Enterprise Messaging Platform

User Manual for WS Interface

Revision History:

Version	Date	Changed by	Change description
1.0	30.11.2010	Tomáš Zeman	The first version.
1.1	12.6.2012	Jan Cervenak	Updated document.
1.2	3.6.2014	Martin Polovincak	Updated chapter 3.3

Table of Contents

1 Introduction

1.1 Purpose

This document is created only for partners who will use a connection to Enterprise Messaging Platform. This document contains time severity of individual steps for connection and by the same document specifies minimal requirements for finally features of apps on partner's side.
Partner has an option for usage all of the features from Enterprise Messaging Platform, but he has realize requirements for every feature from set, which you want to use.

1.2 References

Reference	Referenced document	Version
3GPP23038	3GPP 23.038 – Alphabets and language-specific information.	
COOKIES	Persistent Client State -- HTTP Cookies http://www.netscape.com/newsref/std/cookie_spec.html	
COOKIES2	HTTP State Management Mechanism http://www.ietf.org/rfc/rfc2965.txt	
GSM338	GSM 03.38 – Alphabets and language-specific information.	
GSM340	GSM 03.40 – Technical realization of the Short Message Service	
HTTP	RFC 2616 http://www.w3.org/Protocols/rfc2616/rfc2616.html	
HTTP/TLS	HTTP over TLS http://www.ietf.org/rfc/rfc2818.txt	
RFC2119	Key words for use in RFCs to Indicate Requirement Levels http://www.ietf.org/rfc/rfc2119.txt	
SMSG_WSDL	WSDL descriptor describing EMP web service interface, the descriptor is included in [XSDSCHEMA] zip file as `sMessageService.wsdl`	
SPECPARAMS	`Connection parameters to O2 testing environment for partner` (also supplied with this document) contains the information about connection specific parameters listed at the section 3.6, "Deployment Specific Parameters".	

Reference	Referenced document	Version
TLS	The TLS Protocol Version 1.0 http://www.ietf.org/rfc/rfc2246.txt	
SSL	The SSL Protocol Version 3.0 http://wp.netscape.com/eng/ssl3/draft302.txt	
XMLSCHEMA	XML Schema http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/ http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/	
XSDSCHEMA	This document is supplied with the `sMessageSpec.zip` file, which contains all the XSDs and WSDL examples.	

Table 1: References

1.3 Definitions, Acronyms and Abbreviations

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

The following table lists acronyms used in the document.

Acronym	Description
AO	Application-Originated – message send from application to handset.
BA	Business Application
GSM	Global System for Mobile communication
HTTP	Hypertext Transfer Protocol
M2M	Machine-to-Machine
MMS	Multimedia Messaging Service
MO	Mobile-Originated – message send from handset to application.
MSISDN	Mobile Station International ISDN Number
RFC	Request for Comments
SMS	Short Message Service; widely used abbreviation for messages transferred over this service.
SOAP	Simple Object Access Protocol
SSL	Secure Socket Layer

TCP/IP	Transmission Control Protocol / Internet Protocol
O2	O2 Czech Republic a.s.
EMP	Enterprise Messaging Platform
EMP WS Server	O2 platform providing the functionality of O2 systems through EMP WS interface.
EMP WS Client	An application consuming services provided by EMP WS Server. It is connected to EMP WS Server using SOAP protocol.
URL	Uniform Resource Locator
WS	Web Service
XML	Extensible Markup Language

Table 2: Acronyms

1.4 LEGAL DISCLAIMER

This work, either in whole or in part, must not be reproduced or disclosed to others or used for purposes other than that for which it is supplied, without O2´s prior written permission. O2 has made all reasonable efforts to ensure that the instructions contained in the document are adequate and free of errors and omissions. O2 will, if necessary, explain issues, which may not be covered by the document.

The information in this document is subject to change without notice.

The document may include brand and product names that are trademarks or registered trademarks of their respective holders.

Copyright© O2 Czech Republic a.s. 2010. All rights reserved.

2 Enterprise Messaging Platform

Enterprise Messaging Platform is platform of O2 Czech Republic a.s. allowing customers to build standards based applications for sending and receiving SMS messages through the O2 infrastructure.

The operations mentioned below are provided by two separate interfaces:

- Interface for priority messages (see the section for more information),
- Interface for non-priority messages.

The structure and configuration of the messages are the same for both of them. The `Receive` operation (message or status report) is available only for the non-priority interface, which means that the priority interface is used only for sending of the AO messages.

2.1 EMP Services

All services provided by the platform are listed in the **Chyba! Nenalezen zdroj odkazů.**.

Service	Description
Receiving a message or status report	Available only in the non-priority interface. Supported messages are only SMS messages. The message is sent to the mobile user. BA MAY choose whether to be notified about the message processing or not.
Sending AO message	Supported messages are only SMS messages. BA receives all messages sent to its assigned numbers as well as requested notifications about message processing

Table 3: EMP Services

The services are accessible via web service interface described by [SMSG_WSDL] file. The following chapter explains the WS interface in detail.

3 Web Service Interface

In this chapter there will be only a brief description of the interface. More information can be found in the chapter 4, “EMP WS Message”.

The Web Service (WS) interface provides an implementation for the *message sending* and the *message receiving* (see the chapter 4, “EMP WS Message” for more information). For the purposes of this specification there are defined two types of EMP WS nodes (see the chapter 4, “EMP WS Message” for more information):

- EMP WS Server – Node implementing the web service, typically EMP Service Provider,
- EMP WS Client – Node acting as a WS client, typically EMP Service Consumer.

In the current state of implementation, the EMP WS Server is implemented on the O2 side and partners implement the EMP WS Client.

3.1 Message Sending

Message sending is implemented as a simple SOAP call. There can be only one message in every SOAP call and only one recipient for every message.

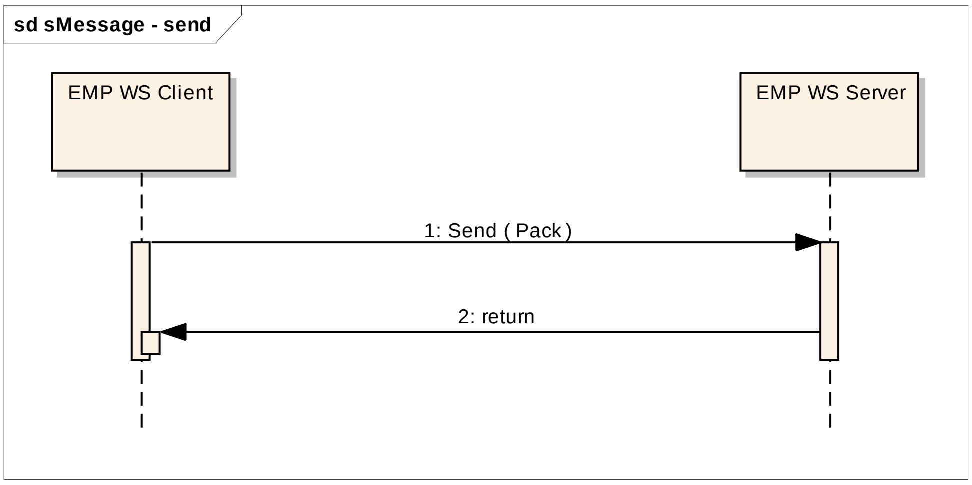

Figure 1: BA sends sMessages

3.2 Message Receiving

Message receiving is implemented using a simple polling mechanism.

EMP WS Client calls the `Receive` operation (see the Example 5.10: Message receiving for more information) to retrieve any message that is available.

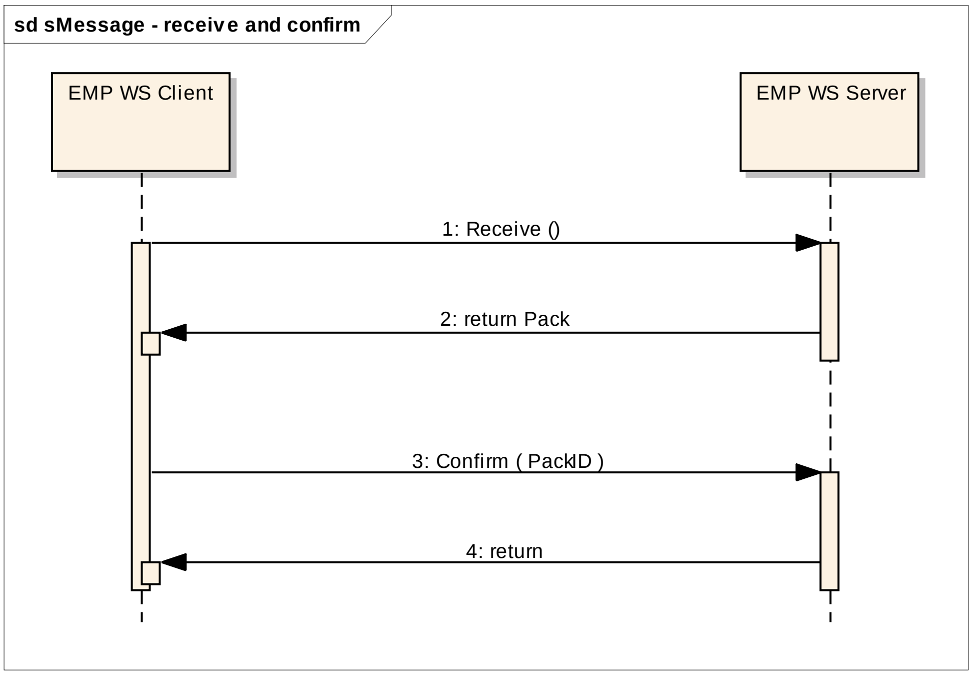

Figure 3: BA receives and confirms sMessages

If there is no message available, the EMP WS Client MUST wait for the agreed amount of time (see the *Waiting time on empty queue* in the section 3.6, "Deployment Specific Parame-

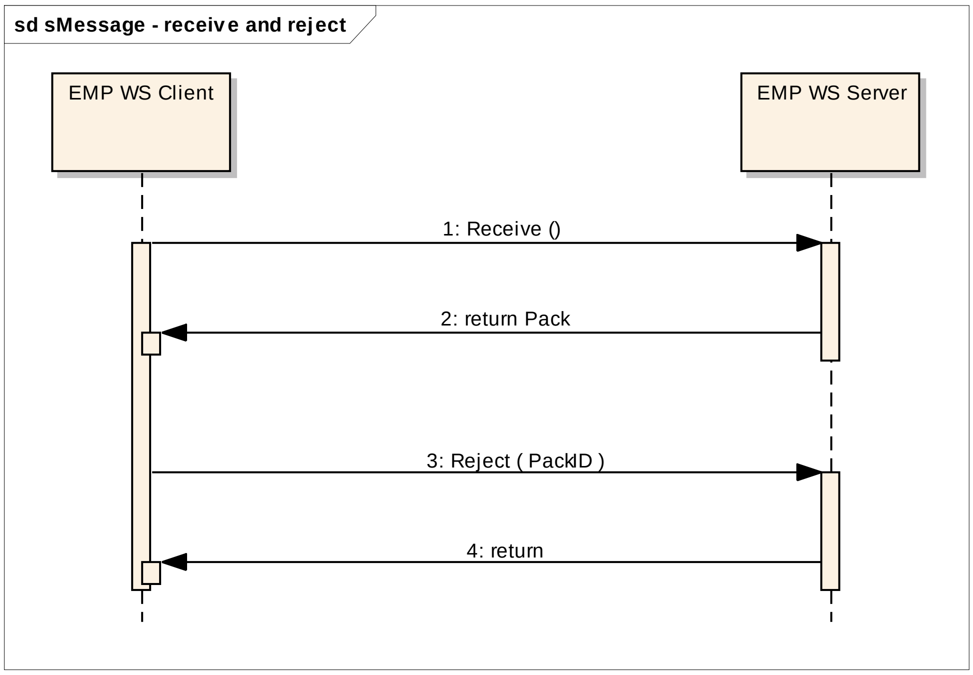

Figure 2: receives and rejects sMessages

ters”) and repeat the call. The Client SHOULD NOT call `Confirm` or `Reject` because there is nothing to confirm or reject.

In case there is a message, the `Receive` operation returns a set of the available messages in a `pack` element (see the chapter 4, “EMP WS Message” for more information). The returned pack contains a special header named `PackID` (see the Example 5.11: Delivery report for more information). This header contains identification of the pack that is used in subsequent call to `Confirm` (see the Example 5.12: Message confirmation for more information).

The client SHOULD forward the retrieved messages securely for processing (i.e. client SHOULD NOT process the messages synchronously) and confirm the reception by calling the `Confirm` operation within an agreed timeout (see *Confirmation timeout* in the section 3.6, “Deployment Specific Parameters”). The confirmation after timeout returns fault and the messages will be delivered again. The EMP WS Client MUST use the same `PackID` as the one returned by operation `Receive` when calls the operations `Confirm` or `Reject`. The client MUST NOT call the `Reject` operation after the `Confirm` operation with the same `PackID` or vice versa.

Note, that if there are many messages in the queue, the client will not be able to receive them until the first message is confirmed.

In case the client cannot ensure processing of the messages (e.g. due to technical problems) it SHOULD call the `Reject` operation to notify the EMP WS Server that the message was not successfully retrieved. In such case the messages will be redelivered in some of the subsequent invocations of the `Receive` operation.

3.3 Priority Interface

EMP has two message interfaces – one for non-priority messages and one for priority messages.

In case that the BA is provisioned to be able to send a priority messages, it can use a separate interface instance (URL) to submit such a message. There is no attribute in the message contents that specifies the message is priority (it is distinguished by submitting it through the priority interface). Please note that the priority interface URL is used for a message sending ONLY and no status/delivery reports will be obtained through the priority interface - all of the message responses will occur in the non-priority interface.

The messages from the interfaces are placed into two separate queues (priority and non-priority queues) in EMP. The messages from the priority queue are handled before messages from the non-priority queue (i.e. no message from the non-priority queue is sent to SMSC until all the messages from the priority queue are sent).

The status and delivery reports for the messages are both directed to the normal priority interface (i.e. even the status and delivery reports for the messages sent through the priority interface are delivered through the non-priority interface).

3.4 The sMessage Endpoint Specification

The web service is specified as a synchronous RPC WS (request-response mode). The communication uses the XML based protocol SOAP 1.1; the SOAP's body uses document-literal style. The description of the WS is provided as a WSDL 1.1 specification.

3.4.1 Operations of the WS

Operation	Parameters (Type)	Returns (Type)	Description
`Send`	`Pack (Pack)`	-	Sends a pack of messages.
`Receive`	-	`Pack (Pack)`	Receives a pack of messages and the id of the pack in the header (`PackID`).
`Confirm`	`PackID (String)`	-	Confirms the pack of messages specified by `PackID`. The `PackID` MUST be same as the one returned by the operation `Receive`.
`Reject`	`PackID (String)`	-	Rejects the pack of messages specified by `PackID`. The `PackID` MUST be same as the one returned by the operation `Receive`.

Table 4: The WS operations

3.4.1.1 Operations usage

According to the Table 4: The WS operations, there are four WS methods, which can be called (their description is in the table above):

- `Send (Pack Pack) : void`
- `Receive : Pack`
- `Confirm (String PackID) : void`
- `Reject (String PackID) : void`

3.4.2 Faults

Error	HTTP Status	SOAP Fault	Fault Code	Description
Request validation failed.	400	No	-	The XML document is not well-formed or does not correspond to WSDL or XML Schemas.
Client authentication failed.	403	No	-	Invalid client certificate or login/password - depends on the authentication method.
General server error.	500	No	-	Some error occurred on the server side. Repeating the call later MAY produce the desired result.
SOAP Request processing error.	500	Yes	`Server`	Some error occurred during the processing of the request. Repeating the call later MAY produce the desired result.
SOAP must understand fault.	500	Yes	`MustUnderstand`	The SOAP request contains a mandatory header block (one that has an attribute `mustUnderstand` with value `"1"`), which the server does not understand.
Too many unconfirmed packs.	500	Yes	`Client.Limit.Unconfirmed`	The client violated the maximum number of unconfirmed packs limit.
Pack confirmation failed.	500	Yes	`Client.ConfirmFailed`	The pack was confirmed later than the confirmation timeout specifies or was already confirmed by another thread.

Throughput limit exceeded.	500	Yes	`Client.Limit.Throughput`	The client violated the maximum throughput limit. In this case, the client SHOULD NOT repeat the call until the returned time-out expires.
Too many concurrent connections.	500	Yes	`Client.Limit.Connections`	The client violated the maximum number of concurrent connections limit.
The Receive operation is not allowed.	500	Yes	`Client.ReceiveNotAllowed`	The `Receive` operation is not allowed (see *The operation receive is disabled by configuration* in the section 3.6, "Deployment Specific Parameters").
The Confirm operation is not allowed.	500	Yes	`Client.ConfirmNotAllowed`	The `Confirm` operation is not allowed (see *The operation confirm is disabled by configuration* in the section 3.6, "Deployment Specific Parameters").
The `Reject` operation is not allowed.	500	Yes	`Client.RejectNotAllowed`	The `Reject` operation is not allowed (see *The operation reject is disabled by configuration* in the section 3.6, "Deployment Specific Parameters").

Table 5: Faults

3.4.2.1 The fault codes meaning

Server

- `{http://schemas.xmlsoap.org/soap/envelope/}Server`
- The fault announces that the call to the service failed, and did not have any affect. Repeating the call later MAY produce the desired result.

Client

- `{http://schemas.xmlsoap.org/soap/envelope/}Client`
- The fault informs that the wrong data were sent to the service, the interface specification was violated or some of the limits were exceeded. The client SHOULD NOT repeat the same call. There is an exception – the `Client.Limit` fault code, which means that the client exceeded one of the limits, so the same call MAY be repeated later.

Throughput limit exceeded fault

This fault MAY contain a timeout value in milliseconds in the element `{http://cz.o2.com/sMessage/sMessageService}RetryAfter` that will be a child of the element `{http://schemas.xmlsoap.org/soap/envelope/}detail` in SOAP fault. If the timeout is present, the EMP WS Client SHOULD NOT repeat the call until the returned timeout expires (see the Example 3.1: Throughput limit exceeded SOAP fault below).

Example 3.1: Throughput limit exceeded SOAP fault

```xml
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <soap:Fault>
      <faultcode>soap:Client.Limit.Throughput</faultcode>
      <faultstring>Throughput limit exceeded.</faultstring>
      <detail>
        <sMsgSvc:RetryAfter
          xmlns:sMsgSvc="http://cz.o2.com/sMessage/sMessageService">
          8156
        </sMsgSvc:RetryAfter>
      </detail>
    </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

Note

Whenever an error occurs, the client MUST wait 15 – 60 seconds before trying to access the server again.

3.5 Requirements for Client Implementation

3.5.1 Multiple parallel threads

To achieve a better performance in terms of number of messages exchanged, the client SHOULD use multiple parallel threads of control (multiple connections) when communicating with the EMP WS Server. There are however specific considerations and requirements when sending and receiving messages.

3.5.1.1 Sending messages

EMP WS Client MAY use multiple threads for the sMessages as long as it does not exceed maximum number of concurrent connections.

3.5.1.2 Receiving messages

EMP WS Client MAY use multiple threads for receiving messages with following limitations:

- The number of parallel connections MUST NOT exceed the maximum number of concurrent connections
- If the EMP WS Client supports cookies (see the section 3.5.2.3, “Cookie support” for more information), calls to Receive and corresponding calls to Confirm MUST be made with the same cookie set

3.5.2 Location transparency & load-balancing

3.5.2.1 Polling multiple servers

When receiving messages, EMP WS Client must allow for polling more than one EMP WS Server.

3.5.2.2 Configurable Server URL

EMP WS Client implementation SHOULD allow for reconfigurable Server URL i.e. for changing the URL without changing code of the application.

3.5.2.3 Cookie support

EMP WS Client SHOULD support HTTP cookies based on [COOKIES] or [COOKIES2] to allow for transparent load balancing on EMP WS Server.

To achieve a better performance in a multithreaded implementation, every thread SHOULD have a separate *cookie repository* (i.e. keeping its own set of cookies).

3.5.3 Security

3.5.3.1 Authentication

In the current version of this specification the HTTP basic authentication in combination with client SSL certificates is used (see the section 3.6, “Deployment Specific Parameters”).

The client application MUST be able to change certificates quickly. This is a part of *Pass to production* tests.

3.5.3.2 Transport Layer Security

HTTP with SSL/TLS transport mechanism (HTTPS) is used to achieve a sufficient level of security on the transport layer [HTTP/TLS].

EMP WS Client MUST use persistent HTTP connections.

EMP WS Client SHOULD use the persistent SSL sessions (persistent connection allows using the same TCP connection for sending and receiving multiple requests and responses) to avoid slow and expensive SSL handshakes. Supported versions of transport layer security protocols are:

- SSL version 3.0 [SSL],
- TLS version 1.0 [TLS].

EMP WS Client MUST check validity of the EMP WS Server certificate in case that

- It was issued by a trusted and previously agreed certificate authority
- Certificate is not expired or not valid yet
- Certificate is issued with a correct and previously agreed identity, which MUST be present in the `Common Name` in the `Subject` field of the certificate or in a `subjectAltName` extension of type `dNSName` [HTTP/TLS]

In case the conditions above mentioned are not met, the client MUST NOT communicate with the server and SHOULD inform O2 representatives.

3.6 Deployment Specific Parameters

The Web Service transport layer can be further parameterized using a set of configuration parameters for a specific service. The actual values for the parameters should be provided in a separate document for every deployment [SPECPARAMS].

Confirmation timeout

The maximum time within the retrieved request SHOULD be confirmed. In case the client fails to confirm the retrieved request within this time, the request is considered rejected and will be redelivered to the client.

Maximum number of concurrent connections

This defines the maximum number of concurrent connections to the server.

Maximum number of unconfirmed packs

This defines the maximum number of packs waiting for confirmation (or rejection).

Maximum throughput

Defines the maximum number of messages sent per second using the `Send` operation. Note, that it is the number of messages and not the number of `Send` calls.

The EMP also provides throttling per payload, i.e. it is possible to define how many SMS, MMS or WAP Push messages can be sent within a specified time window. Corresponding values can be found in the [SPECPARAMS] if the functionality is enabled.

Message Acceptance/Rejection usage

- Default: no usage, requests are logged but ignored
- Specifies how the *Accept/Reject* requests are used in the deployment and what are contents of extra-info fields in the requests (see the section 4.5, "Generic Message Acceptance/Rejection extension" for more information)

SSL encryption used

Defines whether or not the SSL encryption is used.

The operation `confirm` is disabled by configuration

The architecture demands cause specific feature - the disabling of the confirm operation by configuration. Thus it is possible to have more endpoints for incoming data to disposal, but only one for receive and subsequent confirm/reject.

The operation `receive` is disabled by configuration

The architecture demands cause specific feature - the disabling of the receive operation by configuration. Thus it is possible to have more endpoints for incoming data to disposal, but only one for receive.

The operation `reject` is disabled by configuration

The architecture demands cause specific feature - the disabling of the reject operation by configuration. Thus it is possible to have more endpoints for incoming data to disposal, but only one for receive and subsequent confirm/reject.

Uniqueness period

- Default: 60 days
- Period for which the platform keeps data for message uniqueness checks

Waiting time on empty queue

Defines how long a client SHOULD wait before calling the `Receive` operation again if no messages were available.

4 EMP WS Message

The EMP WS message is a SOAP message. It consists of payload and metadata.

Payload

Payloads are defined as XML complex types extending complex type payload from namespace `http://cz.o2.com/sMessage/Payload`.

Metadata

- defines transport binding independent EMP WS message metadata
- optional transport or extension specific message metadata

It is a responsibility of the transport binding specification to specify how the Message is represented in the transport layer.

4.1 Activities

The protocol defines only two logical activities nodes can perform.

- message sending – message is sent out for processing
- message reception – message is received by a node

It is a responsibility of the transport binding specification to specify how nodes perform these activities.

4.2 EMP WS Message Metadata

The EMP WS specification defines a set of attributes of a message independent of a used transport protocol.

4.2.1 Message identification and relationships

MessageID

`MessageID` is a string identifier of the message with a maximum length of 128 characters (see the section 4.3.1, "Message uniqueness" for more information).

RelatesTo

`RelatesTo` is an optional identifier (it has a value of the `MessageID`) of another message this message relates to. This attribute is used in message exchange patterns like *In-Out, In-Optional-Out, Out-In, Out-Optional-In.*

4.2.2 Message authentication

The current version of the specification does not define any message-level security and it is left on the transport binding documents to ensure sufficient security measures.

In the future there may be a common EMP WS message security.

4.3 Node Implementation

4.3.1 Message uniqueness

All nodes MUST equip all the produced messages with a unique `MessageID`.

All nodes (both EMP Service Provider and EMP Service Consumer) MUST either ignore messages with duplicate `MessageIDs` or behave *idempotently* (repeated processing of the same event MUST NOT produce undesirable results - multiplied user interaction, charging, misreporting etc). For the purpose of uniqueness checking the `MessageIDs` are kept at least for a guaranteed period – the length of this period is set by uniqueness period deployment parameter (see the Uniqueness period in the section 3.6, "Deployment Specific Parameters"). It means that it is guaranteed that the message with previously used `MessageID` MUST NOT be processed if the last use of the id occurred within uniqueness period.

4.3.2 Message exchange patterns

On top of the generic message the EMP service specifications may define more complex message exchange patterns like following *In-Out, In-Optional-Out, Out-In, Out-Optional-In.*

4.3.2.1 In-Only

Service Consumer sends a message to the Service Provider. Service Provider processes the message and produces no response message.

4.3.2.2 In-Out

Service Consumer sends a message to the Service Provider. Service Provider processes the message and produces one response message with `RelatesTo` attribute filled with the `MessageID` of the original message.

4.3.2.3 In-Optional-Out

Service Consumer sends a message to the Service Provider. Service Provider processes the message and produces up to one response message with `RelatesTo` attribute filled with the `MessageID` of the original message.

4.3.2.4 Out-Only

M2M Service Provider sends a message to the Service Consumer. Service Consumer processes the message and produces no response message.

4.3.2.5 Out-Optional-In

Service Provider sends a message to the Service Consumer. Service Consumer processes the message and produces up to one response message with `RelatesTo` attribute filled with the `MessageID` of the original message.

4.3.2.6 In-Multi-Out

Service Consumer sends a message to the Service Provider. Service Provider processes the message and produces multiple response messages with `RelatesTo` attributes filled with the `MessageID` of the original message.

4.4 XML Elements and Types

The chapter discusses XML format of the EMP WS message.

Following definitions are based on XML 1.0 [XML] and defined using XML schemas compliant with W3C XML Schema recommendation [XMLSCHEMA].

The XML elements and types are defined in following namespaces:

`http://cz.o2.com/sMessage/Envelope`

- prefix used in this document: `env`
- defines XML type containing full EMP Service Message (`env:Envelope`) and Service Message pack (`env:Pack`)

`http://cz.o2.com/sMessage/Payload`

- prefix used in this document: `m2mPayload`
- defines abstract XML complex type `m2mPayload:Payload`, which is extended by all service specific message payloads

`http://cz.o2.com/sMessage/Acceptance`

- prefix used in this document: `acceptance`
- defines message header element and payload types used in the section 4.5, “Generic Message Acceptance/Rejection extension”.

The defined prefixes are used in this document only for readability's sake. Actual messages MAY use any prefix.

4.4.1 Message Pack

Message pack is an XML representation of multiple messages accompanied with optional metadata. Messages contained in the `Pack` MAY carry various payloads.

The global element of type `smsg:Pack` may serve as an element representing a set of messages (can be used as a top-level element or as a part of a more complex structure).

4.4.2 Envelope

Service message envelope is an XML complex type suitable for transport of the full EMP service message. It serves as a *wrapper* for the elements, which hold the message's data. Every single message MUST be in its own `Envelope` element.

4.4.2.1 Header

The message header contains the common metadata (`MessageID` and optional `RelatesTo`) and optionally other metadata defined in separate extensions.

4.4.2.2 Payload

The payload element has type `m2mPayload:Payload`. It is an abstract type. All of the actual payload types must extend it.

The attribute `xsi:type` `(xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance")` must be used to specify the actual type of the payload being carried in the message.

4.4.3 Acceptance

The application has several control levels and every request must go through all of them. In case the `acceptance:Respond` is enabled (it has `true` value), a client will receive a corresponding answer. If the feature is disabled, the client will not receive anything, even if some error occurred.

4.4.3.1 Element `acceptance:Respond`

Message header element optionally specifying behavior defined in the section 4.5, "Generic Message Acceptance/Rejection extension". The actual usage of this can be seen in the Example 5.1: AO SMS Message XML.

4.4.3.2 Result `acceptance:Accepted`

This is a message payload type. Message containing payload of this type confirms that the message referenced using `smsg:RelatesTo` was or certainly will be processed. See the Example 5.17: Acceptance report – success.

4.4.3.3 Result `acceptance:Rejected`

This is a message payload type. Message containing payload of this type confirms that the message referenced using `smsg:RelatesTo` was rejected from processing. This usually happens when there is an error in the XML request (wrong syntax, missing elements, message limit excess, etc). See the Example 5.18: Acceptance report – failure.

Detailed information about the XML files structure can be found in the [XSDSCHEMA].

4.5 Generic Message Acceptance/Rejection extension

In many real-life scenarios there is often a need for explicit request confirmation or rejection.

A few examples:

- a partner application may want to *accept* an incoming SMS message to certify that the format is valid and O2 can charge the customer,
- O2 accepts (or rejects) SMS messages sent from partner to mobile phone when they (do not) meet agreed requirements i.e. message length, recipient's validity etc.

The actual usage of the Acceptance/Rejection MUST be documented in a deployment specific documentation in parameter *Request Acceptance/Rejection* usage (see the section 3.6, "Deployment Specific Parameters").

The originator of the request MAY ask for an explicit *acceptance/rejection* response using attribute `respond` of the Request (see the Table 6: Possible values of the respond attribute for values description). The receiving party MUST react accordingly in case the reaction is defined in the deployment specific documentation (see the section 3.6, "Deployment Specific Parameters"), otherwise the response MAY not be generated.

The acceptance/rejection extension defines two *result* requests, which are listed in the Table 7: Acceptance/Rejection extension results.

Value	Description
`Never` (default value)	Receiving party MUST NOT generate any acceptance/rejection result (even if the deployment specification describes proper acceptance/rejection functionality).
`Rejects`	Receiving party MUST generate the acceptance/rejection result in case the request gets dropped from processing unless the functionality is not described in the deployment specification.
`Always`	Receiving party MUST generate the acceptance/rejection result unless the functionality is not described in the deployment specification.

Table 6: Possible values of the respond attribute

Result	Description
`Accepted`	The referenced request is *accepted*. This confirms that the referenced request passed the necessary validation and is (or certainly will be) processed. Results of the request processing may follow. See the Example 5.17: Acceptance report – success below.
`Rejected`	The referenced request was *rejected* from processing. This informs the originator that the request failed to pass the processing criteria and was removed from further processing. No further results will be sent. See the Example 5.18: Acceptance report – failure below.

Table 7: Acceptance/Rejection extension results

In case that the Rejected report is received with <acceptance:Description>daily limit exceeded</acceptance:Description> or <acceptance:Description>monthly limit exceeded</acceptance:Description>, valid reaction is requested. In case of daily limit exceeded partner application MUST NOT send any more AO SMS to the end of current day. In case of monthly limit exceeded, partner application MUST NOT send any more AO SMS to the end of current Month.

5 AO SMS Messages

The Send operation request Example 5.1: AO SMS Message XML ensures delivery of AO SMS to a mobile phone with the number +420720333444. All request elements are discussed in detail in this chapter.

Example 5.1: AO SMS Message XML

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <ns4:Send xmlns="http://cz.o2.com/sMessage/Envelope"
xmlns:ns2="http://cz.o2.com/sMessage/Acceptance"
      xmlns:ns3="http://cz.o2.com/Messaging"
xmlns:ns4="http://cz.o2.com/sMessage/sMessageService">
         <ns4:Pack>
            <Envelope>
               <Header>
                  <MessageID>TEST-001</MessageID>
                  <ns2:Respond>Always</ns2:Respond>
               </Header>
               <Payload xsi:type="ns3:SmsMessage"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
                  <ns3:DeliveryReports>true</ns3:DeliveryReports>
                  <ns3:Originator
xsi:type="ns3:SmsApplicationNumberAddress">
                     <ns3:ApplicationNumber>9999600</ns3:ApplicationNumber>
                  </ns3:Originator>
                  <ns3:Recipient xsi:type="ns3:SmsPhoneNumberAddress">
                     <ns3:PhoneNumber>+420123456789</ns3:PhoneNumber>
                  </ns3:Recipient>
                  <ns3:Content xsi:type="ns3:TextSmsContent">
                     <ns3:Text>Test text</ns3:Text>
                  </ns3:Content>
               </Payload>
            </Envelope>
         </ns4:Pack>
      </ns4:Send>
   </S:Body>
</S:Envelope>
```

The elements Envelope, Header and Payload were described in the section 4.4, "XML Elements and Types". See that section or [XSDSCHEMA] for more information.

5.1 Element DeliveryReports

This element has only two possible values – *true* and *false*. It specifies if the client will receive delivery reports or not. Delivery reports indicate whether a message was delivered to the end device or not.

BA MAY request a delivery report, but there MAY be a special price for it.

5.2 Element `Originator`

All messages sent by a O2 business partner to end users, i.e. AO messages, must have populated originator number. The number identifies the message originating application and allows recipients to reply to the originator. Typically the originator number is O2 partner’s application number (aka short number) received from O2. Additionally the partner MAY attach a numeric suffix to this number. The total length of the application number including the suffix MUST NOT be greater than 20.

As the application number is recognized only within the O2 network it is not suitable when the business partner sends messages into other mobile networks too. In that case the partner should ask O2 for a long number, i.e. a standard mobile number in the international format recognized in any mobile network and use it as the originator number.

The third option how to specify the originator number is a `nickname`, which is optional and has a restriction – mobile phone users from any network MUST NOT send messages to the `nickname`.

In a special scenario a partner can specify any valid long number as the originator number. This scenario is not available by default due to security reasons. It can be enabled by O2 upon request for a given business partner.

As was stated before there are three options how to specify the originator:

- `application-number` – short BA number,
- `phone-number` – MSISDN (in the international format),
- `nickname` – alphanumeric number.

Example 5.2: Originator specification with the `application-number`

```
<msg:Originator xsi:type="msg:SmsApplicationNumberAddress">
  <msg:ApplicationNumber>100001</msg:ApplicationNumber>
</msg:Originator>
```

Example 5.3: Originator specification with the `phone-number`

```
<msg:Originator xsi:type="msg:SmsPhoneNumberAddress">
  <msg:PhoneNumber>+420720333444</msg:PhoneNumber>
</msg:Originator>
```

Example 5.4: Originator specification with the `nickname`

```
<msg:Originator xsi:type="msg:SmsAlphanumericAddress">
  <msg:Alphanumeric>NickName</msg:Alphanumeric>
</msg:Originator>
```

5.2.1 Alphanumeric number

The alphanumeric number will be placed into standard attribute originator of the messaging interface. Its length is limited to 11 characters. This does not affect the CDR generation and journaling (the default BA number will be generated in case that the alphanumeric originator is used).

5.3 Element `Recipient`

This element has only one option how to specify a recipient – the MSISDN (the number MUST be in the international format). Every SMS message can have only one recipient.

5.4 Element `Content`

SMS message without any content can’t be send – the content MUST be specified. The type is set in the element `<sms:sms-content>`. There are two kinds of the content:

- Text – used for an ordinary text message
- Binary – used for more complex data (images, etc.)

5.4.1 Text Messages

Example 5.5: Text content specification

```
<msg:Content xsi:type="msg:TextSmsContent">
  <msg:Text>SMS Message Text</msg:Text>
</msg:Content>
```

5.4.2 Binary Messages

The data in binary messages are encoded in so-called *hexa-string*. It is a hexadecimal representation of individual bytes, without spaces. There is no check whether the data is meaningful or valid; it's up to the phone terminal if it can handle the binary message or not. See the Example 5.6: Binary content type specification below.

Example 5.6: Binary content type specification

```
<msg:Content xsi:type="msg:BinarySmsContent">
    <msg:Pid>0</msg:Pid>
    <msg:Udh>0B0504158A00000003690301</msg:Udh>
    <msg:Dcs>245</msg:Dcs>
    <msg:Bytes>30000000020100004 81C01 ......</msg:Bytes>
</msg:Content>
```

5.4.3 Text Message Encoding

GSM standards allow sending SMS text messages in two formats – GSM-7 and UTF-16.

GSM-7

- 7 bit encoding roughly equivalent to 7-bit US-ASCII
- Maximum length of the 7bit SMS message is 160 characters

UTF-16

- UTF-16 character set
- Maximum length of the Unicode message is 70 characters

5.4.3.1 Sending Text Messages in Different Encoding

EMP WS message specification provides several ways how to provide message text content.

GSM-7 encoding

EMP Service Consumer sends the message with a text containing only characters from GSM-7 [GSM338] and the extension defined in [3GPP23038] using the `Gsm7TextSmsContent` (see the [XSDSCHEMA] for more information). The EMP Service Consumer MUST replace all special characters with their equivalents from GSM-7 encoding in this case. If the message text contains characters from outside of the GSM-7 encoding, the processing of the message fails. The advantage of using this encoding is having the full 160 characters available.

Note

Characters from the extension defined in [3GPP23038] are encoded using 2 characters in the physical SMS message, reducing the total maximum length of the message by one character per character from the extension set.

UTF-16

EMP Service Consumer sends the text in full Unicode format (simply as a string in the encoding of the XML document) using the `TextSmsContent`. Any characters from the Unicode set can be used. However the character sets supported by mobile handsets are often limited to European alphabets. The length of the content in this case MUST NOT exceed 70 characters.

Custom encoding

EMP Service Consumer encodes the message into a binary SMS message and sends it as a standard binary SMS using `BinarySmsContent` (see the [XSDSCHEMA] for more information).

5.4.4 Multipart SMS messages

The EMP does not provide automatic support for long message splitting. The EMP Service Consumer is responsible for splitting the long messages into parts and sending them as individual messages. To allow for concatenating of such messages in the handset, the messages MUST be equipped with a *user data header* signaling the multipart message (see the Example 5.8: Content specification with the user data header – part 1 and Example 5.9: Content specification with the *user data header* – part 2). See [GSM340] for complete description.

There is an Example 5.7: The headers of a message consist of 2 GSM messages to show how the header data can be set in a multipart message.

Example 5.7: The headers of a message consist of 2 GSM messages

```
                              message position
                              ↓↓
1st message header: 0500031C0201
2nd message header: 0500031C0202
                            ↑↑
                         total number of messages
```

Since UDH is part of the payload, the number of available characters per segment is lower.

Example 5.8: Content specification with the *user data header* – part 1

```
<msg:Content xsi:type="msg:Gsm7TextSmsContent">
  <msg:Udh>0500031C0201</msg:Udh>
  <msg:Text>Message text - part 1</msg:Text>
</msg:Content>
```

Example 5.9: Content specification with the *user data header* – part 2

```
<msg:Content xsi:type="msg:Gsm7TextSmsContent">
  <msg:Udh>0500031C0202</msg:Udh>
  <msg:Text>Message text - part 2</msg:Text>
</msg:Content>
```

5.5 Receive Operation Flow Example

The `Send` operation Example 5.1: AO SMS Message XML causes generation of several related response messages like an acceptance report and a delivery report. The `Receive` operation must be invoked to obtain these messages. See the Example 5.10: Message receiving below. The operation's response is showed at the Example 5.11: Delivery report.

Example 5.10: Message receiving

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:smsg="http://cz.o2.com/sMessage/sMessageService">
  <soapenv:Header/>
  <soapenv:Body>
    <smsg:Receive/>
  </soapenv:Body>
</soapenv:Envelope>
```

Example 5.11: Delivery report

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <sMsgSvc:ReceiveResponse
xmlns:sMsgSvc="http://cz.o2.com/sMessage/sMessageService">
         <sMsgSvc:Pack>
            <sMsg:Header xmlns:sMsg="http://cz.o2.com/sMessage/Envelope">
               <sMsgSvc:PackID>2d1d1e-rq-
bb5.137bc0192fd.137c60626a4@lxecs401</sMsgSvc:PackID>
            </sMsg:Header>
            <sMsg:Envelope xmlns:sMsg="http://cz.o2.com/sMessage/Envelope">
               <sMsg:Header>
                  <sMsg:MessageID>2d1d1e-rq-
bb5.137bc0192fd.137c60626a4@lxecs401</sMsg:MessageID>
                  <sMsg:RelatesTo>TEST-001</sMsg:RelatesTo>
               </sMsg:Header>
               <sMsg:Payload xsi:type="msg:SmsDeliveryReport"
xmlns:msg="http://cz.o2.com/Messaging"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
                  <msg:Status>Delivered</msg:Status>
                  <msg:Timestamp>2012-06-
07T10:18:40.000+02:00</msg:Timestamp>
                  <msg:Originator
xsi:type="msg:SmsApplicationNumberAddress">
                     <msg:ApplicationNumber>9999600</msg:ApplicationNumber>
                  </msg:Originator>
                  <msg:Recipient xsi:type="msg:SmsPhoneNumberAddress">
                     <msg:PhoneNumber>+420123456789</msg:PhoneNumber>
                  </msg:Recipient>
               </sMsg:Payload>
            </sMsg:Envelope>
         </sMsgSvc:Pack>
      </sMsgSvc:ReceiveResponse>
   </soap:Body>
</soap:Envelope>
```

The response can be either confirmed (see the Example 5.12: Message confirmation and following responses) or rejected (see the Example 5.15: Message rejection and following response) with the correct `PackID`.

Example 5.12: Message confirmation

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:smsg="http://cz.o2.com/sMessage/sMessageService">
  <soapenv:Header/>
  <soapenv:Body>
    <smsg:Confirm>
      <smsg:PackID>
        ad8bb4-rq-20.12c4f691a0e.12c5e51721d@madansportapp02
      </smsg:PackID>
    </smsg:Confirm>
  </soapenv:Body>
</soapenv:Envelope>
```

Example 5.13: Message confirmation – success

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <sMsgSvc:ConfirmResponse
      xmlns:sMsgSvc="http://cz.o2.com/sMessage/sMessageService" />
  </soap:Body>
</soap:Envelope>
```

Example 5.14: Message confirmation – failure

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <soap:Fault>
      <faultcode>soap:Client.ConfirmFailed</faultcode>
      <faultstring>Pack confirmation failed.</faultstring>
    </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

Example 5.15: Message rejection

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:smsg="http://cz.o2.com/sMessage/sMessageService">
  <soapenv:Header />
  <soapenv:Body>
    <smsg:Reject>
      <smsg:PackID>
        ad8bb4-rq-20.12c4f691a0e.12c5e51721d@madansportapp02
      </smsg:PackID>
    </smsg:Reject>
  </soapenv:Body>
</soapenv:Envelope>
```

Example 5.16: Message rejection response

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <sMsgSvc:RejectResponse
      xmlns:sMsgSvc="http://cz.o2.com/sMessage/sMessageService" />
  </soap:Body>
</soap:Envelope>
```

5.5.1 Acceptance reports

If acceptance reports are enabled, they can be fetched from the EMP using the `Receive` operation. They indicate if an error occurred during the message processing. See the Example 5.17: Acceptance report – success and Example 5.18: Acceptance report – failure.

Example 5.17: Acceptance report – success

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <sMsgSvc:ReceiveResponse
xmlns:sMsgSvc="http://cz.o2.com/sMessage/sMessageService">
         <sMsgSvc:Pack>
            <sMsg:Header xmlns:sMsg="http://cz.o2.com/sMessage/Envelope">
               <sMsgSvc:PackID>1865aad-rq-
1c9f.137bc01735a.137c605478c@lxecs401</sMsgSvc:PackID>
            </sMsg:Header>
            <sMsg:Envelope xmlns:sMsg="http://cz.o2.com/sMessage/Envelope">
               <sMsg:Header>
                  <sMsg:MessageID>1865aad-rq-
1c9f.137bc01735a.137c605478c@lxecs401</sMsg:MessageID>
                  <sMsg:RelatesTo>TEST-001</sMsg:RelatesTo>
               </sMsg:Header>
               <sMsg:Payload xsi:type="acceptance:Accepted"
xmlns:acceptance="http://cz.o2.com/sMessage/Acceptance"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
            </sMsg:Envelope>
         </sMsgSvc:Pack>
      </sMsgSvc:ReceiveResponse>
   </soap:Body>
</soap:Envelope>
```

Example 5.18: Acceptance report – failure

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <sMsgSvc:ReceiveResponse
xmlns:sMsgSvc="http://cz.o2.com/sMessage/sMessageService">
         <sMsgSvc:Pack>
            <sMsg:Header xmlns:sMsg="http://cz.o2.com/sMessage/Envelope">
               <sMsgSvc:PackID>e4b8a-rq-
8d.13797d4f027.13797d5bb69@lxecs401</sMsgSvc:PackID>
            </sMsg:Header>
            <sMsg:Envelope xmlns:sMsg="http://cz.o2.com/sMessage/Envelope">
               <sMsg:Header>
                  <sMsg:MessageID>e4b8a-rq-
8d.13797d4f027.13797d5bb69@lxecs401</sMsg:MessageID>
                  <sMsg:RelatesTo>TEST-002</sMsg:RelatesTo>
               </sMsg:Header>
               <sMsg:Payload xsi:type="acceptance:Rejected"
xmlns:acceptance="http://cz.o2.com/sMessage/Acceptance"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
                  <acceptance:Description>unknown nickname
ABCD></acceptance:Description>
               </sMsg:Payload>
            </sMsg:Envelope>
         </sMsgSvc:Pack>
      </sMsgSvc:ReceiveResponse>
   </soap:Body>
```

```
</soap:Envelope>
```

Example 5.19: Acceptance report – failure – daily limit exceeded

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <sMsgSvc:ReceiveResponse
xmlns:sMsgSvc="http://cz.o2.com/sMessage/sMessageService">
         <sMsgSvc:Pack>
            <sMsg:Header xmlns:sMsg="http://cz.o2.com/sMessage/Envelope">
               <sMsgSvc:PackID>1def3f5-rq-
352.127f7149db9.127fc78bd9c@lxecs401</sMsgSvc:PackID>
            </sMsg:Header>
            <sMsg:Envelope xmlns:sMsg="http://cz.o2.com/sMessage/Envelope">
               <sMsg:Header>
                  <sMsg:MessageID>1def3f5-rq-
352.127f7149db9.127fc78bd9c@lxecs401</sMsg:MessageID>
                  <sMsg:RelatesTo>TEST-004</sMsg:RelatesTo>
               </sMsg:Header>
               <sMsg:Payload xsi:type="acceptance:Rejected"
xmlns:acceptance="http://cz.o2.com/sMessage/Acceptance"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
                  <acceptance:Description>daily limit
exceeded</acceptance:Description>
               </sMsg:Payload>
            </sMsg:Envelope>
         </sMsgSvc:Pack>
      </sMsgSvc:ReceiveResponse>
   </soap:Body>
</soap:Envelope>
```

Example 5.20: Acceptance report – failure – monthly limit exceeded

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <sMsgSvc:ReceiveResponse
xmlns:sMsgSvc="http://cz.o2.com/sMessage/sMessageService">
         <sMsgSvc:Pack>
            <sMsg:Header xmlns:sMsg="http://cz.o2.com/sMessage/Envelope">
               <sMsgSvc:PackID>1def3f5-rq-
363.127f7149db9.127fc7c9bf8@lxecs401</sMsgSvc:PackID>
            </sMsg:Header>
            <sMsg:Envelope xmlns:sMsg="http://cz.o2.com/sMessage/Envelope">
               <sMsg:Header>
                  <sMsg:MessageID>1def3f5-rq-
363.127f7149db9.127fc7c9bf8@lxecs401</sMsg:MessageID>
                  <sMsg:RelatesTo>JCaa006</sMsg:RelatesTo>
               </sMsg:Header>
               <sMsg:Payload xsi:type="acceptance:Rejected"
xmlns:acceptance="http://cz.o2.com/sMessage/Acceptance"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
                  <acceptance:Description>monthly limit
exceeded</acceptance:Description>
```

```
               </sMsg:Payload>
            </sMsg:Envelope>
         </sMsgSvc:Pack>
      </sMsgSvc:ReceiveResponse>
   </soap:Body>
</soap:Envelope>
```

6 AO WAP Push Interface

Every partner application has two options:

- send a message,
- process status reports (optional).

XML elements in the WAP Push Interface have basically the same XML structure as the SMS messages – they MUST have the `xsi:type` for the WAP Interface instead of the SMS (see the Example 6.1: WAP Push xsi:type declaration and the complete WAP Push request at Example 6.2: WAP Push message XML).

WAP Push Interface has the same rules as other interfaces of the EMP. It MUST follow the rules described in the chapters 3, 4. Especially the `MessageID` attribute must be unique otherwise the message will not be processed and en error will be generated.

Example 6.1: WAP Push `xsi:type` declaration

```
<env:Payload xsi:type="msg:WapPushMessage">
```

All the XML elements are very similar to the other interfaces – all the information can be found there (see the chapters 3, 4).

WAP Push message is in fact a binary SMS message. Single WAP Push request can be split into several binary SMS messages if the content of the element `<msg:Description>` is longer than 140 characters. The length of the description is not limited, but each 140 characters will result into an extra binary SMS message.

The complete WAP Push request and related responses are shown below.

Example 6.2: WAP Push message XML

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:smes="http://cz.o2.com/sMessage/sMessageService"
xmlns:env="http://cz.o2.com/sMessage/Envelope">
  <soapenv:Header/>
  <soapenv:Body>
    <smes:Send>
      <smes:Pack>
        <env:Envelope>
          <env:Header>
            <env:MessageID>WP001</env:MessageID>
          </env:Header>
          <env:Payload xsi:type="msg:WapPushMessage"
xmlns:msg="http://cz.o2.com/Messaging"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
           <msg:Recipient xsi:type="msg:WapPushPhoneNumberAddress">
               <msg:PhoneNumber>+420123456789</msg:PhoneNumber>
           </msg:Recipient>
           <msg:Content xsi:type="msg:WapPushContent">
               <msg:Url>http://wap.eurotel.cz/</msg:Url>
               <msg:Description>Eurotel Wap</msg:Description>
           </msg:Content>
          </env:Payload>
        </env:Envelope>
      </smes:Pack>
    </smes:Send>
  </soapenv:Body>
</soapenv:Envelope>
```

Example 6.3: WAP Push response – success

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <sMsgSvc:ReceiveResponse
xmlns:sMsgSvc="http://cz.o2.com/sMessage/sMessageService">
         <sMsgSvc:Pack>
            <sMsg:Header xmlns:sMsg="http://cz.o2.com/sMessage/Envelope">
               <sMsgSvc:PackID>28842f-rq-
bc1.137bc0192fd.137c6103dca@lxecs401</sMsgSvc:PackID>
            </sMsg:Header>
            <sMsg:Envelope xmlns:sMsg="http://cz.o2.com/sMessage/Envelope">
               <sMsg:Header>
                  <sMsg:MessageID>28842f-rq-
bc1.137bc0192fd.137c6103dca@lxecs401</sMsg:MessageID>
                  <sMsg:RelatesTo>WP001</sMsg:RelatesTo>
               </sMsg:Header>
               <sMsg:Payload xsi:type="msg:SmsDeliveryReport"
xmlns:msg="http://cz.o2.com/Messaging"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
                  <msg:Status>Delivered</msg:Status>
                  <msg:Timestamp>2012-06-
07T10:29:42.000+02:00</msg:Timestamp>
                  <msg:Originator
xsi:type="msg:SmsApplicationNumberAddress">
                     <msg:ApplicationNumber>9999600</msg:ApplicationNumber>
                  </msg:Originator>
                  <msg:Recipient xsi:type="msg:SmsPhoneNumberAddress">
                     <msg:PhoneNumber>+420123456789</msg:PhoneNumber>
                  </msg:Recipient>
               </sMsg:Payload>
            </sMsg:Envelope>
         </sMsgSvc:Pack>
      </sMsgSvc:ReceiveResponse>
   </soap:Body>
</soap:Envelope>
```

7 MO Messages

BA receives MO messages via the operation `Receive`. The MO messages must comply with the same XML schemas as AO requests thus they have exactly the same structure.

Whenever an MO Message is sent, the EMP produces an XML document (see the example below).

Example 7.1: MO SMS Message XML

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <sMsgSvc:ReceiveResponse
xmlns:sMsgSvc="http://cz.o2.com/sMessage/sMessageService">
         <sMsgSvc:Pack>
            <sMsg:Header xmlns:sMsg="http://cz.o2.com/sMessage/Envelope">
<sMsgSvc:PackID>incomingMsg:SMSC7:0005T00002:120607103600:9999600001:+42060
2196891</sMsgSvc:PackID>
            </sMsg:Header>
            <sMsg:Envelope xmlns:sMsg="http://cz.o2.com/sMessage/Envelope">
               <sMsg:Header>
<sMsg:MessageID>incomingMsg:SMSC7:0005T00002:120607103600:9999600001:+42060
2196891</sMsg:MessageID>
               </sMsg:Header>
               <sMsg:Payload xsi:type="msg:SmsMessage"
xmlns:msg="http://cz.o2.com/Messaging"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
                  <msg:CreationTime>2012-06-
07T10:36:00.000+02:00</msg:CreationTime>
                  <msg:Originator xsi:type="msg:SmsPhoneNumberAddress">
                     <msg:PhoneNumber>+420123456789</msg:PhoneNumber>
                  </msg:Originator>
                  <msg:Recipient
xsi:type="msg:SmsApplicationNumberAddress">
<msg:ApplicationNumber>9999600001</msg:ApplicationNumber>
                  </msg:Recipient>
                  <msg:Content xsi:type="msg:Gsm7TextSmsContent">
                     <msg:Pid>0</msg:Pid>
                     <msg:Text>Test</msg:Text>
                  </msg:Content>
               </sMsg:Payload>
            </sMsg:Envelope>
         </sMsgSvc:Pack>
      </sMsgSvc:ReceiveResponse>
   </soap:Body>
</soap:Envelope>
```