International Standard

ISO 28005-1

Second edition
2024-12

Ships and marine technology — Electronic port clearance (EPC) —

Part 1: Message structures and application programming interfaces

Navires et technologie maritime — Opérations portuaires assistées par systèmes électroniques —

Partie 1: Structures des messages et interfaces de programmation des applications

Reference number
ISO 28005-1:2024(en)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Page

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of ISO document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

ISO draws attention to the possibility that the implementation of this document may involve the use of (a) patent(s). ISO takes no position concerning the evidence, validity or applicability of any claimed patent rights in respect thereof. As of the date of publication of this document, ISO had not received notice of (a) patent(s) which may be required to implement this document. However, implementers are cautioned that this may not represent the latest information, which may be obtained from the patent database available at www.iso.org/patents. ISO shall not be held responsible for identifying any or all such patent rights.

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT), see www.iso.org/iso/foreword.html.

This document was prepared by Technical Committee ISO/TC 8, *Ships and marine technology*, Subcommittee SC 11, *Intermodal and short sea shipping.*

This second edition cancels and replaces the first edition (ISO 28005-1:2013), which has been technically revised.

The main changes are as follows:

— a general introduction to all documents in the ISO 28005 series, including the structure of XML and XSD files and general XML type definitions, has been added in Clauses 4 to 6;

— Clause 7 defining the general design principles for the ISO 28005 series has been added;

— Clause 8 has added general message exchange patterns that can be referenced in message implementation guides;

— Clause 9 has added a more general multi-part message structure based on the HTTP multi-part form structure. This includes a formalization of how attachments to the XML based message body can be added to the message. It also includes possibilities for using EDIFACT or JSON as a message body or as attachments to the standardized XML message body or adding encrypted message parts as attachments (detailed in Clauses 11 and 12);

— the message header has been updated so that it contains sufficient information to do frontend message processing before forwarding the message to the service specific software modules (see Clause 10);

— support for digital signatures has been added in Clause 13. Digital signature certificates can be attached to a message (see Clause 14);

— a definition of a HTTP-based transport protocol has been added in Clauses 15 to 17;

— a possibility for access authorization has been added in Clause 18;

— a definition for the structure of message implementation guides has been added in Clause 19;

— message implementation guides for access authorization have been added in Annex B, and a maritime single window and mandatory reporting systems have been added in Annex C.

A list of all parts in the ISO 28005 series can be found on the ISO website.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

Introduction

This document contains technical specifications that facilitate an efficient exchange of electronic information between ships and shore parties for coastal transit or port calls. It defines the general message format, general message exchange patterns and a message transfer protocol. Other parts of the ISO 28005 series define data models for various types of message transfers.

The message transfer protocol specified in this document uses HTTP over TLS. While many HTTP type application program interfaces (API) are built on the representational state transfer (REST) principle, this document does not use REST. APIs developed according to this document can support physical services such as ordering tugs or pilots, where state changes cannot be guaranteed to be compliant with REST.

This document can be used as a specification of a message format that can be transmitted over other transport protocols than the one defined in this document.

This document is aligned with the IMO FAL guidelines on authentication, integrity and confidentiality in information exchanges via maritime single windows and related services (IMO FAL.5/Circ.46)[1] and the IMO FAL guidelines for harmonized communication and electronic exchange of operational data for port calls, 31 March 2023 (IMO FAL.5/Circ.52).[2] The specifications in this document are aligned with the requirements in ISO 23807.[3]

Ships and marine technology — Electronic port clearance (EPC) —

Part 1:
Message structures and application programming interfaces

1 Scope

This document defines the principles, methods and requirements for message exchanges between ships, ship representatives, and other shore parties via a peer-to-peer communication system. This document defines the message structure, including how the data content is assembled from other parts of the ISO 28005 series, and how digital signatures for authentication, integrity, and confidentiality of the message can be used. It also specifies a transport protocol, the basic message exchange patterns, and the protocol related roles of each party in the message exchange. Furthermore, it specifies how more specific message implementation guides (MIGs) are provided for each type of communication application.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO 3166-1, *Codes for the representation of names of countries and their subdivisions — Part 1: Country code*

ISO 3166-2, *Codes for the representation of names of countries and their subdivisions — Part 2: Country subdivision code*

ISO 6709, *Standard representation of geographic point location by coordinates*

RFC 1952, *GZIP file format specification version 4.3*

RFC 3986, *Uniform resource identifier (URI): Generic syntax.*

RFC 5246, *The transport layer security (TLS) protocol version 1.2*

RFC 7578, *Returning values from forms: multipart/form-data*

RFC 8446, *The transport layer security (TLS) Protocol Version 1.3*

UN/EDIFACT code list 3035 – Party function code qualifier, Release D.00A, https://service.unece.org/trade/untdid/d00a/tred/tred3035.htm

UN/EDIFACT code list 3139 - Contact function code, Release D.23A, https://service.unece.org/trade/untdid/latest/tred/tred3139.htm

3 Terms, definitions, and abbreviated terms

3.1 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

ISO and IEC maintain terminology databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at https://www.electropedia.org/

3.1.1
HTTP
hypertext transfer protocol
client/server communication protocol used to transfer information in the World Wide Web

Note 1 to entry: HTTP is defined in RFC 2068[4] as a generic and stateless application-level protocol for distributed and collaborative, information systems.

Note 2 to entry: In this document, the term HTTP is used to mean HTTP implemented over *transport layer security* (3.1.2), i.e. *HTTPS* (3.1.3), unless otherwise specified.

3.1.2
TLS
transport layer security
protocol for secure communication over the internet

Note 1 to entry: TLS is defined in RFC 5246 (Version 1.2) and RFC 8446 (Version 1.3).

3.1.3
HTTPS
hypertext transfer protocol secure
HTTP (3.1.1) over *transport layer security (TLS)* (3.1.2)

Note 1 to entry: HTTPS is an extension of HTTP used for secure communication and is defined in RFC 2818.[6] It uses TLS to encrypt the communication.

Note 2 to entry: In this document, the term HTTP is used to mean HTTP implemented over TLS, i.e. HTTPS, unless otherwise specified.

3.1.4
PKI
public key infrastructure
set of hardware, software, people, policies and procedures needed to create, manage, distribute, use, store and revoke digital certificates and manage public-key encryption

[SOURCE: ISO 20415:2019, 3.14]

3.1.5
digital signature
data appended to, or cryptographic transformation of, a data unit that allows a recipient of the data unit to prove the source and integrity of the unit and protect against forgery by, for example, the recipient

[SOURCE: ISO 20415:2019, 3.5]

3.1.6
API
application programming interface
collection of communication methods and associated parameters used by a *client* (3.1.11) to exchange information with a *server* (3.1.12)

Note 1 to entry: In this document, the term API is used in the meaning of one specific arrow in the sequence diagram (see Figure 2). Thus, a session will normally require the use of several APIs. This document describes one general *API access point* (3.1.17) that can be used by all APIs by changing the number of message components as well as the contents of one or more message parts.

3.1.7
API access point
application programming interface access point
collection of a specific communication method, one or more general message parts, and related semantics for formatting the message parts that can be used to realize *application programming interfaces (APIs)* (3.1.6)

3.1.8
authorization
granting a *sender* (3.1.9) the right to use one or more *application programming interface access points* (3.1.7)

3.1.9
sender
party that initiates a *session* (3.1.18) by sending a *request* (3.1.15) to a receiver

[SOURCE: IMO FAL.5/Circ.46:2022, modified — the definition has been shortened.]

3.1.10
receiver
party that receives a *request* (3.1.15) from a *sender* (3.1.9), executes a *service* (3.1.14), and returns a status message

[SOURCE: IMO FAL.5/Circ.46:2022, modified — the definition has been shortened.]

3.1.11
client
unit that sends a message to a *server* (3.1.12) and expects a response

Note 1 to entry: The sender and receiver can be both a server and a client.

3.1.12
server
unit that listens for incoming messages from one or more *clients* (3.1.11) and responds to them

Note 1 to entry: The sender and receiver can be both a server and a client.

3.1.13
message
information payload of an *HTTP* (3.1.1) request or response between the *sender* (3.1.9) and *receiver* (3.1.10)

Note 1 to entry: Examples of messages are *request* (3.1.15), *message status* (3.1.16) and *service request status* (3.1.17).

Note 2 to entry: When the full HTTP request or response itself is referred to, the term HTTP message will be used.

3.1.14
service
result of a process performed by a *receiver* (3.1.10) after a *request* (3.1.15) has been delivered by a *sender* (3.1.9)

Note 1 to entry: In this document, service is a general term and includes that the receiver unconditionally accepts an incoming report without doing any processing of the content of the report.

3.1.15
request
message sent by a *sender* (3.1.9) to request a *service* (3.1.14)

[SOURCE: IMO FAL.5/Circ.46:2022, modified — the definition has been shortened.]

3.1.16
message status
message (3.1.13) sent from a *server* (3.1.12) to a *client* (3.1.11) to inform about the status of a received message

Note 1 to entry: A message status can be combined with a *service request status* (3.1.17) in one message.

Note 2 to entry: A message status will normally only consist of a message header. If the message status is combined with a service request status, the message can also contain other message parts.

3.1.17
service request status
message (3.1.13) sent from a *receiver* (3.1.10) to a *sender* (3.1.9) to inform about the status of a *service* (3.1.14)

Note 1 to entry: A service request status can be combined with a *message status* (3.1.16) in one message.

3.1.18
session
sequence of *messages* (3.1.13) that is related to one and the same initial *request* (3.1.15)

[SOURCE: IMO FAL.5/Circ.46:2022]

3.1.19
session context
one or more *sessions* (3.1.18) related to the same overarching operation or event

Note 1 to entry: Session contexts can be nested. As an example, several berth calls can be individual session contexts nested inside a port call session context, which may also be nested inside an entry into national waters session context.

3.1.20
HTTP session
hypertext transfer protocol session
message (3.1.13) exchanges that take place between establishment of an *HTTP* (3.1.1) connection and its closing

Note 1 to entry: One HTTP session can include sending more than one *request* (3.1.15).

3.1.21
MIG
message implementation guide
specific requirements to *message* (3.1.13) exchanges in conjunction with one or more concrete *services* (3.1.14)

3.1.22
EPC
electronic port clearance
services (3.1.14) a ship may request before, during, or after a port call

Note 1 to entry: EPC is used as a general term for all services a ship may request before, during, or after a port call. It is not limited to maritime single windows and other authority operated systems.

3.1.23
GLN
global location number
13-digit decimal number uniquely identifying a geographic position

Note 1 to entry: The list of registered numbers is maintained by GS1.[17]

3.1.24
UN/LOCODE
United Nations code for trade and transport locations
two or three letter code identifying a country and a trade location within it

Note 1 to entry: The countries have a two-letter code as listed in ISO 3166-1. The locations in the respective countries have a three-letter code as maintained by UNECE.[15]

3.1.25
MMSI
maritime mobile service identity
nine-digit decimal number uniquely identifying radiocommunication equipment

Note 1 to entry: MMSI is normally assigned by the national telecommunication authorities.

3.1.26
US-ASCII
7-bit character encoding defined in ISO/IEC 646

3.1.27
X.509
digital signature public certificate as defined in RFC 2459[33]

3.2 Abbreviated terms

For the purposes of this document, the following abbreviated terms apply.

API	application programming interface
DER	Distinguished encoding rules: Binary X.509 certificate format.
EDIFACT	United Nations electronic data interchange for administration, commerce and transport (UN/EDIFACT)
EPC	Electronic port clearance
GISIS	Global integrated shipping information system[16]
GLN	Global location number[17]
GS1	International industry organization maintaining code lists for international business operations NOTE 1 This includes universal product codes (“bar codes”) and GLN (see https://www.gs1.org/).
HTTP	Hypertext transfer protocol
HTTPS	Hypertext transfer protocol secure
IMO	International Maritime Organization
ISPS	International ship and port facility security NOTE 2 A list of ISPS facility codes is available in GISIS[16] under the link to Maritime Security.
JSON	JavaScript[a] object notation
MIG	Message implementation guide
MIME	Multipurpose internet mail extensions (see RFC 2045[7].)
MMSI	Maritime mobile service identity
MRS	Mandatory ship reporting system (see IMO A.851(20)[8])
MSW	Maritime single window
PCS	Port community system

PKI	Public key infrastructure
PMIS	Port management information system
REST	Representational state transfer
TCP/IP	Transfer control protocol over internet protocol (transport protocol for HTTP)
URI	Uniform resource identifier
	NOTE 3 In this document, URI is sometimes used interchangeably with URL. URL normally refers to a specific URI while an URI is meant as a more generic term for any URL.
URL	Uniform resource locator
SMDG	Formerly ship messaging design group, now user group for electronic data interchange (EDI) in the maritime container business. Maintaining the SMDG terminal list.[29]
TOS	Terminal operating system
TLS	Transport layer security
UNECE	United Nations Economic Commission for Europe
UN/LOCODE	United Nations code for trade and transport locations (see UNECE Recommendation 16[10] and list of codes[15])
VTIS	VTS information system
VTS	Vessel traffic services
XML	Extensible markup language (see W3C Recommendation XML 1.0[11])
XSD	XML schema definition language (defined in W3C recommendations XML schema structures,[12] and data types[13]).

[a] JavaScript is a registered trademark of Oracle Corporation. This information is given for the convenience of users of this document and does not constitute an endorsement by ISO or IEC.

4 Structure of XSD data type and object definitions

4.1 General

Each message specified in the ISO 28005 series consists of different parts as described in 7.16. Some of these parts are in XML[11] format. Each of the XML-formatted parts consist of several XML data objects with a corresponding XML data type definition.

The principles used in the ISO 28005 series for describing XML data types in XSD code are described in 4.2. The structure of the clauses in the ISO 28005 series that contain definitions of XML data types are described in 4.3. In addition to defining the syntax of the XML data objects in XSD code, 4.3 also describes how these clauses define the semantics of the data contained in the XML objects.

The XSD code is assembled into an XSD file that can be used for syntactic verification of XML message parts specified in the ISO 28005 series. In 4.4, the individual XSD code definitions are converted into a valid XSD file.

Data objects defined in ISO 28005-2:2021, which have been reused by definitions in this document are specified in 4.5.

4.2 Principles for XML descriptions in the ISO 28005 series

4.2.1 No use of XML attributes

The data types defined by this document do not use XML attributes. All information is contained within XML start and stop tags.

NOTE This makes it easy to create JSON versions from the XSD file defined in 4.4 by replacing XML tag names with the corresponding JSON object or attribute name.

4.2.2 Defaults for minOccurs and maxOccurs

The XSD specification[13] defines the default values for minOccurs and maxOccurs equal to one. This definition is used in the XSD code described in 4.3.3 to shorten the type specifications in the data type definition clauses by sometimes omitting 'minOccurs = "1"' and 'maxOccurs = "1"' where these otherwise would be expected to occur.

4.2.3 Signalling empty XML tags

A client or server can signal that an XML tag is empty (not in use) by not including any content between the start and end XML tags, or by omitting the tags themselves, if the latter is allowed by the corresponding schema definition i.e. that "minOccurs=0" is defined for the tag.

4.2.4 Order of child elements in XSD files

The XSD templates use "<xs:sequence>" to list child elements. This means that the XML file shall have child elements in exactly the same order as the XSD code. Optional child elements (minOccurs = "0") can be omitted.

NOTE The XSD code is automatically generated. The order of child elements shown in the XSD parts of the data type definitions are not always logical.

4.2.5 Character set

The character set used in the XML-formatted parts of the messages specified in the ISO 28005 series (see 7.16) is UTF-8.

4.2.6 Principles for defining types with code lists

Enumerated types, i.e. types that are associated with a fixed set of code values, are defined in one of the following ways.

a) When the code set is small, defined in this document, and not likely to be extended, the permitted code values are listed in the definition of the data type as a token with constraints. This means that the permitted code values will be included in the final XSD file.

b) If the code set is larger, defined in this document, and likely to be expanded in later editions of the relevant part of the ISO 28005 series, the data type is defined as an unconstrained token or number and the code set provided in tabular form. The permitted code values will not be included in the final XSD file.

c) When the code set is not defined in the document but maintained by an organization, the data type is defined as an unconstrained token or number. There will be a reference to where the code set is defined, and if necessary, constraints as to how the code set shall be used in the context of the document. The allowed code values will not be included in the final XSD file.

4.2.7 XSD name space for general XSD data types

All basic data types defined in the XSD data type standard,[14] and which are used in this document, use the name space "xs". Thus, the data type name are prefixed with "xs":. This corresponds to the header of the XSD file defined in 4.4 including the following attribute:

```
<xs:schema ...
  xmlns:xs="https://www.w3.org/2001/XMLSchema" ...
```

4.2.8 ISO 28005 name space

All data types defined in the ISO 28005 series are defined in the namespace "epc". Thus, the data type name is prefixed with "epc":.

This corresponds to the XSD file header including the following attribute:

```
<xs:schema ...
  xmlns:epc="https://standards.iso.org/iso/28005/" ...
```

4.2.9 Use of Xpath expressions

The ISO 28005 series uses Xpath expressions to identify individual data objects in XML message parts. The Xpath expression syntax is defined in the W3C Recommendation XML Path Language (Xpath).[23] The data type `epc:xpath` (see 5.11) has been defined to hold such expressions when used in XML message parts.

EXAMPLE Identification of a company name in the message header SenderId object is "/EPCMessageHeader/SenderId/Company". The full sender identity is "/EPCMessageHeader/SenderId".

4.3 Structure of clauses defining data types

4.3.1 Clause and data type name

All data types defined in the ISO 28005 series are given a name that is also included as the first part of the clause heading where the data type is defined.

The data type name follows the specifications for XML tag names, with the following additional constraints.

a) The data type name shall always end with the string "Type".

   NOTE Some data object names can also have the ending "Type". In that case, the corresponding data type name will have a trailing "TypeType".

b) Enumerated data type names shall end with "ContentType" instead of only "Type".

c) This document uses the "Upper Camel Case" in all core data types, i.e. the first letter is upper case and, when the tag name consists of a number of concatenated words, each of the words starts with an upper-case letter.

d) The name consists only of characters from the sets (A-Z), (a-z) and, exceptionally, (0-9). The names have been selected to be generally understandable in the context of ship-to-shore communication. The names use British English spelling without any special characters. Names are in singular form except where the data element contains a list of items, in which case the tag name is in plural form.

4.3.2 Definition

Each data type has a definition that is intended to give an unambiguous description of what the data element shall contain and in what context it is valid. The definition appears in the first subclause after the clause heading.

4.3.3 Type defined as XSD code

Each data type is defined as a section of XSD code.[14] This section only covers the actual data type definition and is not a valid XML or XSD document in itself.

See 4.4 for details on how a complete XSD file shall be constructed from the type definitions' XSD code.

4.3.4 Representation

The representation paragraph consists of additional requirements that define the semantics of the data type.

This can include how the information is coded, e.g. if distances are in km or nautical miles, or references to code lists that shall be used for enumerated types.

4.4 Creating valid XSD schema files

4.4.1 File structure

The minimum XSD format for defining the XSD schema file is shown in the below example, including comments starting with "<!–" that separate the different parts of the file from each other.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema  targetNamespace="https://www.iso.org/28005-1"
                xmlns:xs="https://www.w3.org/2001/XMLSchema"
                xmlns:epc="https://www.iso.org/28005">

<!--The definition of individual data types>

   <xs:complexType name ="DataElement1Type"> ... </xs:complexType>
   <xs:simpleType name ="DataElement2Type">  ... </xs:simpleType>
       ...
<!--The definition of the message part types>

   <xs:complexType name ="EPCMessageHeaderType">
       ...
   </xs:complexType>
   <xs:complexType name ="EPCMessageBodyType">
          ...
   </xs:complexType>
   <xs:complexType name ="EPCMessageSignatureType">
          ...
   </xs:complexType>

<!--The instantiation of the message parts>

   <xs:element name="EPCMessageHeader"
               type="EPCMessageHeaderTypeType"/>
   <xs:element name="EPCMessageBody" type="EPCMessageBodyType"/>
   <xs:element name="EPCMessageSignature"
               type="EPCMessageSignatureType"/>

</xs:schema>
```

The first four lines is the general preamble in an XSD file. This specifies encoding and any name-spaces being used. UTF-8 encoding shall be used (see also 9.5).

The subsequent lines define the different individual data types. All data types which are defined in this document, as well as in the data definition clauses of other parts of the ISO 28005 series, are expected to be included here.

The next lines define the three different message parts. Currently, these are the ones listed in Table 1.

Table 1 — Message parts defined in the XSD file

Part	Clause	Message part
EPCMessageHeader	10.10	The message header
EPCMessageBody	11.2	The message body part
EPCMessageSignature	14.3	The message signature part

The final part is the definitions related to the instantiation of the message parts.

4.4.2 Numbering of XSD files and message version code

Each part of the ISO 28005 series can contain definitions that change the structure of the XSD file. In this case, the relevant part of the ISO 28005 series is expected to be associated with a new XSD file number as illustrated in Figure 1.

Each XSD file contains definitions for all message parts listed in Table 1 as well as all necessary data type that are used by those message parts. Definitions from one part of the ISO 28005 series will be included in all following XSD file numbers. The first XSD file number that each part in the ISO 28005 series is associated with is specified in Annex A of that part. See 11.2.2 for more details.

NOTE Figure 1 is for illustration only. The actual assignment of XSD file numbers is determined by the parts that together define the related XSD file.

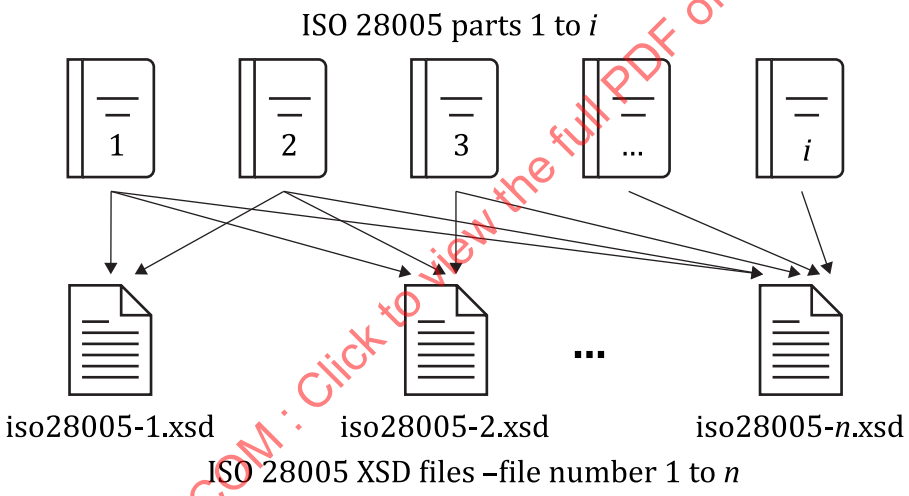


Figure 1 — Structure of XSD files related to the ISO 28005 series

The message version code is a mandatory part of the message header (see 10.10). It is a string consisting of two or three integer numbers separated by a full stop (see 6.19). These numbers, from left to right shall have the following values:

1. The first number shall be '2'. This corresponds to the protocol and message specifications contained in this document.

2. The second number shall correspond to the XSD file number to which the XML message is designed to be conformant. The data elements listed in ISO 28005-2:2021, Annex A as well as the elements listed in Annex A shall all be included in file number 1.

3. The optional third number can be used for local variants of the XSD file that are not defined by the ISO 28005 series. See 11.2 for a description of this mechanism.

The structure of the message header and body is defined so that clients that are designed to understand a specific XSD file number, safely can validate XML messages with the same or lower file number by using the XSD file. Lower numbered XSD files cannot be used to validate higher numbered XML messages.

If the client's parser is constructed to only look for the tags in one version of an XSD file, a client can safely parse also higher numbered XML files if unknown tags are discarded. A client that is designed to understand a higher number XSD file than indicated by an incoming message version code can parse the incoming message, but shall in that case parse the message in accordance with the XSD file number that the incoming message version code specifies, i.e. it shall not look for tags that are only defined in higher numbered XSD files.

NOTE This capability is ensured by only adding new data objects or attributes. Data objects or attributes will not be deleted from definitions, but some objects or attributes can be depreciated in other parts of the ISO 28005 series.

It is always possible to validate local variants with the correspondingly numbered XSD file. However, this will not check the syntax of any additional elements in the local variant. To do this, the XSD file corresponding to the local variant is required.

4.4.3 Location of XSD files

The XSD files can be found an address patterned after the following: https://standards.iso.org/iso/28005/-*n*/ed-*m*/en/. This applies to all parts of the ISO 28005 series, where the desired part number ("*n*") and edition ("*m*") is replaced with the relevant numbers in the URL.

EXAMPLE The XSD file corresponding to the definitions in ISO 28005-2:2021 is available on https://standards.iso.org/iso/28005/-2/ed-2/en/.

4.5 Reference to data types defined in ISO 28005-2:2021

All adapted XSD data types defined in ISO 28005-2:2021, Clause 5 have been reproduced in Clause 5. Several general data types in ISO 28005-2:2021.

Some of the definitions from ISO 28005-2:2021 have been extended in this document. A summary of the changes is listed in Table 2. Definitions in ISO 28005-2:2011[1)] remain valid but the old data types listed in Table 2 may have been depreciated by newer definitions in this document, which should be used for new implementations.

Table 2 — Data types modified from ISO 28005-2:2021

Data type	Definition in ISO 28005-2:2021	Definition in this document	Changes made
`epc:AuthenticatorType`	7.2.6	6.2	Changed authenticator role code list.
`epc:ContactInfoType`	6.3	6.4	Added URL attribute
`epc:LocationType`	7.10	6.9	Added GLN and MRN attributes.
`epc:PostalAddressType`	6.11	6.12	Changed `PostCodeCode` to `PostCode`
`epc:EPCMessageHeaderType`	7.2.2	10	Redefined (including all sub-types)
`epc:EPCRequestBodyType`	Annex A	11.2.1	Redefined and changed to `EPCMessageBodyType`
`epc:JournalNumberType`	7.9.22	6.16	Changed type name to `epc:ReferenceCodeType` (Ship stay reference)
`epc:MessageTypeContentType` `epc:OtherServiceRequestType`	7.7.2, 7.7.3	10	Replaced by `epc:MessageFunctionCodeContentType` `epc:ServiceTypeCodeContentType` and `epc:ServiceCodeContentType` New code values have been defined in Annex G.
`epc:RequestStatusType`	7.7.4	10	Replaced by `epc:StatusType`

1) Withdrawn.

5 Adapted XSD data types for ISO 28005

5.1 General

The adapted XSD data types used in the ISO 28005 series are based on the standard XML types defined in XML Schema Part 2.[13]

5.2 epc:anyURI – Generalized URI

5.2.1 Definition

This data type contains a valid generalized URI. This may be a URL, prefixed by “http”: or “https”:, or a file name, prefixed by “file”:.

5.2.2 Type

```
<xs:simpleType name="anyURI">
   <xs:restriction base="xs:anyURI"/>
</xs:simpleType>
```

5.2.3 Representation

All generalized URIs are permitted in this document. To ensure compatibility with older systems, URLs are restricted to use only the ASCII character set.

NOTE The “file”: type URI is used to refer to message attachments as described in 12.2.

5.3 epc: boolean – Boolean flag

5.3.1 Definition

This data type contains a flag that can have the logical values true or false.

5.3.2 Type

```
<xs:simpleType name="boolean">
   <xs:restriction base="xs:boolean"/>
</xs:simpleType>
```

5.3.3 Representation

All flag values allowed in the XSD standard are also allowed in this document. When the boolean type is used in contexts where the value represents an answer to a yes/no question, the true value shall represent “yes” while the false value shall represent “no”.

5.4 epc:date – General date

5.4.1 Definition

This data type contains a date without additional time of day or time zone information.

5.4.2 Type

```
<xs:simpleType name="date">
  <xs :restriction base="xs:date"/>
</xs:simpleType>
```

5.4.3 Representation

This is a date in the standard XSD format, without any time zone code in the value.

EXAMPLE "2002-09-10" (10th of September 2002).

Senders of date information should not include time zone information. Receivers should be prepared to accept a time zone code, but it shall be disregarded in further processing of the data.

5.5 epc:dateTime – Time and date, with time zone

5.5.1 Definition

This data type contains a date with additional time of day and time zone information.

5.5.2 Type

```
<xs:simpleType name="dateTime">
  <xs:restriction base="xs:dateTime"/>
</xs:simpleType>
```

5.5.3 Representation

This is a date and time in the standard XSD format, with a time zone code in the value.

Senders of date and time information shall include time zone information. Receivers should be prepared to accept values without time zone codes. In this case, the time zone is undefined and proper actions should be taken by the systems processing this data to ensure that this does not cause problems for the intended operation.

EXAMPLE 1 "2012-02-03T23:00:00+02:00" (3 February 2012, local time 23:00 hours in time zone UTC plus two hours).

EXAMPLE 2 "2012-02-03T21:00Z" (same time as Example 1 but transmitted as being in UTC).

NOTE The hour field can be 24 if minutes and seconds are zero. The seconds field can sometimes contain the value 60 (when leap seconds occur). Seconds can contain a decimal part followed by a period (".").

5.6 epc:decimal – decimal number

5.6.1 Definition

This data type is used to specify a quantity.

5.6.2 Type

```
<xs:simpleType name="decimal">
   <xs:restriction base="xs:decimal"/>
</xs:simpleType>
```

5.6.3 Representation

The decimal type represents a subset of the real numbers, which can be represented by decimal numerals. The value space of the decimal is the set of numbers that can be obtained by multiplying an integer i by a non-positive power n of 10, i.e. expressible as $i \times 10^{-n}$ where i and n are integers and $n \geqslant 0$. Precision is not reflected in this value space; the number 2.0 is not distinct from the number 2.00.

The decimal has a lexical representation consisting of a finite-length sequence of decimal digits separated by a period (".") as a decimal indicator. An optional leading sign is allowed. If the sign is omitted, "+" is assumed.

Leading and trailing zeros are optional. If the fractional part is zero, the period and following zeros can be omitted.

EXAMPLE –1.23, 12678967.543233, +100000.00, and 210 are all valid decimal numbers

NOTE The XSD format used in this document requires the use of the period (".") as the decimal sign.

5.7 epc:duration – Time duration

5.7.1 Definition

This data type is used to specify a duration in time.

5.7.2 Type

```
<xs:simpleType name="duration">
  <xs:restriction base="xs:duration"/>
</xs:simpleType>
```

5.7.3 Representation

This type can specify a time period. The general format is "[–]P*n*Y*n*M*n*DT*n*H*n*M*n*[.*n*]S", where [–] is an optional minus sign, *n* is a positive integer number, and [.*n*] is an optional decimal field for seconds. The number *n* ahead of Y, M, D, H and M, respectively, indicates the length of the duration in years, months, days, hours and minutes. If *n* is zero, it can be omitted together with the letter it precedes.

EXAMPLE P3DT10H30M is three days, 10 hours and 30 minutes; –P120D is a negative duration of 120 days.

5.8 epc:int – Integer number

5.8.1 Definition

This data type is used to specify an integer quantity.

5.8.2 Type

```
<xs:simpleType name="int">
  <xs:restriction base="xs:int"/>
</xs:simpleType>
```

5.8.3 Representation

The integer type is an integer in the range from –2147483648 to 2147483647 (inclusive).

5.9 epc:string – General string

5.9.1 Definition

This data type contains a general string that is mainly intended to be read by humans. There are no restrictions on the format of the string. See `epc:token` for a string type that is intended for computer use.

5.9.2 Type

```
<xs:simpleType name="string">
  <xs:restriction base="xs:string"/>
</xs:simpleType>
```

5.9.3 Representation

This is an UTF-8 character string where white space (space, tab, line feed etc.) is preserved.

Applications can put length constraints on the string. The server should be prepared to accept longer strings than what is specified by the application, but longer strings can be truncated. Thus, the client cannot rely on longer strings being fully processed or displayed.

5.10 epc:token – Computer-understandable string

5.10.1 Definition

The token data type is normally used for a text string that is mainly meant to be interpreted by a computer. This includes enumerated codes as well as other structured strings.

5.10.2 Type

```
<xs:simpleType name="token">
  <xs:restriction base="xs:token"/>
</xs:simpleType>
```

5.10.3 Representation

This is represented as a text string that can consist of letters or numbers or both, no leading or trailing spaces and only single spaces inside the string. The use of this data type will also require the definition of the legal tokens or code values.

5.11 epc:xpath – Identification of an XML data item

5.11.1 Definition

The xpath data type is used to identify an XML object in one of the message parts.

5.11.2 Type

```
<xs:simpleType name="xpath">
  <xs:restriction base="xs:token"/>
</xs:simpleType>
```

5.11.3 Representation

This data type is used to identify an XML data object by use of Xpath expressions (see 4.2.9 for the definition of the Xpath format).

6 General ISO 28005 data types

6.1 General

This clause defines a number of general data types that is used both in the message header and message body.

NOTE Some of these definitions can also be found in ISO 28005-2:2021. See 4.5 for an overview.

6.2 epc: AuthenticatorType – Authenticator of information

6.2.1 Definition

This type contains information about the party attesting to the validity of the transmitted information.

6.2.2 Type

```
<xs:complexType name="AuthenticatorType">
  <xs:complexContent>
    <xs:extension base="epc:ContactInfoType">
      <xs:sequence>
         <xs:element name="AuthenticationDate" type="epc:dateTime"
                     minOccurs"0"/>
         <xs:element name="AuthenticatorLocation"
                     type="epc:LocationType" minOccurs="0"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

6.2.3 Representation

The element is an extension of the general contact information type and shall contain the additional data elements:

— `AuthenticationDate`: This is the date and time of the authentication.

— `AuthenticatorLocation`: This is the location of the person when submitting the information.

The “Authenticator party identification code” (IMO 0017 in [14]) use the `CompanyId` attribute in the `ContactInfoType` part of the definition. The code values that shall be used are defined in Annex I.

The “Authenticator role, coded” (IMO 0128 in [14]) use the `ContactType` attribute in the `ContactInfoType` part of the definition. The code values that shall be used are defined in Annex I.

6.3 epc:AuthorizationTokenType – Authorization token

6.3.1 Definition

This type is used to send an authorization token between the sender and receiver.

6.3.2 Type

```
  <xs:simpleType name="AuthorizationTokenType">
    <xs:restriction base="epc:token"/>
  </xs:simpleType>
```

6.3.3 Representation

The token is used to authorize access to the service API. It is retrieved through a special API as described in Clause 18. The token is a text string that only has meaning for the receiver. It should be long and randomly scrambled to make it highly unlikely for other parties to replicate it.

NOTE The API that is used to retrieve and send the authorization codes use secure HTTP and is assumed to be very difficult to eavesdrop on for hostile third parties. However, the sender and receiver are expected to take measures as necessary to protect the authorization code from third parties.

6.4 epc:ContactInfoType – Contact information

6.4.1 Definition

This data type contains contact information for either a person or a company. A person's name can be included if the contact information is for a company.

6.4.2 Type

```
<xs:simpleType name="ContactTypeContentCode">
  <xs:restriction base="xs:token"/>
</xs:simpleType>
<xs:complexType name="ContactInfoType">
  <xs:sequence>
    <xs:element name="Company" type="epc:string" minOccurs="0"/>
    <xs:element name="CompanyId" type="epc:string" minOccurs="0"/>
    <xs:element name="ContactNumbers"
                type="epc:CommunicationNumberType" minOccurs="0"/>
    <xs:element name="ContactType" type="epc:ContactTypeContentCode"
                minOccurs="0"/>
    <xs:element name="Person" type="epc:NameType" minOccurs="0"/>
    <xs:element name="Address" type="epc:PostalAddressType"
                minOccurs="0"/>
   </xs:sequence>
  </xs:complexType>
```

6.4.3 Representation

This is information about a company or person that can be physically contacted. Common information for both persons and companies are addresses and contact numbers. Companies and persons differ in that companies have one name in a string, while persons can have a given name, family name and middle name. A person's name can be submitted for a company contact point, but not vice versa.

The attributes are:

— `Company`: This is the official name of a company.

— `CompanyId`: The recognized identification number of the company, if defined.

NOTE `CompanyId` is normally a business-related. It can be a tax identification number or a local identification number e.g. used in a specific port.

— `ContactNumbers`: Telephone, telefax or other communication channel information.

— `ContactType`: The role of the contact point. This attribute may be required in conjunction with dangerous goods reporting or when the contact point is used as authenticator (see 6.2). The code shall be taken from UN/EDIFACT code lists 3035 or 3139. Codes with a defined meaning in this document are listed in Table I.1.

— `Person`: The name of the person.

— `Address`: The postal address of person or company.

6.5 epc:CommunicationNumberType – Communication number information

6.5.1 Definition

This data type specifies a contact point via telephone or other means.

6.5.2 Type

```
<xs:complexType name="CommunicationNumberType">
  <xs:sequence>
    <xs:element name="BusinessTelephone" type="epc:string"
                minOccurs="0"/>
    <xs:element name="ContactURL" type="epc:anyURI" minOccurs="0"/>
    <xs:element name="EMail" type="epc:anyURI" minOccurs="0"/>
    <xs:element name="HomeTelephone" type="epc:string" minOccurs="0"/>
    <xs:element name="MobileTelephone" type="epc:string"
                minOccurs="0"/>
    <xs:element name="Telefax" type="epc:string" minOccurs="0"/>
```

```
  </xs:sequence>
</xs:complexType>
```

6.5.3 Representation

This element contains a list of contact points for a person or organization. Telephone numbers shall be specified with an international prefix code in the format "+*it*" where *i* is the international prefix and *t* the telephone number without any spaces between or inside either.

The attributes are:

— `BusinessTelephone`: Daytime telephone number.

— `ContactURL`: A web page where information about the party can be obtained.

— `EMail`: Email address as URI, i.e. prefixed with "mailto":.

— `HomeTelephone`: Home telephone number.

— `MobileTelephone`: Telephone number for access outside office.

— `Telefax`: Telefax number.

6.6 epc:CountryCodeContentType – Country identification

6.6.1 Definition

This data type gives a unique and coded representation of a country identity.

6.6.2 Type

```
<xs:simpleType name="CountryCodeContentType">
   <xs:restriction base="xs:token">
     <xs:length value="2"/>
   </xs:restriction>
</xs:simpleType>
```

6.6.3 Representation

The content of the country code element shall be the two-letter country code defined in ISO 3166-1.

6.7 epc:CountrySubdivisionCodeContentType – Country subdivision identification

6.7.1 Definition

This data type gives a unique and coded representation of a country subdivision identity.

6.7.2 Type

```
<xs:simpleType name="CountrySubdivisionCodeContentType">
   <xs:restriction base="xs:token"/>
</xs:simpleType>
```

6.7.3 Representation

The country subdivision code is the final characters of the ISO 3166-2 country subdivision code, following the initial two-letter country code and the hyphen-minus.

NOTE The definition of the subdivision code is a length of maximum six characters where the two first are the country code followed by a hyphen-minus (see ISO 3166-2).

If no code is defined in ISO 3166-2 for the subdivision in question, it is permitted to use a commonly agreed code between the sender and receiver.

6.8 epc:CrewDutyType – Duty onboard or on shore

6.8.1 Definition

This data type specifies the duty of a person.

6.8.2 Type

```
<xs:simpleType name="CrewDutyCodeContentType">
  <xs:restriction base="epc:token">
  </xs:restriction>
</xs:simpleType>

<xs:complexType name="CrewDutyType">
  <xs:sequence>
    <xs:element name="Code" type="epc:CrewDutyCodeContentType"/
                minOccurs="0" />
    <xs:element name="Text" type="epc:string" minOccurs="0" />
  </xs:sequence>
</xs:complexType>
```

6.8.3 Representation

The two fields shall be used as follows:

— `Code`: This is the crew duty type in coded representation. The code list is maintained in the IMO Compendium[14] under IMO object code IMO0043.

— `Text`: This is the crew duty in free text. It is mandatory only if a valid coded value is not supplied.

If the text-field is included and if the crew duty has a code corresponding to a code in the IMO Compendium, the text field should contain the same textual description as in the IMO Compendium.

6.9 epc:LocationType – Identification of a location

6.9.1 Definition

This data type identifies any location, which may also be given a name. This data type is normally used in a specific context to define what type of location this is.

6.9.2 Type

```
<xs:simpleType name="FacilityNameType">
  <xs:list itemType="epc:token"/>
</xs:simpleType>

<xs:simpleType name="GLNCodeContentType">
  <xs:restriction base="epc:token"/>
</xs:simpleType>

<xs:simpleType name="GISISCodeContentType">
```

```
  <xs:restriction base="xs:token"/>
</xs:simpleType>

<xs:complexType name="PositionType">
  <xs:sequence>
     <xs:element name="Latitude" type="epc:LatitudeType"/>
     <xs:element name="Longitude" type="epc:LongitudeType"/>
  </xs:sequence>
</xs:complexType>

<xs:simpleType name="LatitudeType">
  <xs:restriction base="epc:decimal">
    <xs:minInclusive value="-90"/>
    <xs:maxInclusive value="90"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="LongitudeType">
  <xs:restriction base="epc:decimal">
    <xs:minInclusive value="-180"/>
    <xs:maxInclusive value="180"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="UNLoCodeContentType">
  <xs:restriction base="epc:token">
    <xs:length value="3"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="MRNCodeContentType">
  <xs:restriction base="epc:token">
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="SMDGTerminalCodeContentType">
  <xs:restriction base="epc:token">
  </xs:restriction>
</xs:simpleType>


<xs:complexType name="LocationType">
  <xs:sequence>
    <xs:element name="CountryCode" type="epc:CountryCodeContentType"
                minOccurs="0"/>
    <xs:element name="CountrySubdivisionCode"
                type="epc:CountrySubdivisionCodeContentType"
                minOccurs="0"/>
    <xs:element name="FacilityCode" type="epc:GISISCodeContentType"
                minOccurs="0"/>
    <xs:element name="FacilityName" type="epc:FacilityNameType"
                minOccurs="0"/>
    <xs:element name="GLN" type="epc:GLNCodeContentType"
                minOccurs="0"/>
    <xs:element name="MRN" type="epc:MRNCodeContentType"
                minOccurs="0"/>
    <xs:element name="Name" type="xs:string" minOccurs="0"/>
    <xs:element name="NauticalMilesToDestination" type="epc:decimal"
                minOccurs="0"/>
    <xs:element name="Position" type="epc:PositionType" minOccurs="0"/>
    <xs:element name="SMDGTerminalCode"
                type="epc:SMDGTerminalCodeContentType" minOccurs="0"/>
    <xs:element name="UNLoCode" type="epc:UNLoCodeContentType"
                minOccurs="0"/>
    <xs:element name="VisualPosition" type="epc:VisualPositionType"
         minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
```

6.9.3 Representation

The location type allows the user to specify a location by one or more of the following attributes:

— `CountryCode`. Two-letter country code defined in ISO 3166-1.

— `CountrySubdivisionCode`. Country subdivision code defined in ISO 3166-2 (see 6.7 for code values).

— `FacilityCode`: The port facility 4-digit code defined in GISIS.[16] Not all facilities are listed in GISIS. In these cases, the facility name field shall be used instead.

— `FacilityName`: The name of an ISPS facility in a port or terminal. This field is mandatory if the `FacilityCode` is not provided.

— `GLN`: Global location number of the position.[17]

— `MRN`: Marine Resource Name - a text string that identifies a specific marine resource, including positions.[28]

— `Name`: This is a free text name that can be used either as an alternative to the other code types or as an optional name on the location.

— `NauticalMilesToDestination`: As given by name.

— `Position`: Geographic position specifying the latitude and longitude. The latitude and longitude shall be represented using the degrees and decimal degree format specified by ISO 6709, i.e. { ± }dd.dd for latitude and { ± }ddd.dd for longitude, where "+" (plus) indicates north or east and "–"(minus) indicates south or west. The "+" (plus) sign can be omitted.

— `SMDGTerminalCode`: A terminal code defined by SMDG and maintained in their code list.[29]

— `UNLoCode`: Three letter national location code for a port as defined in UN/LOCODE.[15]

— `VisualPosition`: A free text description of location relative to landmarks that can be seen from the reported position.

6.10 epc:NameType – Name of person

6.10.1 Definition

This data type contains the full name of a person.

6.10.2 Type

```
<xs:complexType name="NameType">
  <xs:sequence>
    <xs:element name="FamilyName" type="epc:string" minOccurs="0" />
    <xs:element name="GivenName" type="epc:string" minOccurs="0" />
    <xs:element name="MiddleName" type="epc:string" minOccurs="0" />
   </xs:sequence>
</xs:complexType>
```

6.10.3 Representation

This shall be represented as three text strings. Strings are in free text and possibly not suitable for computer-based comparisons.

NOTE Users of this data element can impose additional restrictions on the data type, e.g. by requiring the spelling to be identical to that used in an electronically readable identification document.

6.11 epc:OrganizationType – Description of an organization

6.11.1 Definition

This data type is used to give details of an organized body such as a business, government body, department or charity.

6.11.2 Type

```
<xs:complexType name="OrganizationType">
  <xs:sequence>
    <xs:element name="Name" type="epc:string" minOccurs="0"/>
    <xs:element name="RegistrationDate" type="epc:date" minOccurs="0"/>
    <xs:element name="TaxIdentifier" type="epc:string"
                minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="RegistrationCountryCode"
                type="epc:CountryCodeContentType" minOccurs="0"/>
    <xs:element name="RegistrationLocationCode"
                type="epc:UNLoCodeContentType" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
```

6.11.3 Representation

The attributes define the following data elements:

— `Name`: Official name of organization.

— `FacilityCode`: The port facility 4-digit code defined in GISIS.[16] Not all facilities are listed in GISIS. In these cases, the facility name field shall be used instead.

— `RegistrationDate`: Date of registration of the organization.

— `TaxIdentifier`: The tax identification code, e.g. value added tax (VAT) code, goods and services tax (GST) code or similar.

— `RegistrationCountryCode`: The two-letter country code (see 6.6).

— `RegistrationLocationCode`: The three-letter port code, if the organization is located in a port (see 6.9).

6.12 epc:PostalAddressType – A postal mail address

6.12.1 Definition

This is the postal address for a person or organization.

6.12.2 Type

```
<xs:complexType name="PostalAddressType">
  <xs:sequence>
     <xs:element name="CityName" type="epc:string" minOccurs="0" />
     <xs:element name="CountrySubdivisionCode"
                 type="epc:CountrySubdivisionCodeContentType"
                 minOccurs="0" />
     <xs:element name="LineFive" type="epc:string" minOccurs="0" />
     <xs:element name="LineFour" type="epc:string" minOccurs="0" />
     <xs:element name="LineOne" type="epc:string" minOccurs="0" />
     <xs:element name="LineThree" type="epc:string" minOccurs="0" />
     <xs:element name="LineTwo" type="epc:string" minOccurs="0" />
     <xs:element name="PostCode" type="epc:token"
                 minOccurs="0" />
     <xs:element name="PostOfficeBox" type="epc:string"
                 minOccurs="0" />
```

```
      <xs:element name="StreetName" type="epc:string" minOccurs="0" />
      <xs:element name="StreetNumber" type="epc:string"
                  minOccurs="0" />
      <xs:element name="CountryCode" type="epc:CountryCodeContentType"
                  minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
```

6.12.3 Representation

This is the description of a postal address. Most fields are intended to be read by humans and are free format, except for the country and country subdivision codes:

— `CityName`: Name of city.

— `CountrySubdivisionCode`: This code shall follow national code lists for country sub-divisions as listed in ISO 3166-2 (see 6.7 for code values).

— `LineN`: Postal address lines, used if street name and street number are not sufficient.

— `PostCode`: Post code of location.

— `PostOfficeBox`: Post box number if in use.

— `StreetName`: Name of street.

— `StreetNumber`: Street number.

— `CountryCode`: This code shall follow national code lists for countries as defined by ISO 3166-1.

The `LineOne` to `LineFive` tags can be used for various contact address information and should in general be printable, one tag data on one line.

NOTE The ordering of address lines in the XSD is alphabetic as the XSD is generated automatically.

6.13 epc:ShipIDType – Ship identity

6.13.1 Definition

This composite element contains data that can be used to identify the ship. Different users of this data structure require different minimum information elements.

6.13.2 Type

```
<xs:complexType name="ShipIDType">
  <xs:sequence>
    <xs:element name="CallSign" type="epc:token" minOccurs="0" />
    <xs:element name="Comment" type="epc:string" minOccurs="0" />
    <xs:element name="IMONumber" type="epc:token" minOccurs="0" />
    <xs:element name="MMSINumber" type="epc:token" minOccurs="0" />
    <xs:element name="ShipName" type="epc:string" minOccurs="0" />
    <xs:element name="RegistrationPort" type="epc:LocationType"
                minOccurs="0" />
  </xs:sequence>
</xs:complexType>
```

6.13.3 Representation

The `ShipIDType` is represented with one or more of the following attributes.

— `CallSign`: This is the call sign for the ship. The call sign is at least four characters long and can consist of both letters and numbers.

— `Comment`: Any other information related to ship identity. For example, this can be used if an identity field is requested by a certain receiver, but is not available for this ship.

— `IMONumber`: This token consists of the string “IMO” followed by the seven-digit IMO number without any embedded separator character.

— `MMSINumber`: This token consists of the nine-digit MMSI number without any separator character.

— `ShipName`: This is the name of the ship. This is in human-readable form and no special restrictions are enforced.

— `RegistrationPort`: This is the port of registration for the ship. It contains the country code and port code for the port of registration. The port name in human-readable format may also be included.

6.14 epc:ReportingSystemType – Name of a reporting system

6.14.1 Definition

This type contains the name of a reporting system.

6.14.2 Type

```
<xs:simpleType name="ReportingSystemType">
  <xs:restriction base="epc:string"/>
</xs:simpleType>
```

6.14.3 Representation

`ReportingSystem` is the name of the reporting system to which the message is sent, if appropriate. This is used for all messages of the MRS type. The name is the official name given in the relevant IMO approval documents. Alternatively, one can use the three-digit codes defined in the IMO compendium for IMO 0322 (Ship reporting system, coded).[14]

EXAMPLE Some examples of reporting system names are “REEFREP”, “BALTREP”, and “GIBREP”.

6.15 epc:AttachmentType – Reference to an attached document

6.15.1 Definition

This type contains a description of and a reference to an attached document.

6.15.2 Type

```
<xs:complexType name="AttachmentType">
  <xs:sequence>
     <xs:element name="Description" type="epc:string" minOccurs="0"/>
     <xs:element name="URI" type="epc:anyURI" minOccurs="0"/>
   </xs:sequence>
</xs:complexType>
```

6.15.3 Representation

The following attributes can be used:

— `Description`: This attribute can give a human-readable description of the attached file or it can contain the attached information itself. If a URI is supplied, the `Description` is for information only and cannot be usefully read by a computer-based client. If the URI is empty or does not exist, the Description field shall contain the attached information.

— `URI`: This attribute shall contain a valid URI as defined in RFC 3986. The data element can be used in two ways:

1) It can use the URI prefix “file”: as in “file:xxx.yyy”, to point to an attached file with the name “xxx.yyy”. In this case, the receiver shall find the attached file in the message attachments as described in Clause 12.

2) Alternatively, the URI can point to a resource available remotely.

6.16 epc:ReferenceCodeType – General reference code

6.16.1 Definition

The reference code type is an arbitrary token that is generated by a party for later reference to a service or data object managed by that party. See 7.13 for an overview of some reference code types.

6.16.2 Type

```
<xs:simpleType name="ReferenceCodeType">
  <xs:restriction base="epc:token"/>
</xs:simpleType>
```

6.16.3 Representation

This is an arbitrary string of printable characters.

NOTE The universally unique identifier (UUID – see Reference [18]) can be used for this purpose, i.e. a 128 bit long number, normally encoded as a 36 character long string. However, in most cases the reference is to a locally managed object and a shorter and more functional local code can be more convenient.

6.17 epc:SystemIdType – Identity code for a software system

6.17.1 Definition

This element identifies a specific software system on land.

6.17.2 Type

```
<xs:simpleType name="SystemTypeContentType">
  <xs:restriction base="epc:token"/>
</xs:simpleType>

<xs:complexType name="SystemIdType">
  <xs:sequence>
    <xs:element name="Name" type="epc:string" minOccurs="0" />
    <xs:element name="Type" type="epc:SystemTypeContentType"
                minOccurs="0" />
    <xs:element name="Location" type="epc:LocationType"
                minOccurs="0" />
 </xs:sequence>
</xs:complexType>
```

6.17.3 Representation

The following elements are used as follows:

— `Name`: Name of the system. A text string that shall be unique within location parameter.

— `Type`: Type of system. Table H.1 defines the codes that shall be used.

— `Location`: Physical place where the system resides.

6.18 epc:SignatureCertificateIdType – Name of digital signature holder

6.18.1 Definition

When digital signatures are used, the identity code entered in the public key certificate shall exactly match that of the person, ship, software system, or the company that signs the message. The certificate identity type is used to refer to this identity code.

6.18.2 Type

```
<xs:complexType name="SignatureCertificateIdType">
  <xs:sequence>
    <xs:element name="CommonName" type="epc:string" minOccurs="0" />
    <xs:element name="CountryCode" type="epc:CountryCodeContentType"
                minOccurs="0" />
    <xs:element name="Organization" type="epc:string" minOccurs="0" />
  </xs:sequence>
</xs:complexType>
```

6.18.3 Representation

The following elements are part of the certificate identity:

— `CommonName`: This string shall be identical to the common name (“CN=”) that is part of the X.509 public key certificate. It can be up to 64 bytes long. International character codes from UTF8 are allowed.

— `CountryCode`: The flag state of the ship or country of legal residence of the identified party. This shall match the country code in the X.509 certificate (“C=”).

— `Organization`: The organization name that shall match the organization name field in the X.509 certificate (“O=”).

This document does not prescribe any specific content or format for these three fields. The fields that are supplied shall exactly match the text in the corresponding X.509 certificate field.

Any standard format of the X.509 fields should be defined in conjunction with the selection of public key infrastructures for the international shipping sector.

NOTE See Reference [19] for a discussion of these general issues. Also see IEC 63173-2 for specific use of these data fields.

6.19 epc:VersionType – Version code

6.19.1 Definition

This type contains a version code that is used in the message header (see 10.10) to inform the server about the version code used by the client to format the XML parts of its message.

6.19.2 Type

```
<xs:simpleType name="VersionType">
  <xs:restriction base="epc:token" />
</xs:simpleType>
```

6.19.3 Representation

The version token contains a version code in the format "*M.N*" or "*M.N.P*" where *M*, *N*, and *P* are positive integers, without leading zeros. The client and server shall use the following rules to determine if the server is able to process the information in a received message.

a) If the "*M*" code is different from what the server can understand, the server shall not try to process the information.

b) If the "*N*" code is higher than what the server is designed to understands, the server shall process the information it understands. There can be additional enumeration codes or tags in the message that shall be discarded.

c) If the "*N*" code sent is lower than what the server understands, it can safely read all information in the message. The server can reject the message if important functions in the server rely on information that can only be transmitted by the higher version code.

d) The "*P*" code shall be ignored by the server if it is not configured to understand it. It can be used by manufacturers to internally identify different revisions of sending systems or to indicate special local implementations functions. The server can reject the message if important functions rely on information that can only be transmitted by the correct local variant code.

NOTE See 4.4.2 for more details on the specific meaning of these version codes and the caveats related to validation of incoming XML messages with XSD schema in cases b) and c).

The receiver shall in all cases return a message status with the receiver's acceptable version code. If the sender's code is acceptable, this shall be returned. If not, the minimum acceptable version code shall be returned. If the message was rejected, the sender can retry sending the message with formatting conformant to the servers reported version code.

Both the sender and receiver shall use the initially accepted version code throughout a session.

7 ISO 28005 design principles

7.1 Harmonization with the IMO reference data model

The data elements defined in the ISO 28005 series are harmonized with the IMO Compendium and its reference data model.[14] Table K.1 contains the cross-references between the data elements defined in this document and the data elements in the IMO reference data model. This mapping can also be found on the IMO website,[14] where mappings from other international standards to the IMO Compendium also are available.

7.2 Fully automated machine to machine

The aim of the ISO 28005 series is to enable fully automated exchange of information between the parties involved in a ship's port call, i.e. fully automated machine-to-machine communication. This requires the use of digital signatures for automatic authentication of the parties as well as for ensuring the integrity of the information transmitted.

Automatic service discovery is not defined by this document, so API access points must be determined by other means (see 7.19). The MIG specifies how the API access points shall be used (see 7.17).

This document assumes that human interventions will be required for more complex error situations. Thus, error message formats are defined as being for human attention, with the following exceptions:

1. Rejected messages due to mismatch in version codes may be automatically corrected by senders by changing the format of the message and resending it (see 6.19).

2. Rejected messages due to missing data ("rejected incomplete" in Table 3) may be automatically corrected by senders by adding missing data and resending the service request.

Both these cases are signalled to the sender by special message status codes (see 7.10.1).

7.3 Using carrier independent and internet-based protocols

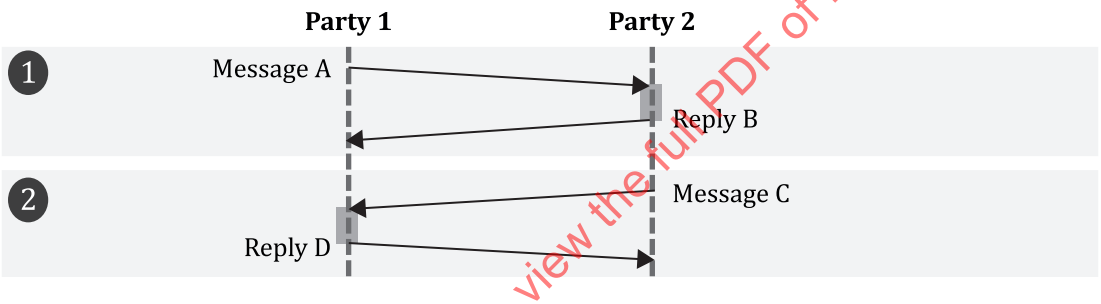
This document is based on internet protocols and is not dependent on any particular physical carrier as long as the capacity and latency of the carrier is suitable for internet protocols. Suitable carriers are most satellite systems as well as mobile and fixed digital data connections.

NOTE The proposed VHF Data Exchange System (VDES) will not be suitable for direct implementation of ISO 28005 message exchanges. However, by using the IMO Compendium as basis for defining VDES-specific data messages, interoperability with other data exchanges within the ISO 28005 series can be ensured.

This document includes provisions for supporting communication to and from ships as described in 7.5. Asynchronous message transfers can be used to cater for situations where the ship may be off-line for shorter or longer periods. It also allows the ship to specify an API access point for returned status messages that can be in the cloud or in a land location. This avoids ships having to provide a continuously available API access point, which can create problems for cyber security, maintaining connection availability, and bandwidth use.

7.4 General format of message sequence diagrams

This document uses message sequence diagrams following the format shown in Figure 2. Each diagram can include several messages exchanged between several different parties. In this example, the diagram includes four messages exchanged between two parties.



Key

party 1, party 2	name of communicating parties
message A, message C	name of messages initiating a synchronous message exchange
reply B, reply D	name of messages that are responses in the synchronous message exchange
⋮	timeline for communicating parties. Time starts at top and goes downwards.
X	labels for lightly shaded parts of the message exchange, referred to as sequence 1 and sequence 2 in the explanatory text
▮	synchronous processing by timeline party

Figure 2 — Example of message sequence diagram

Each vertical dashed line represents a timeline for messages sent to or received by one specific party to the communication. The name of the party is placed above each vertical dashed line.

In the diagram, time starts from the top and travels downwards. While the diagram specifies the chronological ordering of messages, it does not specify any absolute or relative measure of time.

The arrows in Figure 2 show the messages being exchanged between parties and the communication direction. The message is sent from the party at the start of the arrow to the party at the end of the arrow. The label at the start of the arrow specifies the name of the message.

The dark shaded vertical rectangle on the timeline for one party defines a synchronous process whereby other messages will be actively received or sent by the party within a short interval. The absence of a dark

rectangle between messages means that the party is passively waiting for new events, such as an internal event or a new incoming message before new actions are taken.

NOTE Specifically, a synchronous process can mean that other messages can be sent or received during the same HTTP session (see 7.7.3).

The large lightly shaded rectangles show different and possibly optional parts of the message exchange. The numbers inside circles are used as reference to this part of the message exchange in the corresponding textual description.

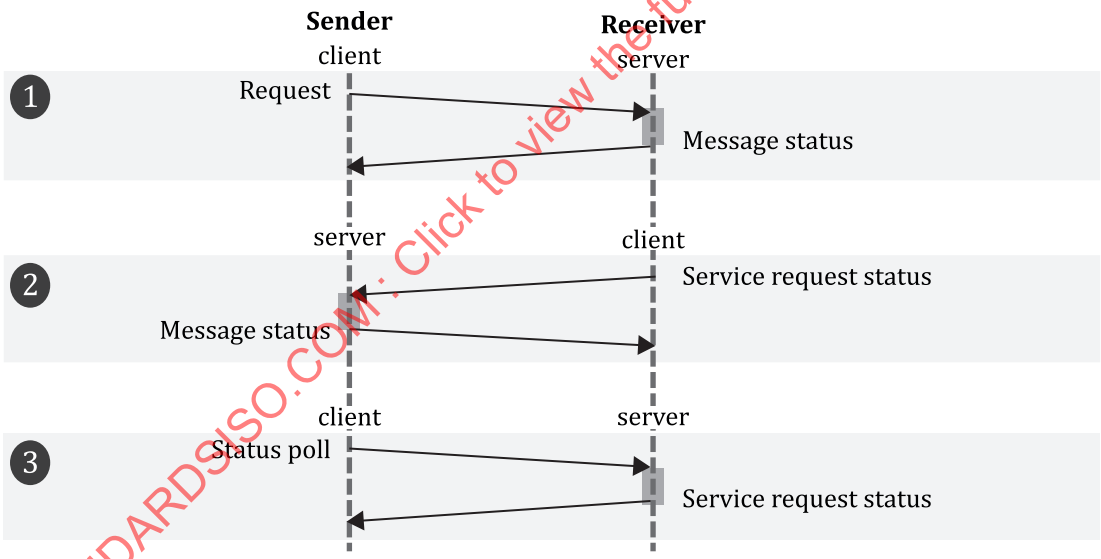
7.5 Sender and receiver versus client and server — asynchronous message transfers

This document follows the conventions from IMO FAL.5/Circ.46.[1] It uses the term sender to represent a requester of a service. This will normally be the ship, its agent or another representative for the ship, e.g. owner or charterer. The provider of the service will be called a receiver.

As service requests can take a significant amount of time to be executed, this document allows for asynchronous notification of service completion. This can be done in one of two ways:

1. By polling from the sender to the receiver on the status of the service request. This is illustrated in sequences 1 and 3 of Figure 3 where the sender is always the client. The technical requirements for this mechanism are specified in Clause 17. This principle does not support the exchange in sequence 2.

2. By the sender specifying a call-back API for the receiver to use when informing the sender about the outcome of the service execution. This is illustrated in sequences 1 and 2 of Figure 3 where the receiver and the sender change roles as client and server. The technical requirements for this mechanism are specified in Clause 16. This principle allows the sender to poll the receiver as is shown in sequence 3.



Key

See Figure 2 for key references.

client, server changing role of the sender and receiver in the three different sequences

Figure 3 — Sender/receiver versus client/server

Sequence 1 of Figure 3 shows an initial request from the sender, where the sender also acts as a client. The receiver, acting as a server, receives the request, schedules it for execution, and immediately returns a message status message to the sender, in the same HTTP session.

The request will at some time be completed by the receiver, at which time the receiver, acting as a client, returns a service request status to the sender. The sender, acting as a server, immediately returns a message status to the receiver (see sequence 2 of Figure 3). This will happen only if the sender has informed the receiver about what API access point the receiver shall use when returning the service request status.

Sequence 3 shows the sender-initiated status poll. This is always allowed, even if the sender has specified an API access point for return calls.

7.6 Generalization of service

This document generalizes any request or information message sent by the sender to the receiver to be a service request.

EXAMPLE Examples of service types can be the authorities' electronic port clearance, mandatory ship reporting, requests for arrival times in port or berth, ordering ship supplies, or services directly related to the ship arrival or departure, such as tugs and linesmen, etc.

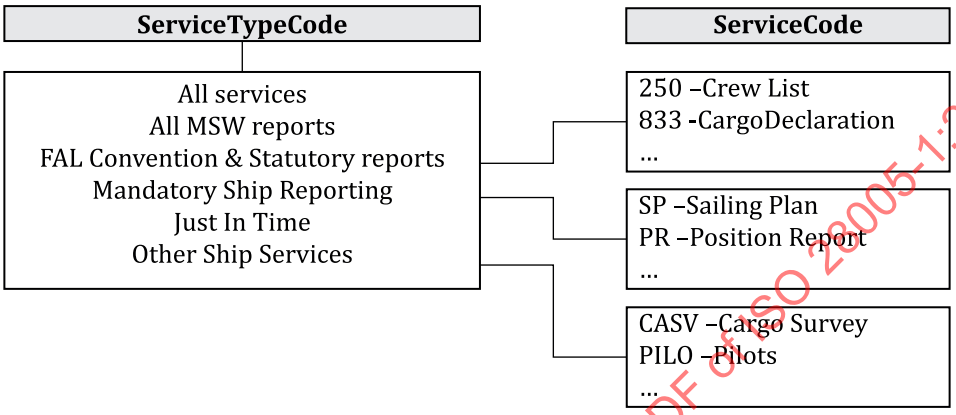


Figure 4 — Examples of service type codes and service codes

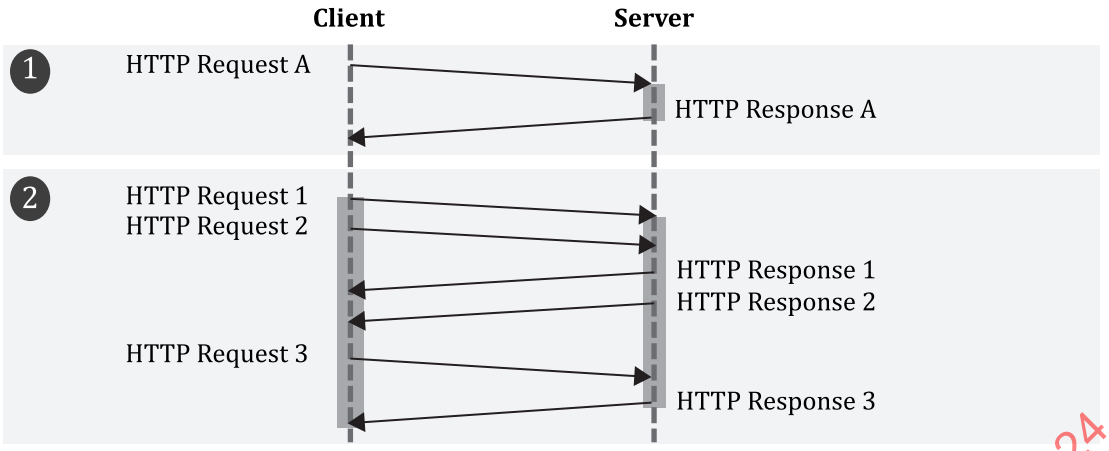
A reference to a specific service code is done through a two-level service identification code located in the message header as illustrated in Figure 4. `ServiceTypeCode` identifies the type of service, e.g. an MSW report, MRS report or other types of ship services, while `ServiceCode` contains the description of the specific service, e.g. an MSW crew list or an MSW cargo declaration.

NOTE The grouping of services into service types is done to facilitate the use of a mix of numeric and alphanumeric code lists.

7.7 Different levels of sessions

7.7.1 HTTP session

The lowest level of a message exchange session is the HTTP session. An example is illustrated in Figure 5, where two different HTTP sessions are shown respectively sequence 1 and sequence 2.



Key

See Figure 2 for key references.

Figure 5 — HTTP session examples

Sequence 1 in Figure 5 shows the most common HTTP session where one HTTP request is replied to with a response, and then the HTTP connection is closed. The server and client can be either sender or receiver in this type of session. The minimum HTTP session includes the incoming message from the client and the response from the server. The latter will be a message status or a service request status message.

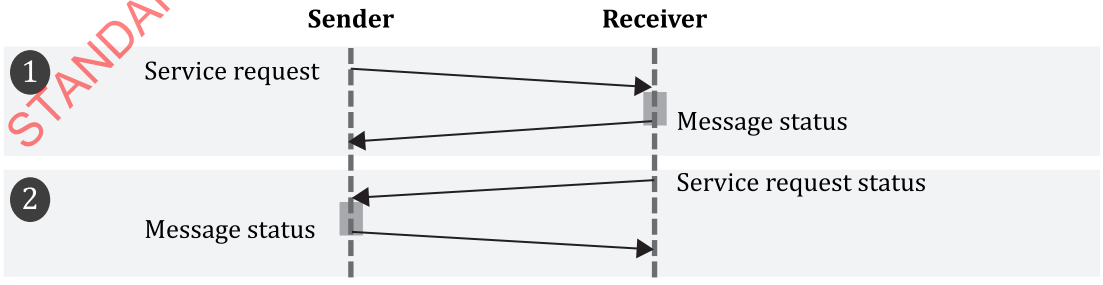
Sequence 2 in Figure 5 shows a legal, but less common case, where the HTTP session is used to send several requests from the client. The HTTP connection and the session is not closed before the final response (response 3) has been sent from the server. This can only be the case when the client is a sender, and the sender has several service requests to send to the same receiver.

To keep the HTTP session open as in sequence 2, it shall be requested from the client by using the keep-alive flag in the HTTP header (see 15.4).

7.7.2 Session

Figure 6 shows a simple session. This consists of a sender requesting a service from a receiver and then getting a message status back, indicating that the service request will be processed. After some time, the receiver will send a service request status to the sender to notify the final status of the service request. The receiver answers with a message status.

The session may also be more complex, with more than one sender, with service request updates and possible service cancellations as described in Clause 8.



Key

See Figure 2 for key references.

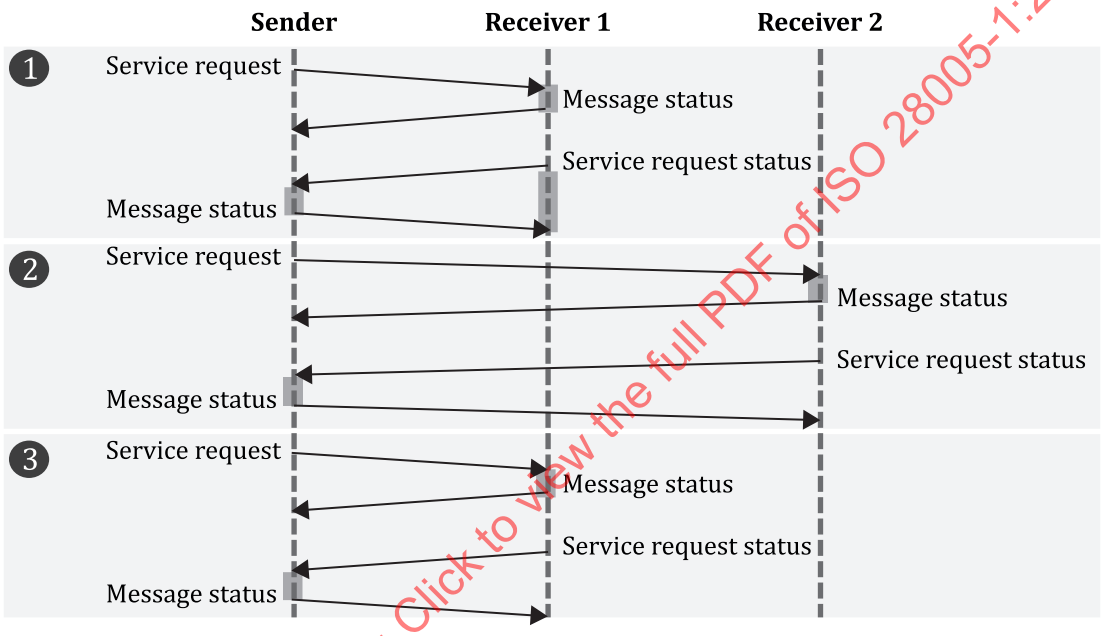
Figure 6 — Simple session example

Each session will be related to the request and execution of one service, e.g. MSW clearance, just-in-time arrival negotiation, or ordering tugs for a port arrival. The duration of a service session is dependent on the service requested. The `ServiceBookingNumber` element in the header will identify the relevant service instance and associated service session. The `Final` data element in the header is used to signal the end of the service session by setting the flag to true. The receiver can also signal that the final status message is sent by setting the `Final` flag in the status message.

In this example, each of sequence 1 and 2 is one HTTP session. The same result can be achieved in an asynchronous message exchange by using sequences 1 and 3 from Figure 3.

7.7.3 Session context

At the top of the session levels is the session context, as illustrated in Figure 7. This is a common context to several different but related service requests sessions. The context can be the stay in territorial waters, a port or terminal call, or a transit through a ship reporting area.



Key

See Figure 2 for key references.

Figure 7 — Session context example

Figure 7 shows three sessions that are connected through a common session context. There can be more than one sender, but always related to the same entity, normally a ship. Other senders can in this case be the ship agent or ship manager, operating on behalf of the ship (see 7.9). There can be one or more receivers, depending on which receiver supplies which service. Figure 7 shows two receivers. The `ShipStayReference` data element in the message header is used to identify the session context.

NOTE The session context can be used to signal to the sender that the submitted data can be reused (see 7.11). It can be used by the receiver to clean up uncompleted requests and other information related to the session context when the session context terminates.

The following requirements apply to the session context:

1. All receivers that use the `ShipStayReference` to maintain session contexts shall coordinate the use of the codes. This includes rules for assigning codes when nested session contexts are in use, e.g. between national clearance, port operations and terminal operations. The rules shall be described in the respective MIG documents. These rules shall define when the outermost session context starts and ends.

   EXAMPLE The `ShipStayReference` code so that the lower level session contexts are signalled by adding extensions to the previous level, e.g. "`OuterSession.Level1.Level2`" etc.

2. A sender shall use an empty `ShipStayReference` code when requesting a new service from outside any outmost session context. If inside the outmost session context, the sender shall use the last received `ShipStayReference` code when requesting a new service.

3. The receiver can define a new or reuse an old `ShipStayReference` code in the message status from a new service request. This code shall be used in all subsequent messages related to this service session.

4. It is possible that the receiver does not return a `ShipStayReference`. If no session context code is returned, the sender shall not include a `ShipStayReference` in any message related to that service.

5. The sender shall clear its last used `ShipStayReference` code when the outermost session context terminates as defined by the MIG documents.

7.8 One service per request and session

Each service request and session is associated with exactly one service. It is not possible to request more than one service in a single request.

NOTE It is possible however, to send a number of different service requests in one HTTP session. Each of these requests will be associated with a different service session (see 7.7.1).

7.9 Linking receivers to service providers

The communication system consists of any number of senders and receivers, where senders can request services from one or more of the receivers as illustrated in Figure 8.

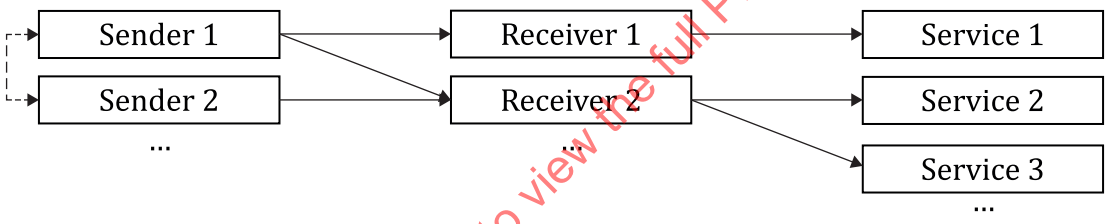


Figure 8 — General communication system topology

This document allows more than one sender to be involved in one session to cater for situations, for example, where different parties represent the same ship. This can be the case when the ship itself, its port agent and charterers cooperate to provide information to the port or order services to the ship. In this case, the senders shall communicate internally to provide the correct reference codes for the session context and the specific session.

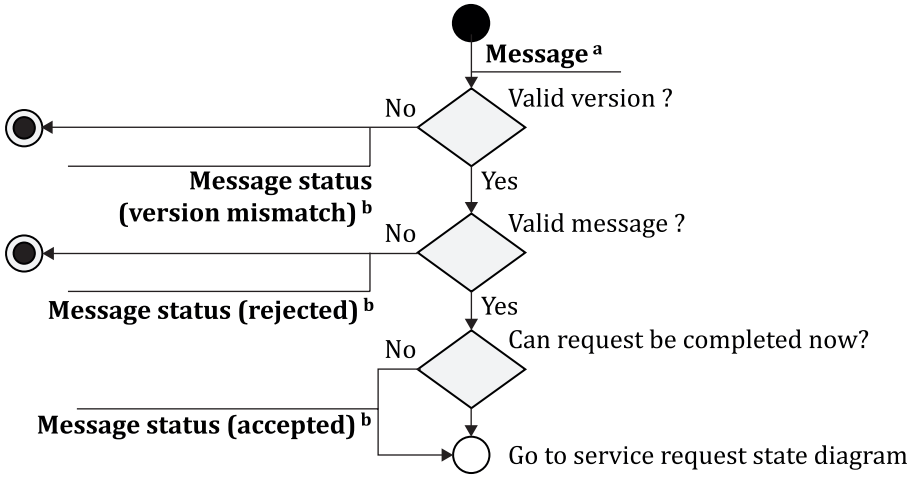
Each receiver can provide one or more services. Each service requested from the same receiver shall be associated with a separate session.

One and the same physical service can be provided through different receivers, but each session shall be associated with only one of these receivers throughout the duration of the session.

7.10 Service request states

7.10.1 Message processing

Many changes in service requests states are triggered by incoming messages from the sender. The principle for the initial processing of these messages is shown in Figure 9.



Key

starting point for state diagram

end point for state diagram

transfer to state diagram in Figure 10

a test to direct further processing

a a message to the receiver

b a message to the sender

Figure 9 — Message processing to service request states

If the message has an unacceptable version code (see 6.19) the receiver shall return the version mismatch status code and insert its minimum acceptable version code in the message header.

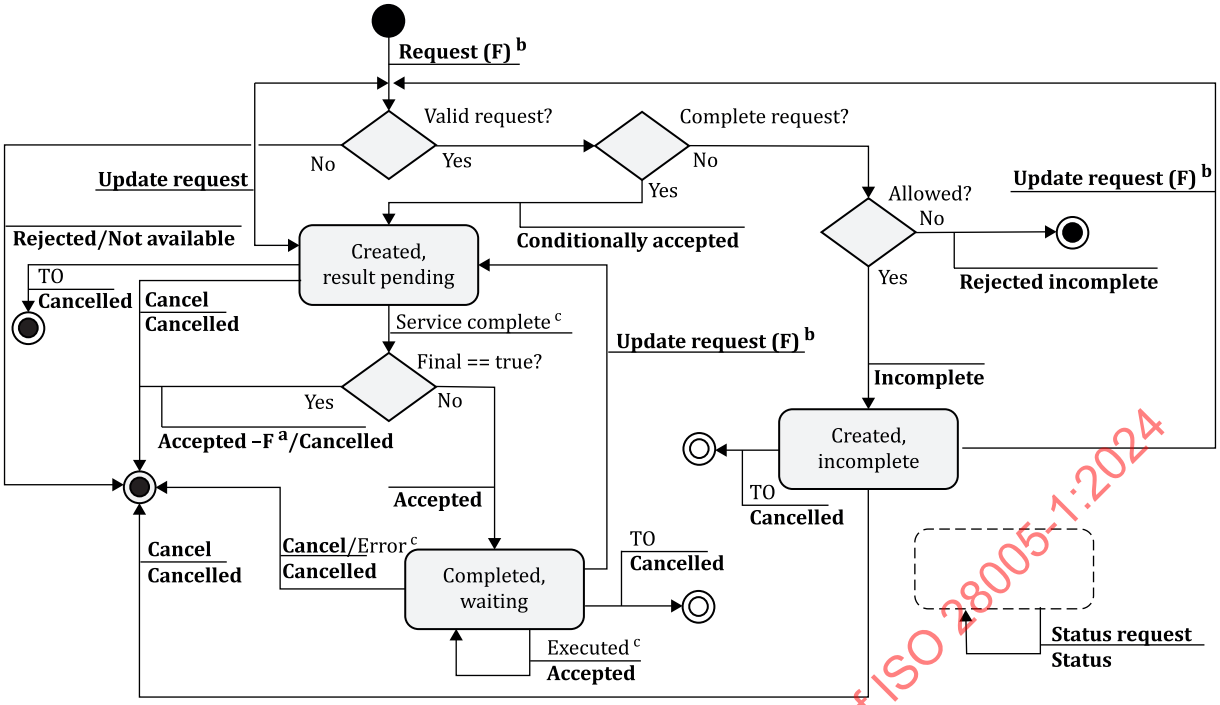
If the message cannot be processed due to errors in format or content, the sender is notified immediately with a message status response. The status, missing and error fields in that message contain information about the error.

Then it is assessed if the further processing of the request can be done within the same HTTP session. If so, the processing is transferred directly to the state transition diagram in Figure 10 where a service request status can be sent in the existing HTTP session. In this case, the message accepted status will be included with the request status.

If the request cannot be processed in the same HTTP session, a message status response is sent to the sender with information about the pending processing of the service request, i.e. an accepted message status and an empty service status.

7.10.2 State diagram for service requests

Once the message is accepted for processing, a service request goes through states as shown in Figure 10. This can be in the same HTTP session (see 7.10.1) or as an asynchronous service status notification process. In both cases, the first status message returned to the sender shall include both a message accepted status and the appropriate service status field in the message header. Further status messages should not include a message status field unless when a new status message is the result of a message from the sender.



Key

See Figure 9 for key references.

◎ optional end point for state diagram (timeout)

TO timeout

a a message with final flag set

b a message that may have final flag set

c internal event, e.g. service error or execution

Figure 10 — Service states

A service request from the sender, if valid, can create a service description. If incomplete, the state will be to wait until more data are forthcoming ("Created, incomplete"). Receivers are allowed to not use this state but terminate the request immediately with an "Rejected incomplete" error code.

When the description is complete, the service is scheduled for execution, and response information is sent to the sender. If the service can be immediately executed, the response is "Accepted", if not the response is "Conditionally accepted". In the latter case the service request will stay in the "Created, result pending" state until such time as the service is completed where the "Accepted" message is sent to the sender.

If the final flag was set by the sender (indicated as an "F" in the incoming message) or the service is of a type that cannot be repeated, e.g. delivering a report or a physical service such as tug or linesmen assistance, this is notified to sender with the final flag set ("Accepted – F") and the service description can be removed. If no final flag was set, the description is put in waiting mode ("Completed, waiting") pending further instructions from the sender or service provider. If the service provider executes the service again ("Executed"), the sender is notified with a new "Accepted" message.

Senders and receivers can also specify a timeout for the service request which will apply to any of the waiting states as described in 7.11. Similarly, the sender can normally cancel a service request at any time. Finally, internal errors in execution may also terminate the service. The response from the receiver in these cases shall be a "Cancel" message.

The sender shall at any time be able to send a "Status request" message to the receiver, which shall respond with a "Status" message to the sender.

7.10.3 Message functions

The different message types and service codes used are listed in Table 3. Column 1 shows the message function codes as used in Figure 9 and Figure 10. Annex E defines the values that shall be used for the message function codes. These codes go into the `MessageFunctionCode` attribute in the message header.

Table 3 — Messages associated with message and service state changes

Message [a]	Type	Message description
Request	R	General service request
Authorization request [b]	R	Access authorization request
Update request	R	Service request information is updated
Cancel	R	A previous service request is cancelled
Status request	R	Query status on a previous service or message request
Incomplete	S	Request is missing information, provide update
Conditionally accepted	S	Service request accepted; result is forthcoming
Rejected	S	Request rejected, error in request information
Rejected incomplete	S	Request rejected, missing request information
Not available	S	Request rejected; service does not exist
Cancelled	S	The service request has been cancelled
Accepted	S	The result of a service request
Authorization [c]	S	The result of an authorization request
Timeout	S	A service request timed out
Status	S	Status on service request returned
Message status	M	Status on incoming message returned

Key

R Request message from sender to receiver

S Response from the receiver on a service-related request

M Message status from Figure 9

[a] As in Figure 9 and 10, see also Annex E for more details on code values.

[b] This message codes is functionally "Request", but is used for access authorization (Clause 18).

[c] This message codes is functionally "Accepted", but is used for access authorization (Clause 18).

7.10.4 Specification of request timeout

The sender can use the `RequestValidityEnd` attribute in the header to define a time-out for its request. This timeout can be redefined by subsequent request updates.

The receiver can also define an internal timeout for completion of service requests after which the service or its request is cancelled. This internal timeout shall be returned to the sender through the `RequestValidityEnd` attribute in the response to the service request, if the receiver's timeout is shorter than the sender specified timeout.

The receiver side timeout can be used as a mechanism to clean up requests that for an unknown reason are never completed or cancelled. The end of a session context can also be used to clean up any uncompleted requests (see 7.7.1). Thus, it is recommended that a timeout is defined by the receiver if not supplied by sender. It is possible that the timeout is not needed if external physical events, e.g. end of session context, can be used to clean up internal states.

The sender shall be informed about a timeout or any other cancellation initiated by the receiver or associated service provider.

7.10.5 Message and service request return values

The message header (see 10.10) contains one `MessageStatus` and one `RequestStatus` data object. These shall respectively contain the message and the service request status codes. The following rules apply:

1. The sender, when returning a message status to the receiver, shall only use the message status fields. The service request fields shall be empty. The sender shall only return a message status message when it acts as a client (see 7.5).

2. A message and/or service status message shall be sent by the receiver whenever a service changes states or as a response to any incoming message from the sender.

3. If a receiver's response is caused by a message from the sender, the receiver shall include the message status fields in the response.

4. The service request status part shall not be filled in if the receiver has no immediate information about the status of the requested service (see 7.10.1).

5. If the sender has not yet received a service booking number (see 7.12), any message from the receiver shall contain the message status fields related to the sender's initial service request.

6. The receiver shall not send a separate message status message if a service request status message will be sent in the same HTTP session (see 7.10.1). In this case, both the message and the service request status fields shall be included in the same message.

7. The message status fields, if included, shall as a minimum contain the client's message reference number and the status code.

8. The service status fields, if included, shall as a minimum contain the service booking number and the status code.

The message function code gives the general information of the status of the message or service request as described in Table 3. See 8.1.5 for additional details on status, missing data fields and error codes.

7.11 Send data once only

As the ship can request several services during a session context, and as each service session can only process one service, the receiver should store the data already submitted so that these can be reused in new service requests and service sessions without the sender having to resubmit data already submitted to the receiver. This data can be deleted when the session context terminates.

The sender may assume that submitted data can be reused as long as the session context (`ShipStayReference`) remains the same value (see 7.7.3). The details of the relationship between session context and data reuse shall be defined in the MIG.

7.12 Message context

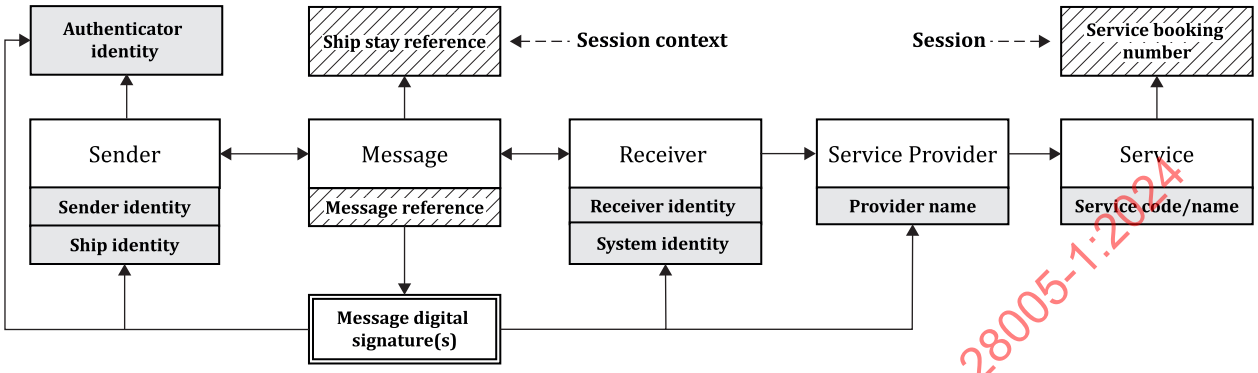
Figure 11 shows the main object classes that are involved in determining the context of a message, i.e. how to interpret contents and determine what process shall handle a specific message. The main object classes are the sender, the receiver, the message, the service provider, and the specific service requested.

Identity information comprises attributes in the main object classes that identify one instance of the class, i.e. an actual party or one of the service provider's actual services. The exception is the authenticator (top left of Figure 11), which is an object in itself but associated to the sender as the person who has confirmed the validity of the message content. The sender and receiver may be a person or organization but may instead, or in addition, be identified as the ship itself or a software system, e.g. a port community system.

Thus, there can be different types of party identities, including:

1. The ship itself, e.g. through a software system on the ship (`epc:ShipIdType`).

2. A person (`epc:NameType`). This can be onboard crew or a shore-based person. This party can be relevant, e.g. when the `Authenticator` attribute is used in the message header.
3. An organization, e.g. ship agent or charterer (`epc:ContactInfoType` or derived types).
4. A software front-end system on shore, e.g. MSW or PCS (`epc:SystemIdType`).
5. A service provider name as an organization (`epc:ContactInfoType` or derived types).



Key

- main object classes that are used or referenced in a communication session
- identification attribute(s) for main object classes, used to identify one specific instance
- data items that are used to reference a specific instance of the main object classes
- digital signatures for one or more parts of the message with link to the signer's identity

Figure 11 — Main objects used in a communication session

Digital signatures can be linked to the different identities. These signatures are used to verify authenticity and integrity of the content of different parts of the message.

The service is identified by a free format name or a code (`epc:ServiceCodedType`), but are not associated with digital signatures. If a digital signature is needed in conjunction with a service, it will be the service provider that signs the message.

Figure 11 generalizes the concept of service to any request that the sender will make from a receiver. This includes MSW clearance, berth allocation, maritime or commercial services. The diagram also allows receivers to be different from service providers, as discussed in 7.6.

A reference code (hashed rectangles in Figure 11) is assigned to each instance of the main objects and are used to keep track of the process flows in the communication session. The session itself is referenced by the service booking number (`ServiceBookingNumber`). The ship stay reference (`ShipStayReference`) can be used to link a session to the session context. The shown reference codes are:

— The message reference, which is assigned to all messages by the client. This is used to report errors in the message as detected by a server. It is also used to link a new service booking number to the request message when this is not done in the first HTTP session (see 7.10.5).

— The service booking number, which is assigned to all service requests by the receiver. It is used by both sender and receiver to refer to that specific service request and defines the communication session (see 7.7.2).

— The ship stay reference is a special reference code that is assigned by the receiver, which defines a session context, e.g. by an MSW or a port authority. It is used to uniquely identify the session context, e.g. a stay in territorial waters or a ship's stay in a specific port (see 7.7.3).

The message should be digitally signed by one or more parties. The MIG can define which parties' signatures that should be used.

7.13 General message structure

The general message structure is illustrated in Figure 12. It is in a standard multi-part format as defined in Clause 9.

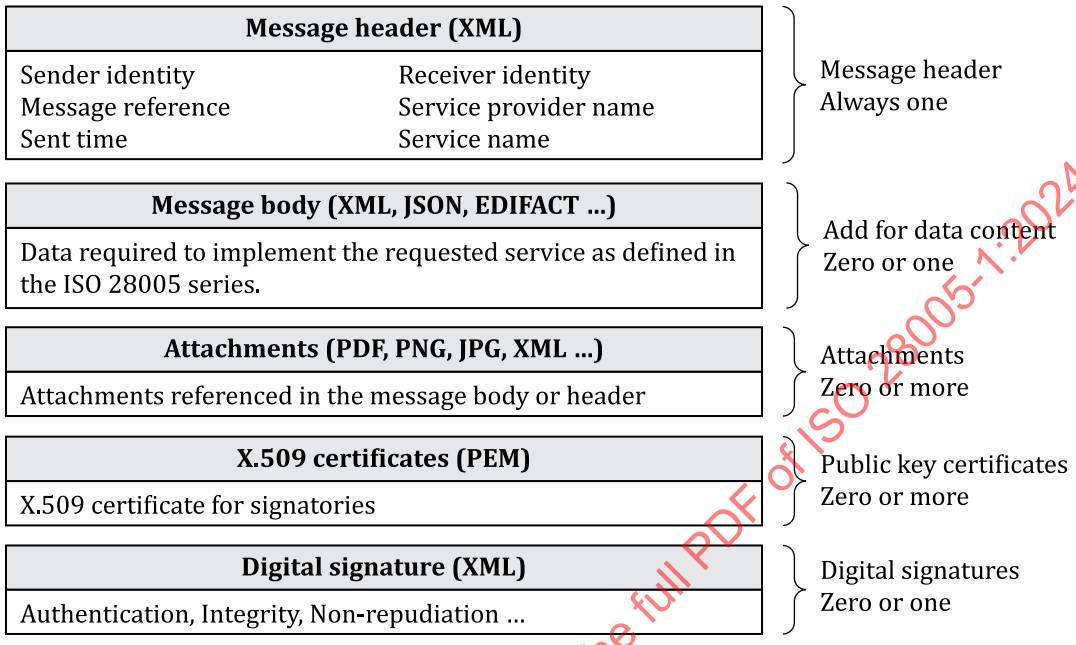


Figure 12 — General message structure

The message consists of:

— a standardized message header;

— a standardized message body (optional);

— one or more attachments (optional);

— public key certificates (optional);

— a standardized set of digital signatures (optional).

The message header includes the information that is needed to determine the further processing of the message. The details of the header data fields are described in Clause 10.

The XML based message body is defined by one XSD covering all possible data elements it can contain (see 11.2). Most of these elements are optional as the required data content differs between service APIs. The data requirements will be described in the MIGs.

This document describes the format of the header, body and signature as XML files. This document allows the body structure to be in other formats, e.g. JSON, ISO 19848 or EDIFACT. The body format is specified in the header as well as by the HTTP directives described in Clause 9.

NOTE 1 EDIFACT messages like BAPLIE (Bayplan/stowage plan occupied and empty locations message) and VERMAS (Verified gross mass message) are widely used in the transport and logistics industry. These messages are currently exchanged in ways not covered in this document. The specifications in this document are intended to be complementary to the ways that the industry currently exchanges EDIFACT messages.

If attachments are included, they should be referred to in other parts of the message (e.g. see 11.2). The attachments can be in any file format supported by HTTP. Further details are given in Clause 12.

EXAMPLE Examples of attachments can be special cargo safety sheets, e.g. in PDF, copies of certain non-digital ship or crew certificates, or pictures of stowaways.

The public key certificates are X.509 certificates in binary format as described in Clause 13. The certificates can be included in the message if there is no agreed-on public key infrastructure (PKI) that can be used to retrieve certificates. These message parts can also be used if the certificate is not available from the agreed-on PKI. Each certificate is a separate message part.

The digital signature message part shall be added if any of the other message parts are signed. 7.19 gives more information on the digital signature. The format of the digital signature message part is defined in Clause 14.

NOTE 2 The content of each message block shown in Figure 12 is for illustration only and does not represent the full contents nor the actual field names. The formal definitions can be found in Clauses 10, 11, 12, 13 and 14.

7.14 Digital signatures

The messaging system allows for the use of a public-private key digital signature system. Each party has its own secret private key and publishes the corresponding public key. The sending party can use the private key to digitally sign outgoing messages. The receiving party uses the sender's public key to verify the signature.

The digital signatures are used for the following purposes:

1. Integrity: A signature can be attached to a message part to verify that no information in that message part has been tampered with.
2. Authentication: The authenticity of the specified sender or authenticator can be verified.
3. Non-repudiation: Both the sender and receiver can store the signed version of a message and its acknowledgement. The signed message and its acknowledgement can be used by both parties to prove that a message was sent and received, and what information it contained.
4. Confidentiality: When sending information through general reporting portals as an MSW or PCS, it can be necessary to encrypt some information. This is to ensure the protection of personal information, i.e. sensitive health information or passenger list details from unauthorized access. This can be done by encrypting the corresponding part of the message and inserting it as an attachment. See 10.8 and 12.2 for a description of the encryption process.

It is not necessary to encrypt data for general protection from external parties, as the use of the secure HTTPS transport protocol is a requirement (see 7.15 and Clause 15).

Clause 14 specifies general requirements to the signature, the public key certificates and the PKI.

7.15 Secure data transfer

This document uses HTTP over TLS to ensure confidentiality of message content with respect to parties which are not part of the message exchange (see Clause 15).

7.16 Additional authorization for accessing API

As a digital signature can be added to all message parts, it is not in general necessary to require authorization when accessing an API, as the identity of the sender can be determined from the signature. However, it will in some cases be necessary to authorize access to the API access point as this can reduce protocol and processing overhead. Clause 18 defines how this shall be done, if implemented.

If authorization is used, it should not be solely rely on as a replacement for digital signatures. Confidentiality and integrity can to some degree be replaced using HTTPS, and authenticity can to some degree be replaced

by authentication. Non-repudiation requires proving the integrity and actual content of each individual message at both the sender's and receiver's side and that requires digital signatures for all messages.

7.17 Message implementation guide

This document defines the general message exchange systems. To provide the information that is necessary to implement a specific application's requirements, a message implementation guide (MIG) shall be provided for each service. Requirements for the MIG are defined in Clause 19.

Annex B contains a MIG for the access authorization service. Annex C contains a MIG for the maritime single window and for mandatory ship reporting systems.

NOTE These last two MIGs provide implementation guides related to two applications for which the data set in ISO 28005-2 has been developed.

7.18 Other formats than XML for the message body

This document defines XML as the default message syntax for the message header, body and signature. 10.4 specifies how other file formats, e.g. ISO 19848, JSON or EDIFACT, can be specified as the ones used in the message body. 11.4 and 11.5 describes how EDIFACT can be used in the message body and as status messages, and 11.6 describes how JSON can be used. 11.7 defines how ISO 19848 can be used as message body.

If other formats than default XML is accepted by an application, the MIG shall specify what formats the application accepts. This may also include other formats than the ones listed in 10.4.

7.19 No explicit service discovery

This document does not specify how senders determine which URL to use for accessing the API access point of receivers. There are several mechanisms that can be used for this purpose:

1. The relevant URLs for a port can be published together with other information needed to plan the port call. This can be, for example, the port information book or guide.

   NOTE 1 Such guides or books are published by many ports. See also the port information manual.[21]

2. E-navigation maritime service 4 (Port Support Services) can be used to publish this information through suitable digital communication services.

   NOTE 2 See the e-navigation implementation plan.[22]

3. Independent service discovery mechanism can also be available.

   NOTE 3 IEC 63173-2 contains an example of such a service.

4. The access authorization request (see Clause 18) can return a list of available services. Given that the authorization URL is known, this can serve as a local service discovery.

8 Message exchange patterns

8.1 General rules

8.1.1 Application of this specification

This clause specifies how messages can be exchanged between the sender and receiver for several different data exchange scenarios. The MIG will specify what pattern is used in a specific application, and Clauses 16 and 17 will define how these exchanges are implemented in an API.

Message names and general patterns are based on IMO FAL.5/Circ.46.[1] However, there are some differences as to how the message names and patterns are implemented, as described in Table 4.

Table 4 — Message names and general patterns: mapping between this document and IMO FAL.5/Circ.46

Cross-reference in IMO FAL.5/Circ.46	Implementation in this document
Figure 1 : Simple information distribution	Implemented in pattern 3 (8.2.3).
Figure 2 : Simple request	Implemented in pattern 4 (8.2.4).
Figure 3 : Basic sequence	Implemented in pattern 1 (8.2.1)
Figure 4 : Updated request	Implemented in pattern 1 (8.2.1).
Figure 5 : Several senders	See 8.1.6 for implementation in this document.
Figure 6 : Through MSW	Implementation specific, not covered by this document.
Figure 7 : Multiple recipients	Implementation specific, not covered by this document.
n/a	Pattern 5 (8.2.4) has been added to allow subscription.
n/a	Pattern 2 (8.2.2) has been added as implied in the circular.

The specific rules for message exchanges in 8.1.2 to 8.1.7 make references to data elements in the message header. The message header is defined in Clause 10 and an overview of the structure of the header is shown in Table 9.

8.1.2 Use of reference codes

An overview of reference codes is shown in Figure 11 and the corresponding context is discussed in 7.7.

The `ShipStayReference` is associated with the session context (see 7.7.3) and contains a receiver specified reference code. 7.7.3 defines how this reference code shall be used.

The `ServiceBookingNumber` is associated with the service session (see 7.7.2), and contains a receiver specified reference code.

The `MessageReference` is defined by the client as a reference to the outgoing message itself. It is used by the server in the `MessageStatus` fields to give feedback on processing of the message. 7.10.5 specifies how this code shall be used.

8.1.3 Use of final flag in message header

The use of the final flag depends to some degree on the application. The general principle is that the final flag shall be used to order or signal the completion of a service as defined in 7.10. This requires some specific behaviour from the parties:

1. The receiver shall always set the final flag in the status message that marks the termination of the service. Termination can mean that: errors in the request make it impossible to fulfil the service request, the service timed out, the session context ended or the service was completed.

   NOTE 1 This can mean that the receiver clears all or some service session related information.

2. The receiver shall always send a final status message when the service terminated. This also applies to services terminated due to termination of the session context, a timeout, or any other server-initiated action.

   NOTE 2 It is not always possible to complete this step if the service was requested by a synchronous sender.

3. The sender can set the final flag if the sender wants the service to end. The receiver shall immediately terminate the service when the message is received and return a status message to the sender indicating the final status of the service by having the final flag set.

   NOTE 3 If insufficient data for the relevant service had been provided when the final flag was sent, this means that the service terminates with an error.

4. A cancellation message from the sender (see 8.2.1) shall imply that the final flag is set, and the receiver shall return a set final flag in the corresponding service status message.

The final flag is not indicated in the sequence diagrams shown in 8.2.1 to 8.2.5, but the above rules are implied.

8.1.4 Use of service timeout or session context end

The sender can specify a timeout for the request and the receiver can also have an internal timeout, unless the session context provides the rules for when requests can be terminated. The latter can also indirectly be the case if the session context has a natural ending point. See 7.11 for more details.

Rules 1 and 2 in 8.1.3 apply to the receiver when the service is terminated due to timeout or session context end.

Timeout or session context end is not shown in the sequence diagrams shown in 8.2.1 to 8.2.5. Rules 1 and 2 in Clause 8.1.3 are implied.

8.1.5 Status and error codes

As shown in Figure 13 on the left, the message header contains three elements that together specifies the status of a received message and of a requested service.

The message and service status elements have no meaning in a sender's message, unless that message is a response to an asynchronous service request update from the receiver. In that case, the message status field can be filled in, e.g. to signal an error in the message from the receiver.

7.10.5 defines the rules that apply to the use of the status code fields.

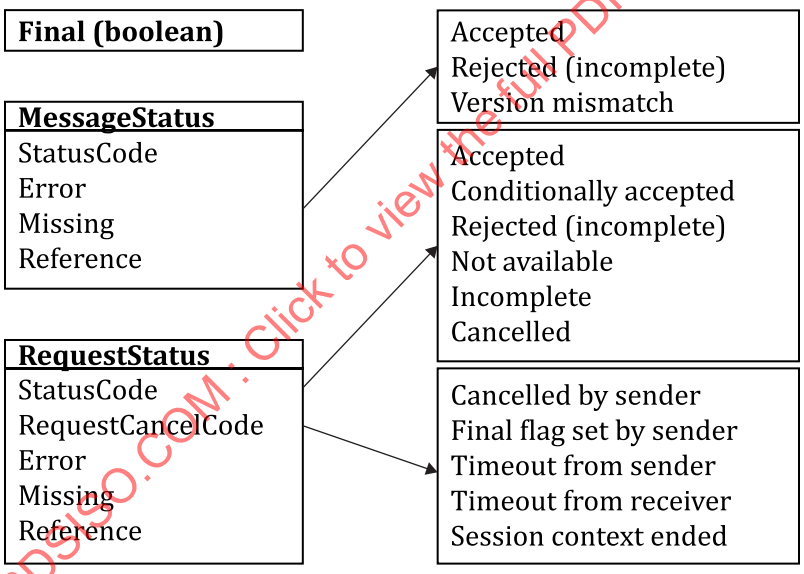


Figure 13 — Status codes in message header

The final flag is used by the sender to signal to the receiver that a service request shall be terminated after the processing of the message, or by the receiver to signal that a service request has been terminated. If the receiver does not set the flag, the service request is still active (see 8.1.3).

Annex F contains the definition of the code values that shall be returned in message or service status fields. All codes are taken from the same code set but are used differently in the message status and the service status messages.

The message status will give the status of the last received message. Table 5 lists the relevant codes for the message status (see also 7.10.1).

Table 5 — Message status codes

Status code	Message status code meaning
Rejected incomplete [a]	Information is missing, see Missing field – message is rejected
Rejected [a]	Message was rejected due to errors in the message, see Error field
Accepted	The message was accepted, see request status for information on service
Version mismatch [a]	Sender's version code not supported by receiver

Key

[a] These codes, if returned as a response to the sender's initial request, mean that the service was not created. If the service was already created, the codes mean that the message was ignored and no change to the service was done.

If the service was created, possibly as a result of the sender's message, the message header can also contain a service request status code. If the service request status field is empty, it means that the status of the service request will be returned later or shall be polled for (see 7.10.1). The message reference code in the message status element of the request status, when polled or delivered, will refer to the sender's initial request message.

If and when the service request status is provided by the receiver, Table 6 contains the relevant codes. The code values "Rejected incomplete" and "Rejected" has the same meaning as for the message status and additional information shall be available in the `Error` or `Missing` data fields of the service request status. See 7.10.2 for details of the service request state transitions.

Table 6 — Service status codes

Status code	Service status code meaning
Accepted	Service has been executed and can be executed again if the Final flag is not set.
Conditionally accepted	Service request accepted and service execution is pending.
Rejected [a]	Service request rejected due to errors, see "Error" field.
Rejected incomplete [a]	Service request rejected due to missing data, see "Missing" field.
Not available [a]	This service does not exist in this system.
Incomplete [b]	Service created; data are missing for the service to be started. See "Missing" field.
Cancelled	Service was cancelled by sender or receiver, see request cancel code.

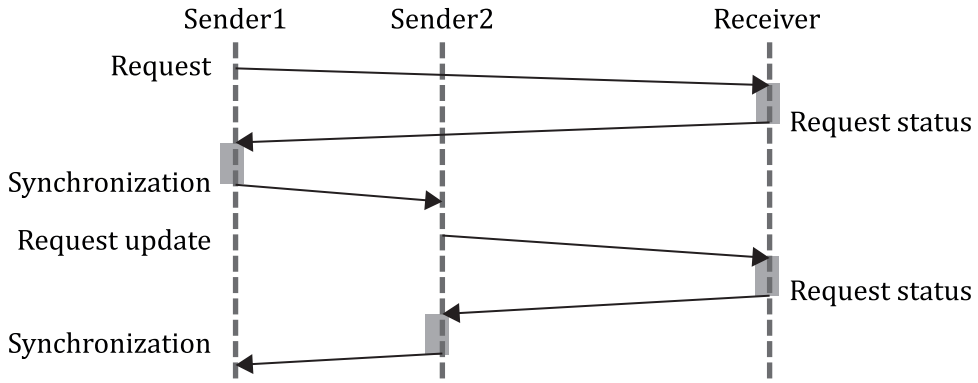
[a] These codes means that the service was not created.

[b] This code means that the service was created, but the execution is pending until additional data are provided.

If the status code is "Cancelled", the `RequestCancelCode` field will contain additional information as illustrated in Figure 13. The code values are listed in Table F.2.

8.1.6 Multiple senders

The sequence diagrams in 8.2 only show one sender. However, this document allows multiple senders to participate in the message exchange. If this principle is used, it requires that the different senders synchronize the exchange between themselves as indicated in Figure 14. This can involve more than two senders as illustrated in Figure 14, in which case all senders are required to synchronize request information with all other senders.



Key

See Figure 2 for key references.

Figure 14 — Synchronization of different senders

The exact information required to synchronize depends on the application and role of each sender. As a minimum, the receiver and the `ServiceBookingNumber` shall be known to all senders.

8.1.7 Interleaving update requests with status messages

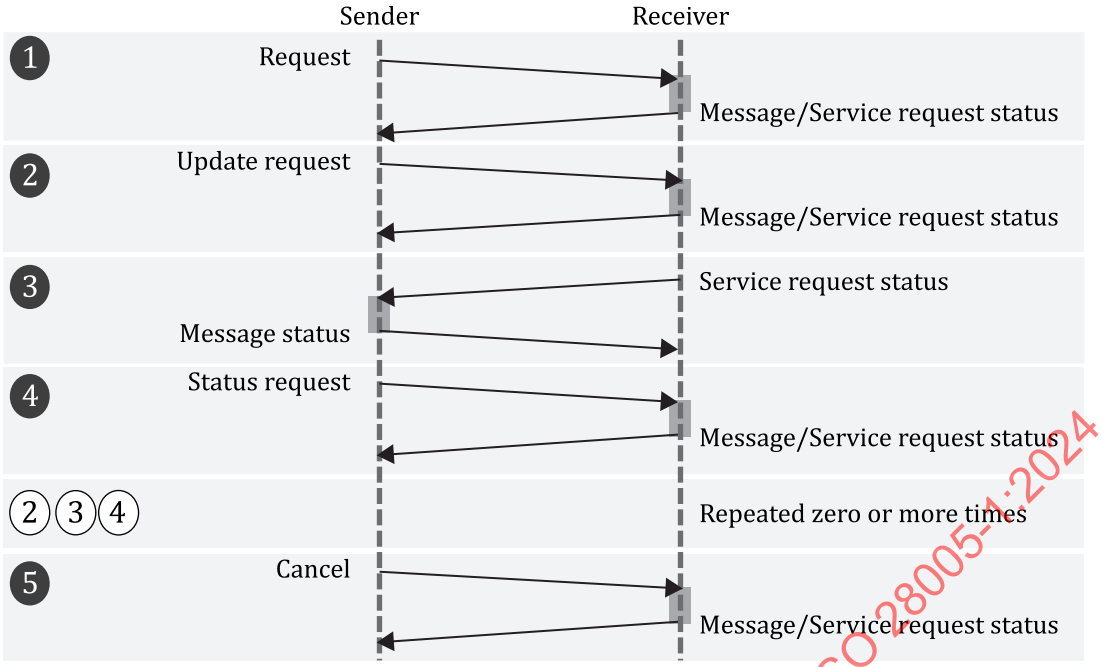
In general, the sender is allowed to send new update requests without waiting for the status message related to the previous update request. This does not apply to the initial request where it is necessary to wait for a service request status message with a valid `ServiceBookingNumber` before new update requests can be sent.

8.2 Sequence diagrams

8.2.1 Pattern 1: Service request and updates

The message exchange pattern that is the most general and expected to be the most common is shown in Figure 15. This is used to send a request from the sender to the receiver, normally including data in a message body, and getting a reply from the receiver, also possibly including data in a message body. The request may be updated by the sender in which case the receiver shall send a new request status.

In its simplest form, the message exchange pattern will only consist of sequence 1 where the response to the service request is returned immediately from the receiver as a message and service request status. This also means that sequence 1 is the only part of the message pattern that is used if the receiver returns an error code for the initial request.



Key

See Figure 2 for key references.

X sequences 1 to 5, which are defined by the corresponding lightly shaded areas of the message exchange

(X) sequences 2, 3 and 4, which can be repeated zero or more times at this point in the message exchange

Figure 15 — Message exchange pattern for service requests and updates

The key references 1, 2, 3, 4, 5 and the corresponding shaded areas in Figure 11 represent different and possibly optional sequences of the message exchange:

1. The initial request is issued by the sender and can result in a complete service execution, in which case the receiver will return a service request status. This may include the Final flag being set, in which case the service has completed. If no service status is returned, the service status will be sent later or polled for (see sequence 3) and only a message status is returned.

2. The sender can in some cases update a request, even if it the service was already completed once. In this case, the same type of return messages as in case 1 will be generated by the receiver. This may be repeated several times. This sequence can also be repeated after sequence 3.

3. If the service execution was not completed in sequences 1 or 2, the service request status will be sent as an asynchronous message by the receiver, when the service is completed or otherwise terminated. This is the normal scenario for asynchronous communication as described in Clause 16. The updated service request status shall be resent after each update request in sequence 2.

4. At any time, the sender can query the receiver for the status of the service request. This sequence is also used in synchronous polling as described in Clause 17.

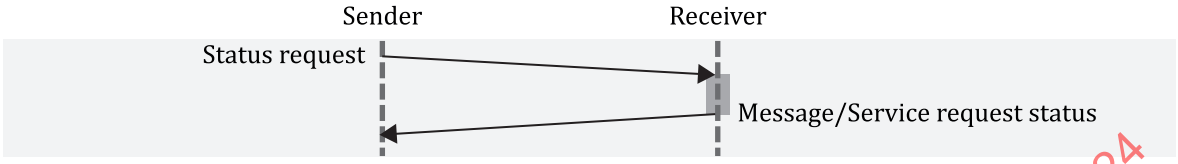
5. The sender can cancel the service at any time, also after a service request status has been received. It is possible that a cancel has no effect on the service if a successful request status was already transmitted.

The implementor of a sender should note that there is a possible race condition where the receiver has sent a service request status, but that this has not been received by the sender before the request update is sent. In this case, the sender will get the service request status on the previous message after the last update message was sent. The message reference codes in the header of the status messages will show what request or update message they refer to.

The receiver is required to send a service request status after each request or request update from the sender.

8.2.2 Pattern 2: Status poll

The sender can at any time perform a status poll to check the status on an ongoing service request or a previously sent message. This can be used in a setting where the sender has the responsibility for checking the service status itself as described in Clause 17, but can also be used in asynchronous message exchanges (sequence 4 in Figure 15). The message pattern is shown in Figure 16.



Key

See Figure 2 for key references.

Figure 16 — Message exchange for status poll

The sender is not allowed to include a message body for this type of request.

There are two different ways this pattern can be used:

1. It can be used to check the status of a service request before a service booking number has been returned by the receiver. In this case, the sender sends the message reference for the initial request message and expects either only a message status or a service and message status back. The service status is returned if the service booking number has been assigned.

2. It can be used to check the status of a service where the receiver already has returned a service booking number. In this case the receiver will return a combined message and service status.

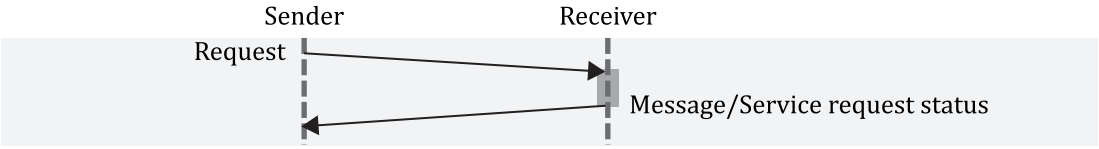
The return value from the receiver will be the current status of the service or the message, or an error code if the service or message does not exist. If a service status is returned, the receiver can, if it is relevant and available, also return a message body that has been created as a result of the service execution.

To be able to return a message status, the receiver shall keep a record of received service requests and the corresponding message references. The record of messages can be deleted when the corresponding session context terminates.

The receiver frontend (see 15.1) or corresponding part of a receiver shall keep track of all service status changes so that a message or status request can be answered synchronously in the same HTTP session. This is necessary for the implementation of synchronous API senders (see Clause 17).

8.2.3 Pattern 3: Simple report

This pattern is shown in Figure 17 and involves one single request from the sender that shall be directly answered with a message and service status from the receiver. This is similar to sequence 1 in Figure 15. This pattern can be used for simple types of reports that do not require any significant processing by the service provider or receiver for acceptance.



Key

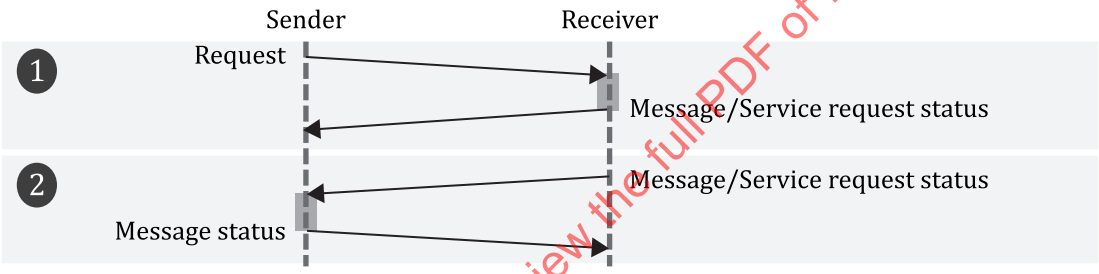
See Figure 2 for key references.

Figure 17 — Message exchange pattern for simple report

The service descriptor (see 7.10.2) will be deleted immediately after the status message has been sent. This service cannot usefully be polled or cancelled, and the client cannot update the request. Any cancel or update request shall get a negative message status return (not available).

8.2.4 Pattern 4: Request information

This pattern is shown in Figure 18 and involves one single request from the sender that may be directly answered with the requested data from the receiver as shown in sequence 1, or asynchronously as shown in sequence 2. Any service data that is included in the message body will in both cases be accompanied by a service request status in the message header.



Key

See Figure 2 for key references.

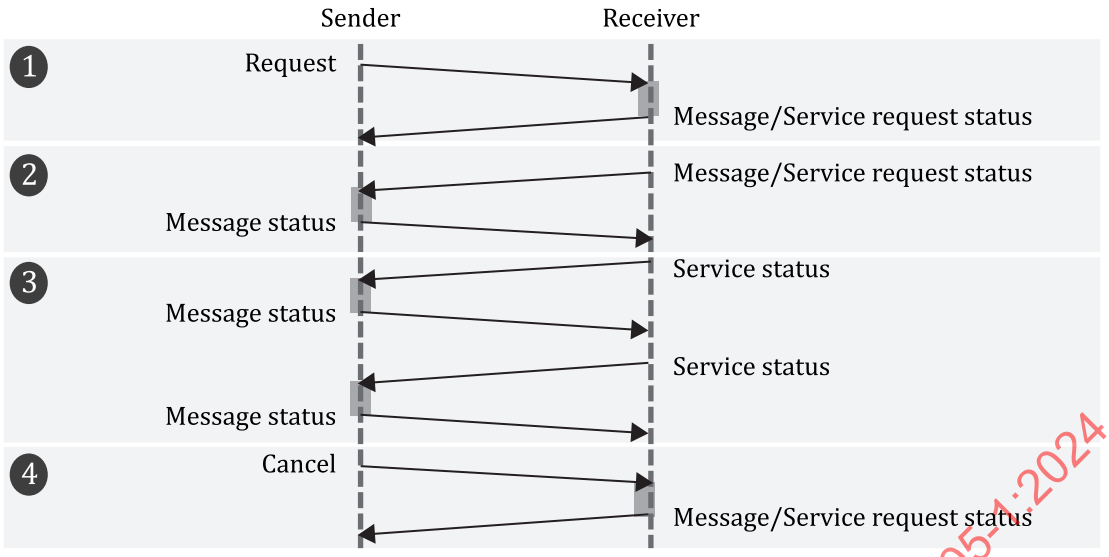
X sequences 1 and 2, which are defined by the corresponding lightly shaded areas of the message exchange

Figure 18 — Message exchange pattern for information requests

The service descriptor (see 7.10.2) will be deleted immediately after the status message have been sent. This process cannot usefully be polled or cancelled, and the client cannot usefully update the request. Any cancel or update request shall get a negative message status return (not available).

8.2.5 Pattern 5: Subscribe to service or information

The pattern shown in Figure 19 is mainly intended for subscribing to information updates from the server but may also involve execution of other, e.g. physical services.



Key

See Figure 2 for key references.

X sequences 1 to 4, which are defined by the corresponding lightly shaded areas of the message exchange

Figure 19 — Message exchange pattern for subscription

Sequence 1 and 2 in Figure 19 shows the normal, possibly asynchronous, response to a request. This is identical to sequence 1 and 2 in Figure 15 (pattern 1) in 8.2.1.

Sequence 3 in Figure 19 shows the subscription phase where the receiver will send a sequence of service status messages, each presumably with different information in the message body. The time interval for the retransmission of data are determined by the sender or may be specified as service data in the message body. This is described in the MIG. The messages from the receiver, except the first, do not contain a message status as they are not related to an action by the sender.

Sequence 4 shows a sender-initiated cancellation of the subscription. This is replied to by a service and message status.

NOTE Due to race conditions before a cancellation takes effect, it is still possible that a service data message is received by the client after the cancel was accepted by the server.

Termination can also be initiated by the receiver, e.g. because of a time-out or other events on the receiver side. This shall be signalled by a service status message from the receiver.

It is not permitted to send a request update. If it is necessary to change a subscription, it shall first be cancelled and then a new request shall be sent.

9 Using HTTP multi-part message

9.1 General

This clause gives an overview of the HTTP multi-part/form-data message structure (see RFC 7578). The media type multipart/form-data follows the multipart MIME media type definitions (see RFC 2046), which means that the data body of the media type multipart/form-data consists of multiple parts separated by a fixed boundary.

9.2 Example of an ISO 28005-1 multi-part message

The messages exchanged between sender and receiver contain different message parts as described in 7.13. The message will use the multi-part format to distinguish the different parts from each other and for transferring information about the content of each part. An example of the format is shown in Figure 20.

```
Content-Type: multipart/form-data; boundary="r4nd0m"
Content-Encoding: gzip

Prose text: This is an electronic message in the ISO 28005-1 format.
The attachments contain the different parts of the message.

--r4nd0m
Content-Type: application/xml; charset=utf-8
Content-Disposition: form-data; name=header;

[XML header goes here]
--r4nd0m
Content-Type: application/xml; charset=utf-8
Content-Disposition: form-data; name=body;

[XML body goes here - zero or one of this block]
--r4nd0m
Content-Type: application/pdf
Content-Disposition: form-data; name=attach1; filename=file1.xxx;

[attachment data goes here - zero or more of these blocks]
--r4nd0m
Content-Type: application/pkix-cert;
Content-Disposition: form-data; name=cert1; filename=cert1.cer;

[A DER encoded X.509 certificate goes here - zero or more]
--r4nd0m
Content-Type: application/xml; charset=utf-8
Content-Disposition: form-data; name=signature;

[XML signature goes here - zero or one of this block]
--r4nd0m--
```

NOTE An almost identical format can be used for email transfer by using the MIME specifications directly.

Figure 20 — Example of multi-part message layout

9.3 to 9.7 explain the use of important directives and other components of the message shown in Figure 20. Titles of clauses contain copied and unique text from the example, which is in general organized from the top to the bottom. Some clauses describe directives that are repeated in different parts of the message. In these cases, the clause will explain the use of the corresponding directive in all parts.

The example in Figure 20 is a minimum valid message. HTTP supports several other directives that may be used by a client, but which do not have any special meaning in this document. All valid HTTP directives shall be accepted by the server and they shall be processed according to rules in RFC 7578.

9.3 Content-Type: multipart/form-data

EXAMPLE Content-Type: multipart/form-data; boundary = ”r4nd0m”

“`Content-Type: multipart/form-data`” is the first-line directive in a multipart message with different encodings in the different parts. The attribute “`boundary=r4nd0m`” specifies the pattern (prefixed with“–”)

that is used to separate the message parts. It is the client's responsibility to make sure that this pattern is unique in the whole multi-part message. The final boundary is terminated by an additional"–" at the end of the string.

The boundary pattern should be long and complicated enough so that it is not confused with a randomly occurring string in any of the message parts. One common solution is to use an UUID pattern.[18]

9.4 Content-Encoding: gzip

EXAMPLE Content-Encoding: gzip

The optional Content-encoding directive can be used to specify compression of a message by specifying the content encoding as "gzip". The compression can be used to reduce the size of large messages.

Both senders and receivers shall accept the gzip file format as defined in RFC 1952.

The Content-Encoding directive shall not be used by the client in any message part in accordance with RFC 7578, which specifies that this directive used in message parts should be discarded by the server.

9.5 Prose text

EXAMPLE Prose text: This is an electronic message in the …

The prose text can be used if there is a possibility that a message is read directly by a human operator. This text is not required.

9.6 Content-Type: application, image or other

EXAMPLE Content-Type: application/xml; charset = utf-8

In the different message parts, this directive specifies the document format of the message part. The formats currently specified for the digitalized and coded message parts in this document are:

1. `application/xml`: This specifies XML encoding. It is recommended to specify the character set used, e.g. UTF-8 as shown in the example (see 4.4).
2. application/json: This specifies a message part encoded in JSON. It is not necessary to specify the character set here as the default is UTF-8 for JSON.
3. `application/EDIFACT`: This specifies a message part in EDIFACT.
4. `application/pkix-cert`: DER encoded X.509 certificate.
5. `application/gzip`: GZIP compressed part of a body message (see 12.1).
6. `application/octet-stream`: Any other type specified, e.g. in special attachment entries. This shall be used for encrypted parts of the message body (see 10.8).

If none of the above codes can be used, the attachment parts can use any recognized format. The body and signature part can also use another recognized format. A list of recognized formats is maintained by the Internet Assigned Numbers Authority.[25] The formats that can be used, if different from the ones specified in this document, shall be documented by the MIG.

NOTE Some examples of formats that can be used in attachments are "application/pdf" for PDF files or "image/pgn" for PGN-encoded pictures.

9.7 Content-Disposition: form-data; name = name; filename = file.name;

EXAMPLE 1 Content-Disposition: form-data; name = header;

EXAMPLE 2 Content-Disposition: form-data; name = attach1; filename = file1.xxx;;

The content disposition directive shall be used in all message parts. It is used to assign a name to each part of the message and to assign a file name to attachments. The file name defines a name for the attachment which is used as reference in the message body (see 12.1), or in the message header (see 10.8).

The client shall use the names “header”, “body” and “signature” in the name fields for the corresponding parts of the message. It shall use “attach*N*” for attachment number *N* and “cert*N*” for certificate number *N*. The implementor shall rely on the fixed ordering of parts as defined in 7.13 and the `MessageManifest` element in the message header (see 10.10) to recognize these parts.

The name and the file name shall use US-ASCII encoding. The structure of the filename is not specified by this document. Each attachment part shall have a file name that is unique within the message.

10 Definitions related to the message header part

10.1 General

The message header structure defined in 10.10 is intended to allow a front-end server to determine the context of the message and to select the service provider to be invoked, so as to allow the further processing of the message (see 15.1), without reading the body part of the message.

10.2 to 10.9 define the data types that are specifically used in the message header structure. Other general data elements that are used in the header are defined in Clauses 5 and 6.

The message header as defined in 10.10 shall be inserted as an XML text file and as the first message part in the HTTP multi-part message. The corresponding content type shall be `application/xml`. See Clause 9 for details.

10.2 epc:MessageFunctionCodeContentType – Message function code

10.2.1 Definition

The message function code specifies the type of message, e.g. request, acknowledgement etc.

10.2.2 Type

```
<xs:simpleType name="MessageFunctionCodeContentType">
  <xs:restriction base="epc:token"/>
</xs:simpleType>
```

10.2.3 Representation

The codes are defined by UNECE data element 1225, Message function code.[36] Annex E defines the code values that shall be supported by users of this document.

10.3 epc:ReplyInformationType – Type of sender response code

10.3.1 Definition

This element identifies the type of requested response and, if necessary, the URL of the receiver API.

10.3.2 Type

```
<xs:complexType name="ReplyInformationType">
  <xs:sequence>
     <xs:element name="ReplyType" type="epc:int"/>
     <xs:element name="ReplyURI" type="epc:anyURI" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
```

10.3.3 Representation

The following elements are used as follows:

— `ReplyType`: How to reply to this message. Table 7 defines the allowable codes.

— `ReplyURI`: A valid URL for a reply if needed, i.e. if `ReplyType` code 2 is used.

NOTE A valid URL is a HTTP URL, e.g. "https://iso.org/iso-28005?service=5". The example shows the use of additional codes after the question mark. The question mark and codes can be omitted or there can be more codes, dependent on the receiver's requirements.

Table 7 — Reply type codes

Code	Reply type
1	The sender will poll the receiver's advertised URL for replies.
2	The sender specifies a HTTPS URL for replies in the attribute `ReplyURL`.

10.4 epc:MessageBodyFormatContentType – Format of body data

10.4.1 Definition

This element identifies the format of the message body.

10.4.2 Type

```
<xs:simpleType name="MessageBodyFormatContentType">
  <xs:restriction base="epc:token"/>
</xs:simpleType>
```

10.4.3 Representation

The format codes are defined in Table 8.

Table 8 — Body format codes

Code	Body format
0	No message body is included. This has the same function as omitting the attribute from the message header.
1	XML as defined in 11.2.
2	UN/EDIFACT message body as defined in 11.4.
3	UN/EDIFACT status message as defined in 11.5.
4	JSON message body as defined in 11.6.
5	ISO 19848 message in XML format as defined in ISO 19848:2024, A.2
6	ISO 19848 message in JSON format as defined in ISO 19848:2024, A.3
> 99	Service specific format. Format will be defined in the corresponding MIG. Any code value from 100 and above can be used for this. The code value to use may be specified in the MIG.

10.5 epc:ServiceTypeCodeContentType – Code for identification of service type

10.5.1 Definition

This data item contains a code that identifies a group of electronic port clearance services, reporting requirements or maritime services. This code is used to allow different code lists for different types of services (see 10.6).

10.5.2 Type

```
<xs:simpleType name="ServiceTypeCodeContentType">
  <xs:restriction base="epc:token"/>
</xs:simpleType>
```

10.5.3 Representation

This data item contains a service type group code as defined in Table G.1. Annex G defines the code values that shall be used.

10.6 epc:ServiceCodeContentType – Code for identification of a service in a group

10.6.1 Definition

This data item contains a code that identifies a specific electronic port clearance service within a service type group as referenced to in 10.5.

10.6.2 Type

```
<xs:simpleType name="ServiceCodeContentType">
  <xs:restriction base="epc:token"/>
</xs:simpleType>
```

10.6.3 Representation

This data item contains a service code. The code values to be used are defined in Annex G. Tables G.2 to G.5 contain the groups of service codes for the types of services referenced in 10.5 and in Table G.1.

NOTE Service codes can be numeric or alphanumeric.

10.7 epc:StatusType – General message and service request status and error codes

10.7.1 Definition

This data type defines status and error codes related to the receipt of a message or execution of a service. This covers various errors or omissions in the message header and signature, general formatting errors in the message body, as well as error codes related to the service execution.

This data object refers to a sent message or a service request. The client that sends the message is also required to check the return codes for errors on protocol level (see Clause 15).

10.7.2 Type

```
<xs:simpleType name="StatusCodeContentType">
  <xs:restriction base="xs:token"/>
</xs:simpleType>

<xs:simpleType name="RequestCancelCodeContentType">
  <xs:restriction base="xs:token"/>
</xs:simpleType>

<xs:complexType name="StatusType">
  <xs:sequence>
    <xs:element name="Error" type="epc:string" minOccurs="0"/>
    <xs:element name="Missing" type="epc:xpath" minOccurs="0"
                maxOccurs="unbounded"/>
    <xs:element name="StatusCode"
                type="epc:StatusCodeContentType"/>
    <xs:element name="Reference" type="epc:ReferenceCodeType"/>
    <xs:element name="RequestCancelCode"
```

```
                type="epc:RequestCancelCodeContentType" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
```

10.7.3 Representation

The data elements are as follows:

— `Error`: This a text string meant for human consumption. It shall be used when the message or request contains errors that cannot be automatically corrected by the sending computer. If the Error field is included and non-empty, corrective actions normally require human intervention. See the "Missing" attribute for machine correctable errors.

— `Missing`: If there were missing information elements in the message, and this can be automatically corrected by the sending computer, these attributes will identify the missing information elements in Xpath notation (see 4.2.9).

— `StatusCode`: This is the status of the message or request. Status codes are taken from UN/EDIFACT consolidated code list 1373.[37] The codes used in this document are defined in Table F.1. See also 8.1.5 for further discussions on the use of this code.

— `Reference`: The reference code for the message or service booking number, to which the status information object relates.

— `RequestCancelCode`: If the status code is “Cancelled”, i.e. the status applies to a service request that was terminated for other reasons than an error, this code shall be used to explain why the request was cancelled. The codes are listed in Table F.2. This field shall only be used in the service request status instance.

10.8 epc:SpecialAttachmentType – Description of special attachment

10.8.1 Definition

It is possible that parts of the message body require protection from unauthorized access by using encryption (see 11.3). It can also be useful to send parts of the message body in other formats than XML, e.g. UN/EDIFACT for passenger or cargo lists. Such message parts can be included as attachments (see 7.13) and shall then be listed in the special attachments data object. If a message part is listed here, the corresponding data objects shall not be included in the general message body structure.

This document does not require a receiver to accept other formats than the plain message body XML structure, so use of this feature is dependent on the published capabilities of the receiver. See Clause 19 for an explanation of how this can be described in the MIG. Instructions related to encrypting message parts may be returned from the access authorization function as described in Clause 18.

10.8.2 Type

```
<xs:simpleType name="AttatchmentTypeContentType">
  <xs:restriction base="xs:token">
    <xs:length value="2"/>
  </xs:restriction>
</xs:simpleType>

<xs:complexType name="SpecialAttachmentType">
  <xs:sequence>
    <xs:element name="AttachmentType"
                type="epc:AttatchmentTypeContentType" minOccurs="0"/>
    <xs:element name="DataElementName" type="epc:xpath" minOccurs="0"/>
    <xs:element name="FileName" type="epc:string" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
```

10.8.3 Representation

The attributes have the following contents:

— `AttachmentType`: This is the type of special attachment. Code values are listed in Table D.1.

— `DataElementName`: This is the Xpath name of the component of the body that has been encrypted (see 4.2.9).

— `FileName`: This is the name of the attachment file as defined in 9.7.

10.9 epc:MessageManifestType – Number of message parts

10.9.1 Definition

This data block contains the number of different types of message parts.

10.9.2 Type

```
<xs:complexType name="MessageManifestType">
  <xs:sequence>
    <xs:element name="HasAttachments" type="epc:int" minOccurs="0"/>
    <xs:element name="HasBody" type="epc:int" minOccurs="0"/>
    <xs:element name="HasCertificates" type="epc:int" minOccurs="0"/>
    <xs:element name="HasSignature" type="epc:int" minOccurs="0"/>
  </xs:sequence>
```

10.9.3 Representation

This data structure contains the number of message parts of each type. The values for each attribute are from zero and higher. If an attribute is omitted or empty, the corresponding value shall be understood as zero. The attributes are as follows:

— `HasBody`: Value zero or one to indicate presence of a body part.

— `HasAttachments`: Value zero or higher to count number of attachments.

— `HasCertificates`: Value zero or higher to count number of X.509 certificates.

— `HasSignature`: Value zero or one to indicate presence of a signature part.

10.10 epc:EPCMessageHeaderType – Standard header for an EPC message

10.10.1 Definition

This data block contains the definition of the message header structure. The same message structure is used by both the sender and receiver, and independently of whether they are acting as client or server. Different attributes will be used in the different cases. This will be described in the relevant MIG.

10.10.2 Type

The formal XSD description follows below. It is complemented by Table 9 to make the description of the attribute elements easier to understand.

```
<xs:complexType name="EPCMessageHeaderType">
 <xs:sequence>
  <xs:element name="Final" type="epc:boolean" minOccurs="0"/>
  <xs:element name="MessageCreatedTime" type="xs:dateTime"
              minOccurs="0"/>
  <xs:element name="ReceiverId" type="epc:ContactInfoType"
              minOccurs="0"/>
  <xs:element name="RequestValidityEnd" type="epc:dateTime"
              minOccurs="0"/>
```

```
  <xs:element name="SenderId" type="epc:ContactInfoType"
              minOccurs="0"/>
  <xs:element name="SentTime" type="epc:dateTime" />
  <xs:element name="ServiceName" type="epc:string" minOccurs="0"/>
  <xs:element name="ServiceProviderName" type="epc:ContactInfoType"
              minOccurs="0"/>
  <xs:element name="ShipId" type="epc:ShipIDType" minOccurs="0"/>
  <xs:element name="ArrivalDeparture" type="epc:ArrivalDepartureType"
              minOccurs="0"/>
  <xs:element name="Authenticator" type="epc:AuthenticatorType"
              minOccurs="0"/>
  <xs:element name="AuthorizationToken"
              type="epc:AuthorizationTokenType" minOccurs="0"/>
  <xs:element name="MessageBodyFormat"
              type="epc:MessageBodyFormatContentType" minOccurs="0"/>
  <xs:element name="MessageFunctionCode"
              type="epc:MessageFunctionCodeContentType" minOccurs="0"/>
  <xs:element name="MessageManifest" type="epc:MessageManifestType"
              minOccurs="0"/>
  <xs:element name="MessageReference" type="epc:ReferenceCodeType"
              minOccurs="0"/>
  <xs:element name="ServiceBookingNumber" type="epc:ReferenceCodeType"
              minOccurs="0"/>
  <xs:element name="ShipStayReference" type="epc:ReferenceCodeType"
              minOccurs="0"/>
  <xs:element name="RequestReplyMethod" type="epc:ReplyInformationType"
              minOccurs="0"/>
  <xs:element name="ReportingSystem" type="epc:ReportingSystemType"
              minOccurs="0"/>
  <xs:element name="RelayReportingSystem"
              type="epc:ReportingSystemType" minOccurs="0"
              maxOccurs="unbounded"/>
  <xs:element name="ServiceCode" type="epc:ServiceCodeContentType"
              minOccurs="0"/>
  <xs:element name="ServiceTypeCode"
              type="epc:ServiceTypeCodeContentType" minOccurs="0"/>
  <xs:element name="SpecialAttachment" type="epc:SpecialAttachmentType"
              minOccurs="0" maxOccurs="unbounded"/>
  <xs:element name="MessageStatus" type="epc:StatusType"
              minOccurs="0"/>
  <xs:element name="RequestStatus" type="epc:StatusType"
              minOccurs="0"/>
  <xs:element name="SystemId" type="epc:SystemIdType" minOccurs="0"/>
  <xs:element name="Version" type="epc:VersionType" />
 </xs:sequence>
</xs:complexType>
<xs:complexType name="EPCMessageSignatureType">



<xs:element name="EPCMessageHeader" type="EPCMessageHeaderType"/>
```

While the XSD follows generated order, Table 9 is organized by function. The Party column specifies if the element is relevant for either a sender (S), or a receiver (R), or both (SR). Column 4 (Notes) provides more detailed explanations that can be found in the table footer.

Table 9 — Functional overview of the EPC header structure

Attribute	Type	Party	Note
Sender information			
SenderId	epc:ContactInfoType	S	
ShipId	epc:ShipIdType	S	
Authenticator	epc:AuthenticatorType	S	
RequestReplyMethod	epc:ReplyInformationType	S	
AuthorizationToken	epc:AuthorizationTokenType	S	
Receiver information			
SystemId	epc:SystemIdType	R	
ReceiverId	epc:ContactInfoType	R	
Message information			
SentTime	epc:dateTime	SR	
MessageCreatedTime	epc:dateTime	S	
MessageReference	epc:ReferenceCodeType	SR	
MessageFunctionCode	epc:MessageFunctionCodeContentType	SR	
Final	epc:boolean	SR	
MessageBodyFormat	epc:MessageBodyFormatContentType	SR	
SpecialAttachments	epc:SpecialAttachmentType	SR	a
MessageManifest	epc:MessageManifestType	SR	
Version	epc:VersionType	SR	
Service request information			
ArrivalDeparture	epc:ArrivalDepartureType	S	
ServiceTypeCode	epc:ServiceTypeCodeContentType	S	
ServiceCode	epc:ServiceCodeContentType	S	b
ServiceName	epc:string	S	b
ServiceProviderName	epc:ContactInfoType	S	
ServiceBookingNumber	epc:ReferenceCodeType	S	c
ShipStayReference	epc:ReferenceCodeType	SR	d
RequestValidityEnd	epc:dateTime	SR	e
ReportingSystem	epc:ReportingSystemType	S	
RelayReportingSystem	epc:ReportingSystemType	S	a
Message and request status information			
MessageStatus	epc:StatusType	SR	
RequestStatus	epc:StatusType	R	

Key

S sender

R receiver

SR sender and receiver

[a] This element can be repeated zero or more times.

[b] These two attributes are mutually exclusive, only one shall be used.

[c] This attribute shall only be used by the sender and only when it has been defined by the receiver. The receiver returns this code in the `RequestStatus` fields as a service request status.

[d] This attribute is returned by the receiver to show the session context that is valid for the service request status, if any is returned. The sender shall use this field as defined in 7.7.3.

[e] This attribute can be set by sender and returned by receiver. Its use is defined in 7.10.4.

10.10.3 Representation

10.10.3.1 General

This data structure contains the information required to process an incoming message and to determine the service provider, without looking into the message body. The attributes are described in 10.10.3.2 to 10.10.3.6, where each attribute is listed in the same order as Table 9. The subclauses have the same title as the lightly shaded sub-header rows in Table 9.

10.10.3.2 Sender information

The following attributes are related to the identification of the sender, sender capabilities and parties associated with the sender.

— `SenderId`: This is the identification of the sender when the sender is an organization or a person. When the sender is an automated system on a ship, the ShipId attribute shall be used instead of, or in addition to, the SenderId.

— `ShipId`: This is the identifier of the sender when the sender is an automated system on the ship. When agents or other persons or organizations are responsible for the message, SenderId shall be used instead.

— `Authenticator`: This is the identity of the person attesting to the validity of the transmitted information, if any. The element also contains information about the associated organization, the location where authentication was provided and the role of the authenticator.

— `RequestReplyMethod`: This is the instruction for the receiver on how to respond to the sender's request.

— `AuthorizationToken`: This is the security code used to authorize access to API (see Clause 18), if required.

10.10.3.3 Receiver information

The following attributes are related to the identification of the receiver and software systems associated with the receiver.

— `SystemId`: The identification of an automated message reception system on shore. If the receiver is operated by a specific person or an organization, `ReceiverId` can also be used.

— `ReceiverId`: This is the identification of the receiver when the receiver is a specific organization or a person.

10.10.3.4 Message information

The following attributes are related to the outgoing message. The values are set by the client.

— `SentTime`: This is the date and time when this message was sent from the client.

— `MessageCreatedTime`: The time and date at which the message body content was assembled.

— `MessageReference`: A client defined code that can be used by the server to refer to this specific message.

— `MessageFunctionCode`: The general function of the message, e.g. if it is a request, an acknowledgement of receipt or a status message.

— `Final`: True if this message signals the end of a service session.

— `MessageBodyFormat`: How the message body is formatted, e.g. in XML or UN/EDIFACT.

— `SpecialAttachments`: This is a list of attachment message parts that are not referenced in the message body.

  These special attachments can be encrypted parts of the message body, see 11.3 or alternatively coded parts of the message body (see 10.8). Attachments that are referenced in the message body, e.g. pictures of stowaways or dangerous material product sheets shall not be listed here (see 12.2).

— `MessageManifest`: A summary of message parts in this transmission. The data element counts the number of message parts of each type.

— `Version`: This is the version code the sender used when formatting the message. See 4.4.2 and 11.2.1 for information on how the version code shall be used. The receiver shall use the same version code as the sender and format messages in accordance with the restrictions that the sender's version code imply.

10.10.3.5 Service request information

The following attributes are related to the service that has been requested by the sender:

— `ArrivalDeparture`: This element specifies if the message refers to an arrival, departure or another type of event.

Some service codes refer to services that are directly associated with arrival or departure. In these cases, this flag is redundant and should be omitted. If it is not omitted, the service code shall take precedence over this flag.

EXAMPLE Examples of such service codes are the MSW services conveyance arrival or departure, and cargo declaration on arrival or departure.

— `ServiceTypeCode`: This is a code identifying the service type to which the message refers. Table G.1 defines the standard code values. Each of the legal codes will refer to a detailed service code in Tables G.2 to G.5.

— `ServiceCode`: This is a code identifying the service to which the message refers. Tables G.2 to G.5 define the standard code values. The service type code determines which table is relevant to use.

— `ServiceName`: This is the name of the service that can be used if there is no code defined to refer to the service.

— `ServiceProviderName`: This is the name and identity of the organization that provides the service which is referred to.

— `ServiceBookingNumber`: This is the reference code for a service. It is defined by the receiver and is used to refer to the specific service instance once it has been confirmed ordered by the receiver.

— `ShipStayReference`: This data element, if included, contains a reference code for the session context, e.g. the port call or another physical event (see 7.7.3 and 8.1.2).

NOTE This can be assigned by the maritime single window when related to the public authorities' clearance of the vessel or by the harbour master when related to the port call itself.

— `RequestValidityEnd`: This is the date and time when the validity of this request expires. This can be used by a sender to limit the time window in which a receiver shall respond to a request. It can also be used by a receiver to notify the sender of the server's corresponding timeout if different from that of the sender (see 8.1.4).

— `ReportingSystem`: Name of reporting system to which the message should be sent, if appropriate. This is used for all messages of the MRS type. Annex C provides details for the use of this attribute.

— `RelayReportingSystem`: Name of reporting system the message is sent to, when the message is to be relayed to `ReportingSystem`. This can be used for messages of the MRS type. Annex C provides details for the use of this attribute.

10.10.3.6 Message and request status information

These attributes are related to the status of received messages and service requests. It is set by the server. See 8.1.5 for details.

— `MessageStatus`: A server's status on the processing of a client's message.

— `RequestStatus`: A receiver's status on the processing of a sender's service request.

11 Definitions related to the message body part

11.1 General

This clause describes how the body part of the EPC message shall be constructed.

— The format of the XML data package is described in 11.2. See 11.3 for a description of how parts of the body can be encrypted.

— Some rules for inclusion of UN/EDIFACT data packages are described in 11.4 and 11.5.

— Some principles for generating JSON data packages are described in 11.6.

Alternative formatting of the message body is specified in the `MessageBodyFormat` attribute in the message header (10.4). Encryption of parts of the message body is specified in the `SpecialAttachment` attribute in the message header (10.8).

If the receiving application do not accept alternative formatting of the message body, an error message shall be returned. The error code Rejected in the service status field shall be used (see 8.1.5).

The header part of the message will always be XML as described in Clause 10. The status and error codes will be in the XML header for both JSON and XML data packages. UN/EDIFACT may use its own formats for the status and error code part of the header (see 11.5).

The message body as defined in 11.2 shall be inserted as an XML text file and as the second message part in the HTTP multi-part message. The corresponding content type shall be `application/xml`. See Clause 9 for details.

11.2 XML message body

11.2.1 epc:EPCMessageBodyType – the XML body data type

11.2.1.1 Definition

This data type defines the structure of the XML body part of the EPC message. The body will be composed of data elements defined in the ISO 28005 series. The data elements that are required to be part of the message body are inserted into the data package.

11.2.1.2 Type

```
<xs:complexType name="DataPackageType">
  <xs:sequence>
      ...  list of elements
  </xs:sequence>
</xs:sequence>

<xs:complexType name="EPCMessageBodyType">
  <xs:sequence>
    <xs:element name="DataPackage" type="epc:DataPackageType">
    <xs:any minOccurs="0" maxOccurs="unbounded"/>
  <xs:sequence/>
</xs:complexType>

<xs:element name="EPCMessageBody" type="EPCMessageBodyType" />
```

11.2.1.3 Representation

The data elements in DataPackage are assembled from the list of data elements defined in Annex A of each part of the ISO 28005 series, as defined in 4.4.2.

The following rules apply:

1. Annex A of each part of the ISO 28005 series specifies the file number in which its data elements are designed to be included.

2. Elements are taken from Annex A of each relevant part of the ISO 28005 series and inserted in alphabetical order in the data package of the XSD file. Relevant parts of the ISO 28005 series are those parts that have a specified file number lower than or equal to the XSD file number.

   NOTE This means that new data elements are inserted in between older data elements in higher numbered XSD files. This can be handled by validation and parsers as discussed in 4.4.2.

3. The data elements in ISO 28005-2:2021, Table A.1 shall be assigned file number 1.

4. All elements shall be defined with a minimum cardinality of zero (“0”) by adding the attribute minOccurs=”0” to the XSD definition. This makes all elements optional, and a valid XML message can be constructed by selecting only the elements that are needed for the message's specific purpose.

5. Elements shall have a maximum cardinality of *n*, which is defined in Annex A of all parts of the ISO 28005 series. This shall be added in the XSD file as the attribute maxOccurs=”*n*”. If the maximum cardinality is one (“1”) the maxOccurs=”1” statement can be omitted.

Additional data elements or packages can be added before or instead of the “any” tag to support local variants of message body formats as discussed in 4.4.2.

11.2.2 Structure of message body definition table

It is expected that each part of the ISO 28005 series contains an annex with a table consisting of the following columns:

NOTE In this document, the table is Table A.1. It can also be found in ISO 28005-2:2021, Table A.1.

1. Core element: This is the name of the data object as it occurs in the message body.

2. Type: This is the type of the data object.

   EXAMPLE In this document, Clauses 5, 6 and 10 contain data type definitions.

3. Card.: This is the cardinality of the attribute in the form “0..*n*” where 0 means that the element is optional and *n* defines the maximum number of occurrences. This can be "*" if there is no bound on the number. The first number is always zero (see list item 4 in 11.2.1).

4. Description: This is a free text brief description of the data object.

In addition, this annex is expected to specify the file number with which this relevant part of the ISO 28005 series is associated.

11.3 Encryption of selected content

As messages specified in this document can be sent through intermediate systems, such as maritime single windows or port community systems, that are not the final receiver, it can be necessary to encrypt some of the information so that it is cannot be understood by the intermediate party.

NOTE 1 In Europe, the General Data Protection Regulation (GDPR)[24] can require sensitive information about persons, such as passenger lists, crew lists, crew effects, and health information to be encrypted.

In such cases, the sensitive information is formatted as a separate message body, using the conventional `epc:MessageBodyType` XSD format (see 11.2). This part is then encrypted and added to the message as an attachment. The name of the attachment is added to the message header as described in 10.8.

The encryption key can be the final receiver's public key (asymmetric encryption). The final receiver may be specified in the response from an authorization request (see 11.3). Other mechanisms not specified in this document can also be used. In the latter case, the MIG shall specify what mechanism to use.

NOTE 2 Symmetric encryption can be used as it is more efficient than asymmetric encryption, but it requires access to a shared encryption and decryption key.

11.4 UN/EDIFACT message body

This document allows an UN/EDIFACT message to be used as the message body. In this case, the message body content is one UN/EDIFACT message with a format as specified in relevant UN/EDIFACT standards.

The corresponding HTTP content-type directive shall be "application/EDIFACT" (see 9.5).

11.5 UN/EDIFACT status message

This document allows an UN/EDIFACT error or acknowledgement message to be included as a status message instead of the normal status fields in the message header (see 10.2). The UN/EDIFACT error or acknowledgement message shall be inserted in the message body as described in 11.4.

NOTE The most relevant status message is the application error and acknowledgement message APERAK.

11.6 JSON message body

This document allows a JSON to be used as an XML message body. This document does not specify any requirements to the JSON data object, and processing of the information will be application dependent.

The corresponding HTTP content-type directive shall be "application/json" (see 9.5).

12 Definitions related to attachment message parts

12.1 General

Any file can be included as an attachment in the HTTP message. If it is a large compressible file, it should be transmitted in compressed form, if the compressed application type can be specified by the content type (see 9.6). The attachment shall be referenced from the message body as defined in 12.2 or through the `SpecialAttachment` data item in the header as defined in 10.8.

The appropriate HTTP directives shall be used (see Clause 9).

12.2 Reference to an attached document in an XML body

The data type "`epcAttachmentType`" is defined in 6.15. It can be used in other parts of ISO 28005 series to refer to a specific attachment, e.g. an image or a printable data sheet.

The rules for referring to an attachment are:

1. The attachment message part shall have the HTTP Content-Disposition directive defined and shall have a file name (see 9.7).

   EXAMPLE 1 "`Content-Disposition: form-data; name=attach`*N*`; filename=data.pdf;`"

2. The relevant attachment tag in the message body or the special attachment data item shall refer to this filename through a "`file:`" URI.

   EXAMPLE 2 The value for the URI for example 1 would be "`file:data.pdf`".

3. The message part shall have a content-type field that is recognized by the receiver as specified in the MIG (see 9.6) and, if relevant, as listed in Table D.1.

The file name has no other inherent meaning in this document and can be any legal HTTP directive string.

13 Definitions related to X.509 certificate message parts

Each of these parts contain one X-509 public key certificate in binary format, encoded according to Distinguished Encoding Rules (DER) as defined in ITU Recommendation X.690.

These certificates can be included in the message in case there is no agreed PKI in use or when the signing party is not listed in the PKI.

The certificates are inserted in the multi-part message after the message body. No particular ordering of the certificate parts is defined. The content type shall be `application/pkix-cert`. See Clause 9 for details.

The public key certificate shall specify the identity of the subject so it is compatible with the identity information defined in 6.18.

The certificate itself should be signed by a party that is trusted by the server.

14 Definitions related to the digital signature message part

14.1 General

The digital signature is an XML structure containing a reference to the message part and the identity of the signer, as well as the signature itself. In this document, the following rules apply to the use of digital signatures:

1. As long as at least one message part is signed by the originator of the message, the signature can be used to verify the authenticity of the whole message.
2. Signing one part of the message guarantees the integrity of that part only. Integrity guarantees are required to enforce non-repudiation of message exchanges. All message parts that can be the object of non-repudiation conflicts should be signed.

   NOTE 1 The use of HTTPS will protect the message parts from modification by third parties, but HTTPS cannot be used for non-repudiation purposes. For this, a digital signature is expected to be used.

3. Encrypted body parts (see 10.8) or attachments that contain digital signatures themselves should not be additionally signed in the signature part of the message.
4. It is permitted to use different certificates to sign different parts of the message.

   NOTE 2 This can be useful in cases, e.g. where the port call data are provided by the ship's agent; the crew and crew effects lists are provided by the ship, and the cargo information is provided by the charterer.

The digital signature as defined in 14.3 shall be inserted as an XML text file and after any certificate parts in the HTTP multi-part message. The corresponding content type shall be `application/xml`. See Clause 9 for details.

14.2 Signers

Signed message parts should normally be signed by the sender of the message as identified in the message header. In some cases, the authenticator as identified in the message header may sign one or more message parts.

It is also allowed that other parties are used as signatories. However, in such cases the server is responsible for verifying that the signer is a trusted party.

The identity of a signer (see 6.18) shall match that of the identity in the corresponding digital certificate.

14.3 epc:EPCMessageSignatureType – Digital signatures of message parts

14.3.1 Definition

This data element contains the digital signatures for selected message parts. It contains a list of signatures for the signed parts of the message.

NOTE If there are no signed parts, the whole signature message part can be omitted.

14.3.2 Type

```
<xs:complexType name="MessagePartSignatureType">
  <xs:sequence>
     <xs:element name="DigestMethod" type="epc:int" minOccurs="0"/>
     <xs:element name="DigestValue" type="epc:string"
                 minOccurs="0"/>
     <xs:element name="File" type="epc:string"/>
                 minOccurs="0"/>
     <xs:element name="Signer"
                 type="epc:SignatureCertificateIdType"/>
                 minOccurs="0"/>
     <xs:element name="Signature" type="epc:string"/>
                 minOccurs="0"/>
     <xs:element name="SignatureMethod" type="epc:int"
                 minOccurs="0"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="EPCMessageSignatureType">
  <xs:sequence>
     <xs:element name="PartSignature"
                 type="epc:MessagePartSignatureType"
                 minOccurs="0" maxOccours="unbounded"/>
  </xs:sequence>
</xs:complexType>

<xs:element name="EPCMessageSignature" type="EPCMessageSignatureType"/>
```

14.3.3 Representation

The elements in each message part signature are used as follows:

— `DigestMethod`: The method used to calculate the message part digest. Table J.1 defines the code values that shall be used.

— `DigestValue`: The digest value encoded as base64 string over the message part. The digest is calculated over the raw file as transmitted, including any compression.

— `File`: The file name of the message part as described in 9.7.

— `Signer`: The identity of the signer as defined in 6.18.

— `Signature`: The actual signature as a base64 encoded text string.

— `SignatureMethod`: The method used to calculate the signature. Table J.2 defines the code values that shall be used.

15 General definitions related to the use of HTTP

15.1 Conceptual structure of a receiver

Figure 21 shows a conceptual structure of a receiver. This is an example of a possible implementation pattern, but other principles for the design of receivers are also possible.

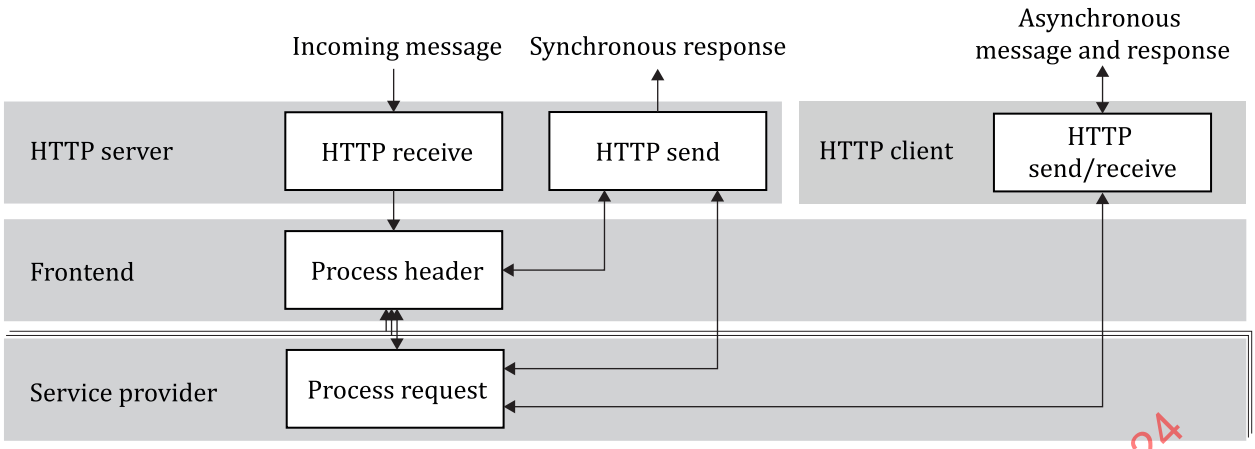


Figure 21 — Conceptual structure of a receiver

The receiver will normally use a top layer which may be implemented by an off-the-shelf HTTP server and client. The HTTP server handles incoming messages and synchronous responses to these messages. The HTTP client is used to send outgoing messages and receive synchronous responses to these.

A receiver can implement a frontend component or other equivalent function that reads the message header and determines what service provider to forward requests to. The frontend should keep track of all request status changes so that simple status requests can be answered to synchronously (see 8.2.2). The frontend can check overall message syntax and the content of the message header. Any errors in the message syntax or header can be returned as a message error and further processing of the message will not be done.

The receiver will be connected to one or more service providers that process the full incoming message and control the execution of services, including request status changes as defined in 7.10.2. The service provider should check the message content and can return an error or a missing data code to the sender (see 10.7). In that case, no further processing of the message will take place. If no errors are found, the processing of the request continues, and request status may change as defined in 7.10.2. Such changes can be returned as a synchronous response if the change happens within the same HTTP session. Otherwise, the service provider can send asynchronous responses to senders that accept asynchronous status reception (Clause 16).

The service provider should immediately notify the receiver frontend when a request status changes. This notification is independent of the use of asynchronous responses and is useful to make it possible for the frontend to immediately answer synchronous service status requests.

15.2 Conceptual structure of a sender

Figure 22 shows a conceptual structure of a sender. This is an example of a possible implementation pattern, but other principles for the design of receivers are also possible.

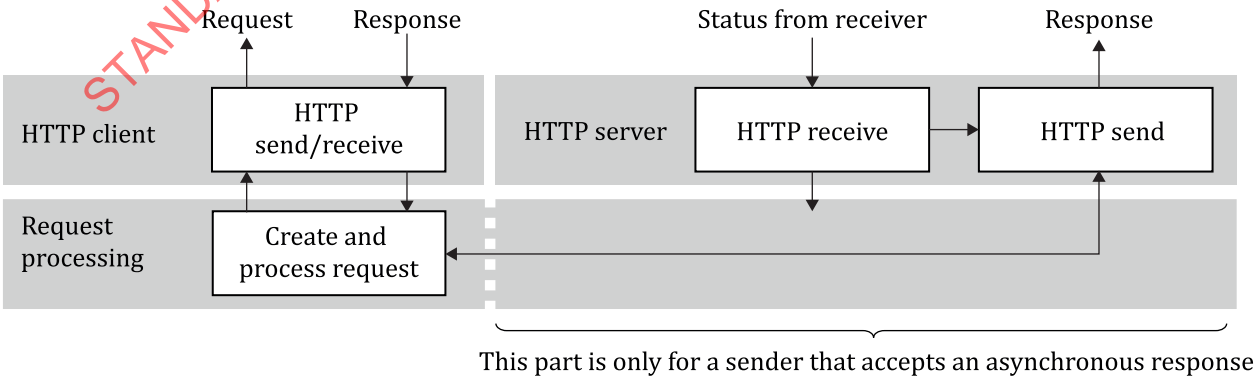


Figure 22 — Conceptual structure of a sender

The sender will normally use a top layer which may be implemented by an off-the-shelf HTTP client and server. An HTTP client implementation is always required and is used to send outgoing messages and receive synchronous responses from the receiver.

The HTTP server implementation is required if the sender accepts asynchronous status reception (see Clause 16). The HTTP server on the sender side shall respond to the synchronous message status whenever a valid message arrives from the receiver. Invalid status messages can be reported back to the receiver, but this is not required.

15.3 Transmission protocol

The transmission protocol is HTTP/1.1 as defined in RFC 7231.[5] All communication shall be encrypted, using Transport Layer Security (TLS) version 1.2 as defined in RFC 5246 or version 1.3 as defined in RFC 8446. RFC 2818[6] defines HTTP over TLS.

The use of TLS requires the use of a digital signature certificate on the HTTP server. The server should avoid using the same certificate as it uses for signing message parts (see 7.14) as its TLS certificate.

NOTE The TLS signature can be exposed to external and hostile parties each time a party makes a HTTP request to the server's URL. As this can aid hackers in determining the private key, this generally requires a more frequent update of the TLS certificate than would otherwise be necessary for the certificate described in 7.14, where the signature is much less frequently exposed.

15.4 Avoid use of HTTP redirect and similar mechanisms

As ship to shore communication via satellite can be expensive and can have limited bandwidth, the implementors of HTTP servers should avoid using mechanisms that force the ship to resend an HTTP request to another URL. This mainly relates to the use of HTTP redirects.

NOTE The access authorization mechanism specified in this document does not use the conventional HTTP authorization mechanisms, which avoids the ship having to resend a request after an "unauthorized" response code from the server.

15.5 Optional use of HTTP keep-alive

The sender may set the keep-alive flag in the HTTP header to signal to the receiver that the TCP connection should be kept open so that the sender can send more than one HTTP request.

The receiver can choose whether to accept the keep-alive flag or not.

NOTE This can be used to save bandwidth, e.g. when several services are ordered by the ship through a PCS. The ship will then avoid establishing a new TCP/IP connection for each service request.

15.6 API access point URL

The receiver's API access point is a URL defined by the receiver. This is also the case for the sender's API access point if the sender accepts asynchronous notifications. This document does not specify any requirements to the format of the URL.

This document does not define how a sender gets the receiver's URL, but it is required that the URL or URLs be published in a manner such that it is convenient for the sender to find it. This information may be published, for example, in a port procedures manual or similar, describing the services available in a port (see 7.19).

15.7 HTTP methods

This document specifies the use of the POST method for all information exchanges.

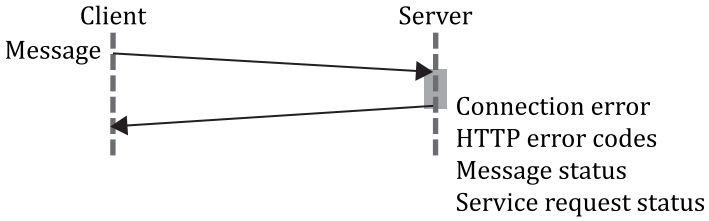
NOTE While it can be desirable to specify REST compliant behaviour and, thus, differentiate between POST, PUT and GET, it is in general not possible to guarantee formal REST behaviour from the different types of equipment that can be connected via the interfaces specified in this document. Thus, POST is specified for all.

Servers should accept both PUT and GET to cater for non-compliant clients. If accepted, the server's behaviour shall not differ between the use of PUT, GET and POST methods with otherwise equivalent message content.

15.8 Different types of synchronous return values

15.8.1 General

This document uses HTTP to exchange messages between a client and a server. This is illustrated in Figure 23, where four different types of returns from the server are shown. The return code on a given line except the first will only be delivered if the previous line return code signalled a valid transaction.



Key

See Figure 2 for key references.

Figure 23 — Message status variants

The four different types of return codes are explained in 15.8.2 to 15.8.5.

15.8.2 Connection error

These are errors that occur at the transport protocol level, e.g. that the connection is denied or closed during a session. In all cases, the client should try to determine the cause of the error and find appropriate corrective actions. This can require assistance from a human operator.

Connection denial can, for example, be caused by an error in the URL, that the TLS certificate that the server uses is invalid, or that the server is momentarily unavailable. The client should correct any client-side errors, and if applicable retry the transaction at a later time.

If the connection is closed during a session, this can mean that the status of a service request remains unknown to the client. The client can use a status poll to check the status of the request (see 8.2.2). Depending on the cause for the connection closing, it can be necessary to complete the status poll after a certain period of time, e.g. if the server crashed during the session.

15.8.3 HTTP error codes

This document only uses HTTP error codes to transfer HTTP protocol related information. All HTTP return codes will indicate the result of the HTTP transaction on the HTTP server level (see 15.1).

Code 200 (OK) or 201 (Created) will be the return code for any successful request.

Other codes will indicate some HTTP anomaly or error that can require human operator assistance to analyse and correct.

15.8.4 Message status

All returns from a server will contain a message status, explaining how the message was processed (see 8.1.5). These codes are mainly the result of processing at the frontend level (see 15.1).

If the message was rejected, no service request status will be returned, and the message will have no effect on the service status.

All errors will be returned as text strings that are intended for human consumption. In addition, missing data fields will be listed in the Missing attributes to the status object and can be automatically corrected by the sender. Likewise, mismatch in version codes will be returned from the receiver in the version code in the message header and can be automatically corrected by the sender. See also 7.2.

15.8.5 Service request status

The service request status will be returned from a receiver as a response to a request or when the status of a service changes. The service request status will often be returned together with the message status. If it is a response to a service request, it can contain error codes or a list of missing data (see 8.1.5).

Errors other than missing data fields will be returned as text strings for human consumption. Missing data fields will be listed in the Missing attributes to the status object.

16 API access points for asynchronous HTTP communication

16.1 General

The asynchronous API access points means that both the sender and the receiver have a URL for delivery of messages. This is the case shown in Figure 2. The sender transfers a return URL to the receiver that the receiver uses to return asynchronous notifications to the sender.

16.2 Message patterns to use

These API access points shall use the exact message patterns defined in Clause 8 for the patterns that are supported by the access point. The polling pattern (see 8.2.2) is required for the receiver. Other patterns shall be implemented by the sender and receiver as described in the MIGs that the access points support.

16.3 No authorization on the sender's URL

The sender should not implement authorization on its return access point.

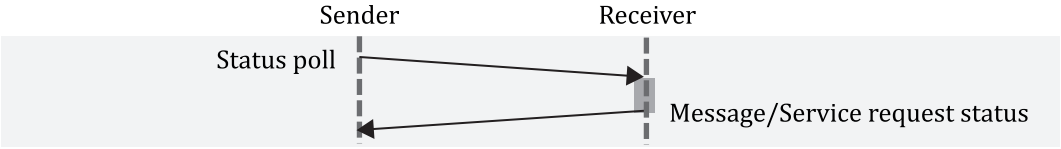
17 API access point for synchronous HTTP communication

17.1 General

The polling or synchronous API access point means that the sender polls the receiver for status or data. In this case, the sender has no URL that the receiver can access. The sender is always a client and the receiver is always a server.

17.2 Message patterns to use

This access point modifies the message patterns defined in Clause 8. Each message transmitted with the receiver as the client shall be replaced by the pattern shown in Figure 24.



Key

See Figure 2 for key references.

Figure 24 — Sender polling pattern

The polling pattern (see 8.2.2) is required for the receiver. Other patterns shall be implemented by the sender and receiver as described in the MIGs that the access point supports.

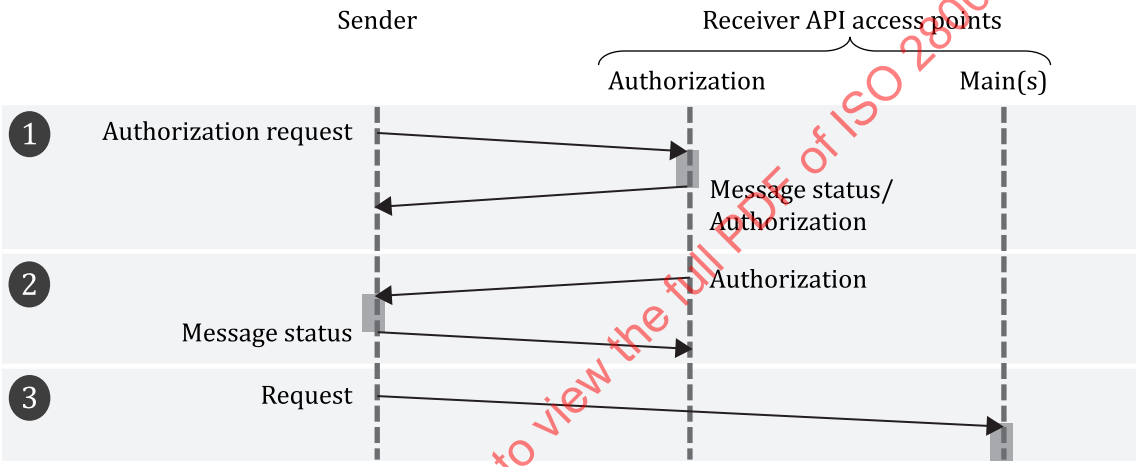
18 Authorization to access API access point

18.1 General

Some receivers can require authorization before access to the main API access point is accepted. The authorization will normally be done through an open access point, different from the service access point. The principle is described in 7.16. The authorization key is inserted in the `AuthorizationToken` field in the header when the open access point is used.

18.2 The message pattern

The message pattern for the authorization case is functionally the same as pattern 1 described in 8.2.1 with some modifications as illustrated in Figure 25. Annex B contains the MIG for the authorization function.



Key

See Figure 2 for key references.

X sequences 1 to 3, which are defined by the corresponding lightly shaded areas of the message exchange

Figure 25 — Authorization message pattern

The main differences from pattern 1 described in 8.2.1 are described below.

1. In sequence 1 and 2 in Figure 25, the authorization requests will normally be done on an API access point different from the main API(s). This API is here called Authorization. The actual service transactions, here described with the initial request in sequence 3, will normally be executed on another access points. This API is here called Main(s). The addresses of one or more main API access points are returned in the authorization message.

2. In sequence 1 in Figure 25, the authorization request is functionally the same as a service request, but with a different message function code as defined in Table E.1. If the authorization can be given in the same HTTP session as the authorization request, the response will be both a message status and an authorization message and sequence 2 is omitted.

   Implementations of the authorization function are not required to accept service request updates and can require that all necessary information for access authorization is included in the initial authorization request. If updates are accepted, this shall be implemented as sequence 2 in pattern 1 as defined in 8.2.1 and use the normal update request function code.

3. In sequence 2 in Figure 25, a positive access authorization is signalled with an Authorization message. This is functionally equivalent to an Accept message but uses a different function code as defined in Table E.1.

18.3 epc:ServiceAuthorizationType – Type of service authorization

18.3.1 Definition

This element provides the authorization code and the access point URL for a list of services that are available from a receiver.

18.3.2 Type

```
<xs:complexType name="ServiceAuthorizationType">
  <xs:sequence>
     <xs:element name="AuthorizationToken"
                 type=”epc:AuthorizationTokenType” minOccurs = "0"/ >
     <xs:element name="AuthorizationValidityEnd" type="epc:dateTime"
                 minOccurs="0"/>
     <xs:element name="EncryptionId"
                 type="epc:SignatureCertificateIdType"
                 minOccurs="0"/>
     <xs:element name="ServiceCode" type="epc:ServiceCodeContentType"
                 minOccurs="0"/>
     <xs:element name="ServiceName" type="epc:string” minOccurs = "0"/ >
     <xs:element name="ServiceTypeCode"
                 type="epc:ServiceTypeCodeContentType”
                 minOccurs="0"/>
     <xs:element name="ServiceURL" type="epc:anyURI" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="ServiceAuthorizationListType">
  <xs:sequence>
    <xs:element name="ServiceAuthorization"
                type="epc:ServiceAuthorizationType”
                maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
```

18.3.3 Representation

The authorization is returned as a list of entries. The elements in each entry are used as follows:

— `AuthorizationToken`: The authorization token to be used for specified service.

— `AuthorizationValidityEnd`: The date and time the token expires, if defined.

— `EncryptionId`: The identity of the receiver of the message (part), when the message (part) shall be encrypted before transmission. This can be used to retrieve the public key that can be used for asymmetric encryption of content.

NOTE The message can contain the corresponding X.509 certificate if it is not already available through a PKI.

— `ServiceCode`: The service code as defined in 10.5.

— `ServiceName`: The service name as defined in 10.5.

— `ServiceTypeCode`: The service type code as defined in 10.5.

— `ServiceURL`: The URL for the API access point where the service can be ordered.

The service can be specified by code or name.

If this API is used for service discovery only, and no authorization is required, the token and its expiry date and time should be omitted.

18.4 The message body

One service authorization list can be sent in a normal EPC body message as described in 11.2. The service authorization list is defined as a message body data object in Annex A.

19 Specifications for the message implementation guide (MIG)

19.1 General structure of MIG

MIGs can be structured as is most convenient for describing the intended functions. However, in this document, the MIGs follow a structure composed of four sections, which are defined in 19.2 to 19.5.

Annex B contains a MIG for the access authorization mechanisms (see Clause 18) and Annex C contains a MIG for MSW and MRS. These annexes can also be used as examples of the structure of a MIG.

19.2 MIG Introduction

The introduction gives an overview of the function defined by the MIG including any normative reference for the documents that are the basis for the description of these functions.

19.3 High level description of use case

This includes additional details related to functionality which are not already covered in the introduction. It also defines the parties involved and the physical system architecture.

19.4 Prerequisites

This section defines all relevant requirements to parties using the MIG that are not directly related to the message sequence diagrams.

19.5 Message sequence diagrams

This section describes the message patterns that are used, if necessary, with additional message sequence diagrams. When a message pattern is referenced, all requirements related to that pattern as defined in 8.2.1 (pattern 1) to 8.2.5 (pattern 5) apply. Special or deviant requirements can be specified in the MIG.

Subclauses in this section of the MIG shall be made for all message types specified by the MIG, where similar message types can be grouped and discussed together. This most commonly results in two message descriptions: one for the requests from the sender, including cancel and status poll; and one for the replies from the receiver.

The message headers are normally described in a table, as exemplified in Table 10.

Table 10 — Example header for MIG

Attribute [a]	Value [b]	Card. [c]
Sender information		
SenderId	Identity of sender	0..1
ShipId	Ship identity	1..1
Message information		
SentTime	Date and time of transmission	1..1
MessageReference	Message reference	1..1
MessageFunctionCode	Authorization request, update request, cancel or status poll	1..1
MessageBodyFormat	As defined by AGR (XML is default)	1..1
SpecialAttachments	If required by AGR – should not be used	0..1
MessageManifest	Structure of message parts	1..1
Version	"2.*n*" [d]	1..1
Service request information		
ServiceBookingNumber	Reference to the service session if not first message	0..1
RequestValidityEnd	Optional timeout	0..1

[a] Attribute is the name of the data object as defined in 10.10.

[b] Value gives additional information about the object.

[c] Cardinality (Card.) specifies the number of occurrences of this object: "0..1" for zero or one; "1:1" for exactly one; "1..*" for one or more; and "0..*" for an arbitrary number, including none.

[d] Replace *n* with the relevant file code (see 6.19).

The table lists all elements in the header that the sender and receiver are required to process. Other elements may be included by the sender or receiver. In that case, these elements should have the same function as defined in Clause 10, but neither party is required to process them. They can be silently ignored.

If the function of a data object differs from what is specified in Clause 10 an additional definition shall be provided in this section of the MIG.

The parts of the message other than the header are briefly described in a separate sub-section. The message body content, in particular, is normally defined by requirements in external references, e.g. the IMO Compendium.[14]

Annex A
(normative)

EPC request body

Table A.1 contains the data elements defined in this document that shall be available in the consolidated message body definitions in the XSD file. The elements in Table A.1 shall be included in file number 1 as discussed in 4.4.2.

The corresponding mandatory part of the version code in the message header shall be “2.1” (see 4.4.2).

Table A.1 — EPC Request Body

Core element	Type	Card.	Description
ServiceAuthorizationList	epc:ServiceAuthorizationListType	0..1	List of services and corresponding authorization tokens.
The format of this table is defined in 11.2.2			

Annex B
(informative)

Message implementation guide for access authorization

B.1 Introduction

This message implementation guide (MIG) specifies how to implement an API to authorize access to a set of session context message exchanges. It is based on FAL.5/Circ.46.[1] The access authorization mechanism is defined in Clause 18.

B.2 High level description of use case

B.2.1 General function

This use case describes a service where a declarant (sender) sends certain information to an authorization granting server (AGR) via the sender's declarant reporting sender (DRS) in order to be granted access to one or more other APIs that can be used in the specified session context.

The use case describes a fully automated machine-to-machine exchange of information. Thus, the parties used in the descriptions are the respective software systems.

B.2.2 Parties involved

B.2.2.1 DRS – Declarant reporting sender

The declarant reporting sender (DRS) is the software system used by the declarant to send information and request access authorization.

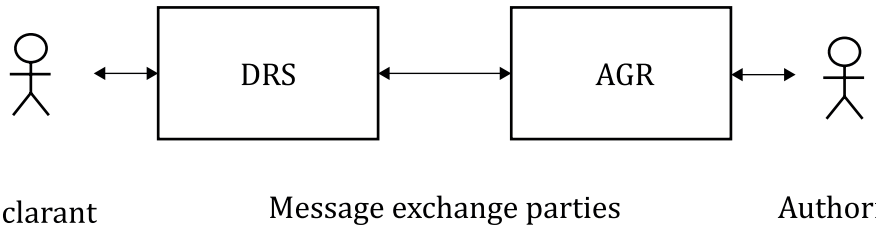
NOTE This can be a computer system on the ship, in the ship agent's office, or in the ship manager's or charterer's office.

B.2.2.2 AGR – Authorization granting receiver

The AGR refers to the software system that receives the authorization request and which can either grant access to API access points, or not.

B.2.3 General architecture

The physical architecture is shown in Figure B.1.



Key

DRS declarant reporting sender AGR authorization granting receiver

Figure B.1 — General message exchange architecture

The declarant can use the DRS to perform the necessary message exchanges with the AGR. The AGR can in turn have a backend where an authority examines submitted information and grants or denies authorization. However, in many cases both the declarant and the authorization processes may be automated so that the message exchange can be done fully automatically.

In the communication between the DRS and the AGR, the DRS is defined as the sender and the AGR as the receiver.

B.3 Prerequisites

B.3.1 Publication of AGR API and data requirements

Details of the AGR API access point shall be published so that the DSR can use it to configure the API parameters. The following information is required:

1. API access point URL that can be used to access the AGR.
2. The information requirements for granting authorization. This should be limited to the data set used in the general arrival declaration as described in the IMO Compendium.[14] If additional information is required, this shall be clearly stated in the service description.

   NOTE 1 This will normally be the identity of the ship, the sender identity, if different from the ship, estimated times of arrivals and departure, agent information if calling on a port, and information about the ship's planned operations.

   NOTE 2 The Missing data fields in the service and message status objects can be used to specify missing data that is required for authorization to be granted.
3. A list of message parts that shall be encrypted for the services where encryption may be required. If symmetric encryption keys are available, the method for acquiring these keys shall be described (see 11.3).

   NOTE 3 The list of message parts that should be encrypted will also be indicated by the list of `EncryptObjects`.
4. The extent of the session context that defines the envelope around the services that the AGR provides details of.

B.3.2 No authorization on AGR API

The AGR should not require any authorization for access to the AGR API itself.

B.3.3 Availability of digital signature certificates

If digital signatures are used, each of the parties shall have a public digital signature certificate that allows the other party to verify the authenticity of the digital signature.

The certificates should normally be available from a public key infrastructure, but can also be included in the request and status messages.

B.3.4 Maintain authorization status through the session context

The AGR is required to keep the authorization status available for a status poll from the DRS during the whole session context as defined by the AGR. This also means that the service context is the same as the session context for this service.

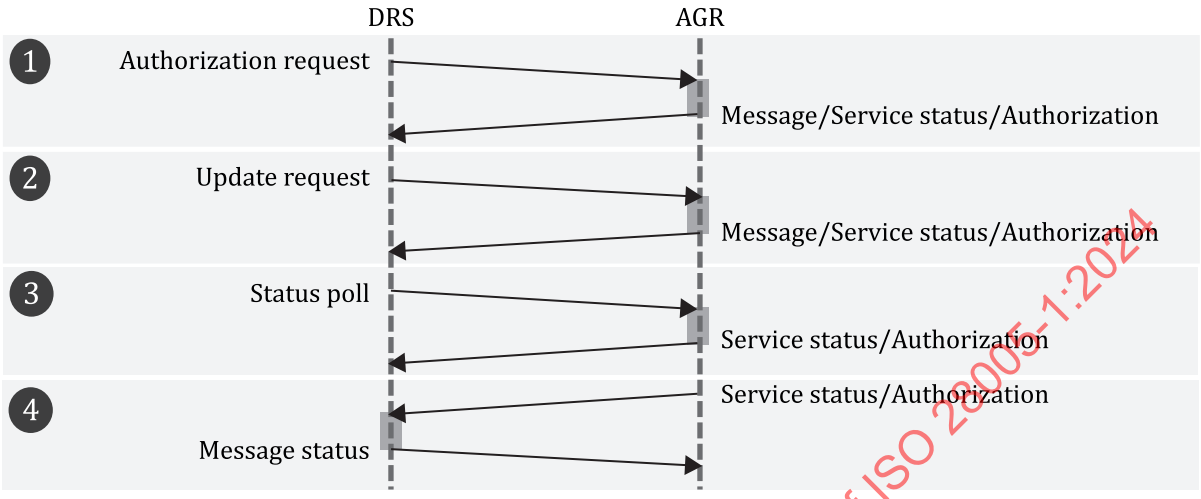
B.4 Message sequence diagrams

B.4.1 The message sequence patterns

Message pattern 1 as defined in 8.2.1 can be used, with the exception that the initial service request is replaced by an authorization request. The final service status is likewise replaced by an authorization

message. The AGR shall also support the status poll function defined in pattern 2 in 8.2.2 (sequence 3 in Figure B.2).

The AGR can alternatively implement the simpler pattern 3 described in 8.2.3. Pattern 3 will correspond to sequence 1 in Figure B.2 where the return message from the AGR is authorization or the service status "rejected". In this case, pattern 2 (sequence 3 in Figure B.2) shall also be supported.



Key

See Figure 2 for key references.

X sequences 1 to 4, which are defined by the corresponding lightly shaded areas of the message exchange

Figure B.2 — Authorization request and acknowledgement

Figure B.2 consists of four numbered sequences. Each sequence is described in the following list. If the simple pattern 1 is implemented by the AGR, only sequence 1 and 3 will be supported.

1. The initial authorization request (see B.4.2), is sent by the DRS. The message is checked by the AGR for validity and if errors are found, a message status is returned with appropriate error codes. If valid, the AGR initiates the service request and establishes the service booking number. If it is possible to immediately grant or deny the authorization, the return from the AGR consists of an authorization message with appropriate status codes. If the service cannot be immediately processed, the AGR returns a message and service status (see B.4.3).

2. If the full pattern is implemented by both parties, the DRS can add additional data to the access request, i.e. if missing data has been signalled in a status message from the AGR.

3. The DRS shall be able to poll the status of the authorization service at any time. If authorization is not granted yet, a service status is returned, otherwise the full authorization message is returned. The poll pattern can be used as long as the session context corresponding to the authorization is active.

4. If authorization was not immediately granted in sequence 1 and when the AGR has finished the authorization processes, the authorization message will be sent if approved. Alternatively, a service status with status rejected will be sent. This contains status codes informing whether access was granted or not.

B.4.2 describes the requests from the DRS in the sequences in Figure B.2. B.4.3 describes the answer from the AGR.

B.4.2 Authorization requests

B.4.2.1 Message header

This message is used to request, update or check status on an access authorization. Table B.1 lists the elements in the message header that may be used in the request.

Table B.1 — Header for authorization request

Attribute [a]	Value [a]	Card.
Sender information		
SenderId	Identity of sender	0..1
ShipId	Ship identity	1..1
Authenticator	The person attesting to information correctness	0..1
RequestReplyMethod	Type of reply required	1..1
Message information		
SentTime	Date and time of transmission	1..1
MessageReference	Message reference	1..1
MessageFunctionCode	Authorization request, update request, cancel or status poll	1..1
MessageBodyFormat	As defined by AGR (XML is default)	1..1
SpecialAttachments	If required by AGR – should not be used	0..1
MessageManifest	Structure of message parts	1..1
Version	"2.*n*" [b]	1..1
Service request information		
ArrivalDeparture	Message related to arrival or departure	0..1
ServiceBookingNumber	Reference to the service session if not first message	0..1
RequestValidityEnd	Optional timeout	0..1

For the table structure, see Table 10 and other definitions in 19.5.

[a] See 10.10 for a definition of attributes and values.

[b] Replace *n* with relevant file code (see 6.19).

ArrivalDeparture shall only be used if the authorization is explicitly related to arrival or departure. Otherwise, this flag should be omitted or empty.

B.4.2.2 The message body and other message parts

For the initial authorization request, the message body should contain the information required by the AGR to grant access. If something is missing, this can be reported by the AGR in the service status missing fields.

The status poll request shall not contain a message body.

The AGR should limit its information requirements to that of the general declaration message as described in the IMO Compendium.[14] If other information is required, this shall be clearly stated in the description of the service.

Attachments may contain an X.509 certificate for the DRS or other senders cooperating with the DRS.

The signature part is optional but should be included to ensure proper authentication of sender and integrity checks of contents. The AGR can deny access if the signature field is empty.

B.4.3 The message status or authorization reply from AGR

B.4.3.1 The message header

The message header is shown in Table B.2.

Table B.2 — AGR message status header

Attribute [a]	Value [a]	Card.
Sender information		
SenderId	Identity of sender	0..1
ShipId	Ship identity	1..1
Receiver information		
SystemId	Type of system the answer is returned from	0..1
ReceiverId	Identity of receiver	0..1
Message information		
SentTime	Date and time of transmission	1..1
MessageReference	Message reference	1..1
MessageFunctionCode	Message status, service status or authorization	1..1
Final	True only for end of session context or for rejected requests	0..1
MessageBodyFormat	As defined by DRS (XML is default), empty if rejected	0..1
MessageManifest	Structure of message parts	1..1
Version	"2.*n*" [b]	1..1
Service request information		
ArrivalDeparture	Message related to arrival or departure	0..1
ServiceBookingNumber	Reference to the service session (may be session context)	1..1
ShipStayReference	Reference to the session context, if used	0..1
RequestValidityEnd	Optional timeout	0..1
ShipStayReference	Session context if defined by AGR	0..1
Message and request status information		
MessageStatus	Status of received message	0..1
RequestStatus	Status of requested service	0..1

For the table structure, see Table 10 and other definitions in 19.5.

[a] See 10.10 for a definition of attributes and values.

[b] Replace *n* with relevant file code (see 6.19).

Message and/or request status is used depending on the type of response. Timeout may be specified if enforced by either DRS or AGR.

B.4.3.2 Message body and other message parts

The message body contains the service authorization data object (see 18.3) if access has been granted. The body is empty in all other cases.

Other message parts can include an X.509 certificate for the AGR if this is not generally available. Other message parts can also contain X.509 certificates for parties that require special encrypted attachments.

The signature part is optional but should be included to ensure proper authentication of the sender and integrity checks of contents.