since their namespaces will not change
- Using this convention eliminates parsing and related issues with two digit versions, leading zeros, etc.
- This convention can also be applied to other Parlay X namespaces such as Common Data.

4.3 Optional Parameters

Some parameters are defined as OPTIONAL. Within a message signature, the XML schema type definition for an optional parameter may use the *nillable* attribute to allow the actual value of the parameter to be "NULL" (or "NIL"). Since this approach is dependant upon the industry adoption of WS-I's Basic Profile 1.0, which is currently not widely implemented, this version of the Parlay X Web Services specification ***does not follow*** this approach. As a temporary measure this specification explicitly defines "NULL" values for parameters tagged OPTIONAL such that, if used, the actual value of the parameter must be considered "NULL".

The following "NULL" values are defined for each OPTIONAL parameter data type:

- Type **String**: the empty string (`""`).
- Type **DateTime**: `"0001-01-01T00:00:00Z"`.

- Type **Integer**: "-1".
- Type **EndUserIdentifier** (which encapsulates a URI): "null://null".

4.4 Document Structure

The present document has the following clauses:

4) Parlay X Web Services
5) Common Data Definitions
6) Third Party Call
7) Network-Initiated Third Party Call
8) SMS
9) Multimedia Message
10) Payment
11) Account Management
12) User Status
13) Terminal Location

Apart from this clause and clause 5, each of the remaining clauses specifies a Parlay X Web Service with the following structure:

- Overview, describing the Web Service, the underlying commercial and/or technical rationale, its relationship to other specifications, and illustrative usage scenario(s).
- A semantic specification of the message-based Parlay X API(s) that constitute the Web Service.
- A definition of the Web-Service-specific data types and exceptions.
- Web Service Syntax, referencing two types of WSDL files for this release of the Parlay X APIs:
  - A modular set of WSDL files that provide an 'rpc/literal' binding and follows the current draft provided by the WS-I Basic Profile. This set of files may be used by tools that support the features included in the Basic Profile definition.
  - A second set of monolithic WSDL files that provide an 'rpc/encoded' binding and use SOAP encoded arrays. This allows developers to use tools that do not yet support the features included in the Basic Profile definition. It is expected that tools will be updated over time, and that developers will migrate to the first set of files, to be compliant with the WS-I Basic Profile.

This document structure, one clause for each Parlay X Web Service, facilitates future releases of the Parlay X specification. It minimizes the scope of editorial work required to either introduce a new Parlay X Web Service, which is accomplished by adding a new document clause, or to update an existing Parlay X Web Service, which is achieved by editing a single, existing document clause.

In addition to the Parlay X Web Service-specific clauses, the present document (in clause 5) provides a listing of data definitions (including exceptions) that are common across multiple Parlay X Web Services. As more Web Services are defined in the future, the number of common data definitions may increase. Thus clause 5 has the following structure:

- Record of Changes, providing a detailed record of any additions, deletions, modifications and other changes that have been applied to the data definitions since its previous public release.
- An English language definition of the common data types and exceptions.
- Common Data Definitions Syntax, referencing two types of WSDL files for this release of the Parlay X APIs.

5 Common Data Definitions

5.1 Common Data Type Definitions

Where possible standard XML Schema data types are used, as defined in http://www.w3.org/TR/xmlschema-2/#built-in-datatypes.

Other data types that are common to multiple Parlay X Web Services are defined in the following subclauses.

5.1.1 EndUserIdentifier

The EndUserIdentifier data type is specified as a URI: [scheme]:[schemeSpecificPart] (RFC 2396 [4], amended by RFC 2732 [5]). Example schemes are tel (RFC 2806 [6]) and sip (RFC 3261 [7]).

5.1.2 ArrayOfEndUserIdentifier

A collection of elements where each element is of data type EndUserIdentifier.

5.1.3 ArrayOfURI

A collection of elements where each element is of data type URI.

5.2 Common Exception Definitions

All of these exceptions are common to multiple Parlay X Web Services. Each exception is assigned an eight character identifier, where:

- The first 3 characters "GEN" identify the exception as generic: i.e. common to multiple Parlay X services. The "GEN" string shall not be assigned to any Parlay X Web Service-specific exception.
- The next 4 digits "1xxx" uniquely identifies the exception within the set of common exceptions. The "1xxx" string may be re-used by any Parlay X Web Service-specific exception defined in this specification.
- The last character identifies the severity of the exception condition, as follows:
  - "F": fatal error, typically indicating an infrastructure problem; the operation triggering the exception should not be retried
  - "E": error, typically indicating an application or user error; the operation triggering the exception has not completed and may be retried
  - "W": warning, typically indicating that an operation has completed, but there are cautions or other caveats.

The common exceptions are as follows:

UNIQUE ID	TEXT STRING	MEANING
GEN1000E	UnknownEndUserException	This fault occurs if the end user identification that is passed is unknown.
GEN1001E	InvalidArgumentException	This fault occurs if an argument passed is semantically incorrect or when the parameter does not conform to the limits specified in the Parlay X specification: e.g. when passing the end user identification: "tel:www.parlay.org".
GEN1002F	ServiceException	This fault is caused by generic platform or network errors.
GEN1003E	PolicyException	This fault is caused by a violation of a policy of the Parlay X Web Service: e.g., when parameter values are used that are outside the scope of the service level agreement.
GEN1004E	ApplicationException	This fault is caused by a generic error in an application web service when processing a message invocation from a Parlay X Web Service. The Parlay X Web Service can log this information and possibly raise an alarm when the number of exceptions reaches a pre-defined threshold.
GEN1005W	MessageTooLongException	This fault is caused if a message to be sent exceeds the maximum length supported by the Web Service; e.g. the message may be too long for a destination terminal device.

5.3 Common Data Definitions Syntax – WSDL

The rpc/literal files include two common files

- parlayx_common_types.xsd
- parlayx_common_faults.wsdl

The rpc/encoded files contain these definitions within each service file.

6 Third Party Call

6.1 Overview

6.1.1 Description

Currently, in order to perform a third party call in telecommunication networks we have to write applications using specific protocols to access Call Control functions provided by network elements (specifically operations to initiate a call from applications). This approach requires a high degree of network expertise. We can also use the OSA gateway approach, invoking standard interfaces to gain access to call control capabilities, but these interfaces are usually perceived to be quite complex by application IT developers. Developers must have advanced telecommunication skills to use Call Control OSA interfaces.

In this subclause we describe a Parlay X Web Service, Third Party Call, for creating and managing a call initiated by an application (third party call). The overall scope of this Web Service is to provide functions to application developers to create a call in a simple way. Using the Third Party Call Web Service, application developers can invoke call handling functions without detailed telecommunication knowledge.

6.1.2 Commercial & Technical Rationale

The basic commercial rationale for developing the Third Party Call Web Service is:

- to increase the use of Call Control capabilities in software applications
- to empower traditional IT developers to produce large numbers of such applications
- to lower the development cost and time for such applications.

6.1.3 Relationship to Similar or Supplanted Specifications

In the PSTN/Internet Interworking (PINT) working group a "request to call" scenario has been defined (RFC 2848 [8]). A request is sent to an IP host that cause a phone call to be made, connecting party A to some remote party B.

The PINT approach is not API-based but instead it proposes an extension to SIP to implement the scenarios identified. The Third Party Call Web Service proposed here while addressing similar scenarios aims at providing a high-level Web Services interface to invoke the service. It does not aim at defining a concrete architecture implementing the functionality. Therefore the two specifications are at different levels of abstraction.

6.1.4 Scenarios

Figure 2 shows an scenario using the Third Party Call Web Service to handle third party call functions. The application invokes a web service to retrieve stock quotes and a Parlay X Interface to initiate a third party call between a broker and his client.

In the scenario, whenever a particular stock quote reaches a threshold value (1) and (2), the client application invokes a third party call between one or more brokers and their corresponding customers to decide actions to be taken. After invocation (3) by the application, the Third Party Call Web Service invokes a Parlay API method (4) using the Parlay/OSA SCS-CC (Call control) interface. This SCS handles the invocation and sends a message (5) to an MSC to set-up a call between user A and user B.

In an alternative scenario, the Parlay API interaction involving steps (4) and (5) could be replaced with a direct interaction between the Third Party Call Web Service and the Mobile network.

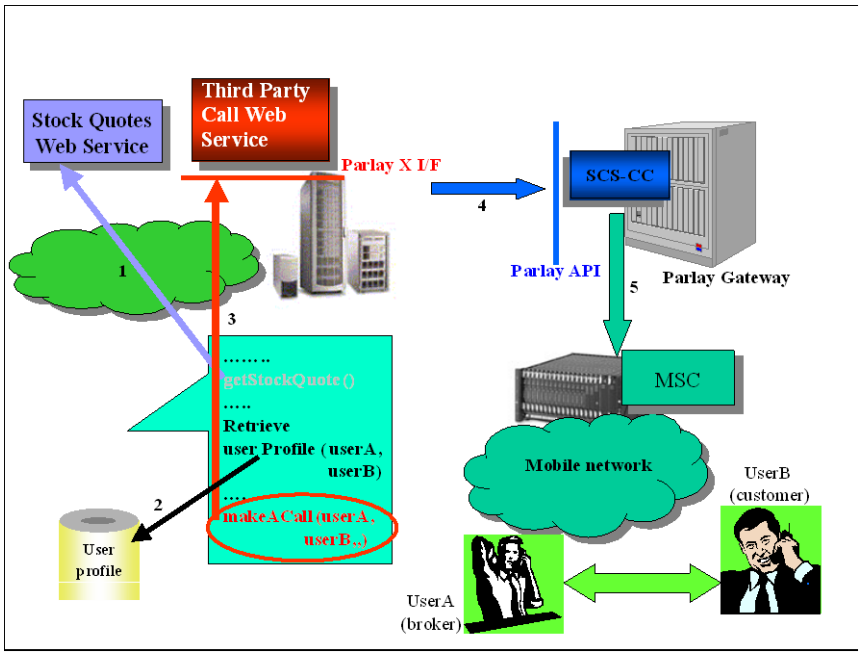

Figure 2: Third Party Call Scenario

6.2 Call API

This subclause describes an initial set of capabilities in terms of message invocations, parameters and data types. The message-based invocations are:

- makeACall
- getCallInformation
- endCall

- cancelCallRequest

6.2.1 Attempt Immediate Call Set-Up Between Two Addresses

a) makeACall(EndUserIdentifier callingParty, EndUserIdentifier calledParty, String charging, out String callIdentifier)

b) Behaviour:

The invocation of **makeACall** requests to set-up a voice call between two addresses, **callingParty** and **calledParty**, provided that the invoking application is allowed to connect them. Optionally the application can also indicate the charging arrangements (**charging**), i.e. the name of an operator-specific charging plan that defines who to charge for the call and how much.

By invoking this operation the application requires to monitor the status of the requested call. The returned parameter, **callIdentifier**, can be used to identify the call. In order to receive the information on call status the application has to explicitly invoke **getCallInformation**.

c) Parameters:

NAME	TYPE	DESCRIPTION
callingParty	EndUserIdentifier	It contains the address of the first user involved in the call
calledParty	EndUserIdentifier	It contains the address of the second user involved in the call
charging	String	OPTIONAL. If present, represents the name of an operator-specific charging plan that defines who to charge for the call and how much. If the named charge plan does not exist, the InvalidArgumentException is thrown. If no charge plan is specified, charging occurs in accordance with an operator-specific charging policy.
callIdentifier	String	OUTPUT. It identifies a specific call request

d) Exceptions:

UnknownEndUserException

InvalidArgumentException

ServiceException

6.2.2 Get Current Status of a Call

a) getCallInformation(String callIdentifier, out CallInformationType callInformation)

b) Behaviour:

The invocation of **getCallInformation** retrieves the current status, **callInformation**, of the call identified by **callIdentifier**. This method can be invoked multiple times by the application even if the call has already ended. However, after the call has ended, status information will be available only for a limited period of time that should be specified in an off-line configuration step.

c) Parameters:

NAME	TYPE	DESCRIPTION
callIdentifier	String	It identifies a specific call request
callInformation	CallInformationType	OUTPUT. It identifies the status of the call

d) Exceptions:

UnknownCallIdentifierException

ServiceException

6.2.3 End a Call

a) endCall(String callIdentifier)

b) Behaviour:

The invocation of **endCall** terminates the call identified by **callIdentifier**. If the call is still in the initial state this method has the same effect as the **cancelCallRequest** method.

c) Parameters:

NAME	TYPE	DESCRIPTION
callIdentifier	String	It identifies a specific call request

d) Exceptions:

CallTerminatedException

UnknownCallIdentifierException

ServiceException

6.2.4 Cancel a Call Request

a) cancelCallRequest(String callIdentifier)

b) Behaviour:

The invocation of **cancelCallRequest** cancels the previously requested call identified by **callIdentifier**. Note that this method differs from the **endCall** method since it only attempts to prevent the call from starting but it does not have any effect if the call has already started.

c) Parameters:

NAME	TYPE	DESCRIPTION
callIdentifier	String	It identifies a specific call request

d) Exceptions:

CallConnectedException

UnknownCallIdentifierException

ServiceException

6.3 Web Service Data Definitions

6.3.1 Data Types

In addition to the Common Data Types defined in clause 5, the following Data Types are specific to this Web Service.

6.3.1.1 CallInformationType

The **CallInformationType** data type is a structure containing the following parameters:

NAME	TYPE	DESCRIPTION
callStatus	CallStatus	It indicates the current status of the call (see possible values below)
startTime	DateTime	When applicable (callStatus <> CallInitial), it indicates the time of the beginning of the call
duration	Integer	When applicable (callStatus = CallTerminated), it indicates the duration of the call expressed in seconds
terminationCause	CallTerminationCause	When applicable (callStatus = CallTerminated), it indicates the cause of the termination of the call (see possible values below)

6.3.1.2 CallStatus

The **CallStatus** data type is an enumeration with the following values:

VALUE	DESCRIPTION
CallInitial	The call is being established
CallConnected	The call is active
CallTerminated	The call was terminated

6.3.1.3 CallTerminationCause

The **CallTerminationCause** data type is an enumeration with the following values:

VALUE	DESCRIPTION
CallingPartyNoAnswer	Calling Party did not answer
CalledPartyNoAnswer	Called Party did not answer
CallingPartyBusy	Calling Party was busy
CalledPartyBusy	Called Party was busy
CallingPartyNotReachable	Calling Party was not reachable
CalledPartyNotReachable	Called Party was not reachable
CallHangUp	The call was terminated by either party hanging up
CallAborted	The call was aborted (any other termination cause)

6.3.2 Exceptions

In addition to the Common Exceptions defined in clause 5.2, there are exceptions specific to this Web Service. Similar to the Common Exceptions, each Web Service-specific exception is assigned an eight-character identifier. This identifier is interpreted as described in clause 5.2, except that the first 3 characters uniquely identify this Web Service.

The following exceptions are specific to this Web Service:

UNIQUE ID	TEXT STRING	MEANING
3PC1000W	CallConnectedException	The call was already active
3PC1001W	CallTerminatedException	The call is already terminated
3PC1002E	UnknownCallIdentifierException	The callIdentifier supplied does not relate to any known call request or has expired.

6.4 Web Service Syntax – WSDL

The W3C WSDL representation of this API is contained in a set of files which accompany the present document (see Annex A).

The rpc/literal files are

- parlayx_third_party_calling_types.xsd
- parlayx_third_party_calling_service_port.wsdl
- parlayx_third_party_calling_service.wsdl

The rpc/encoded file is

- parlayx_third_calling_party_service.wsdl.

7 Network-Initiated Third Party Call

7.1 Overview

7.1.1 Description

Currently, in order to determine the handling of a subscriber initiated call in telecommunication networks we have to write applications using specific protocols to access Call Control functions provided by network elements. This approach requires a high degree of network expertise. We can also use the OSA gateway approach, invoking standard interfaces to gain access to call control capabilities, but these interfaces are usually perceived to be quite complex by application IT developers. Developers must have advanced telecommunication skills to use Call Control OSA interfaces.

In this subclause we will describe a Parlay X Web Service, Network-Initiated Third Party Call, for handling calls initiated by a subscriber in the network. A (third party) application determines how the call should be treated. The overall scope of this Web Service is to provide simple functions to application developers to determine how a call should be treated. Using the Network-Initiated Third Party Call Web Service, application developers can perform simple handling of network-initiated calls without specific Telco knowledge.

7.1.2 Commercial & Technical Rationale

The basic commercial rationale for developing the Network-Initiated Third Party Call Web Service is:

- to increase the use of Call Control capabilities in software applications
- to empower traditional IT developers to produce large numbers of such applications
- to lower the development cost and time for such applications.

7.1.3 Relationship to Similar or Supplanted Specifications

All the capabilities of the Network-Initiated Third Party Call Web Service (and more) can also be achieved with the Parlay/OSA generic call control or multiparty call control services. The Network-Initiated Third Party Call Web Service can be seen as a very limited subset of the network initiated call control functionality present in Parlay/OSA. This has the advantage that the application needs less telecom knowledge. The disadvantage is that the control over the call is much reduced. Basically, a Parlay X application can only choose to release, continue or re-route the call. It does not have control over the specific parameters used in the call (e.g., on the presentation indicators of the addresses), nor can the application control the call over a longer period of time. Furthermore, it is not likely that the robustness and performance requirements of Parlay/OSA services will be matched by the Network-Initiated Third Party Call Web Service.

7.1.4 Scenarios

This subclause gives some possible scenarios using the Network-Initiated Third Party Call Web Service to handle network-initiated calls.

7.1.4.1 Incoming call handling

A subscriber receives a call while he is logged-on to the Internet. Since this occupies his telephone connection, he is regarded as busy by the network. The subscriber has an application that is invoked when somebody tries to call him while he is busy. The application provides the subscriber with a list of choices on how to handle the call (e.g., route the call to voicemail, redirect the call to a secretary, reject the call). Based on the response of the subscriber the call is handled in the network.

Alternatively, the call be re-routed or released depending on the preferences of the subscriber and some context information (e.g., based on the status or location of the subscriber).

7.1.4.2 Service numbers

An application is triggered whenever a certain service number is dialled. This number is used to connect the caller to one of the maintenance personnel. The application redirects the call to the appropriate maintenance person based on, e.g., calling party number, time, location and availability of the maintenance personnel.

7.2 Call API

This subclause describes an initial set of capabilities in terms of message invocations, parameters and data types. The message-based invocations are:

- handleBusy
- handleNotReachable
- handleNoAnswer
- handleCalledNumber
- handleOffHook

These messages are initiated by the Network-Initiated Third Party Call Web Service (running in a Parlay X Gateway) and invoke an application web service(s), as a result of activity in the network. The result of the invocation is used as an indication on how the call should be handled in the network.

Note that because the results of the invocations of the application web service(s) determine call handling in the network, the names of the methods are prefixed with 'handle', rather than 'notify'. The prefix 'notify' would imply a more asynchronous behaviour, whereas 'handle' shows the synchronous nature of these invocations.

The type of events (busy, answer etc.) and related numbers, for which the application web service(s) should be invoked, should be determined by the operator in an off-line process.

7.2.1 Request Application Handling of a 'Busy' Condition

a) handleBusy(EndUserIdentifier callingParty, EndUserIdentifier calledParty, out Action action)

b) Behaviour:

The invocation of **handleBusy** requests the application to inform the gateway how to handle the call between two addresses, the **callingParty** and the **calledParty**, where the **calledParty** is busy when the call is received. The application returns the **action**, which directs the gateway to perform one of the following actions:

- "Continue", resulting in normal handling of the busy event in the network, e.g. playing of a busy tone to the **callingParty**
- "EndCall", resulting in the call being terminated; the exact tone or announcement that will be played to the **callingParty** is operator-specific
- "Route", resulting in the call being re-routed to a **calledParty** specified by the application.

Optionally, in the **action** parameter, the application can also indicate the charging arrangements, i.e. the name of an operator-specific charging plan that defines who to charge for the call and how much.

c) Parameters:

NAME	TYPE	DESCRIPTION
callingParty	EndUserIdentifier	It contains the address of the caller.
calledParty	EndUserIdentifier	It contains the address of the called party. This party is busy.
action	Action	OUTPUT. It indicates the action to be performed by the gateway.

d) Exceptions:

ApplicationException

UnknownEndUserException

InvalidArgumentException

7.2.2 Request Application Handling of a 'Not Reachable' Condition

a) handleNotReachable(EndUserIdentifier callingParty, EndUserIdentifier calledParty, out Action action)

b) Behaviour:

The invocation of **handleNotReachable** requests the application to inform the gateway how to handle the call between two addresses, the **callingParty** and the **calledParty**, where the **calledParty** is not reachable when the call is received. The application returns the **action**, which directs the gateway to perform one of the following actions:

- "Continue", resulting in normal handling of the 'not reachable' event in the network, e.g. playing of a busy tone to the **callingParty**
- "EndCall", resulting in the call being terminated; the exact tone or announcement that will be played to the **callingParty** is operator-specific
- "Route", resulting in the call being re-routed to a **calledParty** specified by the application.

Optionally, in the **action** parameter, the application can also indicate the charging arrangements, i.e. the name of an operator-specific charging plan that defines who to charge for the call and how much.

c) Parameters:

NAME	TYPE	DESCRIPTION
callingParty	EndUserIdentifier	It contains the address of the caller.
calledParty	EndUserIdentifier	It contains the address of the called party. This party is not reachable.
action	Action	OUTPUT. It indicates the action to be performed by the gateway.

d) Exceptions:

ApplicationException

UnknownEndUserException

InvalidArgumentException

7.2.3 Request Application Handling of a 'No Answer' Condition

a) handleNoAnswer(EndUserIdentifier callingParty, EndUserIdentifier calledParty, out Action action)

b) Behaviour:

The invocation of **handleNoAnswer** requests the application to inform the gateway how to handle the call between two addresses, the **callingParty** and the **calledParty**, where the **calledParty** does not answer the received call. The application returns the **action**, which directs the gateway to perform one of the following actions:

- "Continue", resulting in normal handling of the 'no answer' event in the network, e.g. playing of a busy tone to the **callingParty**
- "EndCall", resulting in the call being terminated; the exact tone or announcement that will be played to the **callingParty** is operator-specific
- "Route", resulting in the call being re-routed to a **calledParty** specified by the application.

Optionally, in the **action** parameter, the application can also indicate the charging arrangements, i.e. the name of an operator-specific charging plan that defines who to charge for the call and how much.

c) Parameters:

NAME	TYPE	DESCRIPTION
callingParty	EndUserIdentifier	It contains the address of the caller.
calledParty	EndUserIdentifier	It contains the address of the called party. This party does not answer the call.
action	Action	OUTPUT. It indicates the action to be performed by the gateway.

d) Exceptions:

ApplicationException

UnknownEndUserException

InvalidArgumentException

7.2.4 Request Application Handling of a 'Called Number' Condition

a) handleCalledNumber(EndUserIdentifier callingParty, EndUserIdentifier calledParty, out Action action)

b) Behaviour:

The invocation of **handleCalledNumber** requests the application to inform the gateway how to handle the call between two addresses, the **callingParty** and the **calledParty**. The method is invoked when the **callingParty** tries to call the **calledParty**, but before the network routes the call to the **calledParty**. For example, the **calledParty** does not have to refer to a real end user, i.e., it could be a service number. The application returns the **action**, which directs the gateway to perform one of the following actions:

- "Continue", resulting in normal handling in the network, i.e. the call will be routed to the **calledParty** number, as originally dialled
- "EndCall", resulting in the call being terminated; the exact tone or announcement that will be played to the **callingParty** is operator-specific
- "Route", resulting in the call being re-routed to a **calledParty** specified by the application.

Optionally, in the **action** parameter, the application can also indicate the charging arrangements, i.e. the name of an operator-specific charging plan that defines who to charge for the call and how much.

c) Parameters:

NAME	TYPE	DESCRIPTION
callingParty	EndUserIdentifier	It contains the address of the caller.
calledParty	EndUserIdentifier	It contains the address of the called party.
action	Action	OUTPUT. It indicates the action to be performed by the gateway.

d) Exceptions:

ApplicationException

UnknownEndUserException

InvalidArgumentException

7.2.5 Request Application Handling of an 'Off Hook' Condition

a) handleOffHook(EndUserIdentifier callingParty, out Action action)

b) Behaviour:

The invocation of **handleOffHook** requests the application to inform the gateway how to handle the fact that the **callingParty** tries to initiate a call. The application returns the **action**, which directs the gateway to perform one of the following actions:

- "Continue", resulting in normal handling in the network, i.e. the calling party can enter digits and, when enough digits are entered, the call is routed based on this information
- "EndCall", resulting in the call being terminated; the exact tone or announcement that will be played to the **callingParty** is operator-specific
- "Route", resulting in the call being routed to a **calledParty** specified by the application.

Optionally, in the **action** parameter, the application can also indicate the charging arrangements, i.e. the name of an operator-specific charging plan that defines who to charge for the call and how much.

c) Parameters:

NAME	TYPE	DESCRIPTION
callingParty	EndUserIdentifier	It contains the address of the caller.
action	Action	OUTPUT. It indicates the action to be performed by the gateway.

d) Exceptions:

ApplicationException

UnknownEndUserException

InvalidArgumentException

7.3 Web Service Data Definitions

7.3.1 Data Types

In addition to the Common Data Types defined in clause 5.1, the following Data Types are specific to this Web Service.

7.3.1.1 Action

The **Action** data type is a structure containing the following parameters:

NAME	TYPE	DESCRIPTION
actionToPerform	ActionValues	Indicates the action as described below
routingAddress	EndUserIdentifier	The address to be used in case the action indicates 'Route'
charging	String	OPTIONAL. If present, represents the name of an operator-specific charging plan that defines who to charge for the call and how much. If no charge plan is specified, the charging will be based on an operator-specific charging policy.

7.3.1.2 ActionValues

The **ActionValues** data type is an enumeration with the following values:

VALUE	DESCRIPTION
Route	Request to (re-)route the call to the address indicated with routingAddress.
Continue	Request to continue the call without any changes. This will result in normal handling of the event in the network
EndCall	Request to end the call. This will result in termination of the call. The callingParty will receive a tone or announcement.

7.3.2 Exceptions

All exceptions thrown by this Web Service are Common Exceptions, as defined in clause 5.2.

7.4 Web Service Syntax – WSDL

The W3C WSDL representation of this API is contained in a set of files which accompany the present document (see Annex A).

The rpc/literal files are

- parlayx_network_initiated_call_types.xsd
- parlayx_network_initiated_call_service_port.wsdl
- parlayx_network_initiated_call_service.wsdl

The rpc/encoded file is

- parlayx_network_initiated_call_service.wsdl.

8 SMS

8.1 Overview

8.1.1 Description

Currently, in order to programmatically receive and send SMS it is necessary to write applications using specific protocols to access SMS functions provided by network elements (e.g., SMS-C). This approach requires a high degree of network expertise. Alternatively it is possible to use the Parlay/OSA approach, invoking standard interfaces (e.g., User Interaction or Messaging Service Interfaces) to gain access to SMS capabilities, but these interfaces are usually perceived to be quite complex by IT application developers. Developers must have advanced telecommunication skills to use OSA interfaces.

In this chapter we describe a Parlay X Web Service, SMS for sending and receiving SMS. The overall scope of this Web Service is to provide to application developers primitives to handle SMS in a simple way. In fact, using the SMS Web Service, application developers can invoke SMS functions without specific Telco knowledge.

For sending a message to the network (see clause 8.2 of the present document, Send SMS API), the application invokes a message to send it and must subsequently become active again to poll for delivery status. There is an alternative to this polling mechanism, i.e. an asynchronous notification mechanism implemented with an application-side web service. However it was decided not to provide a notification mechanism in the first release, to make the API as simple as possible, even though the polling mechanism is not as network efficient as the notification mechanism.

For receiving a message from the network, the application may use either polling (see clause 8.4 of the present document, Receive SMS API) or notification (see clause 8.3 of the present document, SMS Notification API) mechanisms. The notification mechanism is more common: network-initiated messages are sent to autonomous application-side web services. Both mechanisms are supported, but the provisioning of the notification-related criteria is not specified.

8.1.2 Commercial & Technical Rationale

The basic commercial rationale for developing the SMS Web Service is:

- to increase the use of SMS capabilities in software applications
- to empower traditional IT developers to produce large numbers of such applications
- to lower the development cost and time for such applications.

8.1.3 Relationship to Similar or Supplanted Specifications

Published web services exist that allow transmission of SMS messages, ring-tones and operator logos. For example, the Xmethods site (http://www.xmethods.com/) hosts more than one web service to handle SMS. These web services provide basic SMS capabilities, each using a different interfaces. The SMS Web Service aims to be a standard way to perform SMS operations and to provide more advanced features.

To specify the format of logos and ringtones, the following alternatives exist:

- 3GPP EMS format (see Note).
- Smart Messaging format.
- Other proprietary formats

NOTE: EMS (Enhanced Messaging Service) is an enhancement to SMS that provides the ability to send a combination of simple melodies, pictures, sounds, animations, modified text and standard text as an integrated message for display on an EMS compliant handset. EMS is standardized in 3GPP TS 23.040 [3] where the coding mechanisms and formats are specified.

Both the standardized EMS format and de facto Smart Messaging formats are supported. As an enhancement to SMS for sending content, dedicated methods are proposed taking into account the different content formats applied on the market.

8.1.4 Scenarios

Figure 3 shows a scenario using the SMS Web Service to send an SMS message from an application. The application invokes a web service to retrieve a weather forecast for a subscriber (1) & (2) and a Parlay X Interface (3) to use the SMS Web Service operations (i.e. to send an SMS). After invocation, the SMS Web Service invokes a Parlay API method (4) using the Parlay/OSA SCS-SMS (User Interaction) interface. This SCS handles the invocation and sends an UCP operation (5) to an SMS-C. Subsequently the weather forecast is delivered (6) to the subscriber.

In an alternative scenario, the Parlay API interaction involving steps (4) and (5) could be replaced with a direct interaction between the SMS Web Service and the Mobile network.

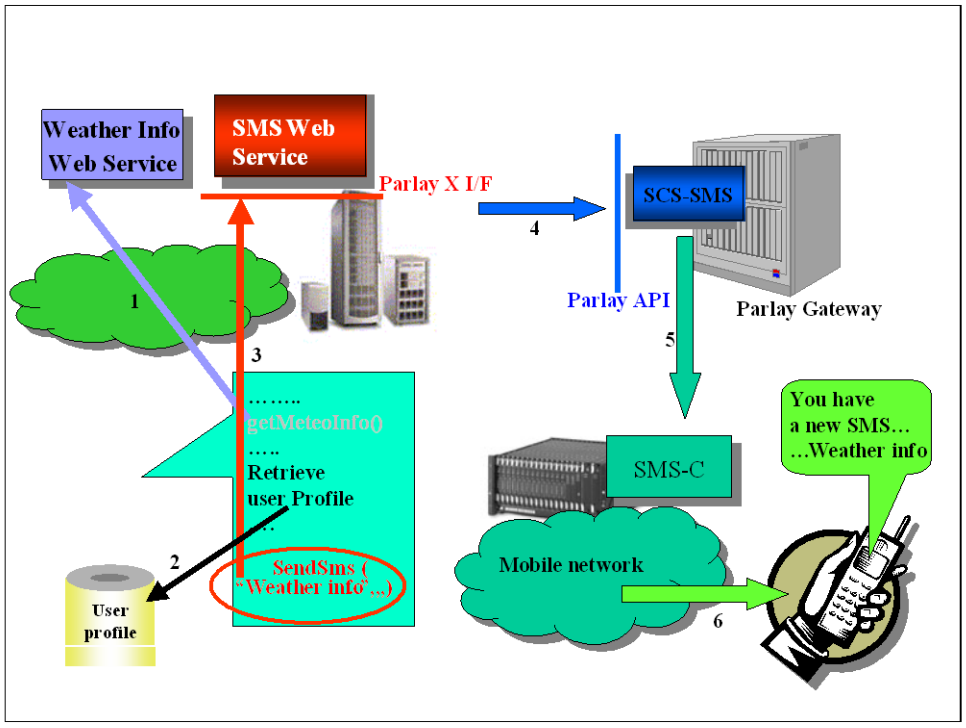

Figure 3: Send SMS Scenario

Figure 4 shows a scenario using the SMS Web Service to deliver a received SMS message to an application. The application receives a Parlay X web service invocation to retrieve an SMS sent by a subscriber (1) & (2). The SMS message contains the e-mail address of the person the user wishes to call. The application invokes a Parlay X Interface (3) to the Third Party Call Web Service in order to initiate the call (4).

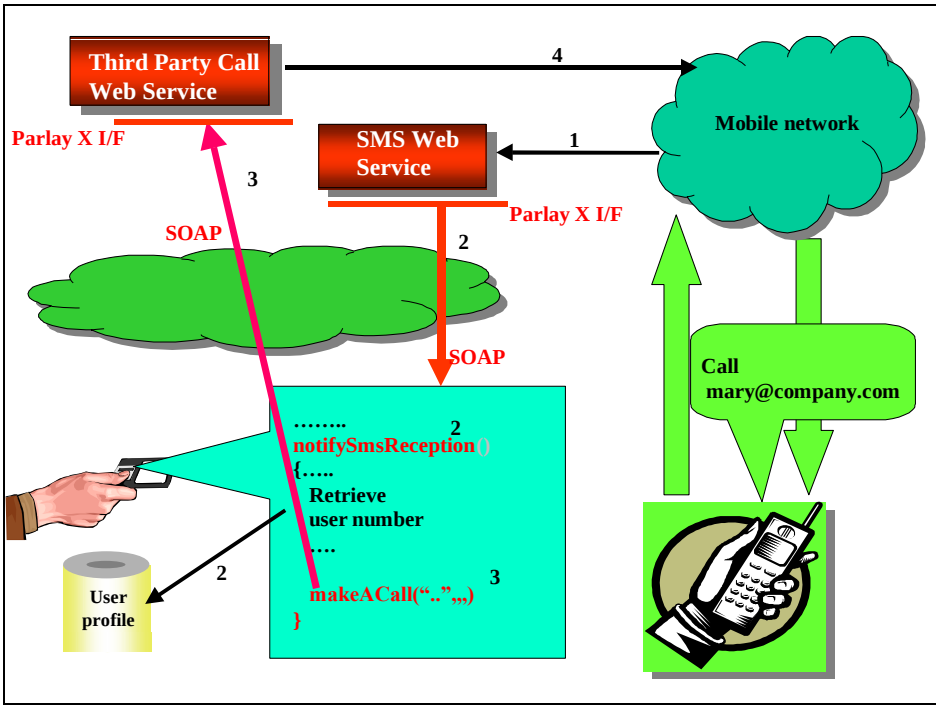

Figure 4: Receive SMS Scenario

8.2 Send SMS API

This subclause describes an initial set of capabilities in terms of message invocations, parameters and data types. The message-based invocations (i.e. of the SMS Web Service by the application) are:

- sendSms
- sendSmsLogo
- sendSmsRingtone
- getSmsDeliveryStatus.

8.2.1 Send an SMS Message

a) sendSms (EndUserIdentifier[] destinationAddressSet, String senderName, String charging, String message, out String requestIdentifier)

b) Behaviour:

The invocation of **sendSms** requests to send an SMS, specified by the String **message** to the specified address (or address set), specified by **destinationAddressSet**. Optionally the application can also indicate the sender name (**senderName**), i.e. the string that is displayed on the user's terminal as the originator of the message, and the charging arrangements (**charging**), i.e. the name of an operator-specific charging plan that defines who to charge for the SMS and how much. By invoking this operation the application requires to receive the notification of the status of the SMS delivery. In order to receive this information the application has to explicitly invoke the **getSmsDeliveryStatus**. The **requestIdentifier**, returned by the invocation, can be used to identify the SMS delivery request.

For GSM systems, if **message** contains characters not in the GSM 7-bit character set, the SMS is sent as a Unicode SMS.

If **message** is longer than the maximum supported length (e.g. for GSM, 160 GSM 7-bit characters or 70 Unicode characters), the message will be sent as several concatenated short messages.

c) Parameters:

NAME	TYPE	DESCRIPTION
destinationAddressSet	Array of EndUserIdentifier	Addresses to which the SMS will be sent
senderName	String	If present, it indicates the SMS sender name, i.e. the string that is displayed on the user's terminal as the originator of the message.
charging	String	OPTIONAL. If present, represents the name of an operator-specific charging plan that defines who to charge for the SMS and how much. If the named charge plan does not exist, the InvalidArgumentException is thrown. If no charge plan is specified, the sending service/application will be charged, based on an operator-specific charging policy.
message	String	Text to be sent in SMS
requestIdentifier	String	OUTPUT. It identifies a specific SMS delivery request

d) Exceptions:

UnknownEndUserException

InvalidArgumentException

ServiceException

MessageTooLongException

PolicyException

8.2.2 Send an SMS Logo

a) sendSmsLogo(EndUserIdentifier[] destinationAddressSet, String senderName, String charging, Base64Binary image, SmsFormat smsFormat, out String requestIdentifier)

b) Behaviour:

The invocation of **sendSmsLogo** requests to send an SMS logo, specified by the byte array **image** to the specified address (or address set), specified by **destinationAddressSet**. Optionally the application can also indicate the sender name (**senderName**), i.e. the string that is displayed on the user's terminal as the originator of the message, and the charging arrangements (**charging**), i.e. the name of an operator-specific charging plan that defines who to charge for the SMS logo and how much. By invoking this operation the application requires to receive the notification of the status of the SMS delivery. In order to receive this information the application has to explicitly invoke the **getSmsDeliveryStatus**. The **requestIdentifier**, returned by the invocation, can be used to identify the SMS delivery request.

c) Parameters:

NAME	TYPE	DESCRIPTION
destinationAddressSet	Array of EndUserIdentifier	Addresses to which the SMS logo will be sent
senderName	String	If present, it indicates SMS sender name, i.e. the string that is displayed on the user's terminal as the originator of the message.
charging	String	OPTIONAL. If present, represents the name of an operator-specific charging plan that defines who to charge for the SMS logo and how much. If the named charge plan does not exist, the InvalidArgumentException is thrown. If no charge plan is specified, the sending service/application will be charged, based on an operator-specific charging policy.
image	Base64Binary	The image in jpeg, gif or png format. The image will be scaled to the proper format.
smsFormat	SmsFormat	Possible values are: 'Ems', 'SmartMessaging'.
requestIdentifier	String	OUTPUT. It identifies a specific SMS delivery request

d) Exceptions:

UnknownEndUserException

InvalidArgumentException

MessageTooLongException

UnsupportedFormatException

ServiceException

PolicyException

8.2.3 Send an SMS Ringtone

a) sendSmsRingtone(EndUserIdentifier[] destinationAddressSet, String senderName, String charging, String ringtone, SmsFormat smsFormat, out String requestIdentifier)

b) Behaviour:

The invocation of **sendSmsRingtone** requests to send an SMS ringtone, specified by the String **ringtone** (in RTX format) to the specified address (or address set), specified by **destinationAddressSet**. Optionally the application can also indicate the sender name (**senderName**) i.e. the string that is displayed on the user's terminal as the originator of the message, and the charging arrangements (**charging**), i.e. the name of an operator-specific charging plan that defines

who to charge for the SMS ringtone and how much. By invoking this operation the application requires to receive the notification of the status of the SMS delivery. In order to receive this information the application has to explicitly invoke the **getSmsDeliveryStatus**. The **requestIdentifier**, returned by the invocation, can be used to identify the SMS delivery request.

Depending on the length of the ringtone, it may be sent as several concatenated short messages.

c) Parameters:

NAME	TYPE	DESCRIPTION
destinationAddressSet	Array of EndUserIdentifier	Addresses to which the SMS ringtone will be sent
senderName	String	If present, it indicates SMS sender name, i.e. the string that is displayed on the user's terminal as the originator of the message.
charging	String	OPTIONAL. If present, represents the name of an operator-specific charging plan that defines who to charge for the SMS ringtone and how much. If the named charge plan does not exist, the InvalidArgumentException is thrown. If no charge plan is specified, the sending service/application will be charged, based on an operator-specific charging policy.
ringtone	String	The ringtone in RTX format (see Note). (http://www.logomanager.co.uk/help/Edit/RTX.html)
smsFormat	SmsFormat	Possible values are: 'Ems', 'SmartMessaging'.
requestIdentifier	String	OUTPUT. It identifies a specific SMS delivery request
NOTE: RTX Ringtone Specification : An RTX file is a text file, containing the ringtone name, a control subclause and a subclause containing a comma separated sequence of ring tone commands.		

d) Exceptions:

UnknownEndUserException

InvalidArgumentException

UnsupportedFormatException

MessageTooLongException

ServiceException

PolicyException

8.2.4 Get Current Status of an SMS Delivery

a) getSmsDeliveryStatus(String requestIdentifier, out DeliveryStatusType[] deliveryStatus)

b) Behaviour:

The invocation of **getSmsDeliveryStatus** requests the status of a previous SMS delivery request identified by **requestIdentifier**. The information on the status is returned in **deliveryStatus**, which is an array of status related to the request identified by **requestIdentifier**. The status is identified by a couplet indicating a user address and the associated delivery status. This method can be invoked multiple times by the application even if the status has reached a final value. However, after the status has reached a final value, status information will be available only for a limited period of time that should be specified in an off-line configuration step. The following four different SMS delivery status have been identified:

- 'Delivered': in case of concatenated messages, only when all the SMS-parts have been successfully delivered.
- 'Delivery Uncertain': e.g. because it was handed off to another network.
- 'Delivery Impossible': unsuccessful delivery; the message could not be delivered before it expired.

- 'MessageWaiting': the message is still queued for delivery.

c) Parameters:

NAME	TYPE	DESCRIPTION
requestIdentifier	String	It identifies a specific SMS delivery request
deliveryStatus	Array of DeliveryStatusType	OUTPUT. It lists the variations on the delivery status of the SMS

d) Exceptions:

UnknownRequestIdentifierException

ServiceException

8.3 SMS Notification API

This subclause describes an initial set of capabilities in terms of message invocations, parameters and data types. The message-based invocations (i.e. of a notification web service by the SMS Web Service) are:

- notifySmsReception

8.3.1 Notify Application of an SMS Message Sent to a Specific Address

a) notifySmsReception(String registrationIdentifier, String smsServiceActivationNumber, EndUserIdentifier senderAddress, String message)

b) Behaviour:

The **notifySmsReception** method must be implemented by a Web Service at the ***application side***. It will be invoked by the Parlay X server to notify the application of the reception of an SMS. The notification will occur if and only if the SMS received fulfils the criteria specified in an off-line provisioning step, identified by the **registrationIdentifier**. The criteria must at least include an **smsServiceActivationNumber**, i.e. the SMS destination address that can be "monitored" by the application. The parameter **senderAddress** contains the address of the sender. The application can apply the appropriate service logic to process the SMS.

c) Parameters:

NAME	TYPE	DESCRIPTION
registrationIdentifier	String	Identifies the off-line provisioning step that enables the application to receive notification of SMS reception according to specified criteria.
smsServiceActivation Number	String	Number associated with the invoked Message service, i.e. the destination address used by the terminal to send the message.
senderAddress	EndUserIdentifier	It indicates the address sending the SMS
message	String	Text received in the SMS

d) Exceptions:

ApplicationException

8.4 Receive SMS API

This subclause describes an initial set of capabilities in terms of message invocations, parameters and data types. The message-based invocations (i.e. of the SMS Web Service by the application) are:

- getReceivedSms.

8.4.1 Retrieve All SMS Messages Sent to a Specific Address

a) getReceivedSms(String registrationIdentifier, out SmsType[] receivedSms)

b) Behaviour:

The invocation of **getReceivedSms** retrieves all the SMS messages received that fulfil the criteria identified by **registrationIdentifier**. The method returns only the list of SMS messages received since the previous invocation of the same method, i.e. each time the method is executed the messages returned are removed from the server. Moreover, each SMS message will be automatically removed from the server after a maximum time interval specified in an off-line configuration step.

The received SMS messages are returned in **receivedSms**. An SMS message is identified by a structure indicating the sender of the SMS message and the content.

c) Parameters:

NAME	TYPE	DESCRIPTION
registrationIdentifier	String	Identifies the off-line provisioning step that enables the application to receive notification of SMS reception according to specified criteria.
receivedSms	Array of SmsType	OUTPUT. It lists the received SMS since last invocation.

d) Exceptions:

UnknownRegistrationIdentifierException

ServiceException

8.5 Web Service Data Definitions

8.5.1 Data Types

In addition to the Common Data Types defined in clause 5.1, the following Data Types are specific to this Web Service.

8.5.1.1 DeliveryStatusType

The **DeliveryStatusType** data type is a structure containing the following parameters:

NAME	TYPE	DESCRIPTION
destinationAddress	EndUserIdentifier	It indicates the destination address to which the notification is related
deliveryStatus	DeliveryStatus	Indicates the delivery result for destinationAddress. Possible values are: 'Delivered', 'DeliveryUncertain', 'DeliveryImpossible'.

8.5.1.2 DeliveryStatus

The **DeliveryStatus** data type is an enumeration with the following values:

VALUE	DESCRIPTION
Delivered	Successful delivery
DeliveryUncertain	Delivery status unknown: e.g. because it was handed off to another network.
DeliveryImpossible	Unsuccessful delivery; the message could not be delivered before it expired.
MessageWaiting	The message is still queued for delivery. This is a temporary state, pending transition to one of the preceding states.

8.5.1.3 SmsType

The **SmsType** data type is a structure containing the following parameters:

NAME	TYPE	DESCRIPTION
message	String	Text received in SMS
senderAddress	EndUserIdentifier	It indicates address sending the SMS

8.5.1.4 SmsFormat

The **SmsFormat** data type is an enumeration with the following values:

VALUE	DESCRIPTION
Ems	**Enhanced Messaging Service**, standardized in 3GPP TS 23.040 [3], which defines a logo/ringtone format
SmartMessaging™	Defines a logo/ringtone format

8.5.2 Exceptions

In addition to the Common Exceptions defined in clause 5.2, there are exceptions specific to this Web Service. Similar to the Common Exceptions, each Web Service-specific exception is assigned an eight-character identifier. This identifier is interpreted as described in clause 5.2, except that the first 3 characters uniquely identify this Web Service.

The following exceptions are specific to this Web Service:

UNIQUE ID	TEXT STRING	MEANING
SMS1000E	UnsupportedFormatException	The smsFormat supplied is not one of the permitted values of the SmsFormat data type.
SMS1001E	UnknownRegistration IdentifierException	The registrationIdentifier supplied is not known by the server
SMS1002E	UnknownRequestIdentifier Exception	The requestIdentifier supplied does not relate to any known SMS request or has expired.

8.6 Web Service Syntax – WSDL

The W3C WSDL representation of this API is contained in a set of files which accompany the present document (see Annex A).

The rpc/literal files are

- parlayx_sms_types.xsd
- parlayx_sms_service_port.wsdl
- parlayx_sms_send_service.wsdl
- parlayx_sms_receive_service.wsdl
- parlayx_sms_notification_service_port.wsdl
- parlayx_sms_notification_service.wsdl

The rpc/encoded files are

- parlayx_sms_service.wsdl
- parlayx_sms_notification_service.wsdl

9 Multimedia Message

9.1 Overview

9.1.1 Description

Currently, in order to programmatically receive and send Multimedia Messages, it is necessary to write applications using specific protocols to access MMS functions provided by network elements (e.g., MMS-C). This approach requires application developers to have a high degree of network expertise.

This contribution defines a Multimedia Message Web Service that can map to SMS, EMS, MMS, IM, E-mail etc.

The choice is between defining one set of APIs per messaging network or a single set common to all networks; e.g. we could define sendMMS, sendEMS, sendSMS, … or just use sendMessage. Although the more specific the API the easier it is to use, there are advantages to a single set of network-neutral APIs. These advantages include:

- improved service portability
- lower complexity, by providing support for generic user terminal capabilities only.

For this version of the Parlay X specification, we provide sets of APIs for two messaging web services: SMS-specific APIs (as described in clause 8) and Multimedia Message APIs (this clause), which provides generic messaging features (including SMS).

For sending a message to the network (see clause 9.2 of the present document, Send Message API), the application invokes a message to send it and must subsequently become active again to poll for delivery status. There is an alternative to this polling mechanism, i.e. an asynchronous notification mechanism implemented with an application-side web service. However it was decided not to provide a notification mechanism in the first release, to make the API as simple as possible, even though the polling mechanism is not as network efficient as the notification mechanism.

For receiving a message from the network, the application may use either polling (see clause 9.3 of the present document, Receive Message API) or notification (see clause 9.4 of the present document, Message Notification API) mechanisms. The notification mechanism is more common: network-initiated messages are sent to autonomous application-side web services. Both mechanisms are supported, but the provisioning of the notification-related criteria is not specified.

9.1.2 Commercial & Technical Rationale

The scope of this Web Service is much more than an enhancement of the Parlay X SMS Web Service. The purpose is not to add more SMS features, but to form a generic multimedia adapted messaging API. The reason to incorporate SMS in the API is mainly to create one set of APIs for messaging instead of one set per network. We believe that the benefits of a single set of APIs, i.e. service portability and the ability to serve different handsets or even multiple sub-networks using common APIs, is highly beneficial for both the Network Operators and the Service Providers.

9.1.3 Relationship to Similar or Supplanted Specifications

This Web Service includes functions implemented in the SMS Web Service.

9.1.4 Scenarios

Figure 5 shows an example scenario using sendMessage and getMessageDeliveryStatus to send data to subscribers and to determine if the data has been received by the subscriber. The application invokes a web service to retrieve a stock quote (1) & (2) and sends the current quote - sendMessage - using the Parlay X Interface (3) of the Multimedia Message Web Service. After invocation, the Multimedia Message Web Service sends the message to an MMS-C using the MM7 interface (4) for onward transmission (5) to the subscriber over the Mobile network

Later, when the next quote is ready, the application checks to see - getMessageDeliveryStatus - if the previous quote has been successfully delivered to the subscriber. If not, it may for instance perform an action (not shown) to provide a

credit for the previous message transmission. This way, the subscriber is only charged for a stock quote if it is delivered on time.

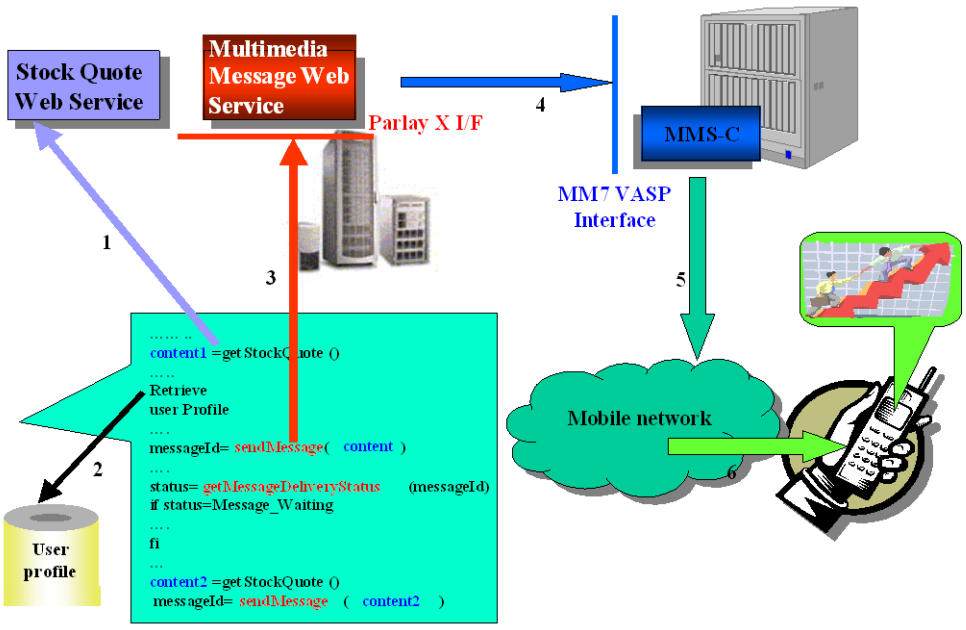

Figure 5: Multimedia Messaging Scenario

9.2 Send Message API

This subclause describes an initial set of capabilities in terms of message invocations, parameters and data types. The message-based invocations (i.e. of the Multimedia Message Web Service by the application) are:

- sendMessage
- getMessageDeliveryStatus

9.2.1 Send a Multimedia Message

a) **sendMessage(EndUserIdentifier [] destinationAddressSet, String senderAddress, String subject, MessagePriority priority, String charging, out String requestIdentifier) Attachment[content]**

b) **Behaviour:**

Request to send a Message to a set of destination addresses, returning a **requestIdentifier** to identify the message. The **requestIdentifier** can subsequently be used by the application to poll for the message status, i.e. using **getMessageDeliveryStatus** to see if the message has been delivered or not. The content is sent as a SOAP-Attachment (see note) encoded using MIME or DIME.

NOTE: SOAP-Attachment is used because specification of the WS-Attachments standard is not yet complete. The decision to use SOAP-Attachment may be revisited in future releases. Please refer to your SOAP/WSDL toolkit documentation for information on populating or retrieving a SOAP-Attachment.

c) Parameters:

NAME	TYPE	DESCRIPTION
destinationAddressSet	Array of EndUserIdentifier	Destination addresses for the Message.
senderAddress	String	OPTIONAL. If present, indicates Message sender address. This parameter is not allowed for all 3rd party providers. Parlay X server needs to handle this according to a SLA for the specific application and its use can therefore result in a PolicyException.
subject	String	OPTIONAL. If present, it indicates the message subject. If mapped to SMS this parameter will be used as the senderAddress, even if a separate senderAddress is provided.
priority	MessagePriority	OPTIONAL. If present, represents the priority of the message. If not defined, the network will assign a priority based on an operator policy.
charging	String	OPTIONAL. If present, represents the name of an operator-specific charging plan that defines who to charge for the message and how much. If the named charge plan does not exist, the InvalidArgumentException is thrown. If no charge plan is specified, the sending service/application will be charged, based on operator-specific charging policy.
requestIdentifier	String	OUTPUT. It is a correlation identifier that is used in a **getMessageDeliveryStatus** message invocation, i.e. to poll for the delivery status of all of the sent Messages.
Input Attachments		
content	MIME or DIME format	Data to be sent with Message, i.e. in MIME or DIME format and sent as a SOAP-Attachment

d) Exceptions

UnknownEndUserException

InvalidArgumentException

ServiceException

PolicyException

MessageTooLongException

9.2.2 Get Current Status of a Multimedia Message Delivery

a) getMessageDeliveryStatus(String requestIdentifier, out DeliveryStatusType[] deliveryStatus)

b) Behaviour:

This is a poll method used by the application to retrieve delivery status for each message sent as a result of a previous **sendMessage** message invocation. The **requestIdentifier** parameter identifies this previous message invocation.

c) Parameters:

NAME	TYPE	DESCRIPTION
requestIdentifier	String	Identifier related to the delivery status request.
deliveryStatus	Array of DeliveryStatusType	OUTPUT. It is an array of status of the messages that were previously sent. Each array element represents a sent message: i.e. its destination address and its delivery status.

d) Exceptions

InvalidArgumentException

ServiceException

Policy Exception

UnknownRequestIdentifierException

9.3 Receive Message API

This subclause describes an initial set of capabilities in terms of message invocations, parameters and data types. The message-based invocations (i.e. of the Multimedia Message Web Service by the application) are:

- getReceivedMessages
- getMessageURIs
- getMessage

9.3.1 Provide Application with Multimedia Messages Sent to the Application

a) getReceivedMessages(String registrationIdentifier, MessagePriority priority, out MessageRef [] messageRef)

b) Behaviour:

This method enables the application to poll for new messages associated with a specific **registrationIdentifier**. If the **registrationIdentifier** is not specified, the Parlay X server will return references to all messages sent to the application. The process of binding different **registrationIdentifier** parameters to applications is an off-line process. The Parlay X gateway shall not allow an application to poll for messages using **registrationIdentifier** parameters that are not associated with the application. The priority parameter may be used by the application to retrieve references to higher priority messages, e.g. if Normal is chosen only references to high priority and normal priority messages are returned. If the priority parameter is omitted all message references are returned.

c) Parameters:

NAME	TYPE	DESCRIPTION
registrationIdentifier	String	Identifies the off-line provisioning step that enables the application to receive notification of Message reception according to specified criteria.
priority	MessagePriority	OPTIONAL. The priority of the messages to poll from the Parlay X gateway. All messages of the specified priority and higher will be retrieved. If not specified, all messages shall be returned, i.e. the same as specifying Low.
messageRef	Array of MessageRef	OUTPUT. It contains an array of messages received according to the specified filter of **registrationIdentifier** and **priority**.

d) Exceptions

InvalidArgumentException

ServiceException

PolicyException

UnknownRegistrationIdentifierException

9.3.2 Retrieve URI References to the Parts of a Multimedia Message

getMessageURIs(String messageRefIdentifier, out MessageURI message)

b) Behaviour:

This method will read the different parts of the message, create local files in the Parlay Gateway and return URI references to them. The application can then simply read each file or just have them presented as links to the end-user. The URIs to the files will be active for an agreed time.

c) Parameters:

NAME	TYPE	DESCRIPTION
messageRefIdentifier	String	The identity of the message to retrieve.
message	MessageURI	OUTPUT. It contains the complete message, i.e. the textual part of the message, if such exists, and a list of file references for the message attachments, if any.

d) Exceptions

InvalidArgumentException

ServiceException

PolicyException

UnknownMessageException

9.3.3 Provide Application with a Multimedia Message as an Attachment

getMessage(String messageRefIdentifier) Attachments[out content]

b) Behaviour:

This method will read the whole message. The data is returned as a SOAP-Attachment (see note) in the return message.

NOTE: SOAP-Attachment is used because specification of the WS-Attachments standard is not yet complete. The decision to use SOAP-Attachment may be revisited in future releases. Please refer to your SOAP/WSDL toolkit documentation for information on populating or retrieving a SOAP-Attachment.

c) Parameters:

NAME	TYPE	DESCRIPTION
messageRefIdentifier	String	The identity of the message
Output Attachments		
content	MIME or DIME format	Data to be returned with Message, i.e. in MIME or DIME format and received as a SOAP-Attachment

d) Exceptions

InvalidArgumentException

ServiceException

PolicyException

UnknownMessageException

9.4 Message Notification API

This subclause describes an initial set of capabilities in terms of message invocations, parameters and data types. The message-based invocations (i.e. of a notification web service by the Multimedia Message Web Service) are:

- notifyMessageReception

9.4.1 Notify Application of a Multimedia Message Sent to a Specific Address

a) notifyMessageReception (String registrationIdentifier, MessageRef messageRef)

b) Behaviour:

This method will have to be implemented by a web service on the client application side. The registration of the URI for this application web service is done off-line. This means that there is a registration mechanism in the Parlay X Gateway that binds different **registrationIdentifier** parameters to applications and their web service URIs.

A client application is notified that a new Message, sent to a specific Service Activation Number, has been received. Using the **registrationIdentifier**, the client application can apply appropriate service logic with specific behaviour.

c) Parameters:

NAME	TYPE	DESCRIPTION
registrationIdentifier	String	A handle connected to the off-line registration of the notifications. This distinguishes registrations that point to the same application web service.
messageRef	MessageRef	This parameter contains all the information associated with the received message.

d) Exceptions

ApplicationException

9.5 Web Service Data Definitions

9.5.1 Data Types

In addition to the Common Data Types defined in clause 5.1, the following Data Types are specific to this Web Service.

9.5.1.1 MessagePriority

The **MessagePriority** data type is an enumeration with the following values:

VALUE	DESCRIPTION
Default	This is the "NULL" value as described in subclause 4.3. This value is applicable if the parameter of type MessagePriority is tagged OPTIONAL.
Low	Low message priority
Normal	Normal message priority
High	High message priority

9.5.1.2 DeliveryStatus

The **DeliveryStatus** data type is an enumeration with the following values:

VALUE	DESCRIPTION
Delivered	Successful delivery
DeliveryUncertain	Delivery status unknown: e.g. because it was handed off to another network.
DeliveryImpossible	Unsuccessful delivery; the message could not be delivered before it expired.
MessageWaiting	The message is still queued for delivery. This is a temporary state, pending transition to one of the preceding states.

9.5.1.3 DeliveryStatusType

The **DeliveryStatusType** data type is a structure containing the following parameters:

NAME	TYPE	DESCRIPTION
destinationAddress	EndUserIdentifier	Address associated with the delivery status. The address field is coded as a URI.
deliveryStatus	DeliveryStatus	Parameter indicating the delivery status.

9.5.1.4 MessageRef

The **MessageRef** data type is a structure containing six parameters as described below.

This data type is used to return the basic message data when polled by the application.

- If a message is a pure text message, the content will be returned in the **message** parameter and the **messageRefIdentifier** parameter will then not be sent.
- If a message contains attachments or other non-text messages the **message** parameter will not be sent; instead the **messageRefIdentifier** will contain a reference to the message stored in the Parlay X gateway.

NAME	TYPE	DESCRIPTION
messageRefIdentifier	String	OPTIONAL: If present, contains a reference to a message stored in the Parlay X gateway. If the message is pure text, this parameter is not present.
messageService ActivationNumber	String	Number associated with the invoked Message service, i.e. the destination address used by the terminal to send the message.
senderAddress	EndUserIdentifier	Indicates message sender address
subject	String	OPTIONAL: If present, indicates the subject of the received message. This parameter will not be used for SMS services.
priority	MessagePriority	The priority of the message: default is Normal
message	String	OPTIONAL: If present, then the **messageRefIdentifier** is not present and this parameter contains the whole message. The type of the message is always pure ASCII text in this case. The message will not be stored in the Parlay X gateway.

9.5.1.5 MessageURI

The **MessageURI** data type is a structure containing the following parameters:

NAME	TYPE	DESCRIPTION
bodyText	String	Contains the message body if it is encoded as ASCII text.
fileReferences	Array of URI	This is an array of URI references to all the attachments in the Multimedia message. These are URIs to different files, e.g. GIF pictures or pure text files.

9.5.2 Exceptions

In addition to the Common Exceptions defined in clause 5.2, there are exceptions specific to this Web Service. Similar to the Common Exceptions, each Web Service-specific exception is assigned an eight-character identifier. This identifier is interpreted as described in clause 5.2, except that the first 3 characters uniquely identify this Web Service.

The following exceptions are specific to this Web Service:

UNIQUE ID	TEXT STRING	MEANING
MMM1000E	UnknownRequestIdentifier Exception	The Parlay X gateway did not recognize the **requestIdentifier** parameter. The message may have timed out or may have never been sent. This fault includes a string that provides additional information
MMM1001E	UnknownRegistration IdentifierException	The provided registration identifier does not exist. This fault includes a string that provides additional information.
MMM1002E	UnknownMessageException	The provided **messageRefIdentifier** was not found in the Parlay X gateway. The message may have been timed out or it may never have been received by the gateway. This fault includes a string that provides additional information

9.6 Web Service Syntax – WSDL

The W3C WSDL representation of this API is contained in a set of files which accompany the present document (see Annex A).

The rpc/literal files are

- parlayx_mm_types.xsd
- parlayx_mm_service_port.wsdl
- parlayx_mm_send_service.wsdl
- parlayx_mm_receive_service.wsdl
- parlayx_mm_notification_service_port.wsdl
- parlayx_mm_notification_service.wsdl

The rpc/encoded file is

- parlayx_mm_service.wsdl
- parlayx_mm_notification_service.wsdl

10 Payment

10.1 Overview

10.1.1 Description

A vast amount of content, both information and entertainment, will be made available to subscribers. To support a business model that enables operators to offer integrated billing, a payment API is crucial. Open and inter-operable "payment APIs" are the key to market growth and investment protection. The Payment Web Service supports payments for any content in an open, Web-like environment.

The Payment Web Service described in this document supports payment reservation, pre-paid payments, and post-paid payments. It supports charging of both volume and currency amounts, a conversion function and a settlement function in case of a financially resolved dispute.

Note that certain parameters are negotiated off line. For example the currency, volume type, default reservation enforcement time, as well as the taxation procedures and parameters.

10.1.2 Commercial & Technical Rationale

The payment process is a complex and critical component of telecom networks. The Payment Web Service:

- further enlarges the market for third party software, services and content by providing essential primitives for integrated billing
- enables charging through trusted and authenticated service or content providers
- allows charging by subscriber address (e.g. MSISDN), against an existing reservation or directly against an account.

10.1.3 Relationship to Similar or Supplanted Specifications

Payment APIs are the focus of a number of industry standardisation initiatives:

- Parlay/OSA have developed powerful, carrier-grade Content-Based Charging (CBC) APIs
- PayCircle intends to provide a full, hand-crafted XML version of the Parlay/OSA CBC APIs
- Within the JCP, the JPay expert group seeks to specify a local Java API that can be implemented on top of either Parlay/OSA CBC, Parlay X Payment Web Service, or PayCircle's Payment Web Service
- OMA is commencing an e-commerce initiative.

The Payment Web Service provides a simpler, higher-level, WSDL-defined interface consistent with the Parlay X philosophy.

10.1.4 Scenarios

This subclause discusses two scenarios: one where the subscriber account is directly charged and one where a subscriber uses a service for which the provider needs a reservation. Note, associated Payment API messages are shown in 'bold' format: e.g. (**chargeAmount**).

10.1.4.1 Scenario Number 1

Assume a subscriber is interested in downloading a ring tone to his mobile (MS). The subscriber selects a ring tone and establishes a trusted relation with the ring tone provider. Essentially, the ring tone provider obtains the address (MSISDN) and other information from the subscriber. The ring tone may be downloaded to the MS using SMS. As soon as the download succeeds, the provider of the ring tone will charge the subscriber (**chargeAmount)**.

10.1.4.2 Scenario Number 2

Assume a subscriber is interested in receiving a stream of, say, a soccer match. The subscriber selects a match and establishes a trusted relation with the provider. Again, the provider obtains the MSISDN and other information from the subscriber. The subscriber wants to know what the service will cost and the provider interacts with the operators rating engine (**getAmount**) taking into account the subscriber's subscription, time of day, etc. The value returned is a currency amount and is printed on the page that is displayed at the MS. The subscriber then decides to stream the match to his MS. Subsequently, the provider will reserve the appropriate amount with the operator (**reserveAmount**) to ensure that the subscriber can fulfil his payment obligations. The match starts and the provider periodically charges against the reservation (**chargeReservation**). The match ends in a draw and is extended with a 'sudden death' phase. The subscriber continues listening, so the existing reservation is enlarged (**reserveAdditionalAmount**). Suddenly, one of the teams scores a goal, so the match abruptly ends, leaving part of the reserved amount unused. The provider now releases the reservation (**releaseReservation**), and the remaining amount is available for future use by the subscriber.

Now we can extend the second scenario by having the subscriber participate in a game of chance in which the provider refunds a percentage of the usage costs (**refundAmount**) based on the ranking of a particular team in this tournament. For example, the subscriber gambling on the team that wins the tournament receives a full refund, while for gambling on the team that finishes in second place, the refund is 50%, etc.

10.2 Amount Charging API

This subclause describes an initial set of capabilities in terms of message invocations, parameters and data types. The message-based invocations are:

- chargeAmount

refundAmount

10.2.1 Charge Currency Amount to an Account

a) chargeAmount(EndUserIdentifier endUserIdentifier, Decimal amount, String billingText, String referenceCode)

b) Behaviour:

This message results in directly charging to the account indicated by the end user identifier. The charge is specified as a currency amount. The billing text field is used for textual information to appear on the bill. The reference code is used to uniquely identify the request; it is the application's responsibility to provide a unique reference code within the scope of the application.

c) Parameters:

NAME	TYPE	DESCRIPTION
endUserIdentifier	EndUserIdentifier	The end user's account to be charged
amount	Decimal	The currency amount of the charge
billingText	String	Textual information to appear on the bill
referenceCode	String	Textual information to uniquely identify the request, e.g. in case of disputes

d) Exceptions:

UnknownEndUserException

InvalidArgumentException

ChargeFailureException

10.2.2 Refund Currency Amount to an Account

a) refundAmount(EndUserIdentifier endUserIdentifier, Decimal amount, String billingText, String referenceCode)

b) Behaviour:

This message results in directly applying a refund to the account indicated by the end user identifier. The refund is specified as a currency amount. The billing text field is used for textual information to appear on the bill. The reference code is used to uniquely identify the request; it is the application's responsibility to provide a unique reference code within the scope of the application.

c) Parameters:

NAME	TYPE	DESCRIPTION
endUserIdentifier	EndUserIdentifier	The end user's account to be refunded
amount	Decimal	The currency amount of the refund
billingText	String	Textual information to appear on the bill
referenceCode	String	Textual information to uniquely identify the request, e.g. in case of disputes

d) Exceptions:

UnknownEndUserException

InvalidArgumentException

ChargeFailureException

10.3 Volume Charging API

This subclause describes an initial set of capabilities in terms of message invocations, parameters and data types. The message-based invocations are:

- chargeVolume
- getAmount
- refundVolume

10.3.1 Charge Volume to an Account

a) chargeVolume(EndUserIdentifier endUserIdentifier, Long volume, String billingText, String referenceCode)

b) Behaviour:

This message results in directly charging to the account indicated by the end user identifier. The charge is specified as a volume. The billing text field is used for textual information to appear on the bill. The reference code is used to uniquely identify the request; it is the application's responsibility to provide a unique reference code within the scope of the application.

c) Parameters:

NAME	TYPE	DESCRIPTION
endUserIdentifier	EndUserIdentifier	The end user's account to be charged
volume	Long	The volume to be charged
billingText	String	Textual information to appear on the bill
referenceCode	String	Textual information to uniquely identify the request, e.g. in case of disputes

d) Exceptions:

UnknownEndUserException

InvalidArgumentException

ChargeFailureException

10.3.2 Convert a Volume to a Currency Amount

a) getAmount(EndUserIdentifier endUserIdentifier, Long volume, out Decimal amount)

b) Behaviour:

This message results in converting the given volume to a currency amount. The end user identifier is given to indicate the subscriber for whom this conversion calculation must be made. The message returns a currency amount if successful.

c) Parameters:

NAME	TYPE	DESCRIPTION
endUserIdentifier	EndUserIdentifier	The end user's account to be charged
volume	Long	The volume to be converted
amount	Decimal	OUTPUT. It is the currency amount resulting from the conversion process

d) Exceptions:

UnknownEndUserException

InvalidArgumentException

ServiceException

10.3.3 Refund Volume to an Account

a) refundVolume(EndUserIdentifier endUserIdentifier, Long volume, String billingText, String referenceCode)

b) Behaviour:

This message results in directly applying a refund to the account indicated by the end user identifier. The refund is specified as a volume. The billing text field is used for textual information to appear on the bill. The reference code is used to uniquely identify the request; it is the application's responsibility to provide a unique reference code within the scope of the application.

c) Parameters:

NAME	TYPE	DESCRIPTION
endUserIdentifier	EndUserIdentifier	The end user's account to be refunded
volume	Long	The volume to be refunded
billingText	String	Textual information to appear on the bill
referenceCode	String	Textual information to uniquely identify the request, e.g. in case of disputes

d) Exceptions:

UnknownEndUserException

InvalidArgumentException

ChargeFailureException

10.4 Reserved Amount Charging API

This subclause describes an initial set of capabilities in terms of message invocations, parameters and data types. The message-based invocations are:

- reserveAmount
- reserveAdditionalAmount
- chargeReservation
- releaseReservation

10.4.1 Reserve a Currency Amount from an Account

c) reserveAmount(EndUserIdentifier endUserIdentifier, Decimal amount, String billingText, out String reservationIdentifier)

b) Behaviour:

This message results in directly reserving an amount for an account indicated by the end user identifier. The reservation is specified as a currency amount. Note that reservations do not last forever; it is assumed the default reservation enforcement time is negotiated off-line. If the reservation times out, the remaining funds will be returned to the account

from which this reservation was made. However, the remaining funds shall preferably be returned explicitly to the account using the **releaseReservation** message. The billing text field is used for textual information to appear on the bill. Subsequent textual information provided during this charging session will be appended to this textual information; one charging session to a reservation will result in only one entry on the bill. In case of success, a reservation id is returned for future reference; e.g. subsequent charging against the existing reservation using the **chargeReservation** message.

c) Parameters:

NAME	TYPE	DESCRIPTION
endUserIdentifier	EndUserIdentifier	The end user's account subject to the reservation
amount	Decimal	The currency amount of the reservation
billingText	String	Textual information to appear on the bill
reservationIdentifier	String	OUTPUT. It is an identifier for the newly created reservation

d) Exceptions:

UnknownEndUserException

InvalidArgumentException

ServiceException

10.4.2 Adjust the Currency Amount of an Existing Reservation

a) reserveAdditionalAmount(String reservationIdentifier, Decimal amount, String billingText)

b) Behaviour:

This message results in the addition/reduction of a currency amount to/from an existing reservation indicated by the reservation id. The reservation is specified as a currency amount. Note that reservations do not last forever; it is assumed the default reservation enforcement time is negotiated off-line. Invoking this message will extend the reservation enforcement time for another off-line-negotiated period. The billing text field is used for appending textual information to appear on the bill. The textual information is appended to the initial textual information given by the **reserveAmount** message; one charging session to a reservation will result in only one entry on the bill. Reserved credit can be returned to the account through the **releaseReservation** message.

c) Parameters:

NAME	TYPE	DESCRIPTION
reservationIdentifier	String	An identifier for the reservation to be amended
amount	Decimal	The currency amount to be added to (or subtracted from) the reservation
billingText	String	Textual information to appear on the bill

d) Exceptions:

UnknownReservationException

InvalidArgumentException

ServiceException

10.4.3 Charge a Currency Amount against an Existing Reservation

a) chargeReservation(String reservationIdentifier, Decimal amount, String billingText, String referenceCode)

b) Behaviour:

This message results in charging to a reservation indicated by the reservation id. Reservations, identified by reservation id, are established through invoking the **reserveAmount** message. The charge is specified as a currency amount. Optionally, the billing text field can be used for appending textual information to appear on the bill. The textual information is appended to the initial textual information given by the **reserveAmount** message; one charging session to a reservation will result in only one entry on the bill. The reference code is used to uniquely identify the request; it is the application's responsibility to provide a unique reference code within the scope of the application.

c) Parameters:

NAME	TYPE	DESCRIPTION
reservationIdentifier	String	An identifier for the reservation to be charged
amount	Decimal	The currency amount of the charge
billingText	String	OPTIONAL. Textual information to appear on the bill
referenceCode	String	Textual information to uniquely identify the request, e.g. in case of disputes

d) Exceptions:

UnknownReservationException

InvalidArgumentException

ChargeFailureException

10.4.4 Release an Existing Reservation

a) releaseReservation(String reservationIdentifier)

b) Behaviour:

Returns funds left in a reservation indicated by reservation id to the account from which this reservation was made. Reservations, identified by reservation id, are established by invoking the reserveAmount message.

c) Parameters:

NAME	TYPE	DESCRIPTION
reservationIdentifier	String	An identifier for the reservation to be released

d) Exceptions:

UnknownReservationException

ServiceException

10.5 Reserved Volume Charging API

This subclause describes an initial set of capabilities in terms of message invocations, parameters and data types. The message-based invocations are:

- getAmount
- reserveVolume

- reserveAdditionalVolume
- chargeReservation
- releaseReservation

10.5.1 Convert a Volume to a Currency Amount

a) getAmount(EndUserIdentifier endUserIdentifier, Long volume, out Decimal amount)

b) Behaviour:

Returns the amount resulting from converting the given volume. The end user identifier is given to indicate the subscriber for whom this calculation must be made. The message returns a currency amount if successful.

c) Parameters:

NAME	TYPE	DESCRIPTION
endUserIdentifier	EndUserIdentifier	The end user's account to be charged
volume	Long	The volume to be converted
amount	Decimal	OUTPUT. It is the currency amount resulting from the conversion process

d) Exceptions:

UnknownEndUserException

InvalidArgumentException

ServiceException

10.5.2 Reserve a Volume from an Account

a) reserveVolume(EndUserIdentifier endUserIdentifier, Long volume, String billingText, out String reservationIdentifier)

b) Behaviour:

Reserves an amount of an account indicated by the end user identifier. The reservation is specified as a volume. Note that reservations do not last forever; it is assumed the default reservation enforcement time is negotiated off-line. If the reservation times out, the remaining volume will be returned to the account from which this reservation was made. However, the remaining volume should preferably be returned explicitly to the account using the **releaseReservation** message. The billing text field is used for textual information to appear on the bill. Subsequent textual information provided during this charging session will be appended to this textual information; one charging session to a reservation will result in only one entry on the bill. In case of success, a reservation identifier is returned for future reference; e.g. subsequent charging against the existing reservation using the **chargeReservation** message.

c) Parameters:

NAME	TYPE	DESCRIPTION
endUserIdentifier	EndUserIdentifier	The end user's account subject to the reservation
volume	Long	The volume of the reservation
billingText	String	Textual information to appear on the bill
reservationIdentifier	String	OUTPUT. It is an identifier for the newly created reservation

d) Exceptions:

UnknownEndUserException

InvalidArgumentException

ServiceException

10.5.3 Adjust the Volume of an Existing Reservation

a) reserveAdditionalVolume(String reservationIdentifier, Long volume, String billingText)

b) Behaviour:

Adds/reduces a volume to an existing reservation indicated by the reservation id. The reservation is specified as a volume. Note that reservations do not last forever; it is assumed the default reservation enforcement time is negotiated off-line. Invoking this message will extend the reservation enforcement time for another off-line-negotiated period. The billing text field is used for appending textual information to appear on the bill. The textual information is appended to the initial textual information given by the **reserveVolume** message; one charging session to a reservation will result in only one entry on the bill. A reserved credit can be returned to the account through the **releaseReservation** message.

c) Parameters:

NAME	TYPE	DESCRIPTION
reservationIdentifier	String	An identifier for the reservation to be amended
volume	Long	The volume to be added to (or subtracted from) the reservation
billingText	String	Textual information to appear on the bill

d) Exceptions:

UnknownReservationException

InvalidArgumentException

ServiceException

10.5.4 Charge a Volume against an Existing Reservation

a) chargeReservation(String reservationIdentifier, Long volume, String billingText, String referenceCode)

b) Behaviour:

This message results in charging to a reservation indicated by the reservation id.. Reservations, identified by reservation id., are established through invoking the **reserveVolume** message. The charge is specified as a volume. Optionally, the billing text field can be used for appending textual information to appear on the bill. The textual information is appended to the initial textual information given by the **reserveVolume** message; one charging session to a reservation will result in only one entry on the bill. The reference code is used to uniquely identify the request; it is the application's responsibility to provide a unique reference code within the scope of the application.

c) Parameters:

NAME	TYPE	DESCRIPTION
reservationIdentifier	String	An identifier for the reservation to be charged
volume	Long	The currency amount of the charge
billingText	String	OPTIONAL. Textual information to appear on the bill
referenceCode	String	Textual information to uniquely identify the request, e.g. in case of disputes

d) Exceptions:

UnknownReservationException

InvalidArgumentException

ChargeFailureException

10.5.5 Release an Existing Reservation

a) releaseReservation(String reservationIdentifier)

b) Behaviour:

Returns funds left in a reservation indicated by reservation id. to the account from which this reservation was made. Reservations, identified by reservation id., are established through invoking the **reserveVolume** message.

c) Parameters:

NAME	TYPE	DESCRIPTION
reservationIdentifier	String	An identifier for the reservation to be released

d) Exceptions:

UnknownReservationException

ServiceException

10.6 Web Service Data Definitions

10.6.1 Data Types

All data types are defined in clause 5.1.

10.6.2 Exceptions

In addition to the Common Exceptions defined in clause 5.2, there are exceptions specific to this Web Service. Similar to the Common Exceptions, each Web Service-specific exception is assigned an eight-character identifier. This identifier is interpreted as described in clause 5.2, except that the first 3 characters uniquely identify this Web Service.

The following exceptions are specific to this Web Service:

UNIQUE ID	TEXT STRING	MEANING
PAY1000E	ChargeFailureException	Indicates that an error occurred when attempting to charge to the account. The charge did not occur.
PAY1001F	UnknownReservation Exception	Indicates that the passed reservation identifier is unknown.
PAY1002E	UnknownReservation Exception	Indicates that the passed reservation identifier is unavailable; it may have timed out according to the policy in place.

10.7 Web Service Syntax – WSDL

The W3C WSDL representation of this API is contained in a set of files which accompany the present document (see Annex A).

The rpc/literal files are

- parlayx_payment_service_port.wsdl
- parlayx_payment_amount_charging_service
- parlayx_payment_volume_charging_service
- parlayx_payment_reserve_amount_charging_service
- parlayx_payment_reserve_volume_charging_service

The rpc/encoded file is

- parlayx_payment_service.wsdl

11 Account management

11.1 Overview

11.1.1 Description

Pre-paid subscribers, whether they have subscribed to pre-paid telephony, SMS, or data service, have credits with their service providers; the consumption of services will lead to reduction of their credit, or the credit may expire. Therefore, from time to time, subscribers may have to recharge their accounts. This occurs through an application that interfaces with the subscriber either directly or indirectly. Examples of direct interaction are voice prompts and WAP/web pages, or even SMS. Typically, such multi-modal applications either request a currency amount and, e.g. credit card information, or a voucher number plus credentials. The voucher number and credentials are then validated and causes a pre-determined currency amount to be transferred.

The Parlay X Account Management API described in this document supports account querying, direct recharging and recharging through vouchers. As a side effect, it may prevent subscribers from having their account balance credits expire.

11.1.2 Commercial & Technical Rationale

The recharging process is a critical component of telecom networks. At present, a number of prepaid services exist: e.g. Voice, SMS, and GPRS. The Account Management Web Service:

- further enlarges the market for third party software and outsourcing as it supports subscriber self service through re-charging, account querying and prevents subscriber credits from expiring
- enables subscriber "self-service" through trusted and authenticated service or content providers
- allows recharging by subscriber address (e.g. MSISDN and PIN).

11.1.3 Relationship to Similar or Supplanted Specifications

Parlay/OSA have developed powerful, carrier-grade Content-Based Charging (CBC) and Account Management (AM) APIs. The CBC APIs can be used to do recharging, but they also enable many other functions (e.g. debiting from accounts) as well. The AM APIs support some subscriber self-service (i.e. query functions), but they are not as comprehensive as the Account Management Web Service.

11.1.4 Scenarios

This subclause discusses three scenarios; one where a subscriber uses a voucher, one where the subscriber directly recharges after the payment is cleared, and one where the subscriber checks the recent transactions. Note, associated Account Management API messages are shown in 'bold' format: e.g. (**getBalance**).

11.1.4.1 Scenario Number 1

The prepaid subscriber wishes to recharge their account with a voucher and query their account balance. The subscriber uses their mobile phone or other wireline phone to interact with an IVR system. In order to recharge their account, the subscriber must enter the voucher number, the MSISDN to be recharged , and PIN(s). The IVR system accesses an external voucher database to validate the voucher number. The subscriber's account balance is then increased with the value of the voucher (**voucherUpdate**). The subscriber queries their account balance (**getBalance**), before and/or after the recharge.

11.1.4.2 Scenario Number 2

Directly recharging (i.e. without a voucher) works much along the same way. In this case, we assume the prepaid subscriber interacts with a web page. After providing the MSISDN, along with the PIN, the user can query the account balance (**getBalance**). For recharging, the subscriber must enter payment details, for example credit card information, from which the payment will be made. After clearing the payment details, the currency amount will be transferred and the subscriber's prepaid account balance expiration date will be reset (**balanceUpdate**). The subscriber also queries their account balance expiration date (**getCreditExpiryDate**), after the recharge.

11.1.4.3 Scenario Number 3

Every time a subscriber makes a telephone call the balance of their prepaid account is decremented with the cost of the call. When a recharge is done, the balance is increased either directly (**balanceUpdate**) or with an amount represented by a voucher (**voucherUpdate**). When a subscriber has doubts about the correctness of the account balance, the subscriber can request the last transactions on the account and verify them (**getHistory**). By offering automated access to this information, a call to the Operator's Help Desk is prevented which saves operational costs.

11.2 Account Management API

This subclause describes an initial set of capabilities in terms of message invocations, parameters and data types. The message-based invocations are:

- getBalance
- getCreditExpiryDate
- balanceUpdate
- voucherUpdate
- getHistory

Note that certain parameters are negotiated off line. For example the reseller identifier, which identifies the reseller of calling cards.

11.2.1 Account Balance Query

getBalance(EndUserIdentifier endUserIdentifier, String endUserPin, out Decimal amount)

b) Behaviour:

This message results in getting account balance indicated by the end user identifier and associated end user PIN. The returned amount is specified as a currency amount.

c) Parameters:

NAME	TYPE	DESCRIPTION
endUserIdentifier	EndUserIdentifier	This parameter identifies the end user's account.
endUserPin	String	Contains the end user's credentials for authorizing access to the account
amount	Decimal	OUTPUT. It is the balance on the end user's account.

d) Exceptions:

InvalidArgumentException

ServiceException

PolicyException

EndUserAuthenticationException

11.2.2 Account Credit Expiration Date Query

a) getCreditExpiryDate(EndUserIdentifier endUserIdentifier, String endUserPin, out DateTime date)

b) Behaviour:

This message results in getting the expiration date of the credit indicated by the end user identifier and associated end user PIN. The returned date is the date the current balance will expire. Nil is returned if the balance does not expire.

c) Parameters:

NAME	TYPE	DESCRIPTION
endUserIdentifier	EndUserIdentifier	This parameter identifies the end user's account.
endUserPin	String	Contains the end user's credentials for authorizing access to the account.
date	DateTime	OUTPUT. It is the date the current balance will expire. Nil is returned if the balance does not expire.

d) Exceptions:

InvalidArgumentException

ServiceException

PolicyException

EndUserAuthenticationException

11.2.3 Account Balance Recharging

a) balanceUpdate(EndUserIdentifier endUserIdentifier, String endUserPin, String referenceCode, Decimal amount, Integer period)

b) Behaviour:

This message results in directly recharging the account indicated by the end user identifier and optional associated end user PIN. The reference code is used to uniquely identify the request; it is the application's responsibility to provide a unique reference code within the scope of the application. The charge is specified as a currency amount. The balance is requested to expire in the number of days indicated by the period parameter. The operator's policies may overrule this parameter. If the optional period parameter is not present, the operator's policy on balance expiration is always in effect.

c) Parameters:

NAME	TYPE	DESCRIPTION
endUserIdentifier	EndUserIdentifier	This parameter identifies the end user's account.
endUserPin	String	OPTIONAL. Contains the end user's credentials for authorizing access to the account.
referenceCode	String	Textual information to uniquely identify the request, e.g. in case of disputes
amount	Decimal	Currency amount that should be added to the end user's account.
period	Integer	OPTIONAL. The balance is requested to expire in the number of days indicated by this parameter. The operator's policies may overrule this parameter. If this optional parameter is not present, the operator's policy on balance expiration is always in effect.

d) Exceptions:

InvalidArgumentException

ServiceException

PolicyException

EndUserAuthenticationException

11.2.4 Account Balance Voucher Recharging

a) voucherUpdate(EndUserIdentifier endUserIdentifier, String endUserPin, String referenceCode, String voucherIdentifier, String voucherPin)

b) Behaviour:

This message results in directly recharging the account indicated by the end user identifier and optional associated end user PIN. The reference code is used to uniquely identify the request; it is the application's responsibility to provide a unique reference code within the scope of the application. A voucher identifier indirectly specifies the charge. The optional voucher PIN code can be used to verify the voucher.

c) Parameters:

NAME	TYPE	DESCRIPTION
endUserIdentifier	EndUserIdentifier	This parameter identifies the end user's account.
endUserPin	String	OPTIONAL. Contains the end user's credentials for authorizing access to the account.
referenceCode	String	Textual information to uniquely identify the request, e.g. in case of disputes
voucherIdentifier	String	This parameter identifies the voucher.
voucherPin	String	OPTIONAL. Contains the voucher's credentials for authentication.

d) Exceptions:

InvalidArgumentException

UnknownVoucherException

ServiceException

PolicyException

EndUserAuthenticationException

11.2.5 Account Transaction History Query

a) getHistory(EndUserIdentifier endUserIdentifier, String endUserPin, DateTime date, Integer maxEntries, out DatedTransaction[] history)

b) Behaviour:

This message results in returning the transaction history of the account indicated by the end user identifier and associated optional end user PIN. The maximum number of entries to return and the start date define the range of transactions that are of interest to the requester.

If the total number of entries in the transaction history, starting at the specified date, is larger than the specified maximum number of entries, only the most recent events are returned. Note that the operator might limit the maximum amount of entries to be returned or the period for which the entries are to be returned.

c) Parameters:

NAME	TYPE	DESCRIPTION
endUserIdentifier	EndUserIdentifier	This parameter identifies the end user's account.
endUserPin	String	OPTIONAL. Contains the end user's credentials for authorizing access to the account.
date	DateTime	OPTIONAL. This parameter indicates the desired starting date for the entries to be returned. If this parameter is not present, it is up to the discretion of the service to decide this date.
maxEntries	Integer	OPTIONAL. This parameter indicates the maximum number of entries that shall be returned. If this parameter is not present, it is up to the discretion of the service to decide how many entries to return.
history	Array of DatedTransaction	OUTPUT. It is a DatedTransaction array that consists of types with a date field and a string field: i.e. the date of the occurrence and the transaction details, respectively.

d) Exceptions:

InvalidArgumentException

ServiceException

PolicyException

EndUserAuthenticationException

11.3 Web Service Data Definitions

11.3.1 Data Types

In addition to the Common Data Types defined in clause 5.1, the following Data Types are specific to this Web Service.

11.3.1.1 DatedTransaction

The **DatedTransaction** data type is a structure containing the following parameters:

NAME	TYPE	DESCRIPTION
transactionDate	DateTime	The date the transaction occurred.
transactionDetails	String	The transaction details.

11.3.2 Exceptions

In addition to the Common Exceptions defined in clause 5.2, there are exceptions specific to this Web Service. Similar to the Common Exceptions, each Web Service-specific exception is assigned an eight-character identifier. This identifier is interpreted as described in clause 5.2, except that the first 3 characters uniquely identify this Web Service.

The following exceptions are specific to this Web Service:

UNIQUE ID	TEXT STRING	MEANING
ACM1000E	UnknownVoucherException	This fault occurs if the voucher identification that is passed is unknown.
ACM1001E	EndUserAuthentication Exception	This fault occurs if either the end user identification that is passed is unknown, the end user's credentials are required but are not passed, or the end user's credentials are passed but are invalid.

11.4 Web Service Syntax – WSDL

The W3C WSDL representation of this API is contained in a set of files which accompany the present document (see Annex A).

The rpc/literal files are

- parlayx_account_management_types.xsd
- parlayx_account_management_service_port.wsdl
- parlayx_account_management_service.wsdl

The rpc/encoded file is

- parlayx_account_management_service.wsdl

12 User status

12.1 Overview

12.1.1 Description

The Parlay X User Status Web Service will be used for getting user status information. The use of the Web Service will not require any specific telecommunication skills.

12.1.2 Commercial & Technical Rationale

The use of a Parlay X User Status Web Service will enable fast and easy development of user status aware applications.

12.1.3 Relationship to Similar or Supplanted Specifications

The Parlay/OSA Mobility (User Status) Service API can also provide user status information, but the Parlay X API for this Web Service is easier to use.

The Parlay/OSA Presence and Availability Management Service specifies a set of interfaces for applications that provide broader presence and availability capabilities than those currently defined for this Web Service. These capabilities will be considered for inclusion in future Parlay X Web Services.

12.1.4 Scenarios

In the following, two sample Scenarios are described.

12.1.4.1 Buddy-list

This application is a mobile version of services like ICQ, MSN Messenger Service and Yahoo! Messenger, which offer text and voice chat and text conferencing. With a mobile terminal the user can be always on.

The user of the service can define one or more buddy-lists containing their friends (alternatively their family, colleagues or a combination). The status of buddies can be shown in the Buddy-list

12.1.4.2 Manual call routing

When a switchboard receives a call, the switchboard operator may obtain the status of a mobile terminal before trying to route the call to it.

12.2 User Status API

This subclause describes an initial set of capabilities in terms of message invocations, parameters and data types. The message-based invocations are:

- getUserStatus

12.2.1 Get User Status

a) getUserStatus(EndUserIdentifier endUser, EndUserIdentifier requester, out UserStatusData userStatus)

b) Behaviour:

Requests the user status information of a user. Before returning the user status indicator, end-user and operator policies must be satisfied.

c) Parameters:

NAME	TYPE	DESCRIPTION
endUser	EndUserIdentifier	It identifies the end user for whom user status information is being requested.
requester	EndUserIdentifier	OPTIONAL. The address of the requester.
userStatus	UserStatusData	OUTPUT. It is the user status of the end user

d) Exceptions:

InvalidArgumentException

UnknownEndUserException

ServiceException

PolicyException

12.3 Web Service Data Definitions

12.3.1 Data Types

In addition to the Common Data Types defined in clause 5.1, the following Data Types are specific to this Web Service.

12.3.1.1 UserStatusData

The **UserStatusData** data type is a structure containing the following parameters:

NAME	TYPE	DESCRIPTION
userStatusIndicator	UserStatusIndicator	It indicates the status of the end user.
additionalUserStatus Information	String	A string to hold additional information if the userStatusIndicator is 'Other'.

12.3.1.2 UserStatusIndicator

The **UserStatusIndicator** data type is an enumeration with the following values:

VALUE	DESCRIPTION
Online	User is online.
Offline	User is offline (mobile terminal switched off/other terminal not connected to the service), or wants to appear to be offline.
Busy	User is busy.
Other	Custom user status information can be retrieved from additionalUserStatusInformation.

12.3.2 Exceptions

All exceptions thrown by this Web Service are Common Exceptions, as defined in clause 5.2.

12.4 Web Service Syntax – WSDL

The W3C WSDL representation of this API is contained in a set of files which accompany the present document (see Annex A).

The rpc/literal files are

- parlayx_user_status_types.xsd
- parlayx_user_status_service_port.wsdl
- parlayx_user_status_service.wsdl

The rpc/encoded file is

parlayx_user_status_service.wsdl

13 Terminal Location

13.1 Overview

13.1.1 Description

The Parlay X Terminal Location Web Service will be used for getting location information. The use of the Web Service will not require any specific telecommunication skills, but knowledge of location co-ordinates will be required.

13.1.2 Commercial & Technical Rationale

The use of a Parlay X Terminal Location Web Service will enable fast and easy development of location aware Applications. The use of the Web Service will only require knowledge of longitude and latitude from the World Geodetic System 1984 (WGS 84), which is the reference system chosen for the coding of locations.

The market for location aware services is growing, and easy access to location information will stimulate the growth of this market.

13.1.3 Relationship to Similar or Supplanted Specifications

The Mobile Positioning System (MPS) offers much of the same functionality that the Parlay X Terminal Location Web Service will offer.

13.1.4 Scenarios

In the following, a sample Scenario is described.

13.1.4.1 Location enabled Buddy-list:

This application is a mobile version of services like ICQ, MSN Messenger Service and Yahoo! Messenger, which offer text and voice chat and text conferencing. With a mobile terminal the user can be always on, and the service can also show where buddies are located.

The user of the service can define one or more buddy-lists containing their friends (alternatively their family, colleagues or a combination). The Buddy List application may access terminal location information, and the following two cases are considered: a user initiates the retrieval of i) its own location and ii) other users' location.

13.2 Terminal Location API

This subclause describes an initial set of capabilities in terms of message invocations, parameters and data types. The message-based invocations are:

- getLocation

13.2.1 Get Location of Terminal

a) getLocation(EndUserIdentifier endUser, EndUserIdentifier requester, LocationAccuracy accuracy, out LocationInfo location)

b) Behaviour:

Requests the location of one terminal. The location is returned if the terminal is available. Before returning the location, end-user and operator policies must be satisfied.

c) Parameters:

NAME	TYPE	DESCRIPTION
endUser	EndUserIdentifier	The address of the terminal for which location information is being requested.
requester	EndUserIdentifier	OPTIONAL. The address of the terminal from which the request is initiated.
accuracy	LocationAccuracy	The desired accuracy. Possible values are: 'Low', 'Medium', 'High'. Each operator must assign a "radius of uncertainty" to each value (e.g. < 3 km, < 1 km, < 10 m, respectively)
location	LocationInfo	OUTPUT. It identifies the location of the terminal.

d) Exceptions:

InvalidArgumentException

UnknownEndUserException

ServiceException

PolicyException

13.3 Web Service Data Definitions

13.3.1 Data Types

In addition to the Common Data Types defined in clause 5.1, the following Data Types are specific to this Web Service.

13.3.1.1 LocationInfo

The **LocationInfo** data type is a structure containing the following parameters:

NAME	TYPE	DESCRIPTION
longitude	Float	Longitude
latitude	Float	Latitude
accuracy	LocationAccuracy	How accurate the Location information is. If the degree of accuracy wanted is available this should be given, if not the best possible accuracy should be returned. Possible values are: 'Low', 'Medium', 'High'.
dateTime	DateTime	Identifies when the location information was obtained.

13.3.1.2 LocationAccuracy

The **LocationAccuracy** data type is an enumeration with the following values: (Each Parlay X Gateway operator must specify the uncertainty radiuses for Low, Medium and High)

VALUE	DESCRIPTION
Low	Low accuracy i.e. < 3 km radius of uncertainty.
Medium	Medium accuracy i.e. < 1 km radius of uncertainty.
High	High accuracy i.e. < 100 m radius of uncertainty.

13.3.2 Exceptions

All exceptions thrown by this Web Service are Common Exceptions, as defined in clause 5.2.

13.4 Web Service Syntax – WSDL

The W3C WSDL representation of this API is contained in a set of files which accompany the present document (see Annex A).

The rpc/literal files are

- parlayx_terminal_location_types.xsd
- parlayx_terminal_location_service_port.wsdl
- parlayx_terminal_location_service.wsdl

The rpc/encoded file is

parlayx_terminal_location_service.wsdl

Annex A (informative): W3C WSDL Description of Web Service Syntax

The W3C WSDL representation of the APIs specified in the present document is contained in a set of files which accompany the present document:

px0326rpcenc.zip

px0326rpclit.zip

Annex B (informative): Change history

Change history							
Date	**TSG #**	**TSG Doc.**	**CR**	**Rev**	**Subject/Comment**	**Old**	**New**
Jul 03	--	--	--	--	1st Draft in 3GPP Spec. format, no number assigned	-	0.0.1
Oct 03	--	--	--	--	Change of Title and removal of PayCircle copyright from Clause 10. Addition of Parlay and PayCircle acknowledgement in Forward clause	0.0.1	0.0.2
Oct 03	--	--	--	--	• Change 3GPP reference to include 29.199; • Removal of PayCircle from the acknowledgement in the Foreword clause. . • Title changed to remove the word 'specification'. • Addition of WSDL files to the zip file.	0.0.2	
Nov 2003	--	--	--	--	• EditHelp document processing & MCC review. • Created reference list (please review the IETF ones !!) • Added a new Annex A (informative): W3C WSDL Description of Web Service Syntax referred to from the various subclauses entitled x.y Web Service Syntax – WSDL • Field codes/automatic numbering/Bookmarking of Clauses/Figures etc. had been removed • Hidden text still exist (to be removed by the authors ?)	0.0.2	0.0.3
Dec 2003	CN_21	NP-030552	--	--	Submitted to CN#22 for Information	1.0.0	
Jan 2004	--	--	--	--	Added The W3C WSDL representation of the APIs specified in the present document is contained in a set of files which accompany the present document: px0326rpcenc.zip px0326rpclit.zip	1.0.1	