The Consultative Committee for Space Data Systems

Draft Recommendation for
Space Data System Standards

MISSION OPERATIONS MESSAGE ABSTRACTION LAYER

CESG APPROVAL COPY

PROPOSED DRAFT RECOMMENDED STANDARD

CCSDS 521.0-P-2.0

PROPOSED PINK BOOK

May 2022

AUTHORITY

Issue:	Proposed Pink Book, Issue 2.0
Date:	May 2022
Location:	Not Applicable

(WHEN THIS RECOMMENDED STANDARD IS FINALIZED, IT WILL CONTAIN THE FOLLOWING STATEMENT OF AUTHORITY:)

This document has been approved for publication by the Management Council of the Consultative Committee for Space Data Systems (CCSDS) and represents the consensus technical agreement of the participating CCSDS Member Agencies. The procedure for review and authorization of CCSDS documents is detailed in *Organization and Processes for the Consultative Committee for Space Data Systems* (CCSDS A02.1-Y-4), and the record of Agency participation in the authorization of this document can be obtained from the CCSDS Secretariat at the email address below.

This document is published and maintained by:

CCSDS Secretariat
National Aeronautics and Space Administration
Washington, DC, USA
Email: secretariat@mailman.ccsds.org

STATEMENT OF INTENT

(WHEN THIS RECOMMENDED STANDARD IS FINALIZED, IT WILL CONTAIN THE FOLLOWING STATEMENT OF INTENT:)

The Consultative Committee for Space Data Systems (CCSDS) is an organization officially established by the management of its members. The Committee meets periodically to address data systems problems that are common to all participants, and to formulate sound technical solutions to these problems. Inasmuch as participation in the CCSDS is completely voluntary, the results of Committee actions are termed **Recommended Standards** and are not considered binding on any Agency.

This **Recommended Standard** is issued by, and represents the consensus of, the CCSDS members. Endorsement of this **Recommendation** is entirely voluntary. Endorsement, however, indicates the following understandings:

- Whenever a member establishes a CCSDS-related **standard**, this **standard** will be in accord with the relevant **Recommended Standard**. Establishing such a **standard** does not preclude other provisions which a member may develop.
- Whenever a member establishes a CCSDS-related **standard**, that member will provide other CCSDS members with the following information:
  - -- The **standard** itself.
  - -- The anticipated date of initial operational capability.
  - -- The anticipated duration of operational service.
- Specific service arrangements shall be made via memoranda of agreement. Neither this **Recommended Standard** nor any ensuing **standard** is a substitute for a memorandum of agreement.

No later than five years from its date of issuance, this **Recommended Standard** will be reviewed by the CCSDS to determine whether it should: (1) remain in effect without change; (2) be changed to reflect the impact of new technologies, new requirements, or new directions; or (3) be retired or canceled.

In those instances when a new version of a **Recommended Standard** is issued, existing CCSDS-related member standards and implementations are not negated or deemed to be non-CCSDS compatible. It is the responsibility of each member to determine when such standards or implementations are to be modified. Each member is, however, strongly encouraged to direct planning for its new standards and implementations towards the later version of the Recommended Standard.

FOREWORD

Through the process of normal evolution, it is expected that expansion, deletion, or modification of this document may occur. This Recommended Standard is therefore subject to CCSDS document management and change control procedures, which are defined in the *Organization and Processes for the Consultative Committee for Space Data Systems* (CCSDS A02.1-Y-4). Current versions of CCSDS documents are maintained at the CCSDS Web site:

http://www.ccsds.org/

Questions relating to the contents or status of this document should be sent to the CCSDS Secretariat at the email address indicated on page i.

At time of publication, the active Member and Observer Agencies of the CCSDS were:

Member Agencies

- Agenzia Spaziale Italiana (ASI)/Italy.
- Canadian Space Agency (CSA)/Canada.
- Centre National d'Etudes Spatiales (CNES)/France.
- China National Space Administration (CNSA)/People's Republic of China.
- Deutsches Zentrum für Luft- und Raumfahrt (DLR)/Germany.
- European Space Agency (ESA)/Europe.
- Federal Space Agency (FSA)/Russian Federation.
- Instituto Nacional de Pesquisas Espaciais (INPE)/Brazil.
- Japan Aerospace Exploration Agency (JAXA)/Japan.
- National Aeronautics and Space Administration (NASA)/USA.
- UK Space Agency/United Kingdom.

Observer Agencies

- Austrian Space Agency (ASA)/Austria.
- Belgian Science Policy Office (BELSPO)/Belgium.
- Central Research Institute of Machine Building (TsNIIMash)/Russian Federation.
- China Satellite Launch and Tracking Control General, Beijing Institute of Tracking and Telecommunications Technology (CLTC/BITTT)/China.
- Chinese Academy of Sciences (CAS)/China.
- China Academy of Space Technology (CAST)/China.
- Commonwealth Scientific and Industrial Research Organization (CSIRO)/Australia.
- Danish National Space Center (DNSC)/Denmark.
- Departamento de Ciência e Tecnologia Aeroespacial (DCTA)/Brazil.
- Electronics and Telecommunications Research Institute (ETRI)/Korea.
- European Organization for the Exploitation of Meteorological Satellites (EUMETSAT)/Europe.
- European Telecommunications Satellite Organization (EUTELSAT)/Europe.
- Geo-Informatics and Space Technology Development Agency (GISTDA)/Thailand.
- Hellenic National Space Committee (HNSC)/Greece.
- Hellenic Space Agency (HSA)/Greece.
- Indian Space Research Organization (ISRO)/India.
- Institute of Space Research (IKI)/Russian Federation.
- Korea Aerospace Research Institute (KARI)/Korea.
- Ministry of Communications (MOC)/Israel.
- Mohammed Bin Rashid Space Centre (MBRSC)/United Arab Emirates.
- National Institute of Information and Communications Technology (NICT)/Japan.
- National Oceanic and Atmospheric Administration (NOAA)/USA.
- National Space Agency of the Republic of Kazakhstan (NSARK)/Kazakhstan.
- National Space Organization (NSPO)/Chinese Taipei.
- Naval Center for Space Technology (NCST)/USA.
- Netherlands Space Office (NSO)/The Netherlands.
- Research Institute for Particle & Nuclear Physics (KFKI)/Hungary.
- Scientific and Technological Research Council of Turkey (TUBITAK)/Turkey.
- South African National Space Agency (SANSA)/Republic of South Africa.
- Space and Upper Atmosphere Research Commission (SUPARCO)/Pakistan.
- Swedish Space Corporation (SSC)/Sweden.
- Swiss Space Office (SSO)/Switzerland.
- United States Geological Survey (USGS)/USA.

PREFACE

This document is a draft CCSDS Recommended Standard. Its 'Pink Book' status indicates that the CCSDS believes the document to be technically mature and has released it for formal review by appropriate technical organizations. As such, its technical contents are not stable, and several iterations of it may occur in response to comments received during the review process.

Implementers are cautioned **not** to fabricate any final equipment in accordance with this document's technical content.

Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

DOCUMENT CONTROL

Document	Title	Date	Status
CCSDS 521.0-B-1	Mission Operations Message Abstraction Layer, Recommended Standard, Issue 1	October 2010	Original issue (superseded)
CCSDS 521.0-B-2	Mission Operations Message Abstraction Layer, Recommended Standard, Issue 2	March 2013	Current issue
CCSDS 521.0-P-2.0	Mission Operations Message Abstraction Layer, Proposed Draft Recommended Standard, Issue 2.0	May 2022	Current proposed draft: – updated as part of the broader update to Mission Operations (MO) concepts, framework, and services, based on thorough and critical review, to what is collectively labeled MO 2.0; – simplified to increase the potential for adoption.

CONTENTS

CONTENTS (continued)

Section Page

Figure

CONTENTS (continued)

Figure Page

Table

CONTENTS (continued)

Table Page

1 INTRODUCTION

1.1 GENERAL

This Recommended Standard defines the Mission Operations (MO) Message Abstraction Layer (MAL) in conformance with the service framework specified in reference [D1], Mission Operations Services Concept.

The MO MAL is a framework that provides generic service patterns to the Mission Operation services defined in reference [D1]. These Mission Operations services are defined in terms of the MAL.

1.2 PURPOSE AND SCOPE

This Recommended Standard defines, in an abstract manner, the MAL in terms of:

a) the concepts that it builds upon;

b) the basic types it provides;

c) the message headers required by the layer;

d) the relationship between, and the valid sequence of, the messages and resulting behaviours.

It does not specify:

a) individual implementations or products;

b) the implementation of entities or interfaces within real systems;

c) the methods or technologies required for communications.

1.3 DOCUMENT STRUCTURE

This Recommended Standard is organised as follows:

a) section 1 provides purpose and scope, and lists definitions, conventions, and references used throughout the Recommended Standard;

b) section 2 presents an overview of the concepts;

c) section 3 specifies the interaction patterns used to define services;

d) section 4 is a formal specification of the MAL data structures and a formal specification of the MO Object concept;

e) section 5 is a formal specification of the MAL errors;

f) section 6 is the formal service specification Extensible Markup Language (XML) schema;

g) section 7 is the formal compliance requirements of the MAL.

1.4 DEFINITION OF TERMS

Software Component (component)**:** A software unit containing the business function. Components offer their function as Services, which can either be used internally or which can be made available for use outside the component through Provided Service Interfaces. Components may also depend on services provided by other components through Consumed Service Interfaces.

Hardware Component: A complex physical entity (such as a spacecraft, a tracking system, or a control system) or an individual physical entity of a system (such as an instrument, a computer, or a piece of communications equipment). A Hardware Component may be composed from other Hardware Components. Each Hardware Component may host one or more Software Components. Each Hardware Component has one or more ports where connections to other Hardware Component are made. Any given Port on the Hardware Component may expose one or more Service Interfaces.

Service: A set of capabilities that a component provides to another component via an interface. A Service is defined in terms of the set of operations that can be invoked and performed through the Service Interface. Service specifications define the capabilities, behaviour, and external interfaces, but do not define the implementation.

Service Interface: A set of interactions provided by a component for participation with another component for some purpose, along with constraints on how they can occur. A Service Interface is an external interface of a Service where the behaviour of the Service Provider Component is exposed. Each Service will have one defined 'Provided Service Interface', and may have one or more 'Consumed Service Interface' and one 'Management Service Interface'.

Provided Service Interface: A Service Interface that exposes the Service function contained in a component for use by Service Consumers. It receives the MAL messages from a Consumed Service Interface and maps them into API calls on the Provider component.

Consumed Service Interface: The API presented to the consumer component that maps from the Service operations to one or more Service Data Units(s) contained in MAL messages that are transported to the Provided Service Interface.

Management Service Interface: A Service Interface that exposes management functions of a Service function contained in a component for use by Service Consumers.

MO Object: The concept that allows complex data models to be defined within MO services. It enables objects to be created in MO and it allows references from other Composites to be created.

Is an "MO Object" some sort of data structure, or data model, or is it just a "concept"? Perhaps the MO Object Concept is just a "concept"? From the rest of this definition it would appear to be a primitive data object, or a complex data object constructed from primitive data objects? Do you also define "Composite Data Object"?

Often the definition of "system" includes "hardware, software, persons, and procedures". In RASDS it is defined as " A set of elements (people, products [hardware and software], facilities, equipment, material, and processes [automated as well as manual procedures]) that are related and whose behavior satisfies customer and/or operational needs." In SLE docs it is defined as "A set of one or more computers, the associated software, peripherals, terminals, human operators, physical processes, information transfer means, etc., that forms an autonomous whole capable of performing information processing and/or information transfer."

Service System: The set of Hardware and Software Components used to implement a Service in a real system. Service Systems may be implemented using one or more Hardware and Software Components.

Service Provider (provider)**:** A component that offers a Service to another by means of one of its Provided Service Interfaces.

Service Consumer (consumer)**:** A component that consumes or uses a Service provided by another component. A component may be a provider of some Services and a consumer of others.

Service Data Unit, (SDU): A unit of data that is sent by a Service Interface, and is transmitted semantically unchanged, to a peer Service Interface.

Binding: The access mechanism for a Service. Bindings are used to locate and access Service Interfaces. Services use bindings to describe the access mechanisms that consumers have to use to call the Service. The binding specifies unambiguously the protocol stack required to access a Service Interface. Bindings may be defined statically at compile time or they may use a variety of dynamic run-time mechanisms (DNS, ports, discovery).

Service Capability Set: A grouping of the service operations that remains sensible and coherent, and also provides a Service Provider with an ability to communicate to a Consumer its capability. The specification of services is based on the expectation that different deployments require different levels of complexity and functionality from a service. To this end a given service may be implementable at one of several distinct levels, corresponding to the inclusion of one or more capability sets.

Service Connection (connection)**:** A communications connection between a Consumed Service Interface and a Provided Service Interface over a specific Binding. When a component consumes the services of a provider component, this link is known as a Service Connection (connection).

Service Extension: Addition of capabilities to a base Service. A Service may extend the capabilities of another Service with additional operations. An extended Service is indistinguishable from the base Service to consumers such that consumers of the base Service can also be consumers of the extended Service without modification.

Protocol Stack: The stack of Protocol Layers required for communication.

Protocol Layer: The implementation of a specific Protocol. It provides a Protocol Service Access Point to layers above and uses the Protocol Service Access Point of the layer below.

Protocol Service Access Point (SAP): The point at which one layer's functions are provided to the layer above. A layer may provide protocol services to one or more higher layers and use the protocol services of one or more lower layers. A SAP defines unambiguously the interface for a protocol that may be used as part of a Service Interface Binding specification.

Protocol: The set of rules and formats (semantic and syntactic) used to determine the communication behaviour of a Protocol Layer in the performance of the layer functions. The state machines that operate and the protocol data units that are exchanged specify a protocol.

Service directory: An entity that provides publish and lookup facilities to service providers and consumers.

NOTE – Strictly speaking, a directory is not required if a well-known service is to be used; however, in most circumstances a directory provides required flexibility in the location of services. Service location can be statically configured, dynamically discovered through a service directory, or a combination of the two; this is an implementation choice. The service directory is itself, by definition, a service.

1.5 NOMENCLATURE

The following conventions apply throughout this Recommended Standard:

a) the words ‘shall’ and ‘must’ imply a binding and verifiable specification;

b) the word ‘should’ implies an optional, but desirable, specification;

c) the word ‘may’ implies an optional specification;

d) the words ‘is’, ‘are’, and ‘will’ imply statements of fact.

1.6 REFERENCES

Are these really the only valid references? What about the ISO BRM or other fundamental references for services, service systems, protocols, data models?

The following documents contain provisions which, through reference in this text, constitute provisions of this Recommended Standard. At the time of publication, the editions indicated were valid. All documents are subject to revision, and users of this Recommended Standard are encouraged to investigate the possibility of applying the most recent editions of the documents indicated below. The CCSDS Secretariat maintains a register of currently valid CCSDS Recommended Standards.

[1] *Mission Operations Reference Model*. Issue 1. Recommendation for Space Data System Practices (Magenta Book), CCSDS 520.1-M-1. Washington, D.C.: CCSDS, July 2010.

[2] “Media Types.” Internet Assigned Numbers Authority. https://www.iana.org/assignments/media-types/media-types.xhtml.

NOTE – Informative references are listed in annex D.

2 OVERVIEW

2.1 GENERAL

The MO service specifications detail a standard set of services. These services form a contract between a consumer of a service and the provider of that service. Each operation of a service has a set behaviour: a message is sent from the consumer to the provider to instigate the operation, and zero or more messages can be exchanged thereafter depending on the specific pattern and operation. This set of messages, and the pattern in which they are exchanged, needs to be defined for each operation of each service.

An interaction pattern details a standard exchange pattern that removes the need for each operation to detail its exchange pattern individually. By defining a pattern and stating that a given operation is defined in terms of that pattern, the operation definition can focus on the specifics of that operation and rely on the standard pattern to facilitate the interaction.

The MAL defines this set of standard interaction patterns as an abstract interface that is used by the MO services. There are three abstract interfaces defined in the MAL specification: the first is the abstract interface of the MAL that is presented to the higher layers, the second is the abstract interface of the Access Control component, and the third is the abstract interface that a Transport layer must provide to the MAL.

I do not believe that either of these have been defined as of this point. What are they? What functions do they provide?

The MO service specifications and the MAL are abstract in their definition; they do not contain any specific information on how to implement them for a particular programming language or transport encoding. Moving from the abstract to the implemented system, two other specifications are needed. One is the Language Mapping that states how the abstract MAL and MO service specifications are to be realised in some specific language: this defines the API in that language. The second is the transport mapping from the abstract MAL data structures into a specific and unambiguous encoding of the messages and to a defined and unambiguous mapping to a specific data transport. It is only when these mappings are defined that is possible to implement services that use the MAL interface and use the transport bindings to exchange data. (For further information on this, see reference [1].)

See earlier comment. Is this a concept or some primitive object definition? Where is “composite” defined”

The MO Object concept allows complex data models to be defined within MO services. It enables objects to be created in MO which can then be referenced from other Composites.

This document contains the formal specification for these abstract interfaces. More information about the MO Concept can be found in reference [D1], and more information about the Reference Model can be found in reference [1].

2.2 ABSTRACT INTERFACE SPECIFICATIONS

2.2.1 GENERAL

Each abstract interface is defined as a set of interaction patterns. For each pattern defined there is a common layout of the document section.

The following subsections describe the sections and diagram formats specific to the interaction patterns.

2.2.2 PATTERN OVERVIEW

The overview section of the pattern contains a message exchange sequence, which shows the interaction sequence between a source consumer and a destination provider.

The main aspects to note are the direction of the arrows (giving message direction) and the order of the messages (top down). The bars under the consumer and provider show synchronicity of the message exchange.

In the following example (figure 2-1) it can be seen that the consumer sends an initial message to the provider, which responds with an acknowledgement. The acknowledgment is in response to the initial message, as shown by the connected bars under the consumer and provider.

At some time in the future the provider will send another response:

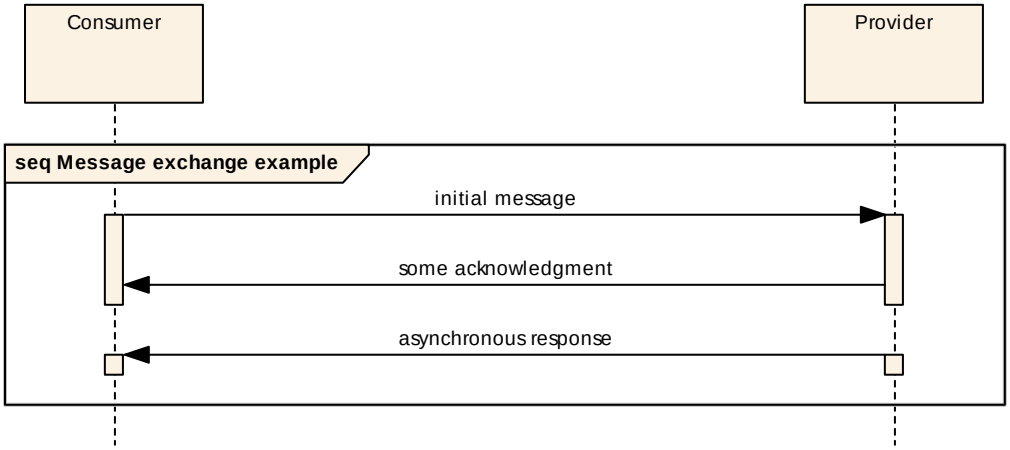


Figure 2-1: Message Exchange Sequence Example

Because the pattern shows the response, the consumer can expect it, but because the bars are not connected it is considered to be indeterminate. This means that although the message will arrive it cannot be determined when arrival will occur. It is important to note that any synchronicity shown is concerned only with messages; it does not imply, or require, a synchronous (blocking) behaviour in either the client or provider, as these are implementation issues.

2.2.3 DESCRIPTION AND USAGE

The Description and Usage sections are used to describe the pattern and define the expected usage respectively.

2.2.4 ERROR HANDLING

The Error Handling section defines the positions in the pattern where errors are permitted to be generated. It uses the same sequence diagram notation as the nominal pattern sequence from the Overview section.

2.2.5 OPERATION TEMPLATE

Each interaction pattern definition contains a table that defines the template for operations that use that pattern:

The term "signature" seems like a strange choice here. Signature typically has a very different meaning. And other docs, such as some SOAs, use "interface binding signature", but that is for a protocol interface. This appears to be a message content spec. As such, isn't this really an "abstract message structure"? To quote "contains the list of types that make up the Body of a particular message and the respective field names"

Table 2-1: Example Operation Template

Operation Identifier	<<Operation name>>	
Interaction Pattern	<<Interaction pattern name>>	
Pattern Sequence	Message	Body ~~Type~~Signature
<<Message direction>>	<<Message name>>	<<Message ~~types~~signature*>>
…	…	…

The message direction denotes the direction of the message relative to the provider of the pattern and is either IN or OUT. So all messages directed towards the provider are IN messages, and all messages directed away from the provider are OUT messages. It is expected that message names match those defined in the Primitives section.

Blue cells (dark grey when printed on a monochrome printer) contain table headings, light grey cells contain fields that are fixed for a pattern, and white cells contain values that must be provided by the operation or structure.

The Body ~~Type~~Signature column contains the list of types that make up the Body of a particular message and the respective field names. Zero to many types may be listed here and together define the body of the indicated message.

This document defines a data type specification language in section 4 (referred to as the MAL data types) that is expected to be the default data type specification language. The Body ~~Types~~Signatures are defined using the MAL data types to define the various base data types, the notation used to describe them, and the rules for their combination into complex data structures. This is separate from the encoding technology used which is responsible for

Are other data type specification languages permitted? What about JSON, XML, ASN.1, …?

mapping the abstract data type notation from the data type specification into an actual 'bits and bytes' representation.

~~Even though the Body Types in this specification are defined using the MAL data types, any data type specification language (such as XML Schema) may be used in an actual service specification.~~

2.2.6 PRIMITIVES

Each pattern defines a set of primitives in a table as below:

Table 2-2: Example Primitive List

Primitive
<<Primitive name>>

For each listed primitive there exists a Request, Message and Indication of the same name. The Requests and Indications are used by the following State Diagrams. Requests are transitions that are triggered by the component being modelled; Indications are transitions that are triggered by an external activity. Messages are the entities that are transferred between the two components to progress the interaction (see figure 2-2).

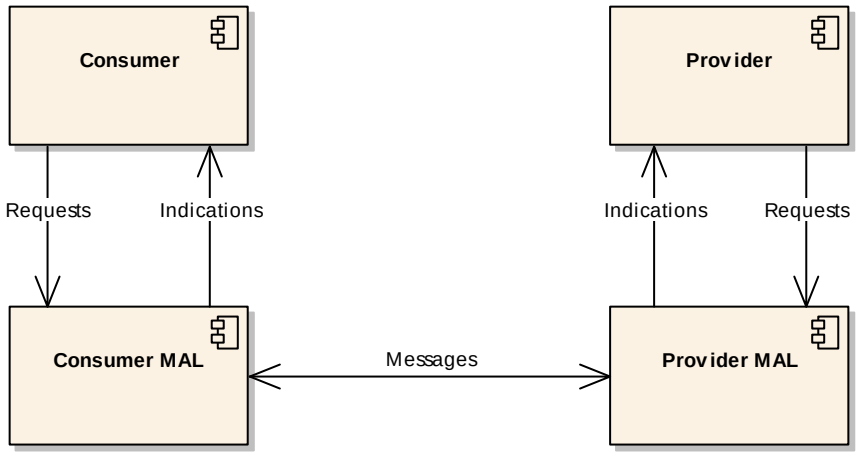


Figure 2-2: Request, Indication, and Message Relationship

Therefore a component issues Requests, which cause Messages to be transmitted, which raise Indications in the destination component.

2.2.7 STATE CHARTS

For each defined interaction there can also be specified one or more state diagrams (see figure 2-3).

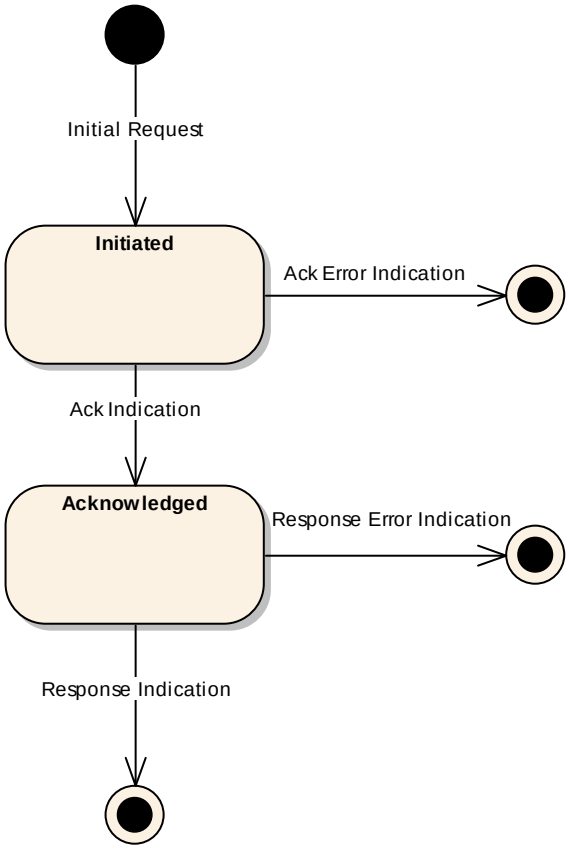


Figure 2-3: Consumer State Diagram Example

The diagrams define the allowed states and transitions between those states. The states are those of the MAL of the relevant component and apply to a single instance of a pattern; for example, figure 2-3 shows the allowed states and transitions for a specific pattern on the consumer, but it is the responsibility of the consumer MAL to maintain that state information and reject invalid state transitions. The state also only applies to a single instance of a pattern; as there can be many interactions concurrently active with one MAL, it needs to be able to track the state of each interaction instance independently.

Figure 2-3 shows the state diagram for the Consumer component from the example in figure 2-1, and includes some error state transitions that are not shown in the example.

I'm noticing that the language used here seems to be conflating the abstract MAL specification with a fully realized instance of the MAL spec as implemented in some given system, using some given language and transport binding. For instance, I do not believe that you can say of the abstract MAL spec that it has to handle multiple concurrent interactions. You could, on the other hand, say that an "Implemented MAL" or a "Real MAL deployment" has to have that property. I think you really need to clarify that distinction in your definitions and in your sections of text that address properties of as built systems.

2.2.8 REQUESTS AND INDICATIONS

2.2.8.1 General

For each pattern there are a set of Requests that can be issued and a set of Indications that can be received. These sets are defined in the Primitives section for the pattern. The allowed state transitions are defined in the State Charts section for the pattern.

Each pattern defines the following subsections for each primitive.

2.2.8.2 Function

The Function subsection defines the use of a specific Request or Indication.

2.2.8.3 Semantics

The Semantics subsection defines what information is required for the Request or Indication. It is equivalent to the arguments of a programming language function call.

2.2.8.4 When Generated

The When Generated subsection specifies the circumstances that trigger the generation of the Request or Indication.

2.2.8.5 Effect on Reception

The Effect on Reception subsection specifies what the effects of the reception of a Request by a MAL or an Indication by an Application are. Usually for a Request is it the transmission of a Message to the destination and for an Indication it is pattern specific.

2.2.8.6 Message Header

The MAL defines a standard message header that is used to manage the interaction. The Message Header subsection specifies the values of the fields of the Message Header to be used for the message. The subsection can specify all the fields, or only fields that are different from another Request/Indication.

The full definition of the message header structure is provided in 3.4.

2.2.9 MESSAGE EXAMPLE DECOMPOSITIONS

Each pattern is illustrated with an example which shows how each message will look when decomposed into a packet structure. It is important to note that such a type of structure is completely dependent on the chosen protocol.

For the example decomposition each message is split into up to three parts (figure 2-4), a standard message header, possibly a standard pattern body, and, finally, possibly a service-specific body:

Figure 2-4: Message Decomposition Key

Each message, in the examples, is split into the relevant fields; the example value is shown in the box, and the field name is given above (figure 2-5):

Message Header															
URI From	*Auth Id*	*URI To*	*Timestamp*	*QoS*	*Priority*	*Domain*	*Zone*	*Session*	*Interaction*	*Stage*	*Trans Id*	*Area*	*Service*	*Operation*	*Version*	*Is Error*
Provider	SC X	Broker		BEST	1	A.B.C	GROUND	LIVE	PUBSUB	3		Example	Example	testPubSub	1	FALSE

Message Header											
From	Auth Id	To	Timestamp	Interaction Type	Interaction Stage	Transaction Id	Area	Service	Operation	Version	Is Error
Provider	SC X	Broker		PUBSUB	3		Example	Example	testPubSub	1	FALSE

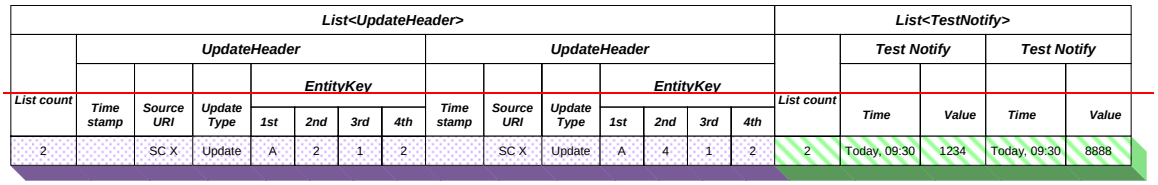
Figure 2-5: Message Header Decomposition Example

This figure introduces a slew of new terms that have not yet been specified and that then appear to be used in the following section without being specified. Examples are: AuthID, Priority, Domain, Zone, Session, Stage Area, Service,...

Field values are there to provide an example, and as such more human-readable values are often used rather than numeric equivalents that may be more efficient. For example, with enumerated values such as ‘Session’ the textual versions of the enumeration are given rather than the numerical versions, as this is more meaningful to a human. This is for example purposes only, however.

When more complex structures are being shown (figure 2-6), a single field can be part of a more complex structure which is itself part of another composite structure:

I suppose that these are instances of complex MO Objects, right? Built from primitive MO Objects?



List<UpdateHeader>														List<TestNotify>				
	UpdateHeader							UpdateHeader							Test Notify		Test Notify	
				EntityKey							EntityKey							
List count	Time stamp	Source URI	Update Type	1st	2nd	3rd	4th	Time stamp	Source URI	Update Type	1st	2nd	3rd	4th	List count	Time	Value	Time	Value
2		SC X	Update	A	2	1	2		SC X	Update	A	4	1	2	2	Today, 09:30	1234	Today, 09:30	8888

Source	keyValues			Test Notify	
	List<Attribute>			Time	Value
	List count	value	value		
SC X	2	param2	3	Today	1234

Figure 2-6: Message Body Decomposition Example

In the example above the ‘Update Type’ field is part of the UpdateHeader structure which is itself part of the overall UpdateHeader list structure. The above example also demonstrates the difference between a pattern body and a service-specific body. The UpdateHeader list is the pattern body and is followed by the service-specific TestNotify part.

NOTE – The structures used in the body of the examples for each pattern are only examples, and as such no explanation of contained values or meanings is provided or required.

2.3 ABSTRACT SERVICE SPECIFICATIONS

Each abstract service specification Blue Book is specified as an Area that consists of one or more services that are composed of a set of operations. Service Areas provide a simple way of grouping related Services together.

Each service specification includes an overview table that details the Area and Service Identifiers and then lists the operations of that service (see table 2-3).

This mentions “Area” and describes it as “one or more services”. This is still pretty vague. Perhaps an example might help make it clearer, especially here in Sec 2, which is usually explanatory material.

Table 2-3: Service Overview Table

Area Identifier	Service Identifier	Area Number	Service Number	~~Area~~Service Version
Interaction Pattern	**Operation Identifier**	**Operation Number**	**Support in replay**	**Capability Set**

The values of table 2-3 are used to populate the [Service Area][Service][Service Operation][~~Area~~Service Version] fields of the Message Header. The Area Number, Service

Number, and Operation Number fields provide an alternative numerical identification scheme parallel to the Identifier scheme.

The choice of whether to use Identifiers, numbers, or another alternative in an actual transport is transport specific, as long as it is possible to correctly determine the area, service, operation, and version values from the transmitted information. The numbers are expected to be used by transports that require a more efficient identification mechanism than Identifiers, and will not overlap with existing service definitions. The set of permissible values are in the UShort range and start at '1'.

The ~~Area~~Service Version is a number that is used to differentiate between issues of a ~~specification~~service. Initially it will be set to the number '1', and future updates to the ~~specification~~service will increase that number. The set of permissible values are in the UOctet range.

Is there a registry of some sort for Areas, Services, Service versions, associated data structures?

Support in replay is used to denote whether a specific operation is allowed to be used in a Replay session and is either of the value 'Yes' or 'No'. For example, only operations that do not modify history are usually supported in a replay session; operations such as setting a parameter would not be supported as they modify history.

The Capability set field is a numerical identifier that holds the Service Capability Set number for that operation. Operations that hold the same number within a service specification are considered to be part of the same Service Capability Set. The set of permissible values are in the UShort range.

3 ABSTRACT SERVICE SPECIFICATIONS

3.1 OVERVIEW

Earlier in the spec this is stated as "three abstract interfaces"…

There are ~~three~~two abstract interfaces defined in this specification. The first is the abstract interface of the MAL. The MAL abstract interface defines the interactions that are presented to the higher layers, the information and interaction between the two interacting components, and also the expected behaviour of the application layer that uses the MAL. It is detailed in 3.5.

This is mentioned earlier without even this level of definition. Its description here is barely more informative. Just what functions does it provide? Access control, authentication, identity verification, "what kind of "security" in addition to this?

The second is the abstract interface of the Access Control component. To support the access control and security aspects of the reference model in reference [1] the MAL requires a standard abstract interface to an Access Control component. The access control component only has the single interaction that is used by the MAL as defined in the Reference Model (reference [1]). The implementation of the Access Control component is deployment specific, as is the security policy and rules to be used; however, the interaction with that component is part of the MAL standard. It is detailed in 3.6.

~~The third is the abstract interface that a Transport layer provides to the MAL. It specifies what facilities must be made available to a compliant MAL and also the required behaviour of the Transport. It is detailed in 3.7.~~

Does this mean that there is no longer a standard abstract Transport Layer? That seems like a major architectural change.

General constraints that apply to all patterns are defined in 3.2, 3.3, and 3.4.

3.2 TRANSACTION HANDLING

3.2.1 The message header shall contain a transaction identifier field providing a mechanism for the originating MAL of a message exchange to uniquely identify the response, or set of responses, to a message.

3.2.2 Each instance of a pattern shall be considered a single transaction.

3.2.3 The transaction identifier shall be:

a) used by the originating MAL to identify messages in a particular instance of an interaction pattern, and

b) returned by the service provider in matching messages (returns from submits, etc.).

3.2.4 The originating MAL shall ensure that the combination of the following Message Header fields form a unique index: pattern type, transaction identifier, the source ~~address, the session, the domain, the network~~, the service area, the service, and operation.

But Domain and Zone still show up in the Message Header. Are they no longer used?

3.3 STATE TRANSITIONS

3.3.1 Each pattern shall specify the set of states that both the provider and consumer can attain during the execution of the pattern.

3.3.2 To move between the states, each pattern shall define the set of legal transitions either as a Request or an Indication.

3.3.3 For a Request to be issued, the source entity shall be in the correct state.

3.3.4 Issuing a Request when in the incorrect state shall cause an INCORRECT_STATE error to be raised by the local MAL, and the pattern shall end at this point.

3.3.5 For Indications, the receiving entity shall be required to be in the correct state.

3.3.6 Reception of an Indication in a state other than the correct one shall cause an INCORRECT_STATE error to be raised by the local MAL, and the pattern shall end at this point.

3.4 MESSAGE COMPOSITION

3.4.1 MESSAGE HEADER FIELD VALUES

3.4.1.1 The header fields, with the values specified in table 3-1, shall be used for all interaction patterns.

3.4.1.2 Each interaction pattern may override the values in table 3-1 with values defined in the relevant Request and Indication section for that pattern.

~~**3.4.1.3** The Domain shall be defined as a list of identifiers.~~

~~**3.4.1.4** The most significant domain part shall be listed first in the list (for example, Agency), and each subsequent domain identifier in the list narrows the preceding domain.~~

~~**3.4.1.5** Each Identifier part of the Domain shall be allowed the full range of Identifier values with the restriction that it is NOT allowed to contain the '.' character.~~

~~**3.4.1.6** The contents of URI From and URI To shall be transport specific (see reference [1] for more information).~~

~~**3.4.1.7** Authentication Id, QoSLevel, Priority, Network Zone, and Session fields shall be populated as defined in reference [1].~~

3.4.1.3 The fields From and To shall hold an Identifier.

Unstated "An Identifier may be a URI or a "textual name"."

3.4.1.4 If the fields From and To hold a URI, then the resolution shall be transport specific (see reference [1] for more information).

What is a "textual name"? How is it defined? Is there a registry of such names? How does that registry work?

3.4.1.5 If the fields From and To hold a textual name, then the name must be resolved by the transport before sending the message (similar to a DNS hostname lookup).

But 3.1 says you are no longer specifying a Transport. This appears to be levying vague "similar to a DNS lookup" requirements onto a Transport layer that no longer exists.

3.4.1.6 The optional Extra field, if used, shall contain a set of named values with optional extra fields.

NOTE – Examples of potential options include: Authentication Id, QoSLevel, Priority, Network Zone, and Session fields.

This all seems pretty vague to me. Do the "names" define the possible fields, is there a list of these, is it extensible, is there ordering or it is, in essence a sort of keyword = value syntax?

~~Table 3-1: MAL Message Header Fields~~

~~Field~~	~~Type~~	~~Value~~
~~URI From~~	~~Identifier~~	~~Message Source URI~~
~~Authentication Id~~	~~Blob~~	~~Source Authentication Identifier~~
~~URI To~~	~~URI~~	~~Message Destination URI~~
~~Timestamp~~	~~Time~~	~~Message generation timestamp~~
~~QoSlevel~~	~~QoSLevel~~	~~The QoS level of the message~~
~~Priority~~	~~UInteger~~	~~The QoS priority of the message~~
~~Domain~~	~~List<Identifier>~~	~~Domain of the message~~
~~Network Zone~~	~~Identifier~~	~~Network zone of the message~~
~~Session~~	~~SessionType~~	~~Type of session of the message~~
~~Session Name~~	~~Identifier~~	~~Name of the session of the message~~
~~Interaction Type~~	~~InteractionType~~	~~Interaction Pattern Type~~
~~Interaction Stage~~	~~UOctet~~	~~Interaction Pattern Stage~~
~~Transaction Id~~	~~Long~~	~~Unique to consumer~~
~~Service Area~~	~~UShort~~	~~Service Area~~
~~Service~~	~~UShort~~	~~Service~~
~~Operation~~	~~UShort~~	~~Service Operation~~
~~Area version~~	~~UOctet~~	~~Areaversion~~
~~Is Error Message~~	~~Boolean~~	~~'True' if this is an error message; else 'False'~~

Table 3-1: MAL Message Header Fields

Field	Type	Value
From	Identifier	Message Source
Authentication Id	Blob	Source Authentication Identifier
To	Identifier	Message Destination
Timestamp	Time	Message generation timestamp
Interaction Type	InteractionType	Interaction Pattern Type
Interaction Stage	UOctet	Interaction Pattern Stage
Transaction Id	Long	Unique to consumer
Service Area	Identifier	Service Area
Service	Identifier	Service
Operation	Identifier	Service Operation
Service version	UOctet	Service version
Is Error Message	Boolean	'True' if this is an error message; else 'False'
Extra	List<NamedValue>	Optional extra fields of the message

3.4.2 MESSAGE BODY

3.4.2.1 The message body shall be composed of the types listed in the Body ~~Type~~Signature column of the operation template in a service specification.

3.4.2.2 Zero to many types may be listed in the Body ~~Type~~Signature column, each type being one part of the message body, and together form the complete body of the indicated message.

3.4.2.3 The listed types should use the MAL data type specification (or service-specific types derived from these) as defined in section 4 of this specification; however, any data type specification (such as XML Schema) may be used.

Ahh, so XML is allowed? Or, as suggested in sec 2.2.5, XML is no longer supported?

3.4.2.4 How the message body is actually encoded shall be dependent on the encoding technology used, but all message parts as presented above are expected to be supplied.

Given the vague nature of the "Extra" fields I do not understand how this could be done in any standard way.

3.4.3 ERROR MESSAGES

3.4.3.1 A service specification shall define which error numbers may be returned for a specific operation and also if any extra information is provided with the error message.

3.4.3.2 The service specification shall specify the data type of the extra information for the specific error.

3.4.3.3 A UInteger value containing the Error Number (see section 5) shall immediately follow the Message Header.

3.4.3.4 The extra information value shall immediately follow the Error Number.

3.4.3.5 If no extra information is provided by an error, then the extra information value shall be set to NULL.

3.5 MAL SERVICE INTERFACE

3.5.1 SEND INTERACTION PATTERN

3.5.1.1 Overview

The SEND pattern is the most basic interaction (figure 3-1); it is a single message from the consumer to the provider. No acknowledgement is sent by the provider.

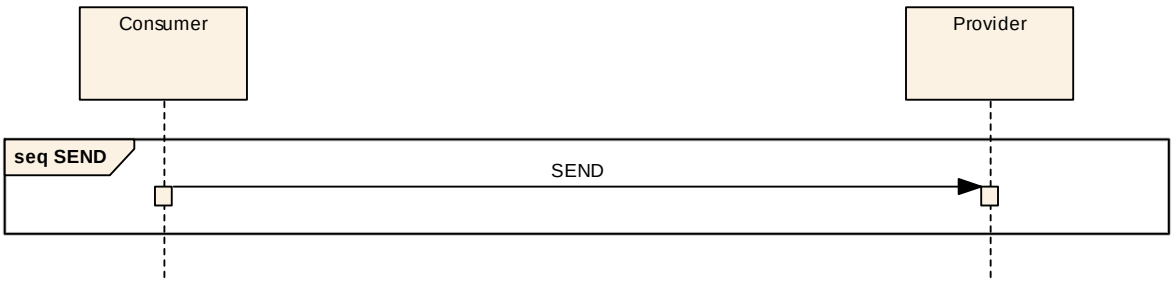


Figure 3-1: SEND Interaction Pattern Message Sequence

3.5.1.2 Description

The SEND pattern is the basic interaction of which all other patterns can be considered extensions. It is the simple passing of a message from a consumer to a provider. Because there is no message 'conversation' implied with a simple SEND, there is no requirement for a transaction identifier in the message. However, one may be specified. No return message is sent from the provider to the consumer, so the consumer has no indication the provider has received the message.

3.5.1.3 Usage

The SEND pattern is expected to be used for non-critical messages where the possible loss of one or more of these messages is not considered critical. An example would be regular heartbeat-type messages.

3.5.1.4 Error Handling

Errors (see section 5) may be generated by the consumer service layers, but no error can be generated by the provider as the interaction pattern does not allow for the provider to return an error.

3.5.1.5 Operation Template

The SEND template only contains the operation name and the body signature containing both the type of the structure and respective field name that is sent as the message body:

Table 3-2: SEND Operation Template

Operation Identifier	<<Operation name>>	
Interaction Pattern	SEND	
Pattern Sequence	Message	Body ~~Type~~Signature
IN	SEND	<<Body ~~type~~signature>>

3.5.1.6 Primitives

Table 3-3: SEND Primitive List

Primitive
SEND

3.5.1.7 State Charts

3.5.1.7.1 Consumer Side

The state chart contains only one transition from the initial state to the final one, so it is not represented. This transition is triggered by the primitive **SEND Request**.

3.5.1.7.2 Provider Side

The state chart contains only one transition from the initial state to the final one, so it is not represented. This transition is triggered by the primitive **SEND Indication**.

3.5.1.8 Requests and Indications

3.5.1.8.1 SEND

3.5.1.8.1.1 Function

3.5.1.8.1.1.1 The **SEND Request** primitive shall be used by the consumer application to initiate a SEND Interaction.

3.5.1.8.1.1.2 The **SEND Indication** primitive shall be used by the provider MAL to deliver a **SEND Message** to a provider and initiate a SEND Interaction.

3.5.1.8.1.2 Semantics

SEND Request and **SEND Indication** shall provide parameters as follows:

(SEND Message Header, SEND Message Body, QoS properties)

3.5.1.8.1.3 When Generated

a) A **SEND Request** may be generated by the consumer application at any time.

b) A **SEND Indication** shall be generated by the provider MAL upon reception of a **SEND Message**, once checked via the Access Control interface, from a consumer.

3.5.1.8.1.4 Effect on Reception

a) Reception of a **SEND Request** shall, once checked via the Access Control interface, cause the consumer MAL to initiate a SEND Interaction by transmitting a **SEND Message** to the provider.

b) The pattern shall end at this point for the consumer.

c) On reception of a **SEND Indication** a provider shall process the operation.

d) The pattern shall end at this point for the provider.

3.5.1.8.1.5 Message Header

a) For the **SEND Message** the message header fields shall be as defined in 3.4 except for the fields in table 3-4.

b) The contents of the **SEND Message** body, as specified in the operation template, shall immediately follow the message header.

Table 3-4: SEND Message Header Fields

Field	Value
~~URI~~ From	Consumer ~~URI~~
Authentication Id	Consumer Authentication Identifier
~~URI~~ To	Provider ~~URI~~
Interaction Type	SEND
Interaction Stage	Not used
Transaction Id	Not used

3.5.1.9 Example

The following example shows a simple example service with a single SEND pattern-based operation:

Operation Identifier	testSend	
Interaction Pattern	SEND	
Pattern Sequence	Message	Body ~~Type~~Signature
IN	SEND	TestBody body

The TestBody structure is defined below:

Name	TestBody		
Extends	Composite		
Short Form Part	Example Only		
Field	Type	Nullable	Comment
FirstItem	String	Yes	Example String item
SecondItem	Integer	Yes	Example Integer item

This corresponds to the following message:

Message Header																*Test Body*	
URI From	*Auth Id*	*URI To*	*Timestamp*	*QoS*	*Priority*	*Domain*	*Zone*	*Session*	*Interaction*	*Stage*	*Trans Id*	*Area*	*Service*	*Operation*	*Version*	*Is Error*	*First Item*	*Second Item*
Consumer	Op A	Provider		BEST	1	A.B.C	GROUND	LIVE	SEND		1233	Example	Example	testSend	1	FALSE	Hello	1234

Message Header												Test Body	
From	Auth Id	To	Timestamp	Interaction Type	Interaction Stage	Trans. Id	Area	Service	Operation	Version	Is Error	First Item	Second Item
Consumer	Op A	Provider		SEND		123	Example	Example	testSend	1	FALSE	Hello	1

NOTE – An actual service specification, rather than the example given here, would fully specify the meaning of, and required values of, all structures used by the service.

3.5.2 SUBMIT INTERACTION PATTERN

3.5.2.1 Overview

The SUBMIT pattern is a simple confirmed message exchange pattern (figure 3-2). The consumer sends a message to a provider and the provider acknowledges it.

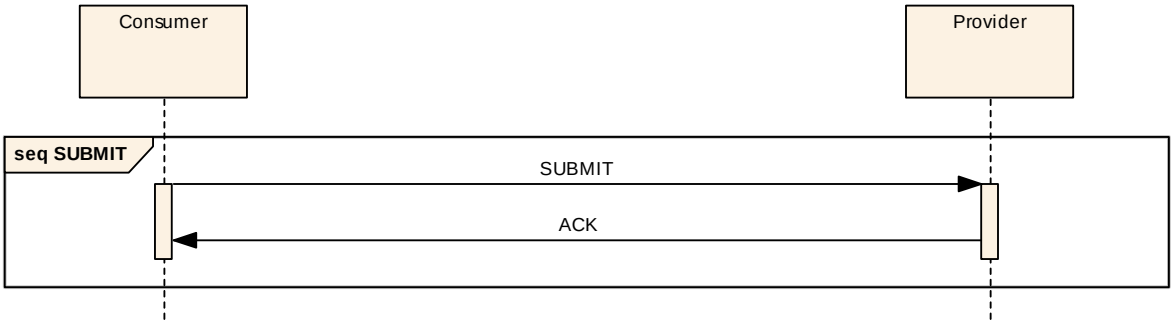


Figure 3-2: SUBMIT Interaction Pattern Message Sequence

The meaning of the return acknowledgement is operation specific. Each operation specification details the exact meaning of the operation's return acknowledgement for its context.

3.5.2.2 Description

The SUBMIT pattern extends the SEND pattern by providing a return acknowledgment message from the provider back to the consumer. The service specification details the meaning of the acknowledgment message for a specific operation.

3.5.2.3 Usage

The SUBMIT pattern is used for simple operations that complete quickly but must be confirmed to the consumer.

3.5.2.4 Error Handling

a) The return acknowledgment shall be replaced with an error message (see section 5) if an error occurs during the processing of the operation as shown in figure 3-3.

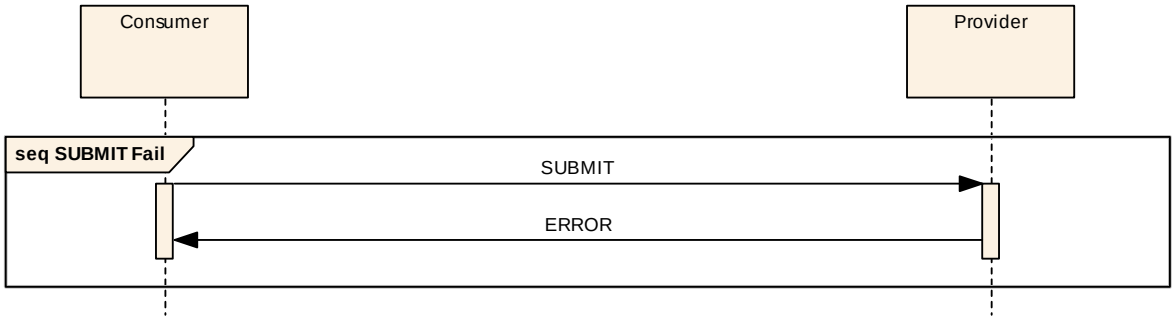


Figure 3-3: SUBMIT Interaction Pattern Error Sequence

b) Either the acknowledgment or the error message shall be returned but never both.

c) The service specification shall specify that the acknowledgement is not returned until all processing that can generate an error has been completed, if a service is required to be able to return an error during the processing of the message.

3.5.2.5 Operation Template

The SUBMIT template only contains the operation name and the body signature containing both the type of the structure and respective field name that is submitted as the Submit message body:

Table 3-5: SUBMIT Operation Template

Operation Identifier	<<Operation name>>	
Interaction Pattern	SUBMIT	
Pattern Sequence	Message	Body ~~Type~~Signature
IN	SUBMIT	<<Body ~~type~~signature>>

The return acknowledgement message does not have a body to keep the operation as simple as possible. Because of this it is not shown in the operation template.

NOTE – If a service-defined return message is required, for example, to return an identifier for the operation, then the REQUEST pattern should be used instead of the SUBMIT pattern.

3.5.2.6 Primitives

Table 3-6: SUBMIT Primitive List

Primitive
SUBMIT
ACK
ERROR

3.5.2.7 State Charts

3.5.2.7.1 Consumer Side

Figure 3-4 shows the consumer side state chart for the SUBMIT pattern:

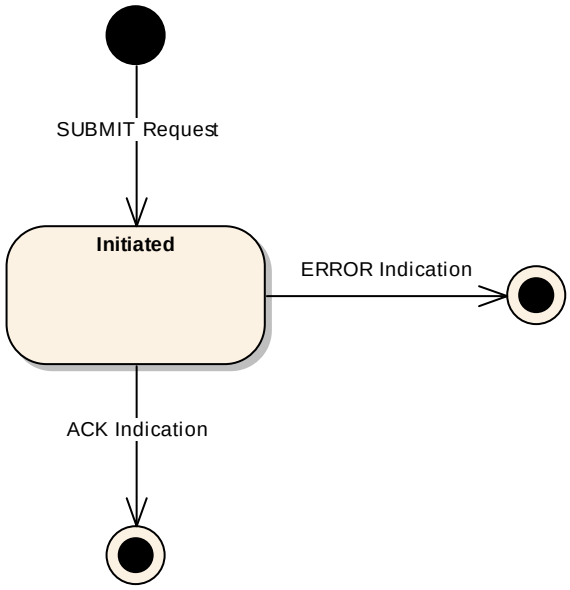


Figure 3-4: SUBMIT Consumer State Chart

a) The initial transition is triggered by the primitive **SUBMIT Request** and shall lead to the **Initiated State**.

b) There are two possible transitions from the **Initiated State**:

   1) the first is the indication of an acknowledgement, **ACK Indication**, and shall lead to the final state;

   2) the second is the indication of an error, **ERROR Indication**, and shall lead to the final state.

3.5.2.7.2 Provider Side

Figure 3-5 shows the provider side state chart for the SUBMIT pattern:

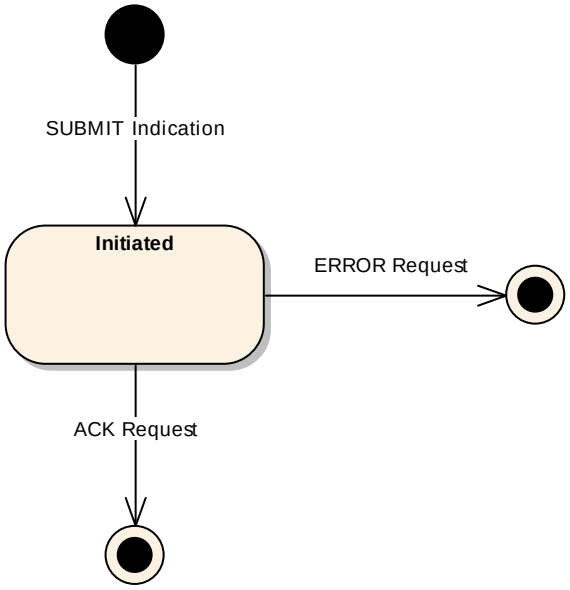


Figure 3-5: SUBMIT Provider State Chart

a) The initial transition is triggered by the primitive **SUBMIT Indication** and shall lead to the **Initiated State**.

b) There are two possible transitions from the **Initiated State**:

   1) the first is the transmission of an acknowledgement, **ACK Request**, and shall lead to the final state;

   2) the second is the transmission of an error, **ERROR Request**, and shall lead to the final state.

3.5.2.8 Requests and Indications

3.5.2.8.1 SUBMIT

3.5.2.8.1.1 Function

a) The **SUBMIT Request** primitive shall be used by the consumer application to initiate a SUBMIT Interaction.

b) The **SUBMIT Indication** primitive shall be used by the provider MAL to deliver a **SUBMIT Message** to a provider and initiate a SUBMIT Interaction.

3.5.2.8.1.2 Semantics

SUBMIT Request and **SUBMIT Indication** shall provide parameters as follows:

(SUBMIT Message Header, SUBMIT Message Body, QoS properties)

3.5.2.8.1.3 When Generated

a) A **SUBMIT Request** may be generated by the consumer application at any time.

b) A **SUBMIT Indication** shall be generated by the provider MAL upon reception of a **SUBMIT Message**, once checked via the Access Control interface, from a consumer.

3.5.2.8.1.4 Effect on Reception

a) Reception of a **SUBMIT Request** shall, once checked via the Access Control interface, cause the consumer MAL to initiate a SUBMIT Interaction by transmitting a **SUBMIT Message** to the provider.

b) The consumer MAL shall then enter the **Initiated State**.

c) On reception of a **SUBMIT Indication** by the provider, the provider MAL shall enter the **Initiated State**.

d) The provider shall then start processing the operation at this point.

3.5.2.8.1.5 Message Header

a) For the **SUBMIT Message** the message header fields shall be as defined in 3.4 except for the fields in table 3-7.

b) The contents of the Message Body, as specified in the operation template, shall immediately follow the message header.

Table 3-7: SUBMIT Message Header Fields

Field	Value
~~URI~~ From	Consumer ~~URI~~
Authentication Id	Consumer Authentication Identifier
~~URI~~ To	Provider ~~URI~~
Interaction Type	SUBMIT
Interaction Stage	1
Transaction Id	Provided by consumer MAL

3.5.2.8.2 ACK

3.5.2.8.2.1 Function

a) The **ACK Request** primitive shall be used by the provider application to acknowledge a SUBMIT Interaction.

b) The **ACK Indication** primitive shall be used by the consumer MAL to deliver an **ACK Message** to a consumer.

3.5.2.8.2.2 Semantics

ACK Request and **ACK Indication** shall provide parameters as follows:

(ACK Message Header, QoS properties)

3.5.2.8.2.3 When Generated

a) An **ACK Request** shall be used by the provider application with the provider MAL in the **Initiated State** to acknowledge a SUBMIT Interaction.

b) An **ACK Indication** shall be generated by the consumer MAL in the **Initiated State** upon reception of an **ACK Message**, once checked via the Access Control interface, from a provider.

3.5.2.8.2.4 Effect on Reception

a) Reception of an **ACK Request** shall, once checked via the Access Control interface, cause the provider MAL to transmit an **ACK Message** to the consumer.

b) The pattern shall end at this point for the provider.

c) On reception of an **ACK Indication** a consumer shall end the interaction pattern with success.

3.5.2.8.2.5 Message Header

For the **ACK message** the message header fields shall be as defined in 3.4 except for the fields in table 3-8.

Table 3-8: Submit ACK Message Header Fields

Field	Value
~~URI~~ From	Provider ~~URI~~
Authentication Id	Provider Authentication Identifier
~~URI~~ To	Consumer ~~URI~~
Interaction Type	SUBMIT
Interaction Stage	2
Transaction Id	Transaction Id from initial message

3.5.2.8.3 ERROR

3.5.2.8.3.1 Function

a) The **ERROR Request** primitive shall be used by the provider application to end a SUBMIT Interaction with an error.

b) The **ERROR Indication** primitive shall be used by the consumer MAL to deliver an **ERROR Message** to a consumer.

3.5.2.8.3.2 Semantics

ERROR Request and **ERROR Indication** shall provide parameters as follows:

(ERROR Message Header, Error Number, Extra Information, QoS properties)

3.5.2.8.3.3 When Generated

a) An **ERROR Request** shall be used by the provider application with the provider MAL in the **Initiated State** to transmit an error.

b) An **ERROR Indication** shall be generated by the consumer MAL in the **Initiated State** upon one of two events:

1) the reception of an **ERROR Message**, once checked via the Access Control interface, from a provider;

2) an error raised by the local communication layer.

3.5.2.8.3.4 Effect on Reception

a) Reception of an **ERROR Request** shall, once checked via the Access Control interface, cause the provider MAL to transmit an **ERROR Message** to the consumer.

b) The pattern shall end at this point for the provider.

c) On reception of an **ERROR Indication** a consumer shall end the interaction with an error.

3.5.2.8.3.5 Message Header

a) For the **ERROR Message** the message header fields shall be the same as for the **ACK Message** except for the 'Is Error Message' field which is set to TRUE.

b) The Error number and any extra information (referred to as the 'Error information' from this point onwards) shall immediately follow the message header.

3.5.2.9 Example

The following example shows a simple example service with a single SUBMIT pattern-based operation:

Operation Identifier	testSubmit	
Interaction Pattern	SUBMIT	
Pattern Sequence	Message	Body ~~Type~~Signature
IN	SUBMIT	TestBody body

The TestBody structure is defined in 3.5.1.9. This corresponds to the following message being transmitted:

Message Header																*Test Body*	
URI From	*Auth Id*	*URI To*	*Timestamp*	*QoS*	*Priority*	*Domain*	*Zone*	*Session*	*Interaction*	*Stage*	*Trans Id*	*Area*	*Service*	*Operation*	*Version*	*Is Error*	*First Item*	*Second Item*
Consumer	Op A	Provider		BEST	1	A.B.C	GROUND	LIVE	SUBMIT	1	1234	Example	Example	testSubmit	1	FALSE	Hello	1234

Message Header												Test Body	
From	Auth Id	To	Timestamp	Interaction Type	Interaction Stage	Trans. Id	Area	Service	Operation	Version	Is Error	First Item	Second Item
Consumer	Op A	Provider		SUBMIT		123	Example	Example	testSubmit	1	FALSE	Hello	1

and corresponds to the following acknowledgement being returned:

Message Header															
URI From	*Auth Id*	*URI To*	*Timestamp*	*QoS*	*Priority*	*Domain*	*Zone*	*Session*	*Interaction*	*Stage*	*Trans Id*	*Area*	*Service*	*Operation*	*Version*	*Is Error*
Provider	SC X	Consumer		BEST	1	A.B.C	GROUND	LIVE	SUBMIT	2	1234	Example	Example	testSubmit	1	FALSE

Message Header											
From	Auth Id	To	Timestamp	Interaction Type	Interaction Stage	Trans. Id	Area	Service	Operation	Version	Is Error
Provider	SC X	Consumer		SUBMIT		123	Example	Example	testSubmit	1	FALSE

NOTE – An actual service specification, rather than the example given here, would fully specify the meaning of, and required values of, all structures used by the service.

3.5.3 REQUEST INTERACTION PATTERN

3.5.3.1 Overview

The REQUEST pattern extends the SUBMIT pattern by replacing the simple acknowledgement with a data response message (figure 3-6). No acknowledgement other than the data response is sent.

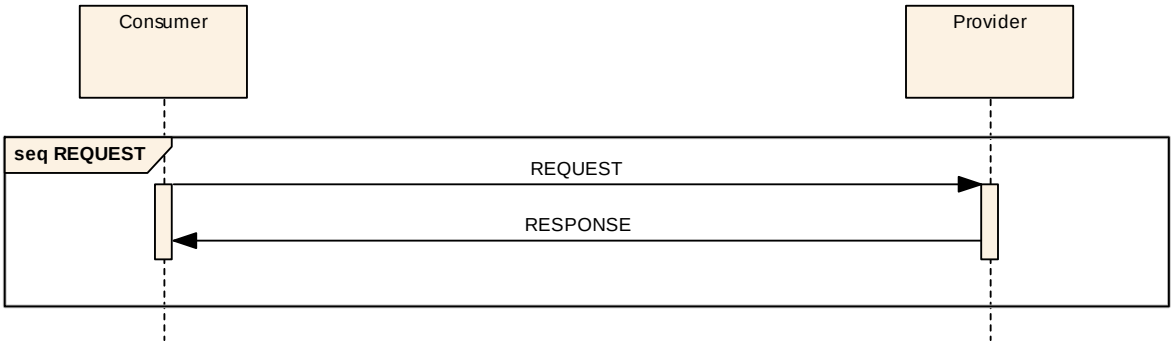


Figure 3-6: REQUEST Interaction Pattern Message Sequence

3.5.3.2 Description

The REQUEST pattern provides a simple request/response message exchange. Unlike the SUBMIT pattern, no acknowledgement is sent upon reception of the request; however, a data response is sent. The lack of an acknowledgement and only the return data response for this pattern means that it is primarily expected to be used for situations where the operation takes minimal time.

3.5.3.3 Usage

It is expected that the REQUEST pattern is to be used only for operations that complete in a relatively short period of time. If a more extended or indeterminate period is possible then the more advanced INVOKE or PROGRESS patterns should be specified.

3.5.3.4 Error Handling

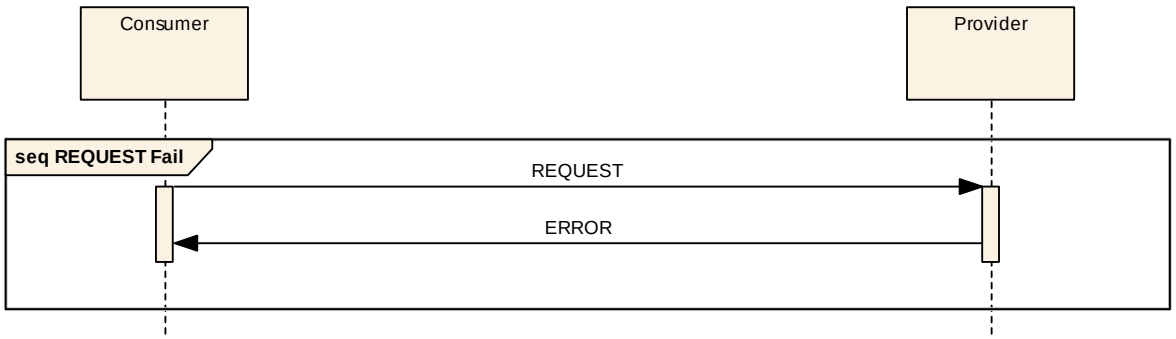


Figure 3-7: REQUEST Interaction Pattern Error Sequence

a) The data response shall be replaced with an error message (see section 5) if an error occurs during the processing of the operation (figure 3-7).

b) Either the data response or the error message shall be returned but never both.

3.5.3.5 Operation Template

The REQUEST pattern template extends the SUBMIT template by adding the requirement for a return ~~type~~message signature:

Table 3-9: REQUEST Operation Template

Operation Identifier	<<Operation name>>	
Interaction Pattern	REQUEST	
Pattern Sequence	Message	Body ~~Type~~Signature
IN	REQUEST	<<Request type>>
OUT	RESPONSE	<<Response type>>

3.5.3.6 Primitives

Table 3-10: REQUEST Primitive List

Primitive
REQUEST
RESPONSE
ERROR

3.5.3.7 State Charts

3.5.3.7.1 Consumer Side

Figure 3-8 shows the consumer side state chart for the REQUEST pattern:

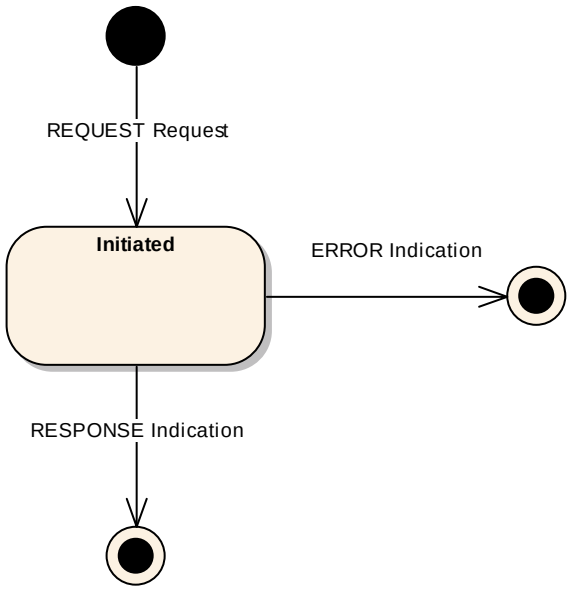


Figure 3-8: REQUEST Consumer State Chart

a) The initial transition is triggered by the primitive **REQUEST Request** and shall lead to the **Initiated State**.

b) There are two possible transitions from the **Initiated State**:

   1) the first is the indication of a response, **RESPONSE Indication**, and shall lead to the final state;

   2) the second is the indication of an error, **ERROR Indication**, and shall lead to the final state.

3.5.3.7.2 Provider Side

Figure 3-9 shows the provider side state chart for the REQUEST pattern:

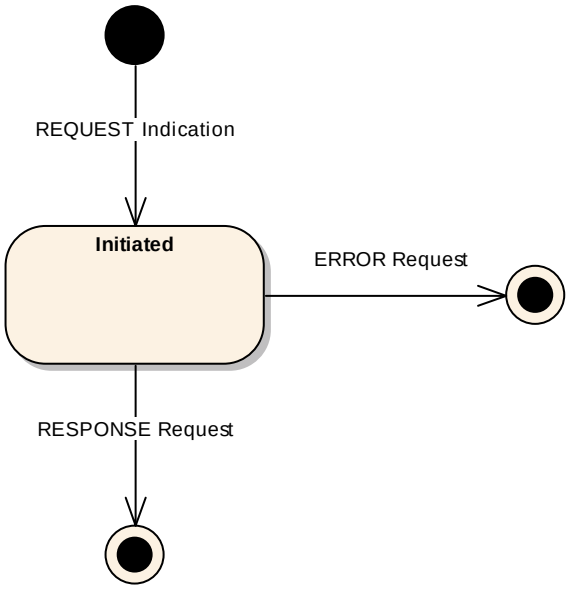


Figure 3-9: REQUEST Provider State Chart

a) The initial transition is triggered by the primitive **REQUEST Indication** and shall lead to the **Initiated State**.

b) There are two possible transitions from the **Initiated State**:

   1) the first is the transmission of a response, **RESPONSE Request**, and shall lead to the final state;

   2) the second is the transmission of an error, **ERROR Request**, and shall lead to the final state.

3.5.3.8 Requests and Indications

3.5.3.8.1 REQUEST

3.5.3.8.1.1 Function

a) The **REQUEST Request** primitive shall be used by the consumer application to initiate a REQUEST Interaction.

b) The **REQUEST Indication** primitive shall be used by the provider MAL to deliver a **REQUEST Message** to a provider and initiate a REQUEST Interaction.

3.5.3.8.1.2 Semantics

REQUEST Request and **REQUEST Indication** shall provide parameters as follows:

(REQUEST Message Header, REQUEST Message Body, QoS properties)

3.5.3.8.1.3 When Generated

a) A **REQUEST Request** may be generated by the consumer application at any time.

b) A **REQUEST Indication** shall be generated by the provider MAL upon reception of a **REQUEST Message**, once checked via the Access Control interface, from a consumer.

3.5.3.8.1.4 Effect on Reception

a) Reception of a **REQUEST Request** shall, once checked via the Access Control interface, cause the consumer MAL to initiate a REQUEST Interaction by transmitting a **REQUEST Message** to the provider.

b) The consumer MAL shall enter the **Initiated State** at this point.

c) On reception of a **REQUEST Indication** by the provider, the provider MAL shall enter the **Initiated State**.

d) The provider shall then start processing the operation at this point.

3.5.3.8.1.5 Message Header

a) For the **REQUEST Message** the message header fields shall be as defined in 3.4 except for the fields in table 3-11.

b) The contents of the Message Body, as specified in the operation template, shall immediately follow the message header.

Table 3-11: REQUEST Message Header Fields

Field	Value
~~URI~~ From	Consumer ~~URI~~
Authentication Id	Consumer Authentication Identifier
~~URI~~ To	Provider ~~URI~~
Session	Session of message
Interaction Type	REQUEST
Interaction Stage	1
Transaction Id	Provided by consumer MAL

3.5.3.8.2 RESPONSE

3.5.3.8.2.1 Function

a) The **RESPONSE Request** primitive shall be used by the provider application to transmit a response to a REQUEST Interaction.

b) The **RESPONSE Indication** primitive shall be used by the consumer MAL to deliver a **RESPONSE Message** to a consumer.

3.5.3.8.2.2 Semantics

RESPONSE Request and **RESPONSE Indication** shall provide parameters as follows:

(RESPONSE Message Header, RESPONSE Message Body, QoS properties)

3.5.3.8.2.3 When Generated

a) A **RESPONSE Request** shall be used by the provider application with the provider MAL in the **Initiated State** to transmit a final response to a REQUEST Interaction.

b) A **RESPONSE Indication** shall be generated by the consumer MAL in the **Initiated State** upon reception of a **RESPONSE Message**, once checked via the Access Control interface, from a provider.

3.5.3.8.2.4 Effect on Reception

a) Reception of a **RESPONSE Request** shall, once checked via the Access Control interface, cause the provider MAL to transmit the supplied response as a **RESPONSE Message** to the consumer.

b) The pattern shall end at this point for the provider.

c) On reception of a **RESPONSE Indication** a consumer shall end the interaction pattern with success.

3.5.3.8.2.5 Message Header

a) For the **RESPONSE Message,** the message header fields shall be as defined in 3.4 except for the fields in table 3-12.

b) The contents of the Message Body, as specified in the operation template, shall immediately follow the message header.

Table 3-12: Request RESPONSE Message Header Fields

Field	Value
~~URI~~ From	Provider ~~URI~~
Authentication Id	Provider Authentication Identifier
~~URI~~ To	Consumer ~~URI~~
Interaction Type	REQUEST
Interaction Stage	2
Transaction Id	Transaction Id from initial message

3.5.3.8.3 ERROR

3.5.3.8.3.1 Function

a) The **ERROR Request** primitive shall be used by the provider application to end a REQUEST Interaction with an error.

b) The **ERROR Indication** primitive shall be used by the consumer MAL to deliver an **ERROR Message** to a consumer.

3.5.3.8.3.2 Semantics

ERROR Request and **ERROR Indication** shall provide parameters as follows:

(ERROR Message Header, Error Number, Extra Information, QoS properties)

3.5.3.8.3.3 When Generated

a) An **ERROR Request** shall be used by the provider application with the provider MAL in the **Initiated State** to transmit an error.

b) An **ERROR Indication** shall be generated by the consumer MAL in the **Initiated State** upon one of two events:

1) the reception of an **ERROR Message**, once checked via the Access Control interface, from a provider;

2) an error raised by the local communication layer.

3.5.3.8.3.4 Effect on Reception

a) Reception of an **ERROR Request** shall, once checked via the Access Control interface, cause the provider MAL to transmit an **ERROR Message** to the consumer.

b) The pattern shall end at this point for the provider.

c) On reception of an **ERROR Indication** a consumer shall end the interaction with an error.

3.5.3.8.3.5 Message Header

a) For the **ERROR Message** the message header fields shall be the same as the **RESPONSE Message** except for the 'Is Error Message' field which is set to TRUE.

b) The Error information shall immediately follow the message header.

3.5.3.9 Example

The following example shows a simple service that defines a single REQUEST operation:

Operation Identifier	testRequest	
Interaction Pattern	REQUEST	
Pattern Sequence	Message	Body ~~Type~~Signature
IN	REQUEST	TestBody body
OUT	RESPONSE	TestResponse response

The TestBody structure is defined in 3.5.1.9 and the TestResponse structure is defined below:

Name	TestResponse		
Extends	Composite		
Short Form Part	Example Only		
Field	Type	Nullable	Comment
RspnItem	Boolean	Yes	Example Boolean item
RspnField	Float	Yes	Example Float item

This corresponds to the following message being transmitted:

Message Header																Test Body	
URI From	Auth Id	URI To	Timestamp	QoS	Priority	Domain	Zone	Session	Interaction	Stage	Trans Id	Area	Service	Operation	Version	Is Error	First Item	Second Item
Consumer	Op A	Provider		BEST	1	A.B.C	GROUND	LIVE	REQUEST	1	1235	Example	Example	testRequest	1	FALSE	Hello	1234

Message Header											Test Body	
From	Auth Id	To	Timestamp	Interaction Type	Interaction Stage	Trans. Id	Area	Service	Operation	Version	Is Error	First Item	Second Item
Consumer	Op A	Provider		REQUEST	1	123	Example	Example	testRequest	1	FALSE	Hello	1

And corresponds to the following response being returned:

Message Header																Test Response	
URI From	Auth Id	URI To	Timestamp	QoS	Priority	Domain	Zone	Session	Interaction	Stage	Trans Id	Area	Service	Operation	Version	Is Error	Rspn Item	Rspn Field
Provider	SC X	Consumer		BEST	1	A.B.C	GROUND	LIVE	REQUEST	2	1235	Example	Example	testRequest	1	FALSE	True	31.0

Message Header											Test Body	
From	Auth Id	To	Timestamp	Interaction Type	Interaction Stage	Trans. Id	Area	Service	Operation	Version	Is Error	Rspn Item	Rspn Field
Provider	SC X	Consumer		REQUEST	2	123	Example	Example	testRequest	1	FALSE	TRUE	31.0

NOTE – An actual service specification, rather than the example given here, would fully specify the meaning of, and required values of, all structures used by the service.

3.5.4 INVOKE INTERACTION PATTERN

3.5.4.1 Overview

The INVOKE pattern extends the REQUEST pattern to add a mandatory acknowledgement of the initial message (figure 3-10). This allows the operation to confirm the receipt of the request before proceeding to process the request.

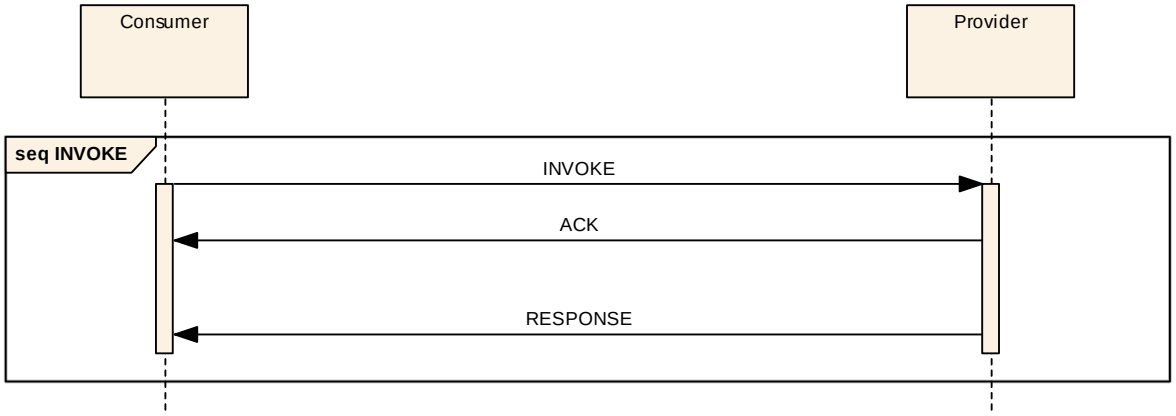


Figure 3-10: INVOKE Interaction Pattern Message Sequence

The acknowledgement message is an extension of the acknowledgment used in the SUBMIT pattern. This pattern allows the acknowledgment to return a service-specific message body. A service-defined acknowledgement is required because the INVOKE pattern is indeterminate. It allows an operation to provide an indication of the operation status upon INVOKE reception. An example would be validation of the invoke arguments.

3.5.4.2 Description

The INVOKE pattern extends the REQUEST pattern with the addition of a mandatory acknowledgement of the initial message.

3.5.4.3 Usage

The INVOKE pattern is expected to be used when there is a significant or indeterminate amount of time taken to process the request and produce the data response. The provision of the service-defined acknowledgement message allows an operation to return supplementary, status, or summary information about the request before processing continues (for example, an identifier used for querying INVOKE status using another operation).

3.5.4.4 Error Handling

The INVOKE pattern can report failure in two distinct ways:

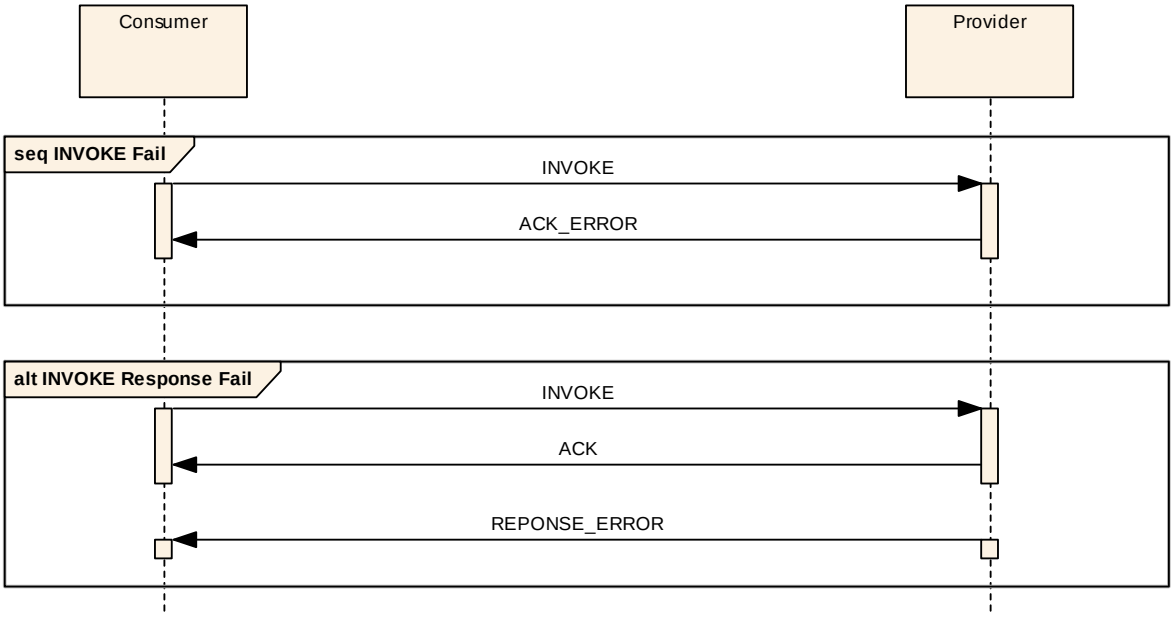


Figure 3-11: INVOKE Interaction Pattern Error Sequence

a) The provider may return an error message (see section 5) in replacement of the ACK message, shown in the first sequence in figure 3-11.

b) The provider may return an error after the acknowledgement is sent in replacement of the RESPONSE message, shown in the second sequence in figure 3-11.

c) After an error message is sent no further messages shall be generated as part of the pattern.

NOTE – It is expected that the first case would be used to report an error with the invoke request and the second to report an error that occurs during processing.

3.5.4.5 Operation Template

The INVOKE template extends the REQUEST pattern template by requiring a ~~structure~~message signature for the acknowledgment message. INVOKE operations take a ~~single~~ message body argument and return an acknowledgment ~~structure and~~followed by a single data response:

Table 3-13: INVOKE Operation Template

Operation Identifier	<<Operation name>>	
Interaction Pattern	INVOKE	
Pattern Sequence	Message	Body ~~Type~~Signature
IN	INVOKE	<<Request type>>
OUT	ACK	<<Ack type>>
OUT	RESPONSE	<<Response type>>

3.5.4.6 Primitives

Table 3-14: INVOKE Primitive List

Primitive
INVOKE
ACK
RESPONSE
ACK_ERROR
RESPONSE_ERROR

3.5.4.7 State Charts

3.5.4.7.1 Consumer Side

Figure 3-12 shows the consumer side state chart for the INVOKE pattern:

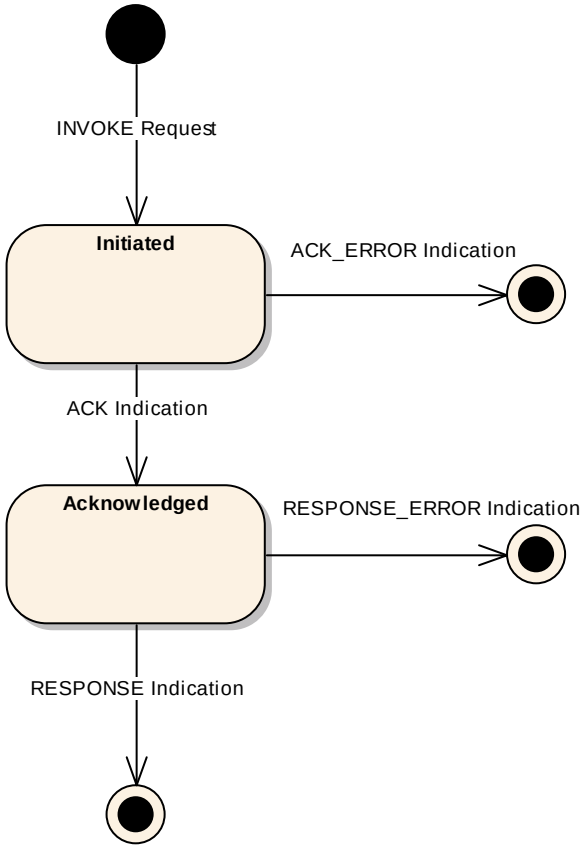


Figure 3-12: INVOKE Consumer State Chart

a) The initial transition is triggered by the primitive **INVOKE Request** and shall lead to the **Initiated State**.

b) There are two possible transitions from the **Initiated State**:

   1) the first is the indication of an acknowledgement, **ACK Indication**, and shall lead to the **Acknowledged State**;

   2) the second is the indication of an error, **ACK_ERROR Indication**, and shall lead to the final state.

c) There are two possible transitions from the **Acknowledged State**:

   1) the first is the indication of a response, **RESPONSE Indication**, and shall lead to the final state;

2) the second is the indication of an error, **RESPONSE_ERROR Indication**, and shall lead to the final state.

3.5.4.7.2 Provider Side

Figure 3-13 shows the provider side state chart for the INVOKE pattern:

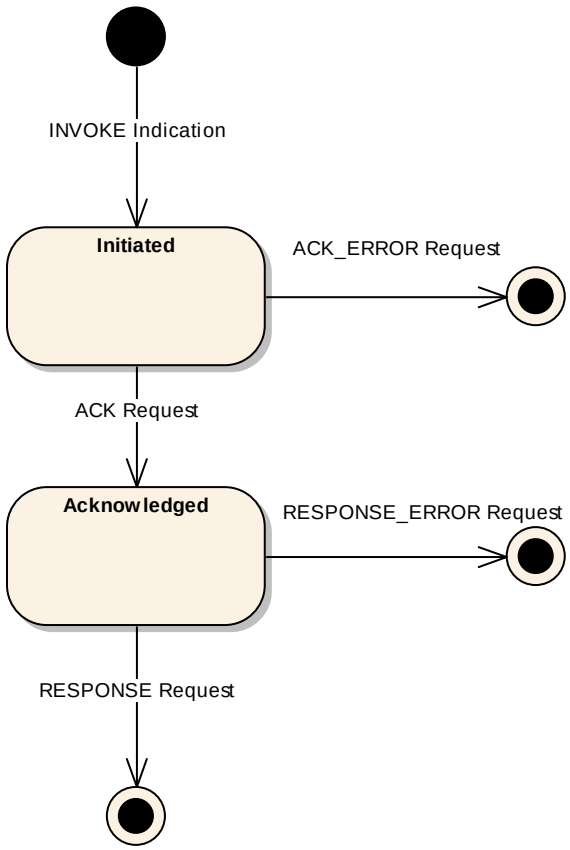


Figure 3-13: INVOKE Provider State Chart

a) The initial transition is triggered by the primitive **INVOKE Indication** and shall lead to the **Initiated State**.

b) There are two possible transitions from the **Initiated State**:

   1) the first is the transmission of an acknowledgement, **ACK Request**, and shall lead to the **Acknowledged State**;

   2) the second is the transmission of an error, **ACK_ERROR Request**, and shall lead to the final state.

c) There are two possible transitions from the **Acknowledged State**:

   1) the first is the transmission of a response, **RESPONSE Request**, and shall lead to the final state;

2) the second is the transmission of an error, **RESPONSE_ERROR Request**, and shall lead to the final state.

3.5.4.8 Requests and Indications

3.5.4.8.1 INVOKE

3.5.4.8.1.1 Function

a) The **INVOKE Request** primitive shall be used by the consumer application to initiate an INVOKE Interaction.

b) The **INVOKE Indication** primitive shall be used by the provider MAL to deliver an **INVOKE Message** to a provider and initiate an INVOKE Interaction.

3.5.4.8.1.2 Semantics

INVOKE Request and **INVOKE Indication** shall provide parameters as follows:

(INVOKE Message Header, INVOKE Message Body, QoS properties)

3.5.4.8.1.3 When Generated

a) An **INVOKE Request** may be generated by the consumer application at any time.

b) An **INVOKE Indication** shall be generated by the provider MAL upon reception of an **INVOKE Message**, once checked via the Access Control interface, from a consumer.

3.5.4.8.1.4 Effect on Reception

a) Reception of an **INVOKE Request** shall, once checked via the Access Control interface, cause the consumer MAL to initiate an INVOKE Interaction by transmitting an **INVOKE Message** to the provider.

b) The consumer MAL shall enter the **Initiated State**.

c) On reception of an **INVOKE Indication** by the provider, the provider MAL shall enter the **Initiated State**.

d) The provider shall then start processing the operation at this point.

3.5.4.8.1.5 Message Header

a) For the **INVOKE Message** the message header fields shall be as defined in 3.4 except for the fields in table 3-15.

b) The contents of the Message Body, as specified in the operation template, shall immediately follow the message header.

Table 3-15: INVOKE Message Header Fields

Field	Value
~~URI~~ From	Consumer ~~URI~~
Authentication Id	Consumer Authentication Identifier
~~URI~~ To	Provider ~~URI~~
Interaction Type	INVOKE
Interaction Stage	1
Transaction Id	Provided by consumer MAL

3.5.4.8.2 ACK

3.5.4.8.2.1 Function

a) The **ACK Request** primitive shall be used by the provider application to acknowledge an INVOKE Interaction.

b) The **ACK Indication** primitive shall be used by the consumer MAL to deliver an **ACK Message** to a consumer.

3.5.4.8.2.2 Semantics

ACK Request and **ACK Indication** shall provide parameters as follows:

(ACK Message Header, **A**CK Message Body, QoS properties)

3.5.4.8.2.3 When Generated

a) An **ACK Request** shall be used by the provider application with the provider MAL in the **Initiated State** to acknowledge the INVOKE Interaction.

b) An **ACK Indication** shall be generated by the consumer MAL in the **Initiated State** upon reception of an **ACK Message**, once checked via the Access Control interface, from a provider.

3.5.4.8.2.4 Effect on Reception

a) Reception of an **ACK Request** shall, once checked via the Access Control interface, cause the provider MAL to transmit the supplied acknowledgement as an **ACK Message** to the consumer.

b) A provider MAL in the **Initiated State** shall enter the **Acknowledged State**.

c) On reception of an **ACK Indication** by the consumer, the consumer MAL shall enter the **Acknowledged State**.

3.5.4.8.2.5 Message Header

a) For the **ACK Message** the message header fields shall be as defined in 3.4 except for the fields in table 3-16.

b) The contents of the Message Body, as specified in the operation template, shall immediately follow the message header.

Table 3-16: Invoke ACK Message Header Fields

Field	Value
~~URI~~ From	Provider ~~URI~~
Authentication Id	Provider Authentication Identifier
~~URI~~ To	Consumer ~~URI~~
Interaction Type	INVOKE
Interaction Stage	2
Transaction Id	Transaction Id from initial message

3.5.4.8.3 ACK_ERROR

3.5.4.8.3.1 Function

a) The **ACK_ERROR Request** primitive shall be used by the provider application to end an INVOKE Interaction with an error.

b) The **ACK_ERROR Indication** primitive shall be used by the consumer MAL to deliver an **ACK_ERROR Message** to a consumer.

3.5.4.8.3.2 Semantics

ACK_ERROR Request and **ACK_ERROR Indication** shall provide parameters as follows:

(ACK_ERROR Message Header, Error Number, Extra Information, QoS properties)

3.5.4.8.3.3 When Generated

a) **ACK_ERROR Request** shall be used by the provider application with the provider MAL in the **Initiated State** to transmit an error.

b) An **ACK_ERROR Indication** shall be generated by the consumer MAL in the **Initiated State** upon one of two events:

   1) the reception of an **ACK_ERROR Message**, once checked via the Access Control interface, from a provider;

   2) an error raised by the local communication layer.

3.5.4.8.3.4 Effect on Reception

a) Reception of an **ACK_ERROR Request** shall, once checked via the Access Control interface, cause the provider MAL to transmit an **ACK_ERROR Message** to the consumer.

b) The pattern shall end at this point for the provider.

c) On reception of an **ACK_ERROR Indication** a consumer shall end the interaction with an error.

3.5.4.8.3.5 Message Header

a) For the **ACK_ERROR Message** the message header fields shall be the same as for the **ACK Message** except for the 'Is Error Message' field which is set to TRUE.

b) The Error information shall immediately follow the message header.

3.5.4.8.4 RESPONSE

3.5.4.8.4.1 Function

a) The **RESPONSE Request** primitive shall be used by the provider application to transmit a response to an INVOKE Interaction.

b) The **RESPONSE Indication** primitive shall be used by the consumer MAL to deliver a **RESPONSE Message** to a consumer.

3.5.4.8.4.2 Semantics

RESPONSE Request and **RESPONSE Indication** shall provide parameters as follows:

(RESPONSE Message Header, RESPONSE Message Body, QoS properties)

3.5.4.8.4.3 When Generated

a) A **RESPONSE Request** shall be used by the provider application with the provider MAL in the **Acknowledged State** to transmit a final response to an INVOKE interaction.

b) A **RESPONSE Indication** shall be generated by the consumer MAL in the **Acknowledged State** upon reception of a **RESPONSE Message**, once checked via the Access Control interface, from a provider.

3.5.4.8.4.4 Effect on Reception

a) Reception of a **RESPONSE Request** shall, once checked via the Access Control interface, cause the provider MAL to transmit the supplied response as a **RESPONSE Message** to the consumer.

b) The pattern shall end at this point for the provider.

c) On reception of a **RESPONSE Indication** a consumer shall end the INVOKE Interaction with success.

3.5.4.8.4.5 Message Header

a) For the **RESPONSE Message** the message header fields shall be as defined in 3.4 except for the fields in table 3-17.

b) The contents of the Message Body, as specified in the operation template, shall immediately follow the message header.

Table 3-17: Invoke RESPONSE Message Header Fields

Field	Value
~~URI~~ From	Provider ~~URI~~
Authentication Id	Provider Authentication Identifier
~~URI~~ To	Consumer ~~URI~~
Interaction Type	INVOKE
Interaction Stage	3
Transaction Id	Transaction Id from initial message

3.5.4.8.5 RESPONSE_ERROR

3.5.4.8.5.1 Function

a) The **RESPONSE_ERROR Request** primitive shall be used by the provider application to end an INVOKE Interaction with an error.

b) The **RESPONSE_ERROR Indication** primitive shall be used by the consumer MAL to deliver a **RESPONSE_ERROR Message** to a consumer.

3.5.4.8.5.2 Semantics

RESPONSE_ERROR Request and **RESPONSE_ERROR Indication** shall provide parameters as follows:

(RESPONSE_ERROR Message Header, Error Number, Extra Information, QoS properties)

3.5.4.8.5.3 When Generated

a) A **RESPONSE_ERROR Request** shall be used by the provider application with the provider MAL in the **Acknowledged State** to transmit an error.

b) A **RESPONSE_ERROR Indication** shall be generated by the consumer MAL in the **Acknowledged State** upon one of two events:

1) the reception of a **RESPONSE_ERROR Message**, once checked via the Access Control interface, from a provider;

2) an error raised by the local communication layer.

3.5.4.8.5.4 Effect on Reception

a) Reception of a **RESPONSE_ERROR Request** shall, once checked via the Access Control interface, cause the provider MAL to transmit a **RESPONSE_ERROR Message** to the consumer.

b) The pattern shall end at this point for the provider.

c) On reception of a **RESPONSE_ERROR Indication** a consumer shall end the interaction with an error.

3.5.4.8.5.5 Message Header

a) For the **RESPONSE_ERROR Message** the message header fields shall be the same as for the **RESPONSE Message** except for the 'Is Error Message' field which is set to TRUE.

b) The Error information shall immediately follow the message header.

3.5.4.9 Example

The following example shows a simple example service that contains a single INVOKE operation. The operation sends a TestBody structure and returns a TestAck acknowledgment structure followed by a TestResponse structure:

Operation Identifier	testInvoke	
Interaction Pattern	INVOKE	
Pattern Sequence	Message	Body ~~Type~~Signature
IN	INVOKE	TestBody body
OUT	ACK	TestAck ack
OUT	RESPONSE	TestResponse response

The TestBody structure is defined in 3.5.1.9 and the TestResponse structure is defined in 3.5.3.9. The TestAck structure is defined below:

Name	TestAck		
Extends	Composite		
Short Form Part	Example Only		
Field	Type	Nullable	Comment
AckId	Identifier	Yes	Example Identifier item

This corresponds to the following message being transmitted:

Message Header																Test Body	
URI From	Auth Id	URI To	Timestamp	QoS	Priority	Domain	Zone	Session	Interaction	Stage	Trans Id	Area	Service	Operation	Version	Is Error	First Item	Second Item
Consumer	Op A	Provider		BEST	1	A.B.C	GROUND	LIVE	INVOKE	1	1236	Example	Example	testInvoke	1	FALSE	Hello	1234

Message Header											Test Body	
From	Auth Id	To	Timestamp	Interaction Type	Interaction Stage	Trans. Id	Area	Service	Operation	Version	Is Error	First Item	Second Item
Consumer	Op A	Provider		INVOKE	1	123	Example	Example	testInvoke	1	FALSE	Hello	1

And corresponds to the following acknowledgement being returned:

Message Header																	Test Ack
URI From	Auth Id	URI To	Timestamp	QoS	Priority	Domain	Zone	Session	Interaction	Stage	Trans Id	Area	Service	Operation	Version	Is Error	Ack Id
Provider	SC X	Consumer		BEST	1	A.B.C	GROUND	LIVE	INVOKE	2	1236	Example	Example	testInvoke	1	FALSE	1234

Message Header											TestAck
From	Auth Id	To	Timestamp	Interaction Type	Interaction Stage	Trans. Id	Area	Service	Operation	Version	Is Error	Ack Id
Provider	SC X	Consumer		INVOKE	2	123	Example	Example	testInvoke	1	FALSE	123

And finally corresponds to the following response being returned:

Message Header																	Test Response	
URI From	Auth Id	URI To	Timestamp	QoS	Priority	Domain	Zone	Session	Interaction	Stage	Trans Id	Area	Service	Operation	Version	Is Error	Rspn Item	Rspn Field
Provider	SC X	Consumer		BEST	1	A.B.C	GROUND	LIVE	INVOKE	3	1236	Example	Example	testInvoke	1	FALSE	True	31.0

Message Header											Test Body	
From	Auth Id	To	Timestamp	Interaction Type	Interaction Stage	Trans. Id	Area	Service	Operation	Version	Is Error	Rspn Item	Rspn Field
Provider	SC X	Consumer		INVOKE	3	123	Example	Example	testInvoke	1	FALSE	TRUE	31.0

NOTE – An actual service specification, rather than the example given here, would fully specify the meaning of, and required values of, all structures used by the service.

3.5.5 PROGRESS INTERACTION PATTERN

3.5.5.1 Overview

The PROGRESS pattern extends the INVOKE pattern to add a set of ~~mandatory~~ progress updates of the request message (figure 3-14). This allows the operation to confirm the receipt of the request before proceeding to process the request and also to show progress of the operation.

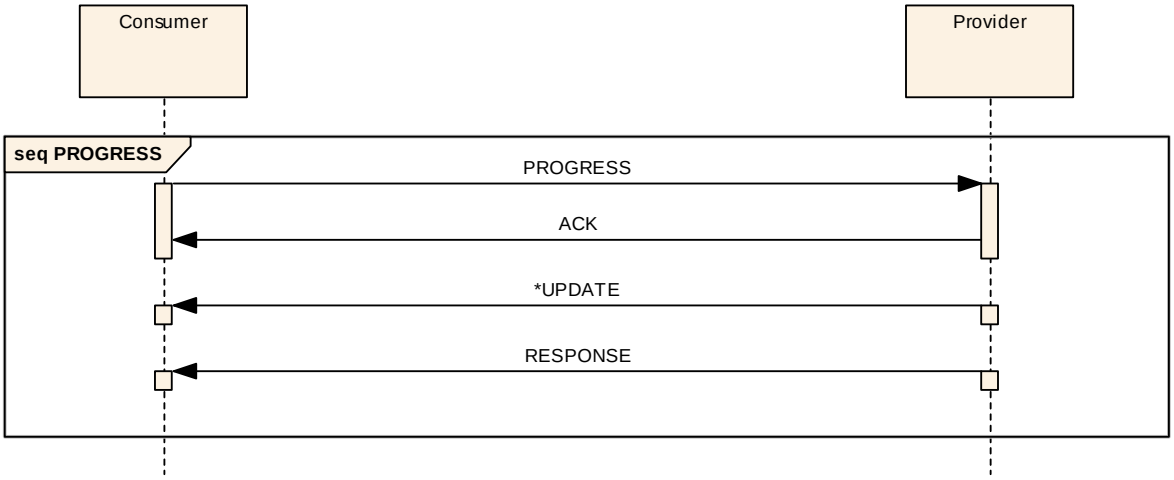


Figure 3-14: PROGRESS Interaction Pattern Message Sequence

3.5.5.2 Description

The PROGRESS pattern extends the INVOKE pattern with the addition of a set of mandatory progress messages. The type of progress messages and their number is defined by the service and not by the pattern.

3.5.5.3 Usage

The PROGRESS pattern is expected to be used when there is a significant or indeterminate amount of time taken to process the request and produce the data response, and where monitoring of the progress of the operation is required or the data response is to be returned in blocks. The order of the update messages is not guaranteed by the MAL; thus it is delegated to the Transport layer on an implementation-specific basis.

3.5.5.4 Error Handling

The PROGRESS pattern can report failure in three distinct ways:

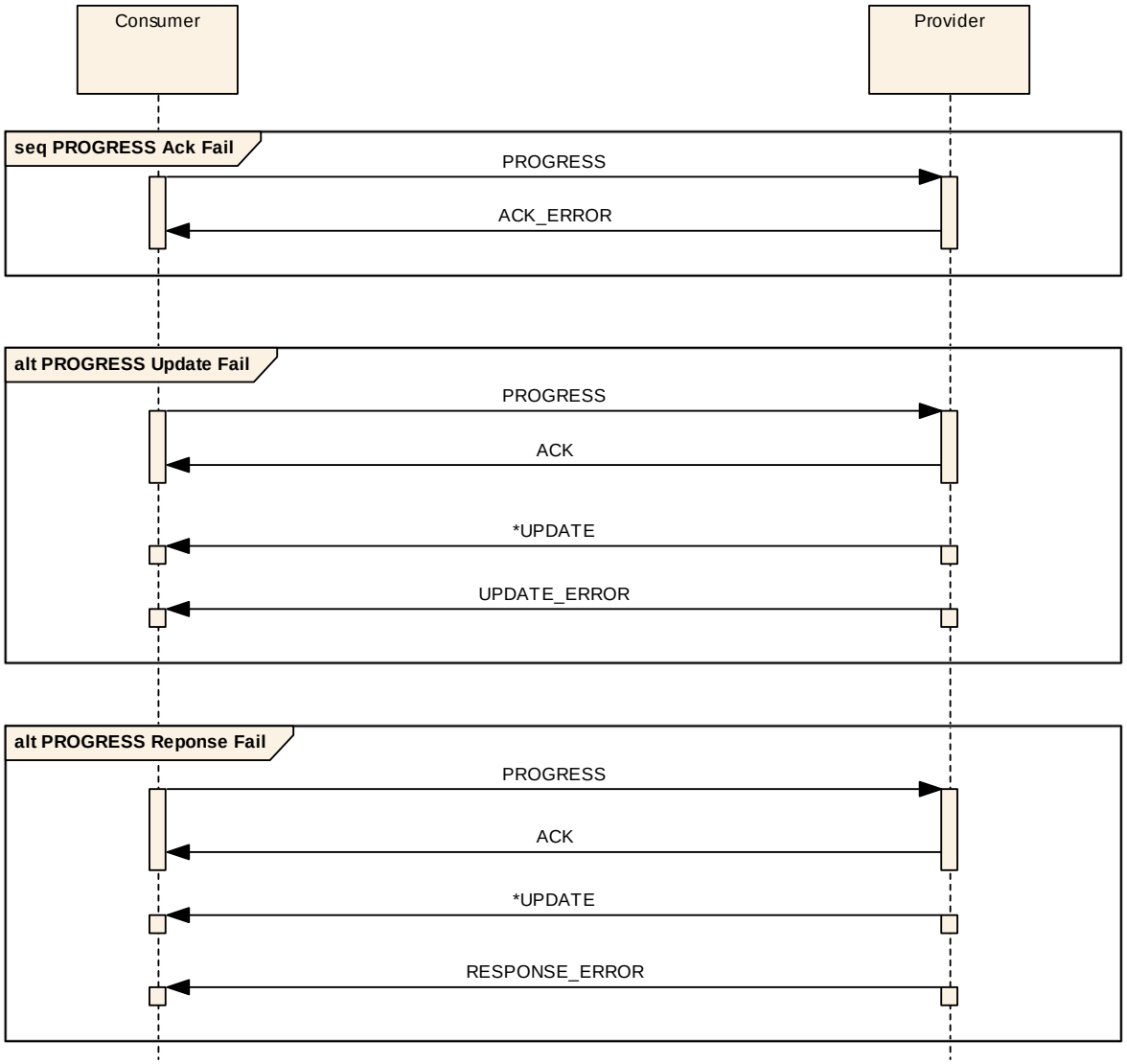


Figure 3-15: PROGRESS Interaction Pattern Error Sequence

a) The acknowledgement may be replaced with an error message (see section 5) shown in the first sequence in figure 3-15.

b) An error may be generated during the processing of the body of the operation, after the initial acknowledgement is sent, as shown in the second sequence in figure 3-15 and replaces any of the progress updates.

c) An error may be generated after the updates and replace the final response as shown in the third sequence in figure 3-15.

d) After an error message is sent, no further messages shall be generated as part of the pattern.

3.5.5.5 Operation Template

The PROGRESS template is similar to the INVOKE pattern template but adds a progress update message:

Table 3-18: PROGRESS Operation Template

Operation Identifier	<<Operation name>>	
Interaction Pattern	PROGRESS	
Pattern Sequence	Message	Body ~~Type~~Signature
IN	PROGRESS	<<Request type>>
OUT	ACK	<<Ack type>>
OUT	UPDATE	<<Update value type>>
OUT	RESPONSE	<<Response type>>

3.5.5.6 Primitives

Table 3-19: PROGRESS Primitive List

Primitive
PROGRESS
ACK
ACK_ERROR
UPDATE
UPDATE_ERROR
RESPONSE
RESPONSE_ERROR

3.5.5.7 State Charts

3.5.5.7.1 Consumer Side

Figure 3-16 shows the consumer side state chart for the PROGRESS pattern:

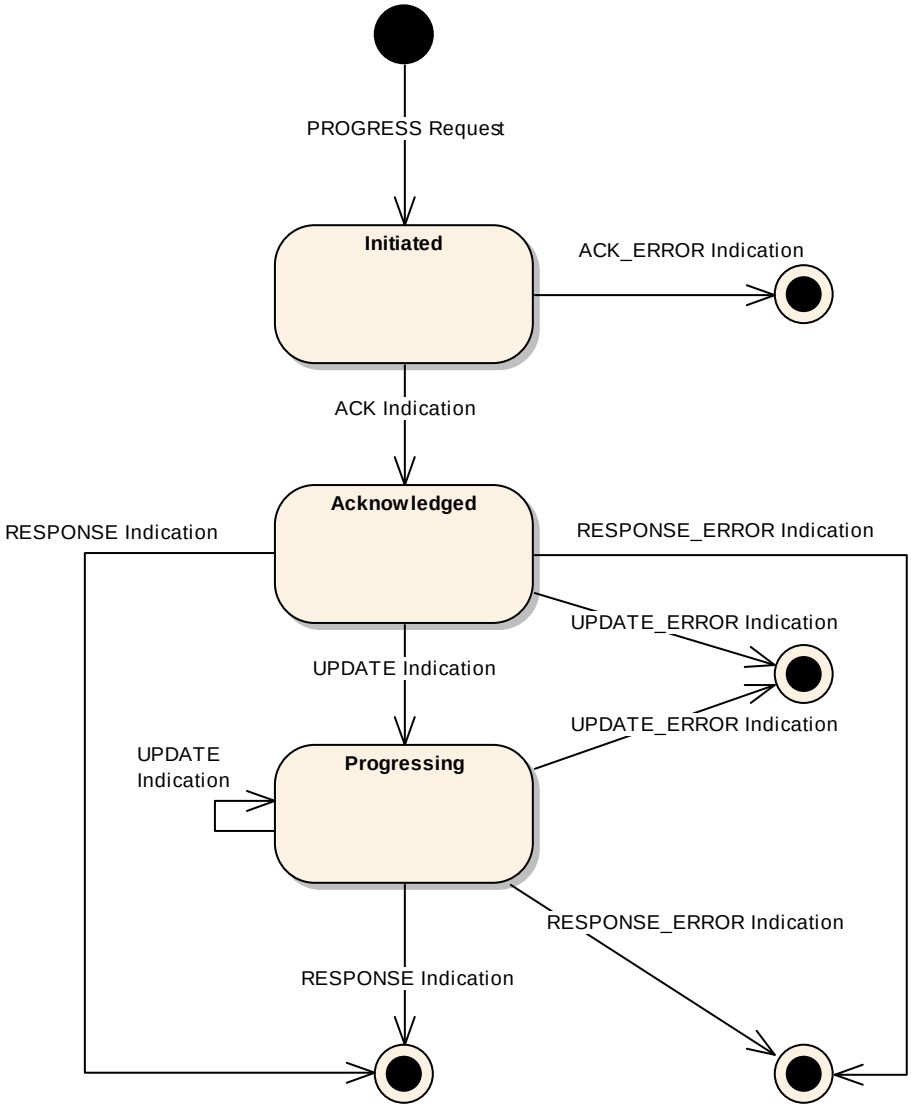


Figure 3-16: PROGRESS Consumer State Chart

a) The initial transition is triggered by the primitive **PROGRESS Request** and shall lead to the **Initiated State**.

b) There are two possible transitions from the **Initiated State**:

   1) the first is the indication of an acknowledgement, **ACK Indication**, and shall lead to the **Acknowledged State**;

   2) the second is the indication of an error, **ACK_ERROR Indication**, and shall lead to the final state.

c) There are four possible transitions from the **Acknowledged State**:

   1) the first is the indication of an update, **UPDATE Indication**, and shall lead to the **Progressing State**;

2) the second is the indication of an error in generating an initial update, **UPDATE_ERROR Indication**, and shall lead to the final state;

3) the third is the indication of a response, **RESPONSE Indication**, and shall lead to the final state;

4) the fourth is the indication of an error in generating the response, **RESPONSE_ERROR Indication**, and shall lead to the final state.

d) There are four possible transitions from the **Progressing State**:

1) the first is the indication of another update, **UPDATE Indication**, and shall lead back to the **Progressing State**;

2) the second is the indication of an error in generating an update, **UPDATE_ERROR Indication**, and shall lead to the final state;

3) the third is the indication of a response, **RESPONSE Indication**, and shall lead to the final state;

4) the fourth is the indication of an error in generating the response, **RESPONSE_ERROR Indication**, and shall lead to the final state.

3.5.5.7.2 Provider Side

Figure 3-17 shows the provider side state chart for the PROGRESS pattern:

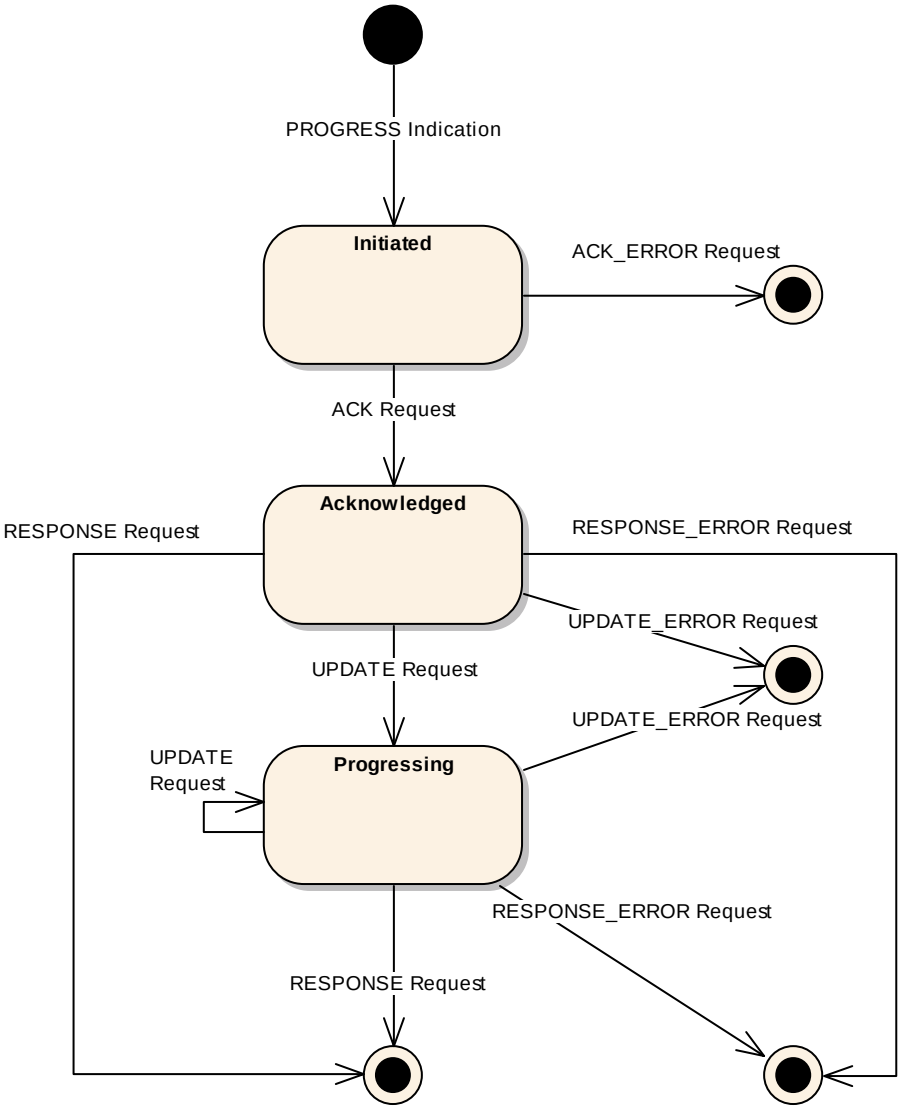


Figure 3-17: PROGRESS Provider State Chart

a) The initial transition is triggered by the primitive **PROGRESS Indication** and shall lead to the **Initiated State**.

b) There are two possible transitions from the **Initiated State**:

   1) the first is the transmission of an acknowledgement, **ACK Request**, and shall lead to the **Acknowledged State**;

   2) the second is the transmission of an error, **ACK_ERROR Request**, and shall lead to the final state.

c) There are four possible transitions from the **Acknowledged State**:

1) the first is the transmission of an update, **UPDATE Request**, and shall lead to the **Progressing State**;

2) the second is the transmission of an error in generating an initial update, **UPDATE_ERROR Request**, and shall lead to the final state;

3) the third is the transmission of a response, **RESPONSE Request**, and shall lead to the final state;

4) the fourth is the transmission of an error in generating the response, **RESPONSE_ERROR Request**, and shall lead to the final state.

d) There are four possible transitions from the **Progressing State**:

1) the first is the transmission of another update, **UPDATE Request**, and shall lead back to the **Progressing State**;

2) the second is the transmission of an error in generating an update, **UPDATE_ERROR Request**, and shall lead to the final state;

3) the third is the transmission of a response, **RESPONSE Request**, and shall lead to the final state;

4) the fourth is the transmission of an error in generating the response, **RESPONSE_ERROR Request**, and shall lead to the final state.

3.5.5.8 Requests and Indications

3.5.5.8.1 PROGRESS

3.5.5.8.1.1 Function

a) The **PROGRESS Request** primitive shall be used by the consumer application to initiate a PROGRESS Interaction.

b) The **PROGRESS Indication** primitive shall be used by the provider MAL to deliver a **PROGRESS Message** to a provider and initiate a PROGRESS Interaction.

3.5.5.8.1.2 Semantics

PROGRESS Request and **PROGRESS Indication** shall provide parameters as follows:

(PROGRESS Message Header, PROGRESS Message Body, QoS properties)

3.5.5.8.1.3 When Generated

a) A **PROGRESS Request** may be generated by the consumer application at any time.

b) A **PROGRESS Indication** shall be generated by the provider MAL upon reception of a **PROGRESS Message**, once checked via the Access Control interface, from a consumer.

3.5.5.8.1.4 Effect on Reception

a) Reception of a **PROGRESS Request** shall, once checked via the Access Control interface, cause the consumer MAL to initiate a PROGRESS Interaction by transmitting a **PROGRESS Message** to the provider.

b) The consumer MAL shall enter the **Initiated State**.

c) On reception of a **PROGRESS Indication** by a provider, the provider MAL shall enter the **Initiated State**.

d) The provider shall then start processing the operation at this point.

3.5.5.8.1.5 Message Header

a) For the **PROGRESS Message** the message header fields shall be as defined in 3.4 except for the fields in table 3-20.

b) The contents of the Message Body, as specified in the operation template, shall immediately follow the message header.

Table 3-20: PROGRESS Message Header Fields

Field	Value
~~URI~~ From	Consumer ~~URI~~
Authentication Id	Consumer Authentication Identifier
~~URI~~ To	Provider ~~URI~~
Interaction Type	PROGRESS
Interaction Stage	1
Transaction Id	Provided by consumer MAL

3.5.5.8.2 ACK

3.5.5.8.2.1 Function

a) The **ACK Request** primitive shall be used by the provider application to acknowledge a PROGRESS Interaction.

b) The **ACK Indication** primitive shall be used by the consumer MAL to deliver an **ACK Message** to a consumer.

3.5.5.8.2.2 Semantics

ACK Request and **ACK Indication** shall provide parameters as follows:

(ACK Message Header, **A**CK Message Body, QoS properties)

3.5.5.8.2.3 When Generated

a) An **ACK Request** shall be used by the provider application with the provider MAL in the **Initiated State** to acknowledge a PROGRESS Interaction.

b) An **ACK Indication** shall be generated by the consumer MAL in the **Initiated State** upon reception of an **ACK Message**, once checked via the Access Control interface, from a provider.

3.5.5.8.2.4 Effect on Reception

a) Reception of an **ACK Request** shall, once checked via the Access Control interface, cause the provider MAL to transmit the supplied acknowledgement as an **ACK Message** to the consumer.

b) A provider MAL in the **Initiated State** shall enter the **Acknowledged State**.

c) On reception of an **ACK Indication** by the consumer, the consumer MAL shall enter the **Acknowledged State**.

3.5.5.8.2.5 Message Header

a) For the **ACK Message** the message header fields shall be as defined in 3.4 except for the fields in table 3-21.

b) The contents of the Message Body, as specified in the operation template, shall immediately follow the message header.

Table 3-21: Progress ACK Message Header Fields

Field	Value
~~URI~~ From	Provider ~~URI~~
Authentication Id	Provider Authentication Identifier
~~URI~~ To	Consumer ~~URI~~
Interaction Type	PROGRESS
Interaction Stage	2
Transaction Id	Transaction Id from initial message

3.5.5.8.3 ACK_ERROR

3.5.5.8.3.1 Function

a) The **ACK_ERROR Request** primitive shall be used by the provider application to end a PROGRESS Interaction with an error.

b) The **ACK_ERROR Indication** primitive shall be used by the consumer MAL to deliver an **ACK_ERROR Message** to a consumer.

3.5.5.8.3.2 Semantics

ACK_ERROR Request and **ACK_ERROR Indication** shall provide parameters as follows:

(ACK_ERROR Message Header, Error Number, Extra Information, QoS properties)

3.5.5.8.3.3 When Generated

a) An **ACK_ERROR Request** shall be used by the provider application with the provider MAL in the **Initiated State** to transmit an error.

b) An **ACK_ERROR Indication** shall be generated by the consumer MAL in the **Initiated State** upon one of two events:

1) the reception of an **ACK_ERROR Message**, once checked via the Access Control interface, from a provider;

2) an error raised by the local communication layer.

3.5.5.8.3.4 Effect on Reception

a) Reception of an **ACK_ERROR Request** shall, once checked via the Access Control interface, cause the provider MAL to transmit an **ACK_ERROR Message** to the consumer.

b) The pattern shall end at this point for the provider.

c) On reception of an **ACK_ERROR Indication** a consumer shall end the interaction with an error.

3.5.5.8.3.5 Message Header

a) For the **ACK_ERROR Message** the message header fields shall be the same as for the **ACK Message** except for the 'Is Error Message' field which is set to TRUE.

b) The Error information shall immediately follow the message header.

3.5.5.8.4 UPDATE

3.5.5.8.4.1 Function

a) The **UPDATE Request** primitive shall be used by the provider application to transmit an update during a PROGRESS Interaction.

b) The **UPDATE Indication** primitive shall be used by the consumer MAL to deliver an **UPDATE Message** to a consumer.

3.5.5.8.4.2 Semantics

UPDATE Request and **UPDATE Indication** shall provide parameters as follows:

(UPDATE Message Header, UPDATE Message Body, QoS properties)

3.5.5.8.4.3 When Generated

a) An **UPDATE Request** shall be used by the provider application with the provider MAL in either the **Acknowledged State** or **Progressing State** to transmit an update to a PROGRESS interaction.

b) An **UPDATE Indication** shall be generated by the consumer MAL in either the **Acknowledged State** or the **Progressing State** upon reception of an **UPDATE Message**, once checked via the Access Control interface, from a provider.

3.5.5.8.4.4 Effect on Reception

a) Reception of an **UPDATE Request** shall, once checked via the Access Control interface, cause the provider MAL to transmit the supplied update as an **UPDATE Message** to the consumer.

b) A provider MAL in the **Acknowledged State** shall enter the **Progressing State**.

c) On reception of an **UPDATE Indication** by the consumer, the consumer MAL in the **Acknowledged State** shall enter the **Progressing State**.

3.5.5.8.4.5 Message Header

a) For the **UPDATE Message**, of which there may be several, the message header fields shall be as defined in 3.4 except for the fields in table 3-22.

b) The contents of the Message Body, as specified in the operation template, shall immediately follow the message header.

Table 3-22: Progress UPDATE Message Header Fields

Field	Value
~~URI~~ From	Provider ~~URI~~
Authentication Id	Provider Authentication Identifier
~~URI~~ To	Consumer ~~URI~~
Interaction Type	PROGRESS
Interaction Stage	3
Transaction Id	Transaction Id from initial message

3.5.5.8.5 UPDATE_ERROR

3.5.5.8.5.1 Function

a) The **UPDATE_ERROR Request** primitive shall be used by the provider application to end a PROGRESS Interaction with an error.

b) The **UPDATE_ERROR Indication** primitive shall be used by the consumer MAL to deliver an **UPDATE_ERROR Message** to a consumer.

3.5.5.8.5.2 Semantics

UPDATE_ERROR Request and **UPDATE_ERROR Indication** shall provide parameters as follows:

(UPDATE_ERROR Message Header, Error Number, Extra Information, QoS properties)

3.5.5.8.5.3 When Generated

a) An **UPDATE_ERROR Request** shall be used by the provider application with the provider MAL in either the **Acknowledged State** or the **Progressing State** to transmit an error.

b) An **UPDATE_ERROR Indication** shall be generated by the consumer MAL in either the **Acknowledged State** or the **Progressing State** upon one of two events:

1) the reception of an **UPDATE_ERROR Message**, once checked via the Access Control interface, from a provider;

2) an error raised by the local communication layer.

3.5.5.8.5.4 Effect on Reception

a) Reception of an **UPDATE_ERROR Request** shall, once checked via the Access Control interface, cause the provider MAL to transmit an **UPDATE_ERROR Message** to the consumer.

b) The pattern shall end at this point for the provider.

c) On reception of an **UPDATE_ERROR Indication** a consumer shall end the interaction with an error.

3.5.5.8.5.5 Message Header

a) For the **UPDATE_ERROR Message** the message header fields shall be the same as for the **UPDATE Message** except for the 'Is Error Message' field which is set to TRUE.

b) The Error information shall immediately follow the message header.

3.5.5.8.6 RESPONSE

3.5.5.8.6.1 Function

a) The **RESPONSE Request** primitive shall be used by the provider application to transmit a response to a PROGRESS Interaction.

b) The **RESPONSE Indication** primitive shall be used by the consumer MAL to deliver a **RESPONSE Message** to a consumer.

3.5.5.8.6.2 Semantics

RESPONSE Request and **RESPONSE Indication** shall provide parameters as follows:

(RESPONSE Message Header, RESPONSE Message Body, QoS properties)

3.5.5.8.6.3 When Generated

a) A **RESPONSE Request** shall be used by the provider application with the provider MAL in either the **Acknowledged State** or **Progressing State** to transmit a final response to a PROGRESS interaction.

b) A **RESPONSE Indication** shall be generated by the consumer MAL in either the **Acknowledged State** or the **Progressing State** upon reception of a **RESPONSE Message**, once checked via the Access Control interface, from a provider.

3.5.5.8.6.4 Effect on Reception

a) Reception of a **RESPONSE Request** shall, once checked via the Access Control interface, cause the provider MAL to transmit the supplied response as a **RESPONSE Message** to the consumer.

b) The pattern shall end at this point for the provider.

c) On reception of a **RESPONSE Indication** a consumer shall end the PROGRESS Interaction with success.

3.5.5.8.6.5 Message Header

a) For the **RESPONSE Message** the message header fields shall be as defined in 3.4 except for the fields in table 3-23.

b) The contents of the Message Body, as specified in the operation template, shall immediately follow the message header.

Table 3-23: Progress RESPONSE Message Header Fields

Field	Value
~~URI~~ From	Provider ~~URI~~
Authentication Id	Provider Authentication Identifier
~~URI~~ To	Consumer ~~URI~~
Interaction Type	PROGRESS
Interaction Stage	4
Transaction Id	Transaction Id from initial message

3.5.5.8.7 RESPONSE_ERROR

3.5.5.8.7.1 Function

a) The **RESPONSE_ERROR Request** primitive shall be used by the provider application to end a PROGRESS Interaction with an error.

b) The **RESPONSE_ERROR Indication** primitive shall be used by the consumer MAL to deliver a **RESPONSE_ERROR Message** to a consumer.

3.5.5.8.7.2 Semantics

RESPONSE_ERROR Request and **RESPONSE_ERROR Indication** shall provide parameters as follows:

(RESPONSE_ERROR Message Header, Error Number, Extra Information, QoS properties)

3.5.5.8.7.3 When Generated

a) A **RESPONSE_ERROR Request** shall be used by the provider application with the provider MAL in either the **Acknowledged State** or **Progressing State** to transmit an error.

b) A **RESPONSE_ERROR Indication** shall be generated by the consumer MAL in either the **Acknowledged State** or the **Progressing State** upon one of two events:

   1) the reception of a **RESPONSE_ERROR Message**, once checked via the Access Control interface, from a provider;

   2) an error raised by the local communication layer.

3.5.5.8.7.4 Effect on Reception

a) Reception of a **RESPONSE_ERROR Request** shall, once checked via the Access Control interface, cause the provider MAL to transmit a **RESPONSE_ERROR Message** to the consumer.

b) The pattern shall end at this point for the provider.

c) On reception of a **RESPONSE_ERROR Indication** a consumer shall end the interaction with an error.

3.5.5.8.7.5 Message Header

a) For the **RESPONSE_ERROR Message** the message header fields shall be the same as for the **RESPONSE Message** except for the ‘Is Error Message’ field which is set to TRUE.

b) The Error information shall immediately follow the message header.

3.5.5.9 Example

The following example shows a simple example service that contains a single PROGRESS operation. The operation sends a TestBody structure, which is acknowledged with a TestAck structure, reports progress with a TestProgress structure, and returns a TestResponse structure:

Operation Identifier	testProgress	
Interaction Pattern	PROGRESS	
Pattern Sequence	Message	Body ~~Type~~Signature
IN	PROGRESS	TestBody body
OUT	ACK	TestAck ack
OUT	UPDATE	TestProgress progress
OUT	RESPONSE	TestResponse response

The TestBody structure is defined in 3.5.1.9, TestAck structure is defined in 3.5.4.9, and the TestResponse structure is defined in 3.5.3.9. The TestProgress structure is defined below:

Name	TestProgress		
Extends	Composite		
Short Form Part	Example Only		
Field	Type	Nullable	Comment
RspnStage	Integer	Yes	Example Integer item

This corresponds to the following message being transmitted:

Message Header																Test Body	
URI From	Auth Id	URI To	Timestamp	Domain	QoS	Priority	Zone	Session	Interaction	Stage	Trans Id	Area	Service	Operation	Version	Is Error	First Item	Second Item
Consumer	Op A	Provider		A.B.C	BEST	1	GROUND	LIVE	PROGRESS	1	1237	Example	Example	testProg	1	FALSE	Hello	1234

Message Header												Test Body	
From	Auth Id	To	Timestamp	Interaction Type	Interaction Stage	Trans. Id	Area	Service	Operation	Version	Is Error	First Item	Second Item
Consumer	Op A	Provider		PROGRESS	1	123	Example	Example	testProg.	1	FALSE	Hello	1

And corresponds to the following acknowledgement being returned:

Message Header																Test Ack
URI From	Auth Id	URI To	Timestamp	Domain	QoS	Priority	Zone	Session	Interaction	Stage	Trans Id	Area	Service	Operation	Version	Is Error	Ack Id
Provider	SC X	Consumer		A.B.C	BEST	1	GROUND	LIVE	PROGRESS	2	1237	Example	Example	testProg	1	FALSE	1234

Message Header											TestAck
From	Auth Id	To	Timestamp	Interaction Type	Interaction Stage	Trans. Id	Area	Service	Operation	Version	Is Error	Ack Id
Provider	SC X	Consumer		PROGRESS	2	123	Example	Example	testProg.	1	FALSE	1234

Progress is reported using the following message; in this example two stages are reported:

Message Header																Test Progress
URI From	Auth Id	URI To	Timestamp	Domain	QoS	Priority	Zone	Session	Interaction	Stage	Trans Id	Area	Service	Operation	Version	Is Error	Rspn Stage
Provider	SC X	Consumer		A.B.C	BEST	1	GROUND	LIVE	PROGRESS	3	1237	Example	Example	testProg	1	FALSE	1

Message Header																Test Progress
URI From	Auth Id	URI To	Timestamp	Domain	QoS	Priority	Zone	Session	Interaction	Stage	Trans Id	Area	Service	Operation	Version	Is Error	Rspn Stage
Provider	SC X	Consumer		A.B.C	BEST	1	GROUND	LIVE	PROGRESS	3	1237	Example	Example	testProg	1	FALSE	2

Message Header											Test Body
From	Auth Id	To	Timestamp	Interaction Type	Interaction Stage	Trans. Id	Area	Service	Operation	Version	Is Error	Rspn Item
Provider	SC X	Consumer		PROGRESS	3	123	Example	Example	testProg.	1	FALSE	1

Message Header											Test Body
From	Auth Id	To	Timestamp	Interaction Type	Interaction Stage	Trans. Id	Area	Service	Operation	Version	Is Error	Rspn Item
Provider	SC X	Consumer		PROGRESS	3	123	Example	Example	testProg.	1	FALSE	2

And finally corresponds to the following response being returned:

Message Header																Test Response	
URI From	Auth Id	URI To	Timestamp	Domain	QoS	Priority	Zone	Session	Interaction	Stage	Trans Id	Area	Service	Operation	Version	Is Error	Rspn Item	Rspn Field
Provider	SC X	Consumer		A.B.C	BEST	1	GROUND	LIVE	PROGRESS	4	1237	Example	Example	testProg	1	FALSE	True	31.0

Message Header											Test Body	
From	Auth Id	To	Timestamp	Interaction Type	Interaction Stage	Trans. Id	Area	Service	Operation	Version	Is Error	Rspn Item	Rspn Field
Provider	SC X	Consumer		PROGRESS	4	123	Example	Example	testProg.	1	FALSE	TRUE	31.0

NOTE – An actual service specification, rather than the example given here, would fully specify the meaning of, and required values of, all structures used by the service.

3.5.6 PUBLISH-SUBSCRIBE INTERACTION PATTERN

3.5.6.1 Overview

The PUBLISH-SUBSCRIBE pattern provides the ability for consumers to register interest in a specific topic and receive updates from one or more providers without requiring awareness of the number of, and location of, these providers.

The basic outline of the pattern is:

a) consumers register interest by sending a subscription list;

b) providers ~~also~~ register on the ~~list of entities they provide~~broker;

c) providers publish updates;

d) updates are then sent to registered consumers; and

e) when no further updates are required the consumer cancels its subscription by deregistering.

The PUBLISH-SUBSCRIBE pattern can be deployed in two distinct ways; the first is where an intermediary shared message broker resides between the consumers and the providers (figure 3-18). In this deployment the shared broker permits a decoupling of the consumer from the providers, receiving the subscriptions from consumers and the updates from providers, and is responsible for dispatching the required updates to registered consumers:

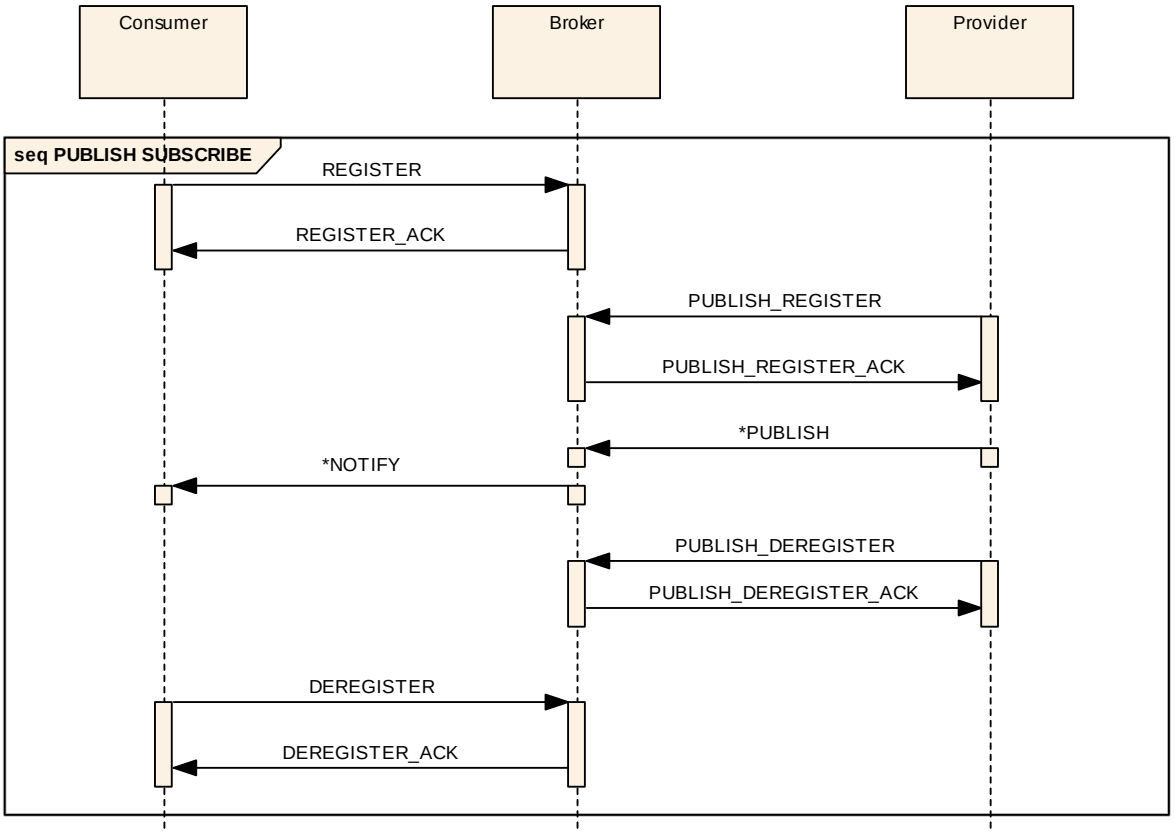


Figure 3-18: PUBLISH-SUBSCRIBE Interaction Pattern Message Sequence

The shared message broker manages the consumer subscription lists; the providers are not aware of what consumers there are and what entities they require. Because of this they must publish all items they are configured to generate to the broker so that it can manage the updates to the consumers.

At a later time, when updates are no longer required, the consumer can deregister for these updates from the message broker.

The broker acknowledges both the consumer and provider register/deregister operations; these are treated like normal SUBMIT messages. No acknowledgement of the publish message is provided by the broker and no acknowledgement by the consumer of the notify message is required.

The second deployment of the PUBLISH-SUBSCRIBE pattern is when there is a single update provider and the message broker is private to it. This is an example of an observe type relationship where a consumer observes some aspect of a specific provider.

In this deployment, because of the one-to-one relationship between the consumer and the provider, the functionality of the message broker does not necessarily require a separate entity. It is completely possible for the message broker logic to reside in the provider application; the provider publishes updates to itself for distribution to consumers (figure 3-19):

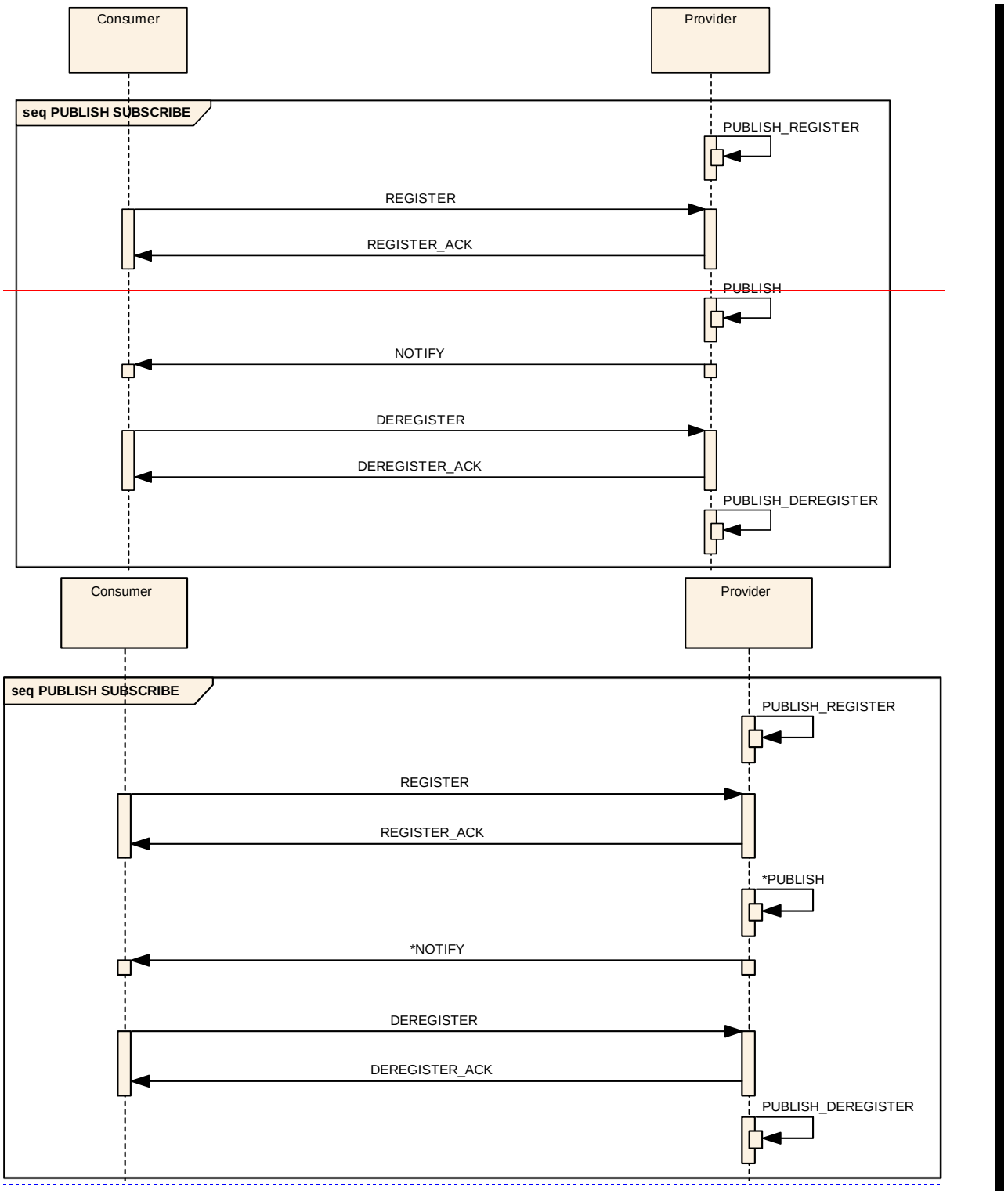


Figure 3-19: PUBLISH-SUBSCRIBE Pattern Alternative Message Sequence

However, because the pattern is still the same, all features of the first deployment (multiple concurrent subscription lists) are still supported.

In terms of implementation, for both the shared or private broker pattern, there are three primary implementation designs (others may exist, however). The first is where the broker logic resides in the MAL layer, the second is where it is natively supported by the message transport technology used, and the third is where it is implemented as a standalone component external to the MAL. In all three cases the pattern of interaction is the same; it is only the physical location of the broker that changes.

3.5.6.2 Description

The PUBLISH-SUBSCRIBE pattern for both the consumer and provider register/deregister has a predefined pattern message structure which allows an implementation of the message broker to manage the mapping of consumers to updates and hide this complexity from Provider applications.

3.5.6.3 Subscriptions

A consumer informs the broker of the updates to send it by the use of subscriptions. Subscriptions are sent during the registration.

a) A single consumer may define many subscription lists concurrently by specifying a different subscription identifier during the registration.

b) To change the contents of a subscription list a consumer may redefine it by registering a new list with the same identifier. The consumer is not required to deregister the subscription first.

c) For a broker the replacement of a subscription is an atomic operation; no updates shall be missed as a result of the subscription replacement operation.

d) The ~~URI of the consumer~~Consumer Entity name and the subscription identifier shall form the unique identifier of the subscription; this means that two consumers cannot share subscriptions, as they have different ~~consumer URIs~~Consumer entity names.

e) The consumer deregister message takes a list of identifiers of the subscription lists to cancel and shall completely remove the subscriptions from the list of active subscriptions.

f) The update rate shall be an aspect of a provider; the consumer cannot specify it.

For example, if a consumer initially registers for updates on the set (A, B, C) but at a later date requires also ‘D’, another register message is required to be sent with the set (A, B, C, D). The register call replaces the previous subscription list with the new list. The deregister message completely cancels the identified lists.

Another way of describing this is the register message defines an update ‘report’ and the deregister message clears that ‘report’ definition.

NOTE – If a single subscription definition lists the same parameter more than once, only one update is sent to the consumer for that list. However, if a consumer defines two concurrent subscription lists that request the same parameters, then the consumer will receive two updates of those common parameters, one for each subscription. If this is an issue, then the consumer still has the option of maintaining a single subscription and splitting the results internally.

~~3.5.6.4 Entity Identification~~

~~The identity of an entity is composed of two parts: specific fields from the message header and an Entity Key (see 4.4.7 for the type definition):~~

~~a) The domain, session type and name, area, service, and operation identifiers of the message header shall form the first part of the Entity identification.~~

~~b) The network identifier shall be ignored and does not form part of the Entity identification.~~

~~c) The Entity Key is a composite structure that contains four sub-keys and shall form the second part of the Entity identification.~~

~~d) Entities shall not use the '*' value for the first sub-key as this is reserved for the wildcard value (see 3.5.6.5).~~

~~e) Entities shall not use the zero value for the second, third, and fourth sub-keys as this is reserved for the wildcard value (see 3.5.6.5).~~

~~f) The combination of the message header fields and the Entity Key in a message shall form the complete Entity identification.~~

~~3.5.6.5 Subscription Matching~~

~~This subsection details the rules for subscription matching in the PUBLISH-SUBSCRIBE pattern. The subscription is matched on three aspects: the Entity Key, the optional subDomain field of the Entity Request, and the message header fields.~~

~~The following paragraphs detail the rules for Entity Key matching:~~

~~a) A sub-key specified in the EntityKey structure shall take one of three types of value: an actual value, a NULL value, and the special wildcard value of '*' (for the first sub-key only) or zero (for the other three sub-keys).~~

~~b) If a sub-key contains a specific value it shall only match a sub-key that contains the same value. This includes an empty '' value for the first sub-key. The matches are case sensitive.~~

~~c) If a sub-key contains a NULL value it shall only match a sub-key that contains NULL.~~

d) If a sub-key contains the wildcard value it shall match a sub-key that contains any value including NULL.

For example, suppose a provider is publishing a set of updates with the following keys (the dot notation is used to separate the sub-keys; all four sub-keys must be provided):

1) A.[null].[null].[null]

2) A.2.[null].[null]

3) A.2.3.[null]

4) A.2.3.4

5) B.[null].[null].[null]

6) Q.2.3.[null]

An EntityRequest using the key of 'A.[null].[null].[null]' would only match to the first update.

An EntityRequest using the key of 'A.0.[null].[null]' would only match to the first and second updates.

An EntityRequest using the key of 'A.0.0.0' would match to all but the last two updates.

An EntityRequest using the key of 'A.2.[null].[null]' would only match the second update.

An EntityRequest using the key of 'A.2.0.[null]' would only match to updates 2 and 3.

An EntityRequest using the key of '*.2.0.[null]' would only match to updates 2, 3 and 6.

An EntityRequest using the key of 'B.0.0.0' would only match to update 5.

NOTE – The meaning of the entity key value is context-specific and must be detailed in the relevant service specification.

The following parapgraphs detail the rules for message header field matching of the subscription and update message:

e) The domain of the update message shall match the domain of the subscription message.

f) If the subscription EntityRequest included a subDomain field, then this shall be appended to the domain of the subscription message to make the complete domain for that request.

g) The final Identifier of the subDomain may be the wildcard character '*'.

h) The session types and names must match.

~~i) The network identifiers shall be ignored.~~

~~j) The area identifiers must match unless the subscription specified True in the allAreas field of the EntityRequest, in which case they shall be ignored.~~

~~k) The service identifiers must match unless the subscription specified True in the allServices field of the EntityRequest, in which case they shall be ignored.~~

~~l) The operation identifiers must match unless the subscription specified True in the allOperations field of the EntityRequest, in which case they shall be ignored.~~

~~For example, suppose a provider is publishing a set of updates with the following domains (the dot notation is used to separate the sub-domains):~~

~~1) spacecraftA~~

~~2) spacecraftA.aocs~~

~~3) spacecraftA.aocs.thrustA~~

~~4) spacecraftA.payload~~

~~5) spacecraftA.payload.cameraA.tempB~~

~~6) spacecraftB~~

~~7) agency.spacecraftA~~

~~Therefore a subscription message with the domain of 'spacecraftA' and a NULL subDomain field would only match the first update.~~

~~A subscription message with the domain of 'spacecraftA' and a subDomain field set to 'aocs' would only match the second update.~~

~~A subscription message with the domain of 'spacecraftA' and a subDomain field set to 'payload.*' would match the fourth and fifth updates.~~

~~A subscription message with the domain of 'spacecraftA' and a subDomain field set to '*' would match updates 1 to 5.~~

3.5.6.4 Published Messages Routing on the Broker

The routing of the published messages in the Broker is based on two parts, (1) specific fields from the message header and (2) the subscription key values:

a) The area, service, and operation identifiers of the message header shall form the first part to be matched.

b) The published subscription key values shall form the second part to be matched with the Subscription filters provided by the consumers.

c) The published subscription key shall not use NULL as a value as this is reserved for the wildcard value (see 3.5.6.5).

3.5.6.5 Subscription Filter Matching on the Broker

For subscription filter matching in the PUBLISH-SUBSCRIBE pattern, the subscription is matched on two aspects, (1) the optional domain field and (2) the optional filters (which are ANDed together):

a) The filters field shall hold a list of filters composed of a name (type: Identifier) and a set of possible values (type: List<MAL::Attribute>) to be matched.

b) Each filter shall hold as a value one of following two types:

1) a specific value, or

2) a NULL value that represents a wildcard.

c) If a filter contains a specific value it shall only match a subkey that contains the same value. This includes an empty ‘’ value for text-based data types. The matches are case sensitive.

d) If a filter contains the NULL wildcard value it shall match a subkey that contains any value including NULL.

If a publish message includes in the Extra field in the MAL Header an entry containing a ‘domain’ name with a respective value, the following rules for domain field matching of the subscription with the publish message shall be applied:

a) The domain of the update message shall match the domain of the subscription message.

b) If the subscription included a domain field, then this shall be matched to the domain of the update message.

c) Any of the Identifiers of the domain may be the wildcard character ‘*’.

For example, a provider might publish a set of updates with the following domains:

1) spacecraftA

2) spacecraftA.aocs

3) spacecraftA.aocs.thrustA

4) spacecraftA.payload

5) spacecraftA.payload.cameraA.tempB

6) spacecraftB

7) agency.spacecraftA

8) spacecraftB.payload.cameraA.tempB

A subscription filter with the domain of ‘spacecraftA’ would only match the first update.

A subscription filter with the domain of ‘spacecraftA.aocs’ would only match the second update.

A subscription filter with the domain of ‘spacecraftA.payload.*’ would match the fourth and fifth updates.

A subscription filter with the domain of ‘*.payload.cameraA.*’ would match the fifth and eighth updates.

A subscription filter with the domain of ‘spacecraftA.*’ would match updates 1 to 5.

3.5.6.6 Usage

The PUBLISH-SUBSCRIBE pattern provides the ability for asynchronous updates to be sent to registered consumers.

3.5.6.7 Error Handling

The PUBLISH-SUBSCRIBE pattern can report failure for a consumer in two distinct ways:

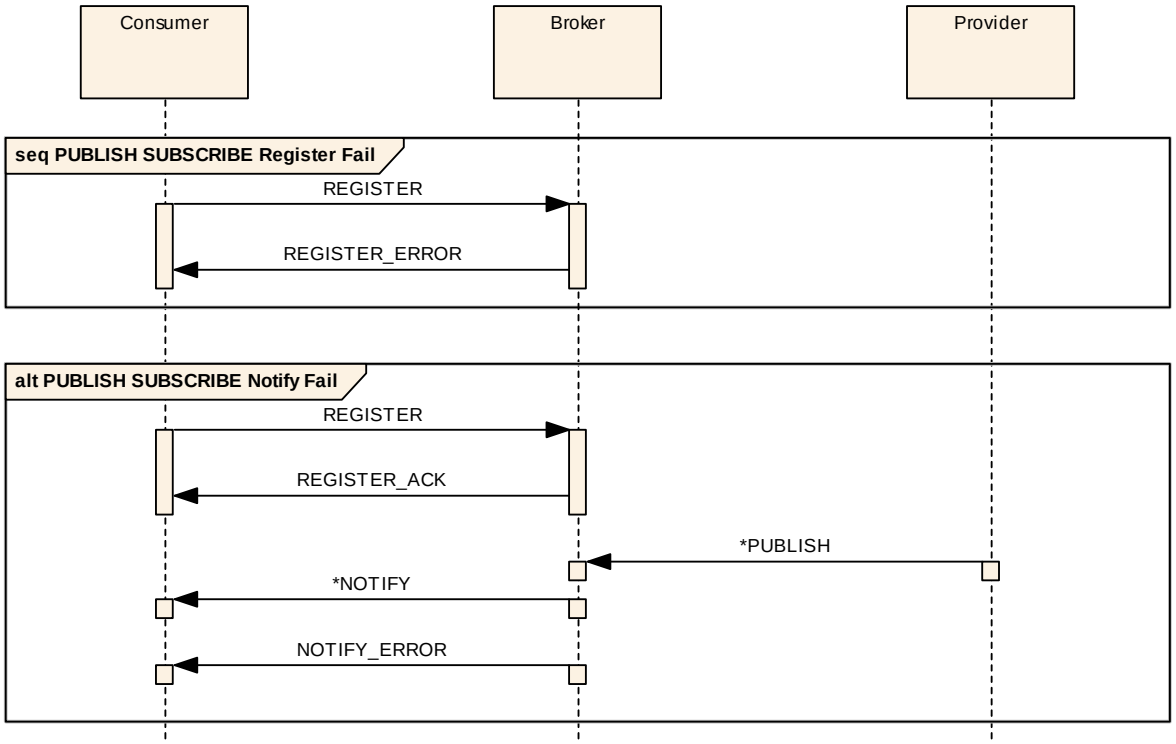


Figure 3-20: PUBLISH-SUBSCRIBE Interaction Pattern Consumer Error Sequence

a) The register acknowledgement may be replaced with an error message (see section 5) shown in the first sequence in figure 3-20.

b) A notify may be replaced by an error by the message broker shown in the second sequence in figure 3-20.

c) After an error message is sent no further messages shall be transmitted to the relevant consumer as part of the pattern.

For a publisher the PUBLISH-SUBSCRIBE pattern can report failure in two distinct ways:

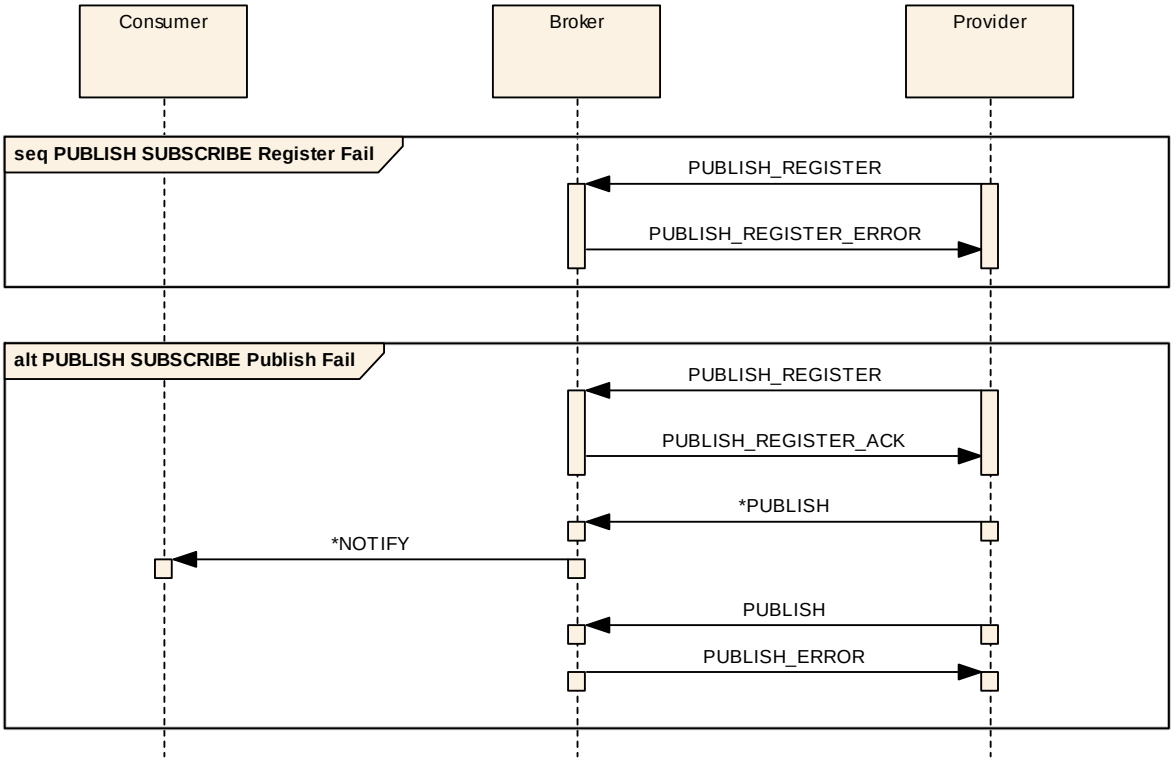


Figure 3-21: PUBLISH-SUBSCRIBE Interaction Pattern Provider Error Sequence

a) The publish register acknowledgement may be replaced with an error message (see section 5) shown in the first sequence in figure 3-21.

b) After a PUBLISH_REGISTER_ERROR message is sent no further messages shall be transmitted to or from the relevant publisher as part of the pattern.

c) When an error is detected by the message broker on reception of a PUBLISH message from a provider, a PUBLISH_ERROR message shall be sent to the provider as shown in the second sequence in figure 3-21. The error is sent asynchronously to the provider.

3.5.6.8 Operation Template

The PUBLISH-SUBSCRIBE template is below:

Table 3-24: PUBLISH-SUBSCRIBE Operation Template

Operation Identifier	<<Operation name>>	
Interaction Pattern	PUBLISH-SUBSCRIBE	
Subscription Keys	<<Subscription Key types and names>>	
Pattern Sequence	Message	Body ~~Type~~Signature
OUT	PUBLISH/NOTIFY	<<Update value ~~type~~>>

a) The Subscription Keys row shall present the list of available subscription keys for the defined operation. The body signature will include the name and its respective data type for each subscription key.

b) The PUBLISH/NOTIFY message shall follow the same rules for message bodies (as defined in 3.4.2.2, where the message body may be composed of several types. Subsection 3.5.6.12 provides an example of this for this interaction pattern.

c) A single update shall require a value for each type listed in the operation template in the case where it is defined using multiple types. Subsection 3.5.6.12 provides an example of this for this interaction pattern.

d) The update value ~~type~~ shall be the ~~structure that~~same as it is passed in the provider-to-broker PUBLISH and broker-to-consumer NOTIFY messages.

e) The contents of these messages are fixed and therefore not present in the operation template for PUBLISH-SUBSCRIBE. The PUBLISH-SUBSCRIBE pattern is actually made up of consumer and provider register/deregister, publish, and notify messages, the templates of which are given below. The message directions (IN/OUT) are all from the point of view of the broker, as it is involved in all operations:

Table 3-25: PUBLISH-SUBSCRIBE Register Operation Template

Operation Identifier	register<< PUBLISH-SUBSCRIBE Operation Name>>	
Interaction Pattern	SUBMIT	
Pattern Sequence	Message	Body ~~Type~~Signature
IN	REGISTER	Subscription subscription

f) The consumer register method takes a subscription which is composed of an identifier, a domain, and a list of filters. The identifier shall be used by the broker to uniquely identify the subscription when combined with the consumer entity name; this allows a consumer to set up several different concurrent subscriptions.

g) Each subscription shall contain a list of filters which contain an identifier key name and a list of values to be matched.

Table 3-26: PUBLISH-SUBSCRIBE Publish Register Operation Template

Operation Identifier	publishRegister<< PUBLISH-SUBSCRIBE Operation Name>>	
Interaction Pattern	SUBMIT	
Pattern Sequence	Message	Body ~~Type~~Signature
IN	PUBLISH_REGISTER	~~List<EntityKey>~~

h) The provider register message takes no arguments. It shall register the provider from which updates will be performed.

Table 3-27: PUBLISH-SUBSCRIBE Publish Operation Template

Operation Identifier	publish<< PUBLISH-SUBSCRIBE Operation Name>>	
Interaction Pattern	SEND	
Pattern Sequence	Message	Body ~~Type~~Signature
IN	PUBLISH	~~List<UpdateHeader>~~ ~~List< <<Update Value Type>> >~~ UpdateHeader header <<Update Value>>

i) The provider-to-broker publish message shall match the provider register message on the area identifier, service identifier, version identifier, and operation identifier.

j) The provider-to-broker publish message domain shall match exactly, or be a sub-domain of (see 3.5.6.5), the domain of the provider register message.

k) A PUBLISH standard pattern body shall contain an UpdateHeader followed by the Update Value defined in the top-level PUBSUB template. For example, a PUBSUB operation defined using the types of [Boolean, Octet, MyComposite] will result in a PUBLISH message with the body containing [UpdateHeader, Boolean, Octet, MyComposite].

l) For each update being published there shall be an UpdateHeader and the corresponding Update Value.

m) Each UpdateHeader shall optionally contain the source of the update, and the update key values.

n) The key values in the UpdateHeader message shall be ordered as defined by the operation so that the corresponding key names can be matched with the entries in the list.

Table 3-28: PUBLISH-SUBSCRIBE Publish Error Operation Template

Operation Identifier	publishError<< PUBLISH-SUBSCRIBE Operation Name>>	
Interaction Pattern	SEND	
Pattern Sequence	Message	Body ~~Type~~Signature
OUT	PUBLISH_ERROR	UInteger ~~Element~~errorNumber

o) If a provider publishes an update with more or less key values than expected, then the broker shall return an UNKNOWN error in a PUBLISH_ERROR message.

Table 3-29: PUBLISH-SUBSCRIBE Notify Operation Template

Operation Identifier	notify<< PUBLISH-SUBSCRIBE Operation Name>>	
Interaction Pattern	SEND	
Pattern Sequence	Message	Body ~~Type~~Signature
OUT	NOTIFY	~~Identifier~~ ~~List<UpdateHeader>~~ ~~List< <<Update Value Type>> >~~ Identifier subscriptionId UpdateHeader updateHeader <<Update Values>>

p) The broker-to-consumer NOTIFY message body shall contain a single Identifier, then an UpdateHeader, followed by the Update Value defined in the top-level PUBSUB template, which holds the update set for a single subscription. For example, a PUBSUB operation defined using the types of [Boolean, Octet, MyComposite] will result in a NOTIFY message with the body containing [Identifier, UpdateHeader, Boolean, Octet, MyComposite].

q) The key values in the UpdateHeader message shall be ordered as defined by the operation so that the corresponding key names can be matched with the entries in the list.

r) The single Identifier in the notify message shall contain the subscription identifier for the subscription being notified.

s) In the NOTIFY message there shall be the update values for each type defined in the top level PUBSUB template.

t) If all entries in the UpdateHeader list are NULL, then the whole list shall be replaced with a NULL in order to increase efficiency.

Table 3-30: PUBLISH-SUBSCRIBE Notify Error Operation Template

Operation Identifier	notifyError<< PUBLISH-SUBSCRIBE Operation Name>>	
Interaction Pattern	SEND	
Pattern Sequence	Message	Body ~~Type~~Signature
OUT	NOTIFY_ERROR	UInteger errorNumber Element extraInformation

u) If the broker wants to shut down, then the broker shall return a SHUTDOWN error in a NOTIFY_ERROR message and an extra information message on the cause.

v) If the broker has an internal error, then the broker shall return an INTERNAL error in a NOTIFY_ERROR message and an extra information message on the cause.

Table 3-31: PUBLISH-SUBSCRIBE Deregister Operation Template

Operation Identifier	deregister<< PUBLISH-SUBSCRIBE Method Name>>	
Interaction Pattern	SUBMIT	
Pattern Sequence	Message	Body ~~Type~~Signature
IN	DEREGISTER	List<Identifier> subscriptionIds

w) The consumer deregister message shall take a list of identifiers of the active subscription lists to cancel.

Table 3-32: PUBLISH-SUBSCRIBE Publish Deregister Operation Template

Operation Identifier	publishDeregister<< PUBLISH-SUBSCRIBE Method Name>>	
Interaction Pattern	SUBMIT	
Pattern Sequence	Message	Body ~~Type~~Signature
IN	PUBLISH_DEREGISTER	

~~The contents of these messages are fixed and therefore not present in the operation template for PUBLISH-SUBSCRIBE. However, they are directly affected by the template as described below:~~

~~a) The consumer register method takes a subscription which is composed of an identifier and a list of entity requests. The identifier shall be used by the broker to uniquely identify the subscription when combined with the consumer URI; this allows a consumer to set up several different concurrent subscriptions.~~

~~b) Each subscription shall contain a list of individual entity requests which contain an optional sub-domain, an entity key and an indication of whether updates are required only on change (creation/modification/deletion update types) or for all updates.~~

c) The provider register message contains a list of entity keys that the provider shall provide updates for.

d) No other entity keys shall be published by that provider.

e) If a provider registers an entity key containing a wildcard for a sub-key then the broker shall allow the provider to publish an update with any value for that sub-key.

f) If a provider publishes an update for an entity that it has not previously registered then the broker shall return an UNKNOWN error in a PUBLISH_ERROR message.

g) The extra information part of the Error message shall be an EntityKey list that contains the list of entity keys that were unknown.

h) The list of entities allowed to be published by a provider may be replaced by another publish register message.

i) This list shall completely replace the existing list for that provider.

j) The provider-to-broker publish message shall match the provider register message on the session type, session name, area identifier, service identifier, and operation identifier.

k) The provider-to-broker publish message domain shall match exactly, or be a sub-domain of (see 3.5.6.5), the domain of the provider register message.

l) A PUBLISH message body shall contain an UpdateHeader list followed by a list of each type defined in the top-level PUBSUB template. For example, a PUBSUB operation defined using the types of [Boolean, Octet, MyComposite] will result in a PUBLISH message with the body containing [List<UpdateHeader>, List<Boolean>, List<Octet>, List<MyComposite>].

m) For each update being published there shall be an entry in the UpdateHeader list and an entry in each update value list.

n) The lists in the PUBLISH message shall be identically ordered so that the corresponding UpdateHeader and update values can be matched.

o) Each UpdateHeader shall contain the timestamp of the update, the sourceURI of the provider (used to identify the source of an update when multiple providers are using a shared broker), an Enumeration denoting if it is considered a new object, an update, a modification, or an object deletion by the provider (context specific), and then the update entity key.

p) The EntityKey of the Update shall not contain the wildcard value.

q) Each update, as listed above, shall be one of four possible types: creation, update, modification, or deletion:

1) ~~A creation notification shall be used when a new object has been created in the provider (for example, a new parameter is now being published by a service provider).~~

2) ~~An update notification shall be used when an object's value has not changed but a new update of it has been generated (for example, a periodic update of a telemetry parameter).~~

3) ~~A modification notification shall be used when an object's value has changed from the previously generated notification (for example, a changed telemetry value update).~~

4) ~~A deletion notification shall be used when an object is being removed from the provider (for example, an existing parameter is no longer being published by a service provider).~~

r) ~~The broker-to-consumer NOTIFY message body shall contain a single Identifier, an UpdateHeader list followed by a list of each type defined in the top-level PUBSUB template, and hold the update set for a single subscription. For example, a PUBSUB operation defined using the types of [Boolean, Octet, MyComposite] will result in a NOTIFY message with the body containing [Identifier, List<UpdateHeader>, List<Boolean>, List<Octet>, List<MyComposite>].~~

s) ~~The single Identifier in the notify message shall contain the subscription identifier for the subscription being notified.~~

t) ~~In the NOTIFY message there shall be an update value list for each type defined in the top level PUBSUB template.~~

u) ~~For each update being notified there shall be an entry in the UpdateHeader list and an entry in the update value lists.~~

v) ~~The lists in the NOTIFY message shall be identically ordered so that the corresponding UpdateHeader and update values can be matched.~~

w) ~~The consumer deregister message shall take a list of identifiers of the active subscription lists to cancel.~~

x) The provider deregister message takes no arguments. It shall completely remove the provider from the set of providers from which updates are permitted.

y) In all cases, for consumer/provider register and deregister, the acknowledge message shall only be sent when the registration/deregistration has completed and is a confirmation that the operation has succeeded.

3.5.6.9 Primitives

Table 3-33: PUBLISH-SUBSCRIBE Primitive List

Primitive
REGISTER
REGISTER_ACK
REGISTER_ERROR
PUBLISH_REGISTER
PUBLISH_REGISTER_ACK
PUBLISH_REGISTER_ERROR
PUBLISH
PUBLISH_ERROR
NOTIFY
NOTIFY_ERROR
DEREGISTER
DEREGISTER_ACK
PUBLISH_DEREGISTER
PUBLISH_DEREGISTER_ACK

3.5.6.10 State Charts

3.5.6.10.1 General

Several state charts are defined:

- one state chart on the consumer side related to the interaction between the consumer and the broker;
- one state chart on the broker side related to the interaction between the consumer and the broker;
- one state chart on the provider side related to the interaction between the provider and the broker;
- another state chart on the broker side related to the interaction between the provider and the broker.

3.5.6.10.2 Consumer Side

Figure 3-22 shows the consumer side state chart for the PUBLISH-SUBSCRIBE pattern:

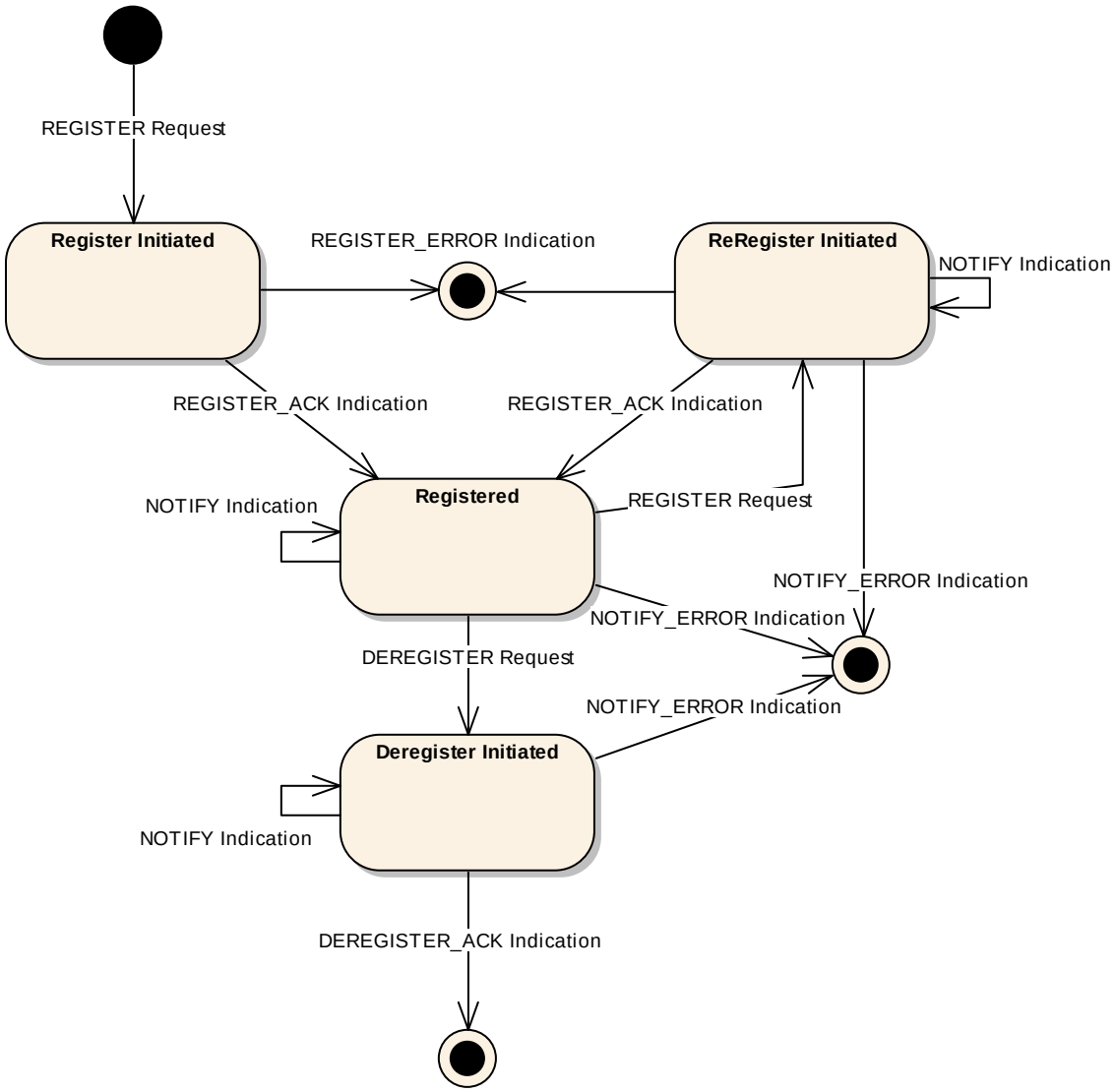


Figure 3-22: PUBLISH-SUBSCRIBE Consumer State Chart

a) The state chart applies to a single subscription for a single consumer, the state chart shall be replicated for each consumer and for each subscription.

b) The transaction identifier of the initial **REGISTER Message** shall be used to identify messages that relate to one PUBLISH-SUBSCRIBE interaction to avoid race conditions.

c) The initial transition is triggered by the primitive **REGISTER Request** and shall lead to the **Register Initiated State**.

d) There are two possible transitions from the **Register Initiated State**:

   1) the first is the indication of an acknowledgement, **REGISTER_ACK Indication**, and shall lead to the **Registered State**;

2) the second is the indication of an error, **REGISTER_ERROR Indication**, and shall lead to the final state.

e) There are four possible transitions from the **Registered State**:

1) the first is the indication of a notification, **NOTIFY Indication**, and shall lead back to the **Registered State**;

2) the second is the indication of an error, **NOTIFY_ERROR Indication**, and shall lead to the final state;

3) the third transition shall be triggered by the primitive **DEREGISTER Request** and shall lead to the **Deregister Initiated State**;

4) the fourth transition shall be triggered by the primitive **REGISTER Request** and shall lead to the **ReRegister Initiated State**.

f) There are four possible transitions from the **ReRegistered Initiated State**:

1) The first is the indication of a notification, **NOTIFY Indication**, and shall lead back to the **ReRegistered Initiated State**. In this case the **NOTIFY Message** will have been sent before the **REGISTER Message** was received by the broker and therefore forms part of the previous subscription. It shall, however, be passed to the consumer application as normal.

2) The second is the indication of a registration error, **REGISTER_ERROR Indication**, and shall lead to the final state.

3) The third is the indication of a notify error, **NOTIFY_ERROR Indication**, and shall lead to the final state. In this case the **NOTIFY_ERROR Message** will have been sent before the **REGISTER Message** was received by the broker and therefore forms part of the previous subscription. It shall, however, be passed to the consumer application as normal. The transaction identifier shall be used by the broker to ensure that the reception of the **REGISTER Message** after the **NOTIFY_ERROR Message** was sent shall not cause the restart of the subscription.

4) The fourth transition is the indication of an acknowledgement, **REGISTER_ACK Indication**, and shall lead to the **Registered State**.

g) There are three possible transitions from the **Deregister Initiated State**:

1) the first is the indication of a notification, **NOTIFY Indication**, and shall lead back to the **Deregister Initiated State**;

2) the second is the indication of an error, **NOTIFY_ERROR Indication**, and shall lead to the final state;

3) the third transition is the indication of an acknowledgement, **DEREGISTER_ACK Indication**, and shall lead to the final state.

3.5.6.10.3 Broker Side (Related to the Consumer)

Figure 3-23 shows the broker to consumer state chart for the PUBLISH-SUBSCRIBE pattern:

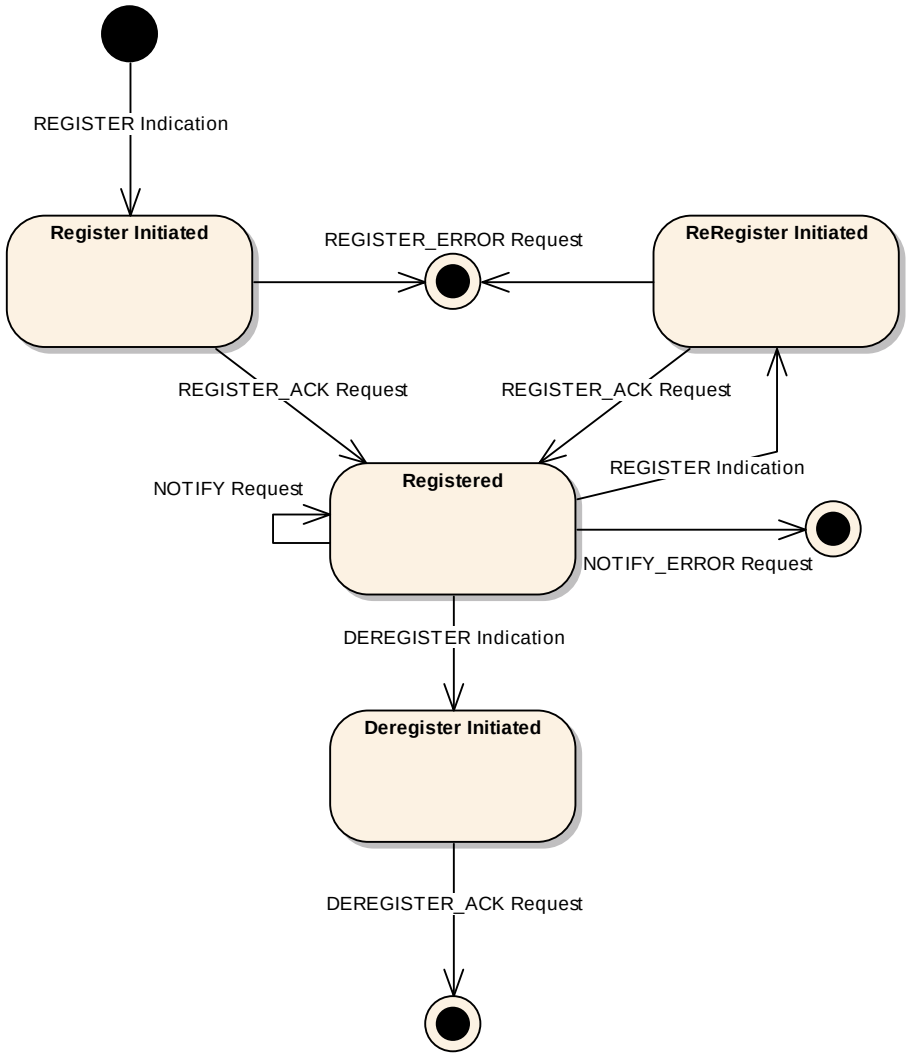


Figure 3-23: PUBLISH-SUBSCRIBE Broker to Consumer State Chart

a) The state chart applies to a single subscription for a single consumer. The state chart shall be replicated for each consumer and for each subscription.

b) The initial transition is triggered by the primitive **REGISTER Indication** and shall lead to the **Register Initiated State**.

c) There are two possible transitions from the **Register Initiated State**:

   1) the first is the triggering of an acknowledgement, **REGISTER_ACK Request**, and shall lead to the **Registered State**;

2) the second is the triggering of an error, **REGISTER_ERROR Request**, and shall lead to the final state.

d) There are four possible transitions from the **Registered State**;

1) the first is the triggering of a notification, **NOTIFY Request**, and shall lead back to the **Registered State**;

2) the second is the triggering of an error, **NOTIFY_ERROR Request**, and shall lead to the final state;

3) the third transition shall be triggered by the primitive **DEREGISTER Indication** and shall lead to the **Deregister Initiated State**;

4) the fourth transition shall be triggered by the primitive **REGISTER Indication** and shall lead to the **ReRegister Initiated State**.

e) There are two possible transitions from the **ReRegister Initiated State**:

1) the first is the triggering of an acknowledgement, **REGISTER_ACK Request**, and shall lead to the **Registered State**;

2) the second is the triggering of an error, **REGISTER_ERROR Request**, and shall lead to the final state.

f) There is only one possible transition from the **Deregister Initiated State**; it is the triggering of an acknowledgement, **DEREGISTER_ACK Request**, and shall lead to the final state.+

3.5.6.10.4 Provider Side

Figure 3-24 shows the provider side state chart for the PUBLISH-SUBSCRIBE pattern:

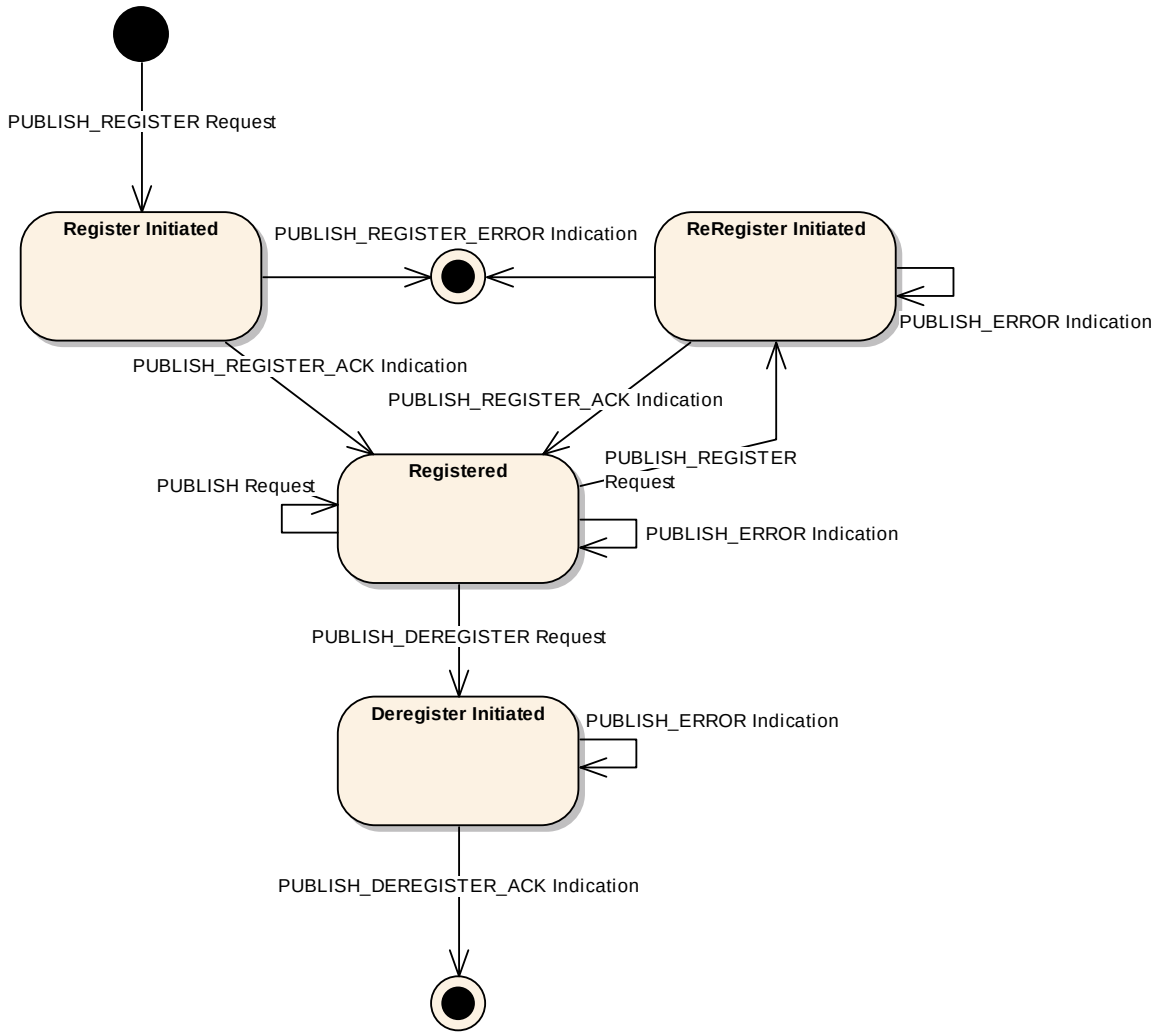


Figure 3-24: PUBLISH-SUBSCRIBE Provider State Chart

a) The state chart applies to a single provider. The state chart shall be replicated for each provider.

b) The initial transition is triggered by the primitive **PUBLISH_REGISTER Request** and shall lead to the **Register Initiated State**.

c) There are two possible transitions from the **Register Initiated State**:

   1) the first is the indication of an acknowledgement, **PUBLISH_REGISTER_ACK Indication**, and shall lead to the **Registered State**;

   2) the second is the indication of an error, **PUBLISH_REGISTER_ERROR Indication**, and shall lead to the final state.

d) There are four possible transitions from the **Registered State**:

1) the first is the triggering of a publish, **PUBLISH Request**, and shall lead back to the **Registered State**;

2) the second is the indication of an error, **PUBLISH_ERROR Indication**, and shall lead back to the **Registered State**;

3) the third transition shall be triggered by the primitive **PUBLISH_REGISTER Request** and shall lead to the **ReRegister Initiated State**;

4) the fourth transition shall be triggered by the primitive **PUBLISH_DEREGISTER Request** and shall lead to the **Deregister Initiated State**.

e) There are three possible transitions from the **ReRegister Initiated State**:

1) The first is the indication of an acknowledgement, **PUBLISH_REGISTER_ACK Indication**, and shall lead to the **Registered State**.

2) The second is the indication of a registration error, **PUBLISH_REGISTER_ERROR Indication**, and shall lead to the final state.

3) The third is the indication of a publish error, **PUBLISH_ERROR Indication**, and shall lead back to the **Re-Registered State**. In this case the **PUBLISH_ERROR Message** will have been sent before the **PUBLISH_REGISTER Message** was received by the broker and therefore forms part of the previous subscription.

f) There are two possible transitions from the **Deregister Initiated State**:

1) the first is the indication of an acknowledgement, **PUBLISH_DEREGISTER_ACK Indication**, and shall lead to the final state;

2) the second is the indication of a publish error, **PUBLISH_ERROR Indication**, and shall lead back to the **Deregister Initiated State**.

3.5.6.10.5 Broker Side (Related to the Provider)

Figure 3-25 shows the broker to provider state chart for the PUBLISH-SUBSCRIBE pattern:

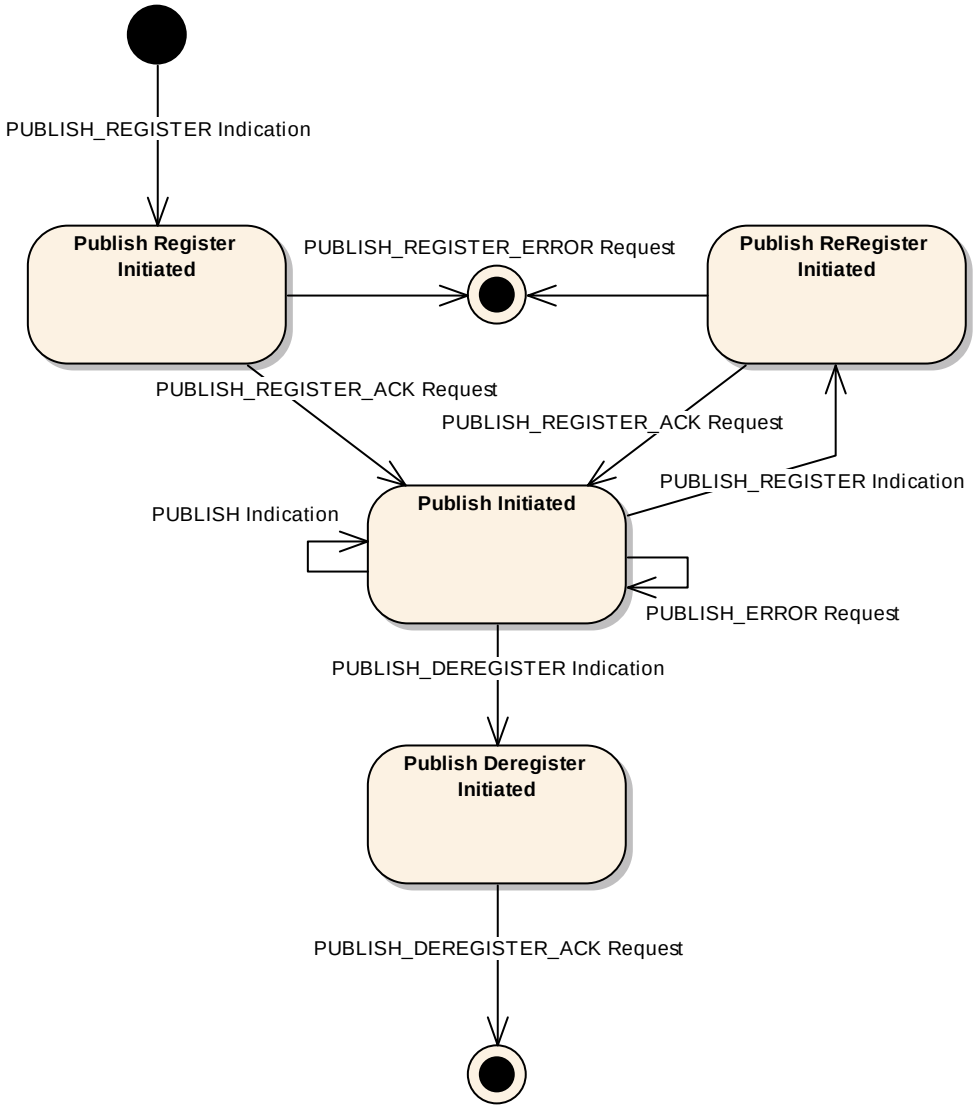


Figure 3-25: PUBLISH-SUBSCRIBE Broker to Provider State Chart

a) The state chart applies to a single provider. The state chart shall be replicated for each provider.

b) The initial transition is triggered by the primitive **PUBLISH_REGISTER Indication** and shall lead to the **Publish Register Initiated State**.

c) There are two possible transitions from the **Publish Register Initiated State**:

   1) the first is the triggering of an acknowledgement, **PUBLISH_REGISTER_ACK Request**, and shall lead to the **Publish Initiated State**;

2) the second is the triggering of an error, **PUBLISH_REGISTER_ERROR Request**, and shall lead to the final state.

d) There are four possible transitions from the **Publish Initiated State**:

1) the first is the indication of a publish, **PUBLISH Indication**, and shall lead back to the **Publish Initiated State**;

2) the second is the triggering of an error, **PUBLISH_ERROR Request**, and shall lead back to the **Publish Initiated State**

3) the third transition shall be triggered by the primitive **PUBLISH_DEREGISTER Indication** and shall lead to the **Publish Deregister Initiated State**;

4) the fourth transition shall be triggered by the primitive **PUBLISH_REGISTER Indication** and shall lead to the **Publish ReRegister Initiated State**.

e) There are two possible transitions from the **Publish ReRegister Initiated State**:

1) the first is the triggering of an acknowledgement, **PUBLISH_REGISTER_ACK Request**, and shall lead to the **Publish Initiated State**;

2) the second is the triggering of an error, **PUBLISH_REGISTER_ERROR Request**, and shall lead to the final state.

f) There is only one possible transition from the **Publish Deregister Initiated State**; it is the triggering of an acknowledgement, **PUBLISH_DEREGISTER_ACK Request**, and shall lead to the final state.

3.5.6.11 Requests and Indications

3.5.6.11.1 REGISTER

3.5.6.11.1.1 Function

a) The **REGISTER Request** primitive shall be used by the consumer application to initiate a PUBSUB Interaction with a subscription or to update an existing subscription for the specified subscription identifier.

b) The **REGISTER Indication** primitive shall be used by the broker MAL to deliver a **REGISTER Message** to a broker and initiate a PUBSUB Interaction or to update an existing subscription for the specified subscription identifier.

3.5.6.11.1.2 Semantics

REGISTER Request and **REGISTER Indication** shall provide parameters as follows:

(REGISTER Message Header, Subscription, QoS properties)

3.5.6.11.1.3 When Generated

a) A **REGISTER Request** may be used by the consumer application at any time to start a new subscription or when the consumer MAL is in the **Registered State** to update an existing subscription.

b) A **REGISTER Indication** shall be generated by the broker MAL for a new subscription or when the broker MAL is in the **Registered State** for existing subscriptions for that consumer upon reception of a **REGISTER Message**, once checked via the Access Control interface, from a consumer.

3.5.6.11.1.4 Effect on Reception

a) Reception of a **REGISTER Request** shall, once checked via the Access Control interface, cause the consumer MAL to initiate a PUBSUB Interaction by transmitting a **REGISTER Message** to the broker for a new subscription.

b) The consumer MAL shall enter the **Register Initiated State** in this case.

c) If the consumer MAL is in the **Registered State**, reception of a **REGISTER Request** shall, once checked via the Access Control interface, cause the consumer MAL to update the subscription by transmitting a **REGISTER Message** to the broker.

d) The consumer MAL shall enter the **ReRegister Initiated State** in this case.

e) On reception of a **REGISTER Message** by the broker MAL, once the message has been checked via the Access Control interface, the broker MAL shall enter the **Register Initiated State** if this is the first message in a new PUBSUB Interaction or enter the **ReRegister Initiated State** if currently in the **Registered State** and then pass a **REGISTER Indication** to the broker.

f) The broker shall store the subscription of the consumer or update an existing subscription from the same consumer if the subscription identifier is already used by that consumer.

g) The broker shall start processing the subscription at this point.

3.5.6.11.1.5 Message Header

a) For the **REGISTER Message** the message header fields shall be as defined in 3.4 except for the fields in table 3-34.

b) The contents of the Subscription structure, as specified in the operation template, shall immediately follow the message header.

Table 3-34: REGISTER Message Header Fields

Field	Value
~~URI~~ From	Consumer ~~URI~~
Authentication Id	Consumer Authentication Identifier
~~URI~~ To	Broker ~~URI~~
Interaction Type	PUBSUB
Interaction Stage	1
Transaction Id	Provided by consumer MAL

3.5.6.11.2 REGISTER_ACK

3.5.6.11.2.1 Function

a) The **REGISTER_ACK Request** primitive shall be used by the broker to acknowledge a PUBSUB Interaction subscription.

b) The **REGISTER_ACK Indication** primitive shall be used by the consumer MAL to deliver a **REGISTER_ACK Message** to a consumer.

3.5.6.11.2.2 Semantics

REGISTER_ACK Request and **REGISTER_ACK Indication** shall provide parameters as follows:

(REGISTER_ACK Message Header, QoS properties)

3.5.6.11.2.3 When Generated

a) A **REGISTER_ACK Request** shall be used by the broker with the broker MAL in either the **Register Initiated State** or **ReRegister Initiated State** to acknowledge the successful registration by a consumer in a PUBSUB Interaction.

b) A **REGISTER_ACK Indication** shall be generated by the consumer MAL in either the **Register Initiated State** or **ReRegister Initiated State** upon reception of a **REGISTER_ACK Message**, once checked via the Access Control interface, from a broker.

3.5.6.11.2.4 Effect on Reception

a) Reception of a **REGISTER_ACK Request** shall, once checked via the Access Control interface, cause the broker MAL to transmit the supplied acknowledgement as a **REGISTER_ACK Message** to the consumer.

b) A broker MAL in the **Register Initiated State** or **ReRegister Initiated State** shall enter the **Registered State**.

c) On reception of a **REGISTER_ACK Message** by the consumer MAL, once the message has been checked via the Access Control interface, the consumer MAL shall enter the **Registered State** and pass a **REGISTER_ACK Indication** to the consumer application.

3.5.6.11.2.5 Message Header

For the **REGISTER_ACK Message** the message header fields shall be as defined in 3.4 except for the fields in table 3-35.

Table 3-35: REGISTER_ACK Message Header Fields

Field	Value
~~URI~~ From	Broker ~~URI~~
Authentication Id	Broker Authentication Identifier
~~URI~~ To	Consumer ~~URI~~
QoSlevel	QoS level from first REGISTER message
Priority	Priority from first REGISTER message
Interaction Type	PUBSUB
Interaction Stage	2
Transaction Id	Transaction Id from initial message

3.5.6.11.3 REGISTER_ERROR

3.5.6.11.3.1 Function

a) The **REGISTER_ERROR Request** primitive shall be used by the broker to reject a subscription request and end a PUBSUB Interaction with an error.

b) The **REGISTER_ERROR Indication** primitive shall be used by the consumer MAL to deliver a **REGISTER_ERROR Message** to a consumer.

3.5.6.11.3.2 Semantics

REGISTER_ERROR Request and **REGISTER_ERROR Indication** shall provide parameters as follows:

(REGISTER_ERROR Message Header, Error Number, Extra Information, QoS properties)

3.5.6.11.3.3 When Generated

a) A **REGISTER_ERROR Request** shall be used by the broker with the broker MAL in either the **Register Initiated State** or **ReRegister Initiated State** to reject a register request.

b) The broker may reject the register attempt for one of the following reasons:

   1) More subscriptions than the broker can support shall cause a TOO_MANY error to be sent. The extra information field of the error contains an Integer which provides the maximum number of subscriptions supported.

   2) Shutdown, the broker is shutting down. A SHUTDOWN error message shall be returned.

   3) Internal error, the broker has experienced an internal error. An INTERNAL error message shall be returned.

   4) UNKNOWN shall not be generated for subscriptions that identify keys that have not been currently registered by a publisher. This is because the consumer cannot know what entities are currently provided and which providers are currently registered.

c) A **REGISTER_ERROR Indication** shall be generated by the consumer MAL in either the **Register Initiated State** or **ReRegister Initiated State** upon one of two events:

   1) the reception of a **REGISTER_ERROR Message**, once checked via the Access Control interface, from a broker;

   2) an error raised by the local communication layer.

3.5.6.11.3.4 Effect on Reception

a) Reception of a **REGISTER_ERROR Request** shall, once checked via the Access Control interface, cause the broker MAL to transmit a **REGISTER_ERROR Message** to the consumer.

b) The pattern shall end at this point for the broker.

c) On reception of a **REGISTER_ERROR Message** by the consumer MAL, once the message has been checked via the Access Control interface, the consumer MAL shall end the interaction by passing the error to the consumer application.

3.5.6.11.3.5 Message Header

a) For the **REGISTER_ERROR Message** the message header fields shall be the same as for the **REGISTER_ACK Message** except for the 'Is Error Message' field which is set to TRUE.

b) The Error information shall immediately follow the message header.

3.5.6.11.4 PUBLISH_REGISTER

3.5.6.11.4.1 Function

a) The **PUBLISH_REGISTER Request** primitive shall be used by the provider application to initiate a PUBSUB Interaction with the list of entities being published or to update an existing list.

b) The **PUBLISH_REGISTER Indication** primitive shall be used by the broker MAL to deliver a **PUBLISH_REGISTER Message** to a broker and initiate a PUBSUB Interaction or to update an existing publish list if one exists for the specified provider.

3.5.6.11.4.2 Semantics

PUBLISH_REGISTER Request and **PUBLISH_REGISTER Indication** shall provide parameters as follows:

(PUBLISH_REGISTER Message Header, ~~List<EntityKey>,~~ QoS properties)

3.5.6.11.4.3 When Generated

a) A **PUBLISH_REGISTER Request** may be generated by the provider application at any time to start a publishing session or when the provider MAL is already in the **Registered State** to update an existing publish list.

b) A **PUBLISH_REGISTER Indication** shall be generated by the broker MAL if a new publisher is detected or in the **Publish Registered State** for the provider upon reception of a **PUBLISH_REGISTER Message**, once checked via the Access Control interface, from a provider.

3.5.6.11.4.4 Effect on Reception

a) Reception of a **PUBLISH_REGISTER Request** shall, once checked via the Access Control interface, cause the provider MAL to initiate a PUBSUB Interaction by transmitting a **PUBLISH_REGISTER Message** to the broker.

b) The provider MAL shall enter the **Register Initiated State** in this case.

c) If the provider MAL is already in the **Registered State**, reception of a **PUBLISH_REGISTER Request** shall, once checked via the Access Control interface, cause the provider MAL to ~~update the publishing list by transmitting~~transmit a **PUBLISH_REGISTER Message** to the broker.

d) The provider MAL shall enter the **ReRegister Initiated State** in this case.

e) On reception of a **PUBLISH_REGISTER Message** by the broker MAL, once the message has been checked via the Access Control interface, the broker MAL shall enter the **Publish Register Initiated State** if this is the first message in a new PUBSUB Interaction with the provider or enter the **Publish ReRegister Initiated State** if currently in the **Publish Registered State** with that provider and then pass a **PUBLISH_REGISTER Indication** to the broker.

~~f) The broker shall store the publish list of the provider or update the existing list from the same provider.~~

3.5.6.11.4.5 Message Header

For the **PUBLISH_REGISTER Message** the message header fields shall be as defined in 3.4 except for the fields in table 3-36.

~~b) The contents of the EntityKey list, as specified in the operation template, shall immediately follow the message header.~~

Table 3-36: PUBLISH_REGISTER Message Header Fields

Field	Value
~~URI~~ From	Provider ~~URI~~
Authentication Id	Provider Authentication Identifier
~~URI~~ To	Broker ~~URI~~
Interaction Type	PUBSUB
Interaction Stage	3
Transaction Id	Provided by provider MAL

3.5.6.11.5 PUBLISH_REGISTER_ACK

3.5.6.11.5.1 Function

a) The **PUBLISH_REGISTER_ACK Request** primitive shall be used by the broker to acknowledge a PUBSUB Interaction publish list from a provider.

b) The **PUBLISH_REGISTER_ACK Indication** primitive shall be used by the provider MAL to deliver a **PUBLISH_REGISTER_ACK Message** to a provider.

3.5.6.11.5.2 Semantics

PUBLISH_REGISTER_ACK Request and **PUBLISH_REGISTER_ACK Indication** shall provide parameters as follows:

(PUBLISH_REGISTER Message Header, QoS properties)

3.5.6.11.5.3 When Generated

a) A **PUBLISH_REGISTER_ACK Request** shall be used by the broker with the broker MAL in either the **Publish Register Initiated State** or **Publish ReRegister Initiated State** to acknowledge the successful registration by a provider in a PUBSUB Interaction.

b) A **PUBLISH_REGISTER_ACK Indication** shall be generated by the provider MAL in either the **Register Initiated State** or **ReRegister Initiated State** upon reception of a **PUBLISH_REGISTER_ACK Message**, once checked via the Access Control interface, from a broker.

3.5.6.11.5.4 Effect on Reception

a) Reception of a **PUBLISH_REGISTER_ACK Request** shall, once checked via the Access Control interface, cause the broker MAL to transmit the supplied acknowledgement as a **PUBLISH_REGISTER_ACK Message** to the provider.

b) A broker MAL in the **Publish Register Initiated State** or **Publish ReRegister Initiated State** shall then enter the **Registered State**.

c) On reception of a **PUBLISH_REGISTER_ACK Message** by the provider MAL, once the message has been checked via the Access Control interface, the provider MAL shall enter the **Registered State** and pass a **PUBLISH_REGISTER_ACK Indication** to the provider application.

3.5.6.11.5.5 Message Header

For the **PUBLISH_REGISTER_ACK Message** the message header fields shall be as defined in 3.4 except for the fields in table 3-37.

Table 3-37: PUBLISH_REGISTER_ACK Message Header Fields

Field	Value
~~URI~~ From	Broker ~~URI~~
Authentication Id	Broker Authentication Identifier
~~URI~~ To	Provider ~~URI~~
QoSlevel	QoS level from first PUBLISH_REGISTER message
Priority	Priority from first PUBLISH_REGISTER message
Interaction Type	PUBSUB
Interaction Stage	4
Transaction Id	Transaction Id from provider register message

3.5.6.11.6 PUBLISH_REGISTER_ERROR

3.5.6.11.6.1 Function

a) The **PUBLISH_REGISTER_ERROR Request** primitive shall be used by the broker to reject a publish registration request from a provider and end a PUBSUB Interaction with an error.

b) The **PUBLISH_REGISTER_ERROR Indication** primitive shall be used by the provider MAL to deliver a **PUBLISH_REGISTER_ERROR Message** to a provider.

3.5.6.11.6.2 Semantics

PUBLISH_REGISTER_ERROR Request and **PUBLISH_REGISTER_ERROR Indication** shall provide parameters as follows:

(PUBLISH_REGISTER_ERROR Message Header, Error Number, Extra Information, QoS properties)

3.5.6.11.6.3 When Generated

a) A **PUBLISH_REGISTER_ERROR Request** shall be used by the broker with the broker MAL in either the **Publish Register Initiated State** or **Publish ReRegister Initiated State** to reject a register request.

b) The broker may reject the register attempt for one of the following reasons:

1) More providers than the broker can support shall cause a TOO_MANY error to be sent. The extra information field of the error contains an Integer which provides the maximum number of providers supported.

2) Shutdown, the broker is shutting down. A SHUTDOWN error message shall be returned.

3) Internal error, the broker has experienced an internal error. An INTERNAL error message shall be returned.

c) A **PUBLISH_REGISTER_ERROR Indication** shall be generated by the provider MAL in either the **Register Initiated State** or **ReRegister Initiated State** upon one of two events:

1) the reception of a **PUBLISH_REGISTER_ERROR Message**, once checked via the Access Control interface, from a broker;

2) an error raised by the local communication layer.

3.5.6.11.6.4 Effect on Reception

a) Reception of a **PUBLISH_REGISTER_ERROR Request** shall, once checked via the Access Control interface, cause the broker MAL to transmit a **PUBLISH_REGISTER_ERROR Message** to the provider.

b) The pattern shall end at this point for the broker.

c) On reception of a **PUBLISH_REGISTER_ERROR Message** by the provider MAL, once the message has been checked via the Access Control interface, the provider MAL shall end the interaction by passing the error indication to the provider application.

3.5.6.11.6.5 Message Header

a) For the **PUBLISH_REGISTER_ERROR Message** the message header fields shall be the same as for the **PUBLISH_REGISTER_ACK Message** except for the 'Is Error Message' field which is set to TRUE.

b) The Error information structure shall immediately follow the message header.

3.5.6.11.7 PUBLISH

3.5.6.11.7.1 Function

a) The **PUBLISH Request** primitive shall be used by the provider to publish ~~a list of entity updates~~an entity update.

b) The **PUBLISH Indication** primitive shall be used by the broker MAL to deliver a **PUBLISH Message** to a broker.

3.5.6.11.7.2 Semantics

PUBLISH Request and **PUBLISH Indication** shall provide parameters as follows:

~~(PUBLISH Message Header, List<UpdateHeader>,~~

~~List< <<Update Value Type>> >, … List< <<Update Value Type N>> >, QoS properties)~~
(PUBLISH Message Header, UpdateHeader,

<<Update Value Type>>, <<Update Value Type N>>, QoS properties)

3.5.6.11.7.3 When Generated

a) A **PUBLISH Request** shall be used by the provider with the provider MAL in the **Registered State** to publish ~~a list of entity updates~~an entity update.

b) A **PUBLISH Indication** shall be generated by the broker MAL in the **Publish Initiated State** upon reception of a **PUBLISH Message**, once checked via the Access Control interface, from a provider.

3.5.6.11.7.4 Effect on Reception

a) Reception of a **PUBLISH Request** shall, once checked via the Access Control interface, cause the provider MAL to transmit the supplied update ~~list~~ as a **PUBLISH Message** to the broker.

b) On reception of a **PUBLISH Message** by the broker MAL, once the message has been checked via the Access Control interface, the broker MAL shall pass a **PUBLISH Indication** to the broker application.

c) The broker application shall match the ~~Updates in the list~~Update to the subscriptions of registered consumers that are in the **Registered State** using the match rules in 3.5.6.4.

d) The matched update~~s~~ shall be transmitted to the relevant consumers by the broker using a **NOTIFY Request**.

3.5.6.11.7.5 Message Header

a) For the **PUBLISH Message** the message header fields shall be as defined in 3.4 except for the fields in table 3-38.

b) The contents of the message body, as specified in the operation template, shall immediately follow the message header.

~~c) The transaction identifier shall be the same as in the initial publish register message. If several register requests have been sent (without any deregister requests in between), the transaction identifier of the first request shall be used.~~

Table 3-38: PUBLISH Message Header Fields

Field	Value
~~URI~~ From	Provider ~~URI~~
Authentication Id	Provider Authentication Identifier
~~URI~~ To	Broker ~~URI~~
Interaction Type	PUBSUB
Interaction Stage	5
Transaction Id	Transaction Id from provider register message

3.5.6.11.8 PUBLISH_ERROR

3.5.6.11.8.1 Function

a) The **PUBLISH_ERROR Request** primitive shall be used by the broker to reject a publish request from a provider.

b) The **PUBLISH_ERROR Indication** primitive shall be used by the provider MAL to deliver a **PUBLISH_ERROR Message** to a provider.

3.5.6.11.8.2 Semantics

PUBLISH_ERROR Request and **PUBLISH_ERROR Indication** shall provide parameters as follows:

(PUBLISH_ERROR Message Header, Error Number, ~~Extra Information,~~ QoS properties)

3.5.6.11.8.3 When Generated

a) A **PUBLISH_ERROR Request** shall be used by the broker with the broker MAL in the **Publish Initiated State** to reject a publish request.

b) The broker may reject the publish attempt for one of the following reasons:

1) Unknown provider: a provider has not previously registered for publishing. An INCORRECT_STATE error message shall be returned.

~~2) Unknown Entity: a provider has attempted to publish a previously unregistered Entity. An UNKNOWN error shall be returned in this case; the extra information field of the error contains an EntityKey list which contains the list of unknown EntityKeys.~~

2) Unknown Subscription Key: a provider has attempted to publish a an incorrect number of Subscription Keys. An UNKNOWN error shall be returned in this case.

3) Shutdown: the broker is shutting down. A SHUTDOWN error message shall be returned.

4) Internal error: the broker has experienced an internal error. An INTERNAL error message shall be returned.

c) Communications/Encoding/Access Control errors shall be returned to a provider using this error message. This allows a provider to receive notification of underlying errors without the publishing overhead of acknowledging each **PUBLISH Message**.

d) A **PUBLISH_ERROR Indication** shall be generated by the provider MAL in either the **ReRegister Initiated State**, **Registered State** or **Deregister Initiated State** upon one of two events:

1) the reception of a **PUBLISH_ERROR Message**, once checked via the Access Control interface, from a broker;

2) an error raised by the local communication layer.

3.5.6.11.8.4 Effect on Reception

a) Reception of a **PUBLISH_ERROR Request** shall, once checked via the Access Control interface, cause the broker MAL to transmit a **PUBLISH_ERROR Message** to the provider.

b) On reception of a **PUBLISH_ERROR Message** by the provider MAL, once the message has been checked via the Access Control interface, the provider MAL shall pass the error indication to the provider application.

3.5.6.11.8.5 Message Header

a) For the **PUBLISH_ERROR Message** the message header fields shall be the same as for the **PUBLISH Message** except for the fields in table 3-39.

b) In the case where an error is being return without a previous PUBLISH_REGISTER message, the QoSlevel, Priority, and Transaction Identifier shall be taken from the PUBLISH message.

c) The Error information shall immediately follow the message header.

Table 3-39: PUBLISH_ERROR Message Header Fields

Field	Value
~~URI~~ From	Broker ~~URI~~
Authentication Id	Broker Authentication Identifier
~~URI~~ To	Provider ~~URI~~
QoSlevel	QoS level from first PUBLISH_REGISTER message
Priority	Priority from first PUBLISH_REGISTER message
Is Error Message	TRUE

3.5.6.11.9 NOTIFY

3.5.6.11.9.1 Function

a) The **NOTIFY Request** primitive shall be used by the broker to transmit ~~a list of entity updates~~an entity update to a consumer.

b) The **NOTIFY Indication** primitive shall be used by the consumer MAL to deliver a **NOTIFY Message** to a consumer.

3.5.6.11.9.2 Semantics

NOTIFY Request and **NOTIFY Indication** shall provide parameters as follows:

~~(NOTIFY Message Header, Identifier, List<UpdateHeader>,~~

~~List< <<Update Value Type>> >, … List< <<Update Value Type N>> >, QoS properties)~~
(NOTIFY Message Header, Identifier, UpdateHeader,

<<Update Value Type>>, … <<Update Value Type N>>, QoS properties)

3.5.6.11.9.3 When Generated

a) A **NOTIFY Request** shall be used by the broker with the broker MAL in the **Registered State** to transmit ~~a list of~~an entity update~~s~~ from a provider **PUBLISH Message** to a consumer.

b) A **NOTIFY Indication** shall be generated by the consumer MAL in either the **ReRegister Initiated State**, **Registered State** or **Deregister Initiated State** upon reception of a **NOTIFY Message**, once checked via the Access Control interface, from a broker.

3.5.6.11.9.4 Effect on Reception

a) Reception of a **NOTIFY Request** shall, once checked via the Access Control interface, cause the broker MAL to transmit the supplied subscription update ~~list~~ as a **NOTIFY Message** to the consumer.

b) On reception of a **NOTIFY Message** by the consumer MAL, once the message has been checked via the Access Control interface, the consumer MAL shall pass a **NOTIFY Indication** to the consumer application.

3.5.6.11.9.5 Message Header

a) For the **NOTIFY Message** the message header fields shall be as defined in 3.4 except for the fields in table 3-40.

b) The contents of the subscription update, as specified in the operation template, shall immediately follow the message header.

c) The transaction identifier shall be the same as in the initial register message that created the first subscription. If several register requests have been sent for the same subscription (without any deregister requests in between), the transaction identifier of the first request shall be used.

Table 3-40: NOTIFY Message Header Fields

Field	Value
~~URI~~ From	Broker ~~URI~~
Authentication Id	Broker Authentication Identifier
~~URI~~ To	Consumer ~~URI~~
QoSlevel	QoS level from first REGISTER message
Priority	Priority from first REGISTER message
Interaction Type	PUBSUB
Interaction Stage	6
Transaction Id	Transaction Id from consumer register message

3.5.6.11.10 NOTIFY_ERROR

3.5.6.11.10.1 Function

a) The **NOTIFY_ERROR Request** primitive shall be used by the broker to inform a consumer of an error and end a PUBSUB Interaction.

b) The **NOTIFY_ERROR Indication** primitive shall be used by the consumer MAL to deliver a **NOTIFY_ERROR Message** to a consumer.

3.5.6.11.10.2 Semantics

NOTIFY_ERROR Request and **NOTIFY_ERROR Indication** shall provide parameters as follows:

(NOTIFY_ERROR Message Header, Error Number, ~~Extra Information,~~ QoS properties)

3.5.6.11.10.3 When Generated

a) A **NOTIFY_ERROR Request** shall be used by the broker with the broker MAL in the **Registered State** to inform a consumer of an error.

b) The broker may send an error for one of the following reasons:

1) Shutdown: the broker is shutting down. A SHUTDOWN error message shall be returned.

2) Internal error: the broker has experienced an internal error. An INTERNAL error message shall be returned.

c) A **NOTIFY_ERROR Indication** shall be generated by the consumer MAL in either the **ReRegister Initiated State**, **Registered State** or **Deregister Initiated State** upon one of two events:

1) the reception of a **NOTIFY_ERROR Message**, once checked via the Access Control interface, from a broker;

2) an error raised by the local communication layer.

3.5.6.11.10.4 Effect on Reception

a) Reception of a **NOTIFY_ERROR Request** shall, once checked via the Access Control interface, cause the broker MAL to transmit a **NOTIFY_ERROR Message** to the consumer.

b) The pattern shall end at this point for the broker.

c) On reception of a **NOTIFY_ERROR Message** by the consumer MAL, once the message has been checked via the Access Control interface, the consumer MAL shall end the interaction by passing the error to the consumer application.

3.5.6.11.10.5 Message Header

a) For the **NOTIFY_ERROR Message** the message header fields shall be the same as for the **NOTIFY Message** except for the ‘Is Error Message’ field which is set to TRUE.

b) The Error information shall immediately follow the message header.

3.5.6.11.11 DEREGISTER

3.5.6.11.11.1 Function

a) The **DEREGISTER Request** primitive shall be used by the consumer to end subscriptions for the specified subscription identifiers.

b) The **DEREGISTER Indication** primitive shall be used by the broker MAL to deliver a **DEREGISTER Message** to a broker and end existing subscriptions for the specified subscription identifiers.

3.5.6.11.11.2 Semantics

DEREGISTER Request and **DEREGISTER Indication** shall provide parameters as follows:

(DEREGISTER Message Header, List<Identifier>, QoS properties)

3.5.6.11.11.3 When Generated

a) A **DEREGISTER Request** shall be used by the consumer application with the consumer MAL in the **Registered State** at any time to end a subscription.

b) A **DEREGISTER Indication** shall be generated by the broker MAL in the **Registered State** upon reception of a **DEREGISTER Message**, once checked via the Access Control interface, from a consumer.

3.5.6.11.11.4 Effect on Reception

a) Reception of a **DEREGISTER Request** shall, once checked via the Access Control interface, cause the consumer MAL in the **Registered State** to end the set of subscriptions by transmitting a **DEREGISTER Message** to the broker.

b) The consumer MAL shall enter the **Deregister Initiated State** for each identified subscription.

c) On reception of a **DEREGISTER Message** by the broker MAL, once the message has been checked via the Access Control interface, the broker MAL shall enter the **Deregister Initiated State** for each of the identified subscriptions and pass a DEREGISTER Indication to the broker.

d) The broker shall then remove the specified subscriptions of the consumer.

3.5.6.11.11.5 Message Header

a) For the **DEREGISTER Message** the message header fields shall be as defined in 3.4 except for the fields in table 3-41.

b) The contents of the subscription Identifier list, as specified in the operation template, shall immediately follow the message header.

Table 3-41: DEREGISTER Message Header Fields

Field	Value
~~URI~~ From	Consumer ~~URI~~
Authentication Id	Consumer Authentication Identifier
~~URI~~ To	Broker ~~URI~~
Interaction Type	PUBSUB
Interaction Stage	7
Transaction Id	Provided by consumer MAL

3.5.6.11.12 DEREGISTER_ACK

3.5.6.11.12.1 Function

a) The **DEREGISTER_ACK Request** primitive shall be used by the broker to acknowledge the termination of a PUBSUB Interaction subscription by the consumer.

b) The **DEREGISTER_ACK Indication** primitive shall be used by the consumer MAL to deliver a **DEREGISTER_ACK Message** to a consumer.

3.5.6.11.12.2 Semantics

DEREGISTER_ACK Request and **DEREGISTER_ACK Indication** shall provide parameters as follows:

(DEREGISTER_ACK Message Header, QoS properties)

3.5.6.11.12.3 When Generated

a) A **DEREGISTER_ACK Request** shall be used by the broker with a broker MAL in the **Deregister Initiated State** to acknowledge the successful deregistration by a consumer of a set of subscriptions in a PUBSUB Interaction.

b) A **DEREGISTER_ACK Indication** shall be generated by the consumer MAL in the **Deregister Initiated State** upon reception of a **DEREGISTER_ACK Message**, once checked via the Access Control interface, from a broker.

3.5.6.11.12.4 Effect on Reception

a) Reception of a **DEREGISTER_ACK Request** shall, once checked via the Access Control interface, cause the broker MAL to transmit a **DEREGISTER_ACK Message** to the consumer.

b) The interaction shall end at this point for the broker for each of the identified subscriptions.

c) On reception of a **DEREGISTER_ACK Message** by the consumer MAL in the **Deregister Initiated State**, once the message has been checked via the Access Control interface, the consumer MAL shall pass the **DEREGISTER_ACK Indication** to the consumer application.

d) The interaction shall end here for the consumer for the set of subscriptions in the initiating **DEREGISTER Message**.

3.5.6.11.12.5 Message Header

For the **DEREGISTER_ACK Message** the message header fields shall be as defined in 3.4 except for the fields in table 3-42.

Table 3-42: DEREGISTER_ACK Message Header Fields

Field	Value
~~URI~~ From	Broker ~~URI~~
Authentication Id	Broker Authentication Identifier
~~URI~~ To	Consumer ~~URI~~
QoSlevel	QoS level from first REGISTER message
Priority	Priority from first REGISTER message
Interaction Type	PUBSUB
Interaction Stage	8
Transaction Id	Transaction Id from initial message

3.5.6.11.13 PUBLISH_DEREGISTER

3.5.6.11.13.1 Function

a) The **PUBLISH_DEREGISTER Request** primitive shall be used by the provider to end a PUBSUB Interaction.

b) The **PUBLISH_DEREGISTER Indication** primitive shall be used by the broker MAL to deliver a **PUBLISH_DEREGISTER Message** to a broker and end a PUBSUB Interaction for the specified provider.

3.5.6.11.13.2 Semantics

PUBLISH_DEREGISTER Request and **PUBLISH_DEREGISTER Indication** shall provide parameters as follows:

(PUBLISH_DEREGISTER Message Header, QoS properties)

3.5.6.11.13.3 When Generated

a) A **PUBLISH_DEREGISTER Request** shall be used by the provider application with the provider MAL in the **Registered State** to end the interaction.

b) A **PUBLISH_DEREGISTER Indication** shall be generated by the broker MAL in the **Publish Initiated State** upon reception of a **PUBLISH_DEREGISTER Message**, once checked via the Access Control interface, from a provider.

3.5.6.11.13.4 Effect on Reception

a) Reception of a **PUBLISH_DEREGISTER Request** shall, once checked via the Access Control interface, cause the provider MAL in the **Registered State** to request the end of the interaction by transmitting a **PUBLISH_DEREGISTER Message** to the broker.

b) The provider MAL shall enter the **Deregister Initiated State**.

c) On reception of a **PUBLISH_DEREGISTER Message** by the broker MAL, once the message has been checked via the Access Control interface, the broker MAL shall enter the **Publish Deregister Initiated State**.

d) The broker shall then remove the provider from the list of allowed publishers.

3.5.6.11.13.5 Message Header

For the **PUBLISH_DEREGISTER Message** the message header fields shall be as defined in 3.4 except for the fields in table 3-43.

Table 3-43: PUBLISH_DEREGISTER Message Header Fields

Field	Value
~~URI~~ From	Provider ~~URI~~
Authentication Id	Provider Authentication Identifier
~~URI~~ To	Broker ~~URI~~
Interaction Type	PUBSUB
Interaction Stage	9
Transaction Id	Provided by provider MAL

3.5.6.11.14 PUBLISH_DEREGISTER_ACK

3.5.6.11.14.1 Function

a) The **PUBLISH_DEREGISTER_ACK Request** primitive shall be used by the broker to acknowledge the termination of a PUBSUB Interaction by the provider.

b) The **PUBLISH_DEREGISTER_ACK Indication** primitive shall be used by the provider MAL to deliver a **PUBLISH_DEREGISTER_ACK Message** to a provider.

3.5.6.11.14.2 Semantics

PUBLISH_DEREGISTER_ACK Request and **PUBLISH_DEREGISTER_ACK Indication** shall provide parameters as follows:

(PUBLISH_DEREGISTER_ACK Message Header, QoS properties)

3.5.6.11.14.3 When Generated

a) A **PUBLISH_DEREGISTER_ACK Request** shall be used by the broker with the broker MAL in the **Publish Deregister Initiated State** to acknowledge the successful deregistration by a provider in a PUBSUB Interaction.

b) A **PUBLISH_DEREGISTER_ACK Indication** shall be generated by the provider MAL in the **Deregister Initiated State** upon reception of a **PUBLISH_DEREGISTER_ACK Message**, once checked via the Access Control interface, from a broker.

3.5.6.11.14.4 Effect on Reception

a) Reception of a **PUBLISH_DEREGISTER_ACK Request** shall, once checked via the Access Control interface, cause the broker MAL to transmit the supplied acknowledgement as a **PUBLISH_DEREGISTER_ACK Message** to the provider.

b) The interaction shall end at this point for the broker.

c) On reception of a **PUBLISH_DEREGISTER_ACK Message** by the provider MAL in the **Deregister Initiated State**, once the message has been checked via the Access Control interface, the provider MAL shall pass the **PUBLISH_DEREGISTER_ACK Indication** to the provider application.

d) The interaction shall end here for the provider.

3.5.6.11.14.5 Message Header

For the **PUBLISH_DEREGISTER_ACK Message** the message header fields shall be as defined in 3.4 except for the fields in table 3-44.

Table 3-44: PUBLISH_DEREGISTER_ACK Message Header Fields

Field	Value
~~URI~~ From	Broker~~ URI~~
Authentication Id	Broker Authentication Identifier
~~URI~~ To	Provider~~ URI~~
QoSlevel	QoS level from first PUBLISH_REGISTER message
Priority	Priority from first PUBLISH_REGISTER message
Interaction Type	PUBSUB
Interaction Stage	10
Transaction Id	Transaction Id from provider deregister message

3.5.6.12 Example

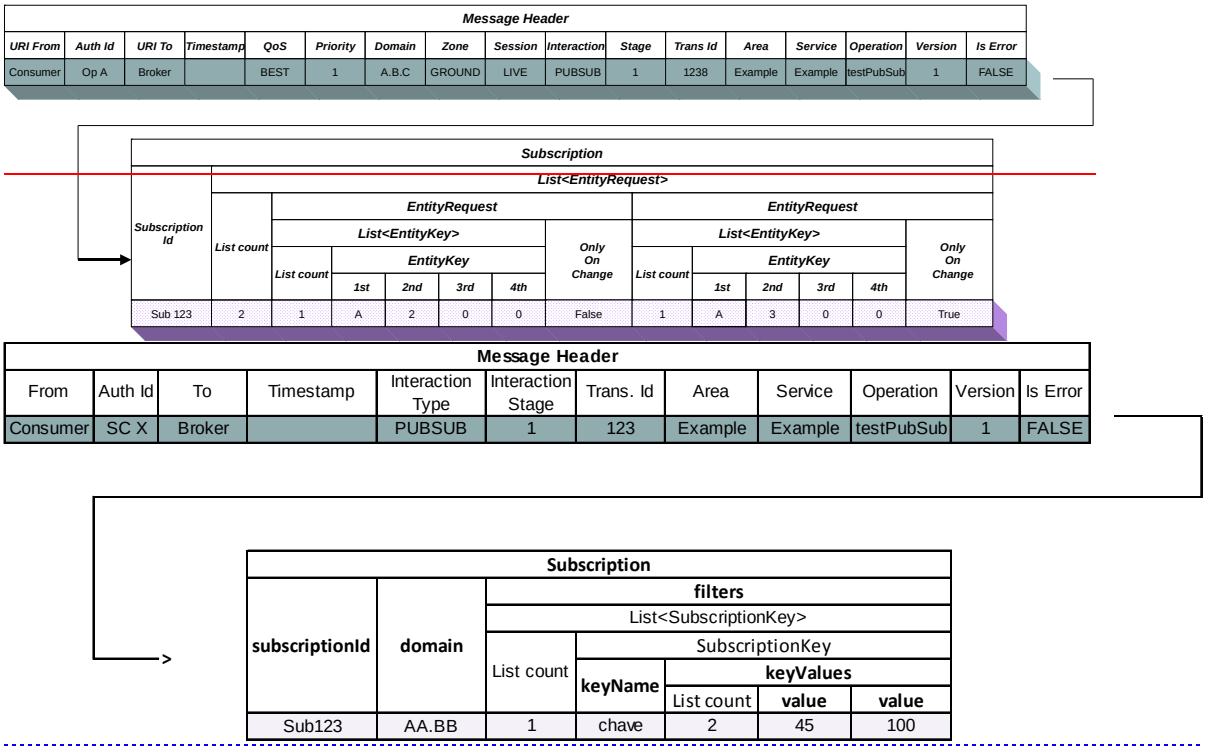
The following example shows a simple example service that contains a single PUBLISH-SUBSCRIBE operation:

Operation Identifier	testPubSub	
Interaction Pattern	PUBLISH-SUBSCRIBE	
Subscription Keys	Identifier key1 Long key2	
Pattern Sequence	Message	Body ~~Type~~Signature
OUT	PUBLISH/NOTIFY	Boolean field1 Integer field2 TestNotify field3

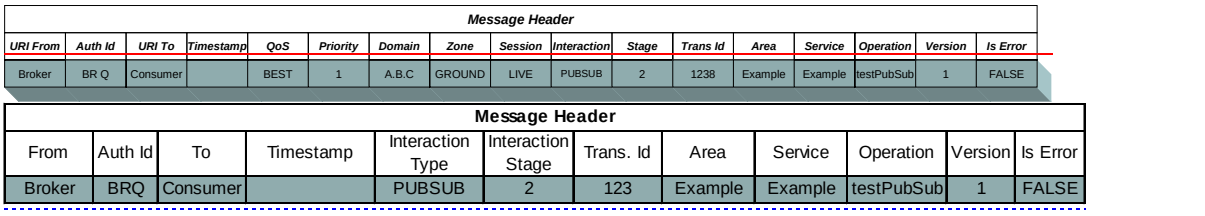
The TestNotify structure is defined below:

Name	TestNotify		
Extends	Composite		
Short Form Part	Example Only		
Field	Type	Nullable	Comment
time	Time	Yes	Example Time item
value	Integer	Yes	Example Integer item

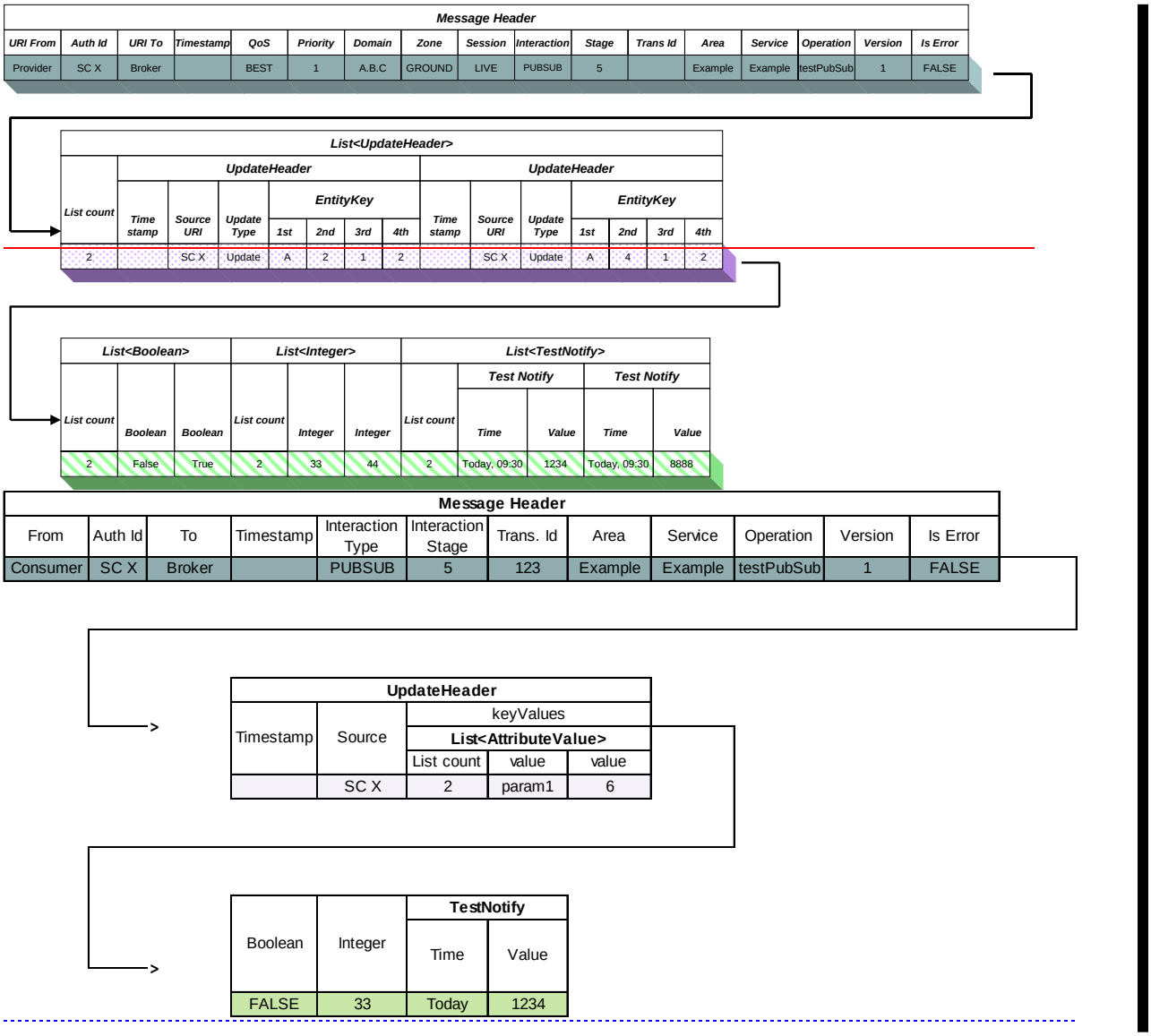
To register for this PUBLISH-SUBSCRIBE pattern, a consumer shall send the following message:



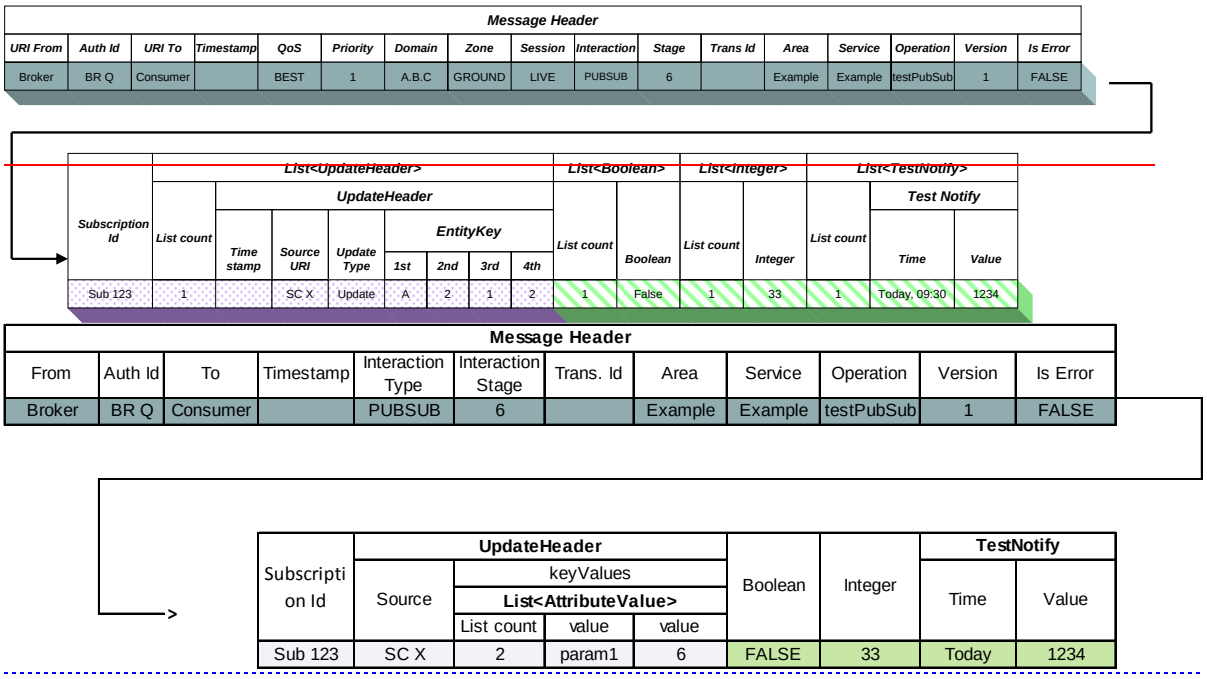
It contains a standard pattern body for the register message. This shall result in the following acknowledgement message being sent:



To publish an update on this topic a previously registered provider would send the following message to the message broker. In this example two updates are being generated at the same time:



When an update is required the following notify message will be sent from the message broker to the consumer:



Message Header																
URI From	Auth Id	URI To	Timestamp	QoS	Priority	Domain	Zone	Session	Interaction	Stage	Trans Id	Area	Service	Operation	Version	Is Error
Broker	BR Q	Consumer		BEST	1	A.B.C	GROUND	LIVE	PUBSUB	6		Example	Example	testPubSub	1	FALSE

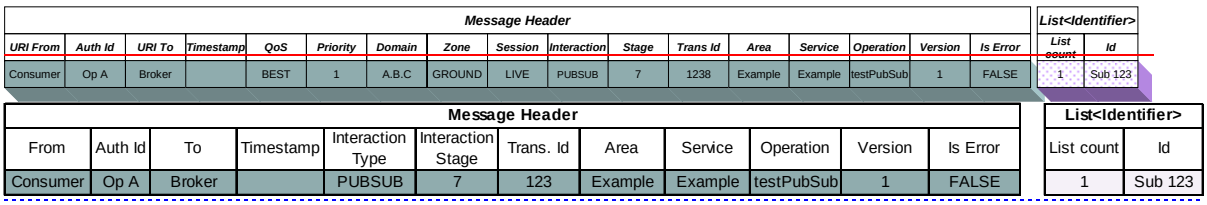
Subscription Id	List<UpdateHeader>								List<Boolean>		List<Integer>		List<TestNotify>		
	List count	UpdateHeader							List count	Boolean	List count	Integer	List count	Test Notify	
		Time stamp	Source URI	Update Type	EntityKey 1st	EntityKey 2nd	EntityKey 3rd	EntityKey 4th						Time	Value
Sub 123	1		SC X	Update	A	2	1	2	1	False	1	33	1	Today, 09:30	1234

Message Header											
From	Auth Id	To	Timestamp	Interaction Type	Interaction Stage	Trans. Id	Area	Service	Operation	Version	Is Error
Broker	BR Q	Consumer		PUBSUB	6		Example	Example	testPubSub	1	FALSE

Subscription Id	UpdateHeader				Boolean	Integer	TestNotify	
	Source	keyValues					Time	Value
		List<AttributeValue>						
		List count	value	value				
Sub 123	SC X	2	param1	6	FALSE	33	Today	1234

No acknowledgment of the notify message shall be sent by the consumer.

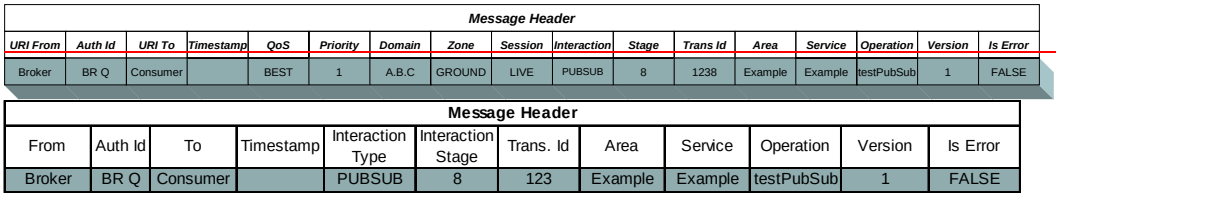
When the consumer wants to deregister, the following message would be sent:



Message Header																	List<Identifier>	
URI From	Auth Id	URI To	Timestamp	QoS	Priority	Domain	Zone	Session	Interaction	Stage	Trans Id	Area	Service	Operation	Version	Is Error	List count	Id
Consumer	Op A	Broker		BEST	1	A.B.C	GROUND	LIVE	PUBSUB	7	1238	Example	Example	testPubSub	1	FALSE	1	Sub 123

Message Header											List<Identifier>	
From	Auth Id	To	Timestamp	Interaction Type	Interaction Stage	Trans. Id	Area	Service	Operation	Version	Is Error	List count	Id
Consumer	Op A	Broker		PUBSUB	7	123	Example	Example	testPubSub	1	FALSE	1	Sub 123

Which would result in the following acknowledgment being received:



Message Header																
URI From	Auth Id	URI To	Timestamp	QoS	Priority	Domain	Zone	Session	Interaction	Stage	Trans Id	Area	Service	Operation	Version	Is Error
Broker	BR Q	Consumer		BEST	1	A.B.C	GROUND	LIVE	PUBSUB	8	1238	Example	Example	testPubSub	1	FALSE

Message Header											
From	Auth Id	To	Timestamp	Interaction Type	Interaction Stage	Trans. Id	Area	Service	Operation	Version	Is Error
Broker	BR Q	Consumer		PUBSUB	8	123	Example	Example	testPubSub	1	FALSE

NOTE – An actual service specification, rather than the example given here, would fully specify the meaning of, and required values of, all structures used by the service.

3.6 ACCESS CONTROL INTERFACE

3.6.1 CHECK MESSAGE INTERACTION

3.6.1.1 Overview

The CHECK pattern is similar to the REQUEST Interaction Pattern: a MAL Message is passed in and the Access Control component responds with a return MAL message (figure 3-26). No acknowledgement other than the response is sent.

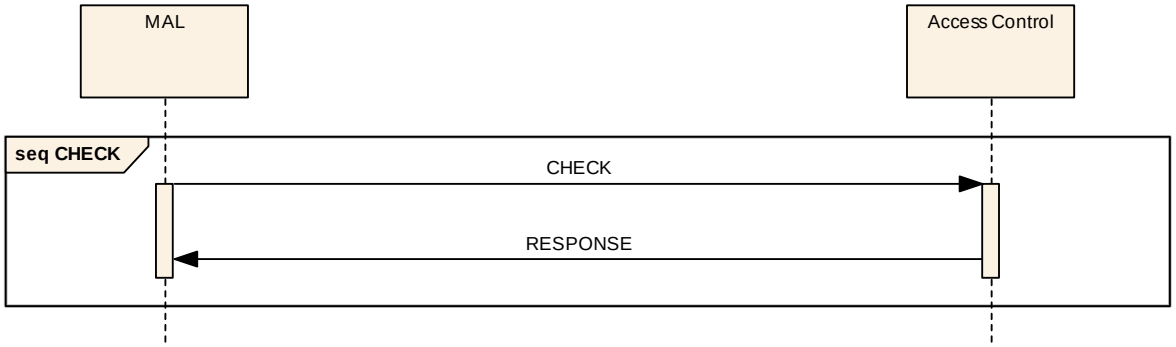


Figure 3-26: CHECK Access Control Pattern Message Sequence

3.6.1.2 Description

The CHECK pattern provides a simple request/response message exchange that is used by a MAL implementation to perform Access Control.

NOTE – It is not expected that this pattern will be implemented via a message transport, as it is expected to be implemented as a local API used by the MAL. It is shown here as a pattern for consistency.

3.6.1.3 Usage

a) The CHECK pattern shall be used only by the MAL for the checking of incoming and outgoing messages. It is not used by any other component.

b) A compliant MAL implementation shall follow the sequences defined in sections 4 and 5 of reference [1] for interacting with an access control component.

c) A broker in a Publish Subscribe pattern shall also submit any NOTIFY and PUBLISH Messages to its Access Control component.

d) Access to sensitive data distributed via the PUBSUB pattern shall be filtered at this point if required.

NOTE – For the restriction of sensitive data it is legitimate for a broker to integrate the CHECK logic into the subscription matching logic.

3.6.1.4 Error Handling

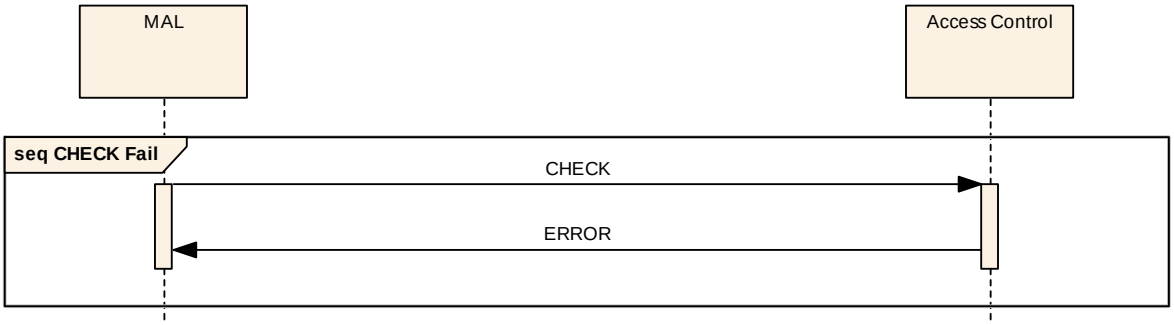


Figure 3-27: CHECK Access Control Pattern Error Sequence

a) The response shall be replaced with an error message (see section 5) if an error occurs during the processing of the operation (figure 3-27).

b) Either the response or the error message shall be returned but never both.

3.6.1.5 Operation Template

The CHECK pattern template is below:

Table 3-45: CHECK Operation Template

Interaction Pattern	CHECK	
Pattern Sequence	Message	Body ~~Type~~Signature
IN	CHECK	MAL message
OUT	RESPONSE	MAL message

3.6.1.6 Primitives

The pattern is not an abstract interface that provides interoperability. However, it is a function that is required by the MAL.

Table 3-46: CHECK Primitive List

Primitive
CHECK
RESPONSE
ERROR

3.6.1.7 State Charts

No state charts are provided for this pattern. The MAL shall initiate the pattern using the CHECK primitive and shall block until either the Response or Error Indication is received.

3.6.1.8 Requests and Indications

3.6.1.8.1 CHECK

3.6.1.8.1.1 Function

The **CHECK Request** primitive shall be used by the MAL to initiate a CHECK Interaction.

3.6.1.8.1.2 Semantics

CHECK Request shall provide parameters as follows:

(MAL Message, QoS properties)

3.6.1.8.1.3 When Generated

a) **CHECK Request** shall be generated by the MAL after reception of a message from either the application or from a transport.

b) It is implementation and deployment specific what checks an implementation of the Access Control component shall perform upon reception of a **CHECK Request**.

3.6.1.8.2 RESPONSE

3.6.1.8.2.1 Function

The **RESPONSE Indication** primitive shall be used by the Access Control component to deliver a **RESPONSE Message** to the MAL.

3.6.1.8.2.2 Semantics

RESPONSE Indication shall provide parameters as follows:

(MAL Message, QoS properties)

3.6.1.8.2.3 When Generated

RESPONSE Indication shall be generated upon reception of a **RESPONSE Message** from the Access Control component.

3.6.1.8.2.4 Effect on Reception

a) On reception of a **RESPONSE Indication** the MAL shall end the interaction pattern with success.

b) The returned message shall then be used by the MAL from that point onwards.

3.6.1.8.3 ERROR

3.6.1.8.3.1 Function

The **ERROR Indication** primitive shall be used by the Access Control component to end a CHECK Interaction with an error.

3.6.1.8.3.2 Semantics

ERROR Indication shall provide parameters as follows:

(Error Number, Extra Information, QoS properties)

3.6.1.8.3.3 When Generated

a) An **ERROR Indication** shall be generated upon reception of an **ERROR Message** from the Access Control component.

b) The Access Control component shall return AUTHENTICATION_FAIL if the supplied message fails authentication checks. These checks are implementation and deployment specific.

c) The Access Control component shall return AUTHORISATION_FAIL if the authenticated supplied message fails authorisation checks. These checks are implementation and deployment specific.

3.6.1.8.3.4 Effect on Reception

a) On reception of an **ERROR Indication** the MAL shall end the interaction with an error.

b) If the Access Control error is to be transmitted to another MAL then the resultant error message shall not be passed to the Access Control component as it originated from that.

c) The behaviour of the MAL from this point onwards is defined in reference [1].

~~3.7 TRANSPORT INTERFACE~~

~~3.7.1 GENERAL~~

~~This subsection defines the abstract interface that a Transport layer provides to the MAL. It specifies what facilities must be made available to a compliant MAL and also the required behaviour of the Transport.~~

~~The specification of this for a particular technology is called a Transport Specification. It defines the mapping from the abstract MAL data structures into a specific and unambiguous encoding of the messages for that specific data transport. (For further information on this, see reference [1].)~~

~~a) A Transport Specification shall list, for each supported QoS, the MAL errors it may raise.~~

~~b) A Transport Specification shall define, for each supported QoS, the conditions which trigger the raising of the listed MAL errors.~~

~~c) A Transport Specification shall define, for each supported QoS, the QoS properties it supports.~~

~~d) A Transport Specification shall define, for each supported QoS, the effect the QoS properties have on its behaviour.~~

~~e) A Transport Specification shall define, for each supported QoS, the message timeout behaviour.~~

~~f) If Transport Specification supports the QUEUED QoS, then it shall define its behaviour in terms of message persistence and purging.~~

3.7.2 SUPPORTEDQOS INTERACTION

3.7.2.1 Overview

The SUPPORTEDQOS pattern is similar to the REQUEST Interaction Pattern. No acknowledgement other than the response is sent (figure 3-28).

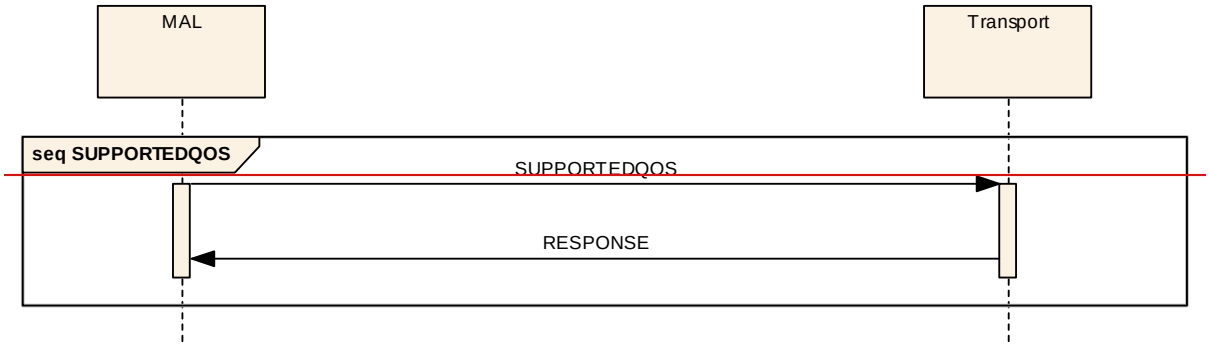


Figure 3-28: SUPPORTEDQOS Transport Pattern Message Sequence

3.7.2.2 Description

The SUPPORTEDQOS pattern provides a simple request/response message exchange that is used by a MAL implementation to determine which QoS levels are supported by a specific Transport layer implementation.

NOTE – It is not expected that this pattern will be implemented via a message, as it is expected to be implemented as a local API used by the MAL to access the Transport layer. It is shown here as a pattern for consistency.

3.7.2.3 Usage

The SUPPORTEDQOS pattern shall be used only by the MAL for the checking of supported QoS levels. It is not used by any other component.

3.7.2.4 Error Handling

No Errors shall be raised.

3.7.2.5 Operation Template

The SUPPORTEDQOS pattern template is below:

Table 3-47: SUPPORTEDQOS Operation Template

Interaction Pattern	SUPPORTEDQOS	
Pattern Sequence	Message	Body Type
IN	SUPPORTEDQOS	QoSLevel
OUT	RESPONSE	Boolean

3.7.2.6 Primitives

The pattern is not an abstract interface that provides interoperability. However, it is a function that is required by the MAL.

Table 3-48: SUPPORTEDQOS Primitive List

Primitive
SUPPORTEDQOS
RESPONSE

3.7.2.7 State Charts

No state charts are provided for this pattern. The MAL shall initiate the pattern using the initial primitive and shall block until a response is received.

3.7.2.8 Requests and Indications

3.7.2.8.1 SUPPORTEDQOS

3.7.2.8.1.1 Function

The **SUPPORTEDQOS Request** primitive shall be used by the MAL to determine if a Transport supports a specific QoS level.

3.7.2.8.1.2 Semantics

SUPPORTEDQOS Request shall provide parameters as follows:

(QoSLevel)

~~**3.7.2.8.1.3 When Generated**~~

~~**SUPPORTEDQOS Request** shall be generated by the MAL when first interacting with a specific Transport.~~

~~**3.7.2.8.1.4 Effect on Reception**~~

~~Reception of a **SUPPORTEDQOS Request** shall result in a return of a **RESPONSE Indication**.~~

~~**3.7.2.8.2 RESPONSE**~~

~~**3.7.2.8.2.1 Function**~~

~~a) The **RESPONSE Indication** primitive shall be used by a Transport layer to deliver the response to a **SUPPORTEDQOS Request** to the MAL.~~

~~b) If the Transport supports the QoS level it shall return 'True' in the indication, 'False' otherwise.~~

~~**3.7.2.8.2.2 Semantics**~~

~~**RESPONSE Indication** shall provide parameters as follows:~~

~~(Boolean)~~

~~**3.7.2.8.2.3 When Generated**~~

~~**RESPONSE Indication** shall be generated by the Transport layer in response to a **SUPPORTEDQOS Request**.~~

~~**3.7.2.8.2.4 Effect on Reception**~~

~~The effect on reception of a **RESPONSE Indication** by the MAL is MAL implementation specific.~~

3.7.3 SUPPORTEDIP INTERACTION

3.7.3.1 Overview

The SUPPORTEDIP pattern is similar to the REQUEST Interaction Pattern. No acknowledgement other than the response is sent (figure 3-29).

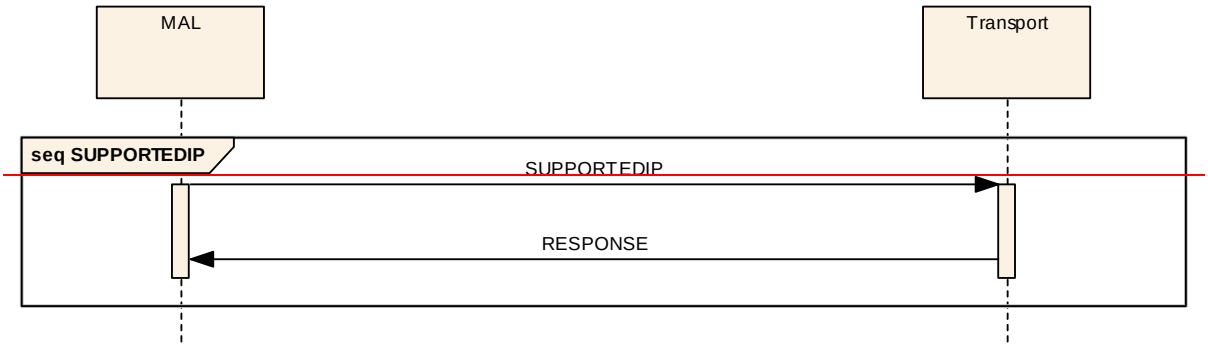


Figure 3-29: SUPPORTEDIP Transport Pattern Message Sequence

3.7.3.2 Description

The SUPPORTEDIP pattern provides a simple request/response message exchange that is used by a MAL implementation to determine which MAL Interaction Patterns are supported by a specific Transport layer implementation.

NOTE It is not expected that this pattern will be implemented via a message, as it is expected to be implemented as a local API used by the MAL to access the Transport layer. It is shown here as a pattern for consistency.

3.7.3.3 Usage

The SUPPORTEDIP pattern is only used by the MAL for the checking of supported interaction patterns. It is not expected to be used by any other component.

3.7.3.4 Error Handling

No Errors shall be raised.

3.7.3.5 Operation Template

The SUPPORTEDIP pattern template is below:

Table 3-49: SUPPORTEDIP Operation Template

Interaction Pattern	SUPPORTEDIP	
Pattern Sequence	Message	Body Type
IN	SUPPORTEDIP	InteractionType
OUT	RESPONSE	Boolean

3.7.3.6 Primitives

The pattern is not an abstract interface that provides interoperability. However, it is a function that is required by the MAL.

Table 3-50: SUPPORTEDIP Primitive List

Primitive
SUPPORTEDIP
RESPONSE

3.7.3.7 State Charts

No state charts are provided for this pattern. The MAL shall initiate the pattern using the initial primitive and block until a response is received.

3.7.3.8 Requests and Indications

3.7.3.8.1 SUPPORTEDIP

3.7.3.8.1.1 Function

The **SUPPORTEDIP Request** primitive shall be used by the MAL to determine if a Transport supports a specific interaction pattern.

3.7.3.8.1.2 Semantics

SUPPORTEDIP Request shall provide parameters as follows:

(InteractionType)

~~**3.7.3.8.1.3 When Generated**~~

~~**SUPPORTEDIP Request** is generated by the MAL when first interacting with a specific Transport.~~

~~**3.7.3.8.1.4 Effect on Reception**~~

~~Reception of a **SUPPORTEDIP Request** shall result in a return of a **RESPONSE Indication**.~~

~~**3.7.3.8.2 RESPONSE**~~

~~**3.7.3.8.2.1 Function**~~

~~a) The **RESPONSE Indication** primitive shall be used to deliver the response to a **SUPPORTEDIP Request** to the MAL.~~

~~b) If the Transport supports the interaction pattern it shall return 'True' in the indication, 'False' otherwise.~~

~~c) It is expected that Transports shall return 'True' for interaction patterns SEND, SUBMIT, REQUEST, INVOKE, and PROGRESS, as these do not require any special processing by the Transport layer.~~

~~d) Transports shall only return 'True' for support of the PUBLISH-SUBSCRIBE interaction pattern if they support the broker aspect natively.~~

~~**3.7.3.8.2.2 Semantics**~~

~~**RESPONSE Indication** shall provide parameters as follows:~~

~~(Boolean)~~

~~**3.7.3.8.2.3 When Generated**~~

~~**RESPONSE Indication** shall be generated by the Transport layer in response to a **SUPPORTEDIP Request**.~~

~~**3.7.3.8.2.4 Effect on Reception**~~

~~On reception of a **RESPONSE Indication** with the 'False' value for the PUBLISH-SUBSCRIBE pattern, the MAL shall implement the PUBLISH-SUBSCRIBE pattern internally.~~

3.7.4 TRANSMIT INTERACTION

3.7.4.1 Overview

The TRANSMIT pattern (figure 3-30) is similar to the SUBMIT Interaction Pattern.

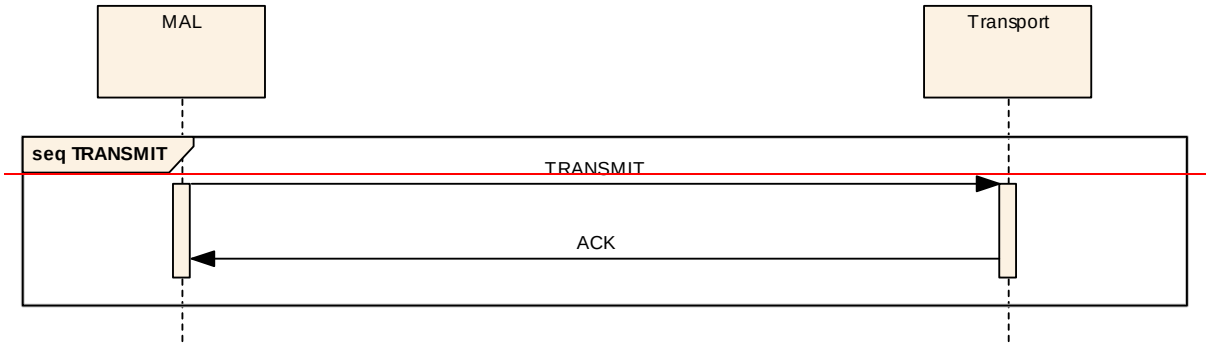


Figure 3-30: TRANSMIT Transport Pattern Message Sequence

3.7.4.2 Description

The TRANSMIT pattern provides a simple transmit/acknowledgement message exchange that is used by a MAL implementation to pass MAL messages to the Transport layer for transmission.

NOTE – It is not expected that this pattern will be implemented via a message, as it is expected to be implemented as a local API used by the MAL to access the Transport layer. It is shown here as a pattern for consistency.

3.7.4.3 Usage

The TRANSMIT pattern is only expected to be used by the MAL for the transmission of MAL Messages. It not expected be used by any other component.

3.7.4.4 Error Handling

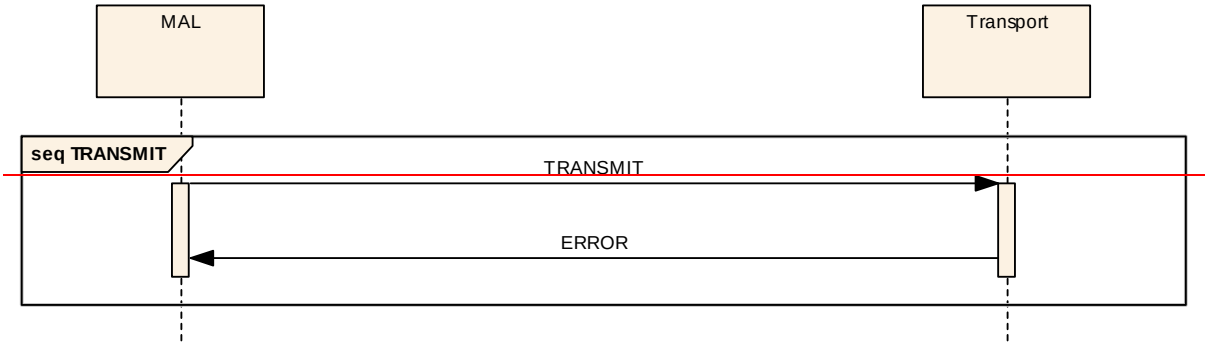


Figure 3-31: TRANSMIT Transport Pattern Error Sequence

a) The acknowledgement shall be replaced with an error message (see section 5) if an error occurs during the processing of the operation (figure 3-31).

b) Either the acknowledgement or the error message shall be returned but never both.

3.7.4.5 Operation Template

The TRANSMIT pattern template is below:

Table 3-51: TRANSMIT Operation Template

Interaction Pattern	TRANSMIT	
Pattern Sequence	Message	Body Type
IN	TRANSMIT	MAL Message
OUT	ACK	

3.7.4.6 Primitives

The pattern is not an abstract interface that provides interoperability. However, it is a function that is required by the MAL.

Table 3-52: TRANSMIT Primitive List

Primitive
TRANSMIT
ACK
ERROR

3.7.4.7 State Charts

No state charts are provided for this pattern. The MAL shall initiate the pattern using the initial primitive and block until a response is received.

3.7.4.8 Requests and Indications

3.7.4.8.1 TRANSMIT

3.7.4.8.1.1 Function

The **TRANSMIT Request** primitive shall be used by the MAL to pass a MAL Message to the Transport layer for transmission.

3.7.4.8.1.2 Semantics

TRANSMIT Request shall provide parameters as follows:

(MAL Message, QoS properties)

3.7.4.8.1.3 When Generated

TRANSMIT Request shall be generated by the MAL when using the Transport layer to transmit messages.

3.7.4.8.1.4 Effect on Reception

a) On reception of a **TRANSMIT Request** a Transport layer shall return an **ACK Indication** if the transmission of the message is successful as far as can be determined initially by the Transport.

b) The determination of success is Transport specific. For example, for a message broker-based transport, this may mean successful transmission to the local broker.

~~**3.7.4.8.2 ACK**~~

~~**3.7.4.8.2.1 Function**~~

~~The **ACK Indication** primitive shall be used by the Transport layer to indicate to the MAL a successful transmission of a message in response to a **TRANSMIT Request**.~~

~~**3.7.4.8.2.2 Semantics**~~

~~**ACK Indication** does not use any parameters.~~

~~**3.7.4.8.2.3 When Generated**~~

~~An **ACK Indication** shall be generated by the Transport layer in response to a **TRANSMIT Request** when the request was successful.~~

~~**3.7.4.8.2.4 Effect on Reception**~~

~~The effect on reception of an **ACK Indication** by the MAL is defined in reference [1].~~

~~**3.7.4.8.3 ERROR**~~

~~**3.7.4.8.3.1 Function**~~

~~The **ERROR Indication** primitive shall be used by the Transport layer to deliver a transmission failure **ERROR Message** to the MAL.~~

~~**3.7.4.8.3.2 Semantics**~~

~~**ERROR Indication** shall provide parameters as follows:~~

~~(ERROR Message Header, Error Number, Extra Information, QoS properties)~~

~~**3.7.4.8.3.3 When Generated**~~

~~An **ERROR Indication** shall be generated by the Transport layer if there is an error during transmission.~~

~~**3.7.4.8.3.4 Effect on Reception**~~

~~The effect on reception of an **ERROR Indication** by the MAL is defined in reference [1].~~

3.7.5 TRANSMITMULTIPLE INTERACTION

3.7.5.1 Overview

The TRANSMITMULTIPLE pattern (figure 3-32) is similar to the SUBMIT Interaction Pattern.

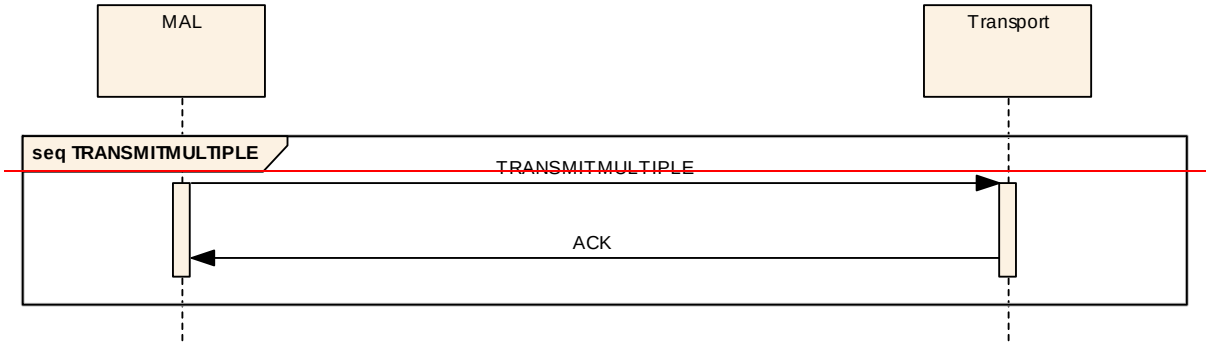


Figure 3-32: TRANSMITMULTIPLE Transport Pattern Message Sequence

3.7.5.2 Description

The TRANSMITMULTIPLE pattern provides a simple transmit/acknowledgement message exchange that is used by a MAL implementation to pass a set of MAL messages to the Transport layer for transmission.

NOTE — It is not expected that this pattern will be implemented via a message, as it is expected to be implemented as a local API used by the MAL to access the Transport layer. It is shown here as a pattern for consistency.

3.7.5.3 Usage

The TRANSMITMULTIPLE pattern is only expected to be used by the MAL for the transmission of MAL Messages. It is not expected to be used by any other component.

3.7.5.4 Error Handling

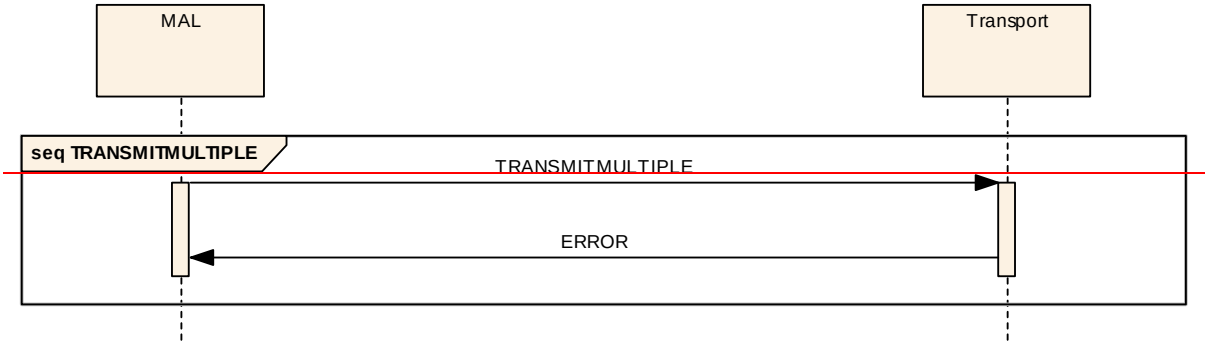


Figure 3-33: TRANSMITMULTIPLE Transport Pattern Error Sequence

a) The acknowledgement shall be replaced with an error message (see section 5) if an error occurs during the processing of the operation (figure 3-33).

b) Either the acknowledgement or the error message shall be returned but never both.

3.7.5.5 Operation Template

The TRANSMITMULTIPLE pattern template is below:

Table 3-53: TRANSMITMULTIPLE Operation Template

Interaction Pattern	TRANSMITMULTIPLE	
Pattern Sequence	Message	Body Type
IN	TRANSMITMULTIPLE	MAL Message List
OUT	ACK	

3.7.5.6 Primitives

The pattern is not an abstract interface that provides interoperability. However, it is a function that is required by the MAL.

Table 3-54: TRANSMITMULTIPLE Primitive List

Primitive
TRANSMITMULTIPLE
ACK
ERROR

3.7.5.7 State Charts

No state charts are provided for this pattern. The MAL shall initiate the pattern using the initial primitive and block until a response is received.

3.7.5.8 Requests and Indications

3.7.5.8.1 TRANSMITMULTIPLE

3.7.5.8.1.1 Function

The **TRANSMITMULTIPLE Request** primitive shall be used by the MAL to pass a list of MAL Messages to the Transport layer for transmission.

3.7.5.8.1.2 Semantics

TRANSMITMULTIPLE Request shall provide parameters as follows:

List of (MAL Message, QoS properties)

3.7.5.8.1.3 When Generated

a) A **TRANSMITMULTIPLE Request** shall be generated by the MAL when using the Transport layer to transmit messages.

b) The pattern may be used when using a MAL-level broker for a Publish/Subscribe Notify message, i.e., one Notify for each consumer.

3.7.5.8.1.4 Effect on Reception

a) On reception of a **TRANSMITMULTIPLE Request** a Transport layer shall return an **ACK Indication** if the transmission of the messages are all successful as far as can be determined initially.

b) The determination of success shall be Transport specific. For example, for a message broker-based transport, this may mean successful transmission to the local broker.

~~**3.7.5.8.2 ACK**~~

~~**3.7.5.8.2.1 Function**~~

~~The **ACK Indication** primitive shall be used by the Transport layer to indicate to the MAL a successful transmission of a list of messages in response to a **TRANSMITMULTIPLE Request**.~~

~~**3.7.5.8.2.2 Semantics**~~

~~**ACK Indication** does not use any parameters.~~

~~**3.7.5.8.2.3 When Generated**~~

~~An **ACK Indication** shall be generated by the Transport layer in response to a **TRANSMITMULTIPLE Request** when the request was successful.~~

~~**3.7.5.8.2.4 Effect on Reception**~~

~~The effect on reception of an **ACK Indication** by the MAL shall be as defined for the TRANSMIT pattern **ACK Indication**.~~

~~**3.7.5.8.3 ERROR**~~

~~**3.7.5.8.3.1 Function**~~

~~The **ERROR Indication** primitive shall be used by the Transport layer to deliver a transmission failure **ERROR Message** to the MAL.~~

~~**3.7.5.8.3.2 Semantics**~~

~~**ERROR Indication** shall provide parameters as follows:~~

~~List of (ERROR Message Header, Error Number, Extra Information, QoS properties)~~

~~**3.7.5.8.3.3 When Generated**~~

~~An **ERROR Indication** shall be generated by the Transport layer if there is an error during transmission of one or more of the messages.~~

3.7.5.8.3.4 Effect on Reception

a) The effect on reception of an **ERROR Indication** by the MAL shall be as defined for the TRANSMIT pattern **ERROR Indication**.

b) The MAL shall assume a successful transmission for all other messages.

3.7.6 RECEIVE INTERACTION

3.7.6.1 Overview

The RECEIVE pattern is a simple one-way pattern where the Transport layer passes a received message to the MAL (figure 3-34). No acknowledgement is sent by the MAL.

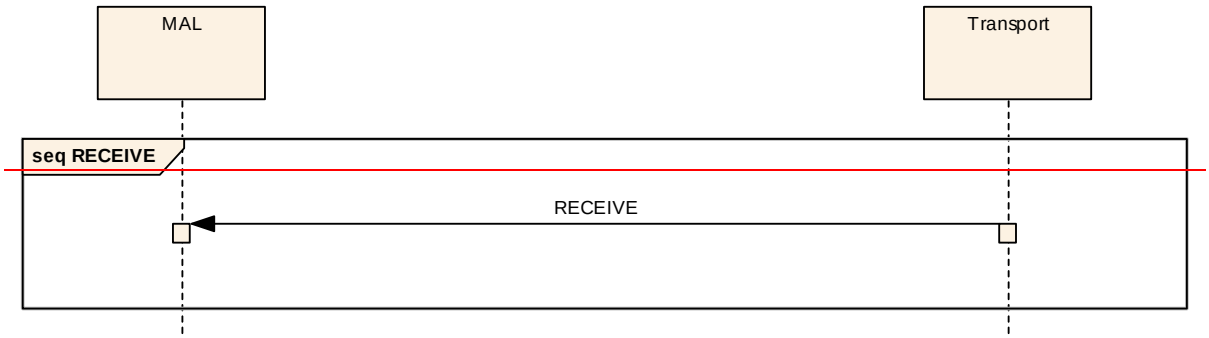


Figure 3-34: RECEIVE Transport Pattern Message Sequence

3.7.6.2 Description

The RECEIVE pattern provides a simple mechanism that is used by the Transport layer to pass a received message to the MAL.

NOTE It is not expected that this pattern will be implemented via a message, as it is expected to be implemented as a local API used by the MAL to access the Transport layer. It is shown here as a pattern for consistency.

3.7.6.3 Usage

The RECEIVE pattern is only expected to be used by the Transport layer for the passing of message to the MAL. It is not expected to be used by any other component.

3.7.6.4 Error Handling

No Errors shall be raised.

3.7.6.5 Operation Template

The RECEIVE pattern template is below:

Table 3-55: RECEIVE Operation Template

Interaction Pattern	RECEIVE	
Pattern Sequence	Message	Body Type
OUT	RECEIVE	MAL Message

3.7.6.6 Primitives

The pattern is not an abstract interface that provides interoperability. However, it is a function that is required by the MAL.

Table 3-56: RECEIVE Primitive List

Primitive
RECEIVE

3.7.6.7 State Charts

No state charts are provided for this pattern. The Transport layer shall initiate the pattern using an implementation-dependent mechanism, and the MAL is notified via the single Indication.

3.7.6.8 Requests and Indications

3.7.6.8.1 RECEIVE

3.7.6.8.1.1 Function

The **RECEIVE Indication** primitive shall be used by the Transport layer to deliver an incoming message to the MAL.

3.7.6.8.1.2 Semantics

RECEIVE Indication shall provide parameters as follows:

(MAL Message, QoS properties)

3.7.6.8.1.3 When Generated

RECEIVE Indication shall be generated in either one of two situations:

a) upon reception of a message by the Transport layer (this may be an error message);

b) by the Transport layer directly in case of an error in response to a **TRANSMIT Request** or **TRANSMITMULTIPLE Request**. In this case the Transport layer shall create the appropriate MAL Interaction Pattern Error response.

3.7.6.8.1.4 Effect on Reception

The effect on reception of a **RECEIVE Indication** by the MAL is defined in reference [1].

3.7.7 RECEIVEMULTIPLE INTERACTION

3.7.7.1 Overview

The RECEIVEMULTIPLE pattern is a simple one-way pattern where the Transport layer passes a set of received messages to the MAL (figure 3-35). No acknowledgement is sent by the MAL.

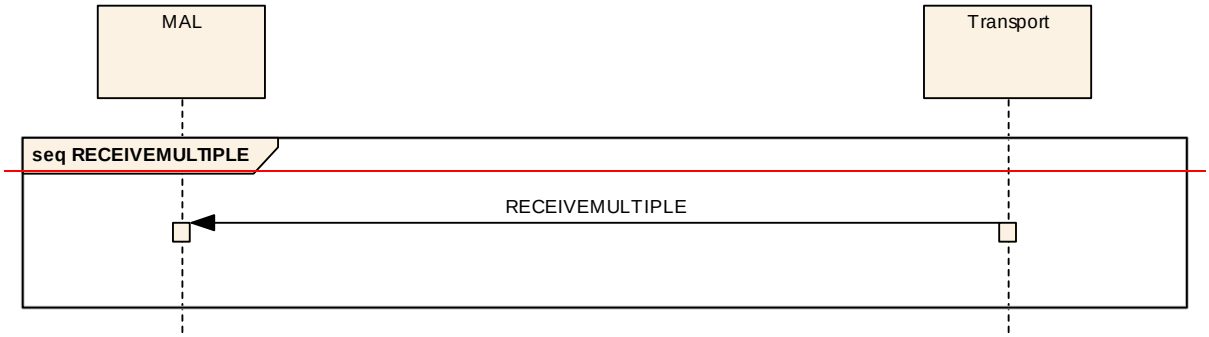


Figure 3-35: RECEIVEMULTIPLE Transport Pattern Message Sequence

3.7.7.2 Description

The RECEIVEMULTIPLE pattern provides a simple mechanism that is used by the Transport layer to pass a set of received messages to the MAL.

NOTE – It is not expected that this pattern will be implemented via a message, as it is expected to be implemented as a local API used by the MAL to access the Transport layer. It is shown here as a pattern for consistency.

3.7.7.3 Usage

The RECEIVEMULTIPLE pattern is only expected to be used by the Transport layer for the passing of messages to the MAL. It is not expected to be used by any other component.

3.7.7.4 Error Handling

No Errors shall be raised.

3.7.7.5 Operation Template

The RECEIVEMULTIPLE pattern template is below:

Table 3-57: RECEIVEMULTIPLE Operation Template

Interaction Pattern	RECEIVEMULTIPLE	
Pattern Sequence	Message	Body Type
OUT	RECEIVEMULTIPLE	MAL Message List

3.7.7.6 Primitives

The pattern is not an abstract interface that provides interoperability. However, it is a function that is required by the MAL.

Table 3-58: RECEIVEMULTIPLE Primitive List

Primitive
RECEIVEMULTIPLE

3.7.7.7 State Charts

No state charts are provided for this pattern. The Transport layer shall initiate the pattern using an implementation dependent mechanism and the MAL is notified via the single Indication.

~~**3.7.7.8 Requests and Indications**~~

~~**3.7.7.8.1 RECEIVEMULTIPLE**~~

~~**3.7.7.8.1.1 Function**~~

~~The **RECEIVEMULTIPLE Indication** primitive shall be used by the Transport layer to deliver a list of incoming messages to the MAL.~~

~~**3.7.7.8.1.2 Semantics**~~

~~**RECEIVEMULTIPLE Indication** shall provide parameters as follows:~~

~~List of (Header, Message Body, QoS properties)~~

~~**3.7.7.8.1.3 When Generated**~~

~~A **RECEIVEMULTIPLE Indication** shall be generated upon reception of a set of messages by the Transport layer (this may include error messages).~~

~~**3.7.7.8.1.4 Effect on Reception**~~

~~The **RECEIVEMULTIPLE Indication** is a convenience pattern that allows a Transport layer implementation to pass a set of messages in one interaction rather than in multiple interactions using the **RECEIVE Interaction.** The MAL shall treat each message received as if it had received each one individually.~~

3.7 MO OBJECTS

3.7.1 OVERVIEW

In order to specify complex data models within MO services, the Recommended Standard introduces the concept of MO Objects.

An MO Object is an extension of a Composite data type with the first field being its identity. The MAL::Object type shall be used in order to specify MO Objects and its respective data structures. This data type allows the creation of data structures that are MO Objects.

The first field of the MO Object is its identity represented by the ObjectIdentity MAL type. This enables the possibility of uniquely identify an MO Object from other data entities. The identity of an MO Object is represented in more detail in the next section.

MO Objects are entities that can be explicitly referenced in other Composite data structures with the use of the ObjectRef MAL data type. The ObjectRef type shall be represented in specifications as ObjectRef<T> as presented in 4.1.6.3.

The ability to uniquely identify, and reference MO Objects allows the creation of complex data models on the MO services Recommended Standards.

3.7.2 OBJECT IDENTITY

Each MO Object has an Object Identity which allows them to be uniquely identified across different domains. The Object Identity is composed of a Domain, an Area, a Type, a Key, and an Object Version.

Table 3-47: Object Identity Fields

Field	Type	Value
Domain	List<Identifier>	Domain of the object
Area	Identifier	Area of the object
Type	Identifier	Type of the object
Key	Identifier	Key of the Object
Object Version	UInteger	Version of the Object, starting at 1

An ObjectIdentity MAL Composite is defined in this Recommended Standard and it includes the exact same fields of the Object Identity as presented above. This allows its direct usage in MAL Composite data structures in order to represent the identity of a certain MO Object.

The Domain shall be formed as follows:

a) The Domain shall be defined as a list of identifiers.

b) The most significant domain part shall be listed first in the list (for example, Agency), and each subsequent domain identifier in the list narrows the preceding domain.

c) Each Identifier part of the Domain shall be allowed the full range of Identifier values with the restriction that it is NOT allowed to contain the '.' character.

The combination of the fields Area and Type shall be used to determine the structure of the Object Body.

The Key field is unique within the scope of the Domain, for a given Area and Type.

If an MO Object does not have versions, then the Version field shall be populated with a '1'.

3.7.3 REPRESENTATION EXAMPLE

As previously defined, an MO Object can be defined using the MAL::Object data type. The following table presents an example of how an MO Object is expected to be represented in the specifications.

The MO Object is called ExampleStructure and includes a first field with an ObjectIdentity MAL type, and two additional example fields.

Name	ExampleStructure		
Extends	Object		
Short Form Part	123		
Field	**Type**	**Nullable**	**Comment**
identity	ObjectIdentity	No	The identity of this MO Object.
myField1	Integer	Yes	An example of a field in the MO Object.
myField2	String	Yes	An example of a field in the MO Object.

4 MAL DATA TYPE SPECIFICATION

4.1 OVERVIEW

4.1.1 GENERAL

The specification of the abstract interfaces and services details the structures passed as the message bodies and message returns of the interaction patterns and operations. This section details the MAL data type specification, the rules for its use with the MAL, the types and structures it contains, and the rules allowed for the combination of these.

Other data type specification languages (such as XML Schema) may be used to specify the message bodies in service specifications; however, support for these other data type specification languages in a particular technology mapping may not be universal and may limit use of a specification to specific deployments (such as ground only).

4.1.2 ABSTRACT AND CONCRETE TYPES

The MAL type specification allows the definition of two basic categories of types, the first being abstract types, and the second being concrete types.

Abstract types are represented in a table as illustrated below:

Name	<<Name>>
Extends	<<Extension Type Name>>
Abstract	

The name of the type is given in the Name field and the name of the abstract type that this type extends is given in the ‘Extends’ field.

Concrete types are represented in a table as illustrated below:

Name	<<Name>>
Extends	<<Extension Type Name>>
Short Form Part	<<Short Form Part>>

Each concrete type is defined with a numerical short form part that is expected to be used by efficient encodings when required to hold type information. Each specification that defines types shall number each concrete type in the appropriate field starting from 1.

To avoid number conflicts between types defined in different areas~~/services~~, the absolute short form of the type is obtained by combining the area number (as a UShort), the ~~service number~~area version (as a UShort), and the short form part from the type definition (as an Integer)~~, and the area version. If the type does not belong to a service, but is defined at the area level, then the service UShort shall be set to zero~~.

In the context of the MAL specification the area number of '1' is used.

4.1.3 FUNDAMENTALS

The base type for all types and structures is Element. Two other fundamental types exist, Composite and Attribute. Only the MAL specification (this document) is allowed to define fundamental types.

Fundament types are represented in a table as illustrated below:

Name	Element
Extends	
Abstract	

4.1.4 ATTRIBUTES

Attributes are the simplest MAL type; they cannot be decomposed into any smaller elements and are used to build more complex structures.

Only the MAL specification (this document) is allowed to define attribute types.

Attribute types are represented in a table as illustrated below:

Name	<<Attribute Name>>
Extends	Attribute
Short Form Part	<<Short Form Part>>

All attribute types extend the fundamental type 'Attribute'. They are also defined in 4.2.4, but the actual representation, or encoding, of them is completely dependent on the language and transport/encoding mapping used. However, because the limits and behaviour of the types are constant (defined here), the informational content is preserved when moving between mappings.

For example, a Boolean value may be represented by a single bit in some encodings or by the text strings 'True/False' in others; however, the meaning of the value is identical regardless of the encoding used.

4.1.5 COMPOSITES

4.1.5.1 General

Composites are represented in a table as illustrated below:

Name	TestBody		
Extends	Composite		
Short Form Part	123		
Field	Type	Nullable	Comment
FirstItem	String	Yes	Example String item.
SecondItem	Integer	Yes	Example Integer item.

Name	ExampleStructure		
Extends	Composite		
Short Form Part	456		
Field	Type	Nullable	Comment
first_item	String	Yes	Example String item.
second_item	Integer	Yes	Example Integer item.
third_item	TestBody	Yes	Contained structure.

The ExampleStructure would decompose to a sequence of:

Field	Type	Comment
first_item	String	from ExampleStructure.
second_item	Integer	from ExampleStructure.
FirstItem	String	from contained TestBody.
SecondItem	Integer	from contained TestBody.

The Nullable' column indicates whether the field is permitted to contain a Null value. If the field is set to Yes then it is allowed to contain a Null value, if it is No then it is not.

~~4.1.6 COMPOSITE EXTENSION~~

~~A Composite can extend another Composite like below (multiple extension is not permitted nor is extension of a non-abstract composite):~~

4.1.5.2 Polymorphism—Composite Extension

Composite structures can be extended; however, there are some restrictions:

- an abstract Composite can be extended by another abstract Composite;
- an abstract Composite can be extended by a concrete Composite;
- a concrete Composite can only extend one unique abstract Composite; therefore, a concrete Composite cannot extend multiple Composites at the same time;
- a concrete Composite cannot be extended further.

NOTE – It is not permitted to extend a non-abstract composite as an issue arises when the extended composite is used in place of the base composite (which is not marked as abstract). Because there is no indication to the receiving application that the message contains extra information, it is not possible to decode the message correctly. Some transport encodings may be able to support this, but many will not be able to. Therefore only abstract composites shall be extended.

The following structures present one example of an abstract Composite being extended by a concrete Composite.

Name	AbstractComposite		
Extends	Composite		
Abstract			
Field	Type	Nullable	Comment
FirstItem	String	Yes	Example String item.
SecondItem	Integer	Yes	Example Integer item.

Name	ComplexStructure		
Extends	**AbstractComposite**		
Short Form Part	345		
Field	Type	Nullable	Comment
extra_item	Boolean	Yes	Extra Boolean item.
second_item	Integer	Yes	Example Integer item.
third_item	TestBody	Yes	Contained structure.

This shows that ComplexStructure can be considered an extension of AbstractComposite and contains all the contents of AbstractComposite plus its own extra items:

Field	Type	Comment
FirstItem	String	from AbstractComposite.
SecondItem	Integer	from AbstractComposite.
extra_item	Boolean	from ComplexStructure.
second_item	Integer	from ComplexStructure.
FirstItem	String	from ComplexStructure contained TestBody.
SecondItem	Integer	from ComplexStructure contained TestBody.

~~It is not permitted to extend a non-abstract composite as an issue arises when the extended composite is used in place of the base composite (which is not marked as abstract). Because there is no indication to the receiving application that the message contains extra information, it is not possible to decode the message correctly. Some transport encodings may be able to support this, but many will not be able to. Therefore only abstract composites shall be extended.~~

~~4.1.6 CONTAINING ABSTRACT ELEMENTS~~

~~The MAL data type specification does not permit composites to contain an abstract type; all fields of a composite must be concrete types. It is permitted to contain a concrete type that extends an abstract type.~~

~~The exception to this is the containment of the abstract Attribute type. It is permitted to create composites that have fields defined as Attribute. This is because the set of possible Attribute types is fixed.~~

NOTE – Good Practices for Service Design:

A sound architectural design favours the ‘composition over inheritance’ principle (or composite reuse principle). This principle states that polymorphic behaviour and code reuse should be achieved by its composition (by containing other composite structures) rather than inheritance from a base or parent structure. This is an often-stated principle in object-oriented programming. MO Architects are advised to take this principle into consideration when defining new services.

4.1.5.3 Polymorphism—Abstract Elements in Composites

The MAL data type specification does allow composites to contain an abstract type. The fields within a composite can be abstract or concrete types. When defining an abstract field within a composite, the composite itself does not become abstract.

This includes the containment of the abstract Attribute type, and abstract Composite type. It is also permitted to create composites that have fields defined as Attribute.

For example the following is permitted:

Name	ConcreteCompositeA		
Extends	Composite		
Short Form Part	654321		
Field	Type	Nullable	Comment
SomeItem	Attribute	Yes	Example attribute item.

~~However, the following is not permitted:~~

And the following is also permitted:

Name	ConcreteCompositeB		
Extends	Composite		
Short Form Part	123456		
Field	Type	Nullable	Comment
AnotherItem	~~Element~~TheAbstractComposite	Yes	~~Illegal~~Example abstract type.
FurtherItem	~~AbstractComposite~~MyAbstractComposite	Yes	~~Illegal~~Example abstract type.

~~The MAL data type specification does not allow the containment of abstract elements; however, other data type specification languages may.~~

4.1.6 REPRESENTING ENUMERATIONS

4.1.6.1 General

Enumerations are defined sets of possible values. All enumerations are extensions of the fundamental Element type and therefore can only be used to replace abstract Elements in message bodies. They are represented as shown below:

Name	ExampleEnum	
Short Form Part	789	
Enumeration Value	Numeric Value	Comment
FIRST	1	First enumeration possible value.
SECOND	2	Second enumeration value.
OTHER	3	Etc.
~~DELETION~~LAST	4	~~Object has been deleted~~Last enumeration value.

The 'Numeric Value' field is a simplified numerical version of the enumeration and has the ~~UInteger~~UShort range. It is expected to be used in efficient encodings and transport mappings.

4.1.6.2 Representing Lists

A list is an arbitrary-length sequence of items. Lists can be created of any ~~non-abstract~~ Attribute, Enumeration, or Composite~~ and are logically extensions of the basic Composite structure. Only lists of concrete types may be used~~. Lists are represented in the specifications as below:

List<ExampleStructure>

The List is shown as having a 'type' of ExampleStructure and is therefore a list of ExampleStructures. A list has a length part, but whether an actual length field is required is dependent on the ~~protocol~~encoding specification. For example, an efficient packet-based protocol may include an initial length field to allow correct decommutation, whereas an XML-based protocol would not require this because XML tags denote the end of the list.

The 'Short Form Part' for a List is the negative version of the 'Short Form Part' of the list type. For example, as the short form part for Integer is '11', the short form part of a list of Integers would be '-11'.

4.1.6.3 Polymorphism—Abstract Elements in Lists

The MAL data type specification allows abstract Lists to be specified. An example is presented:

- elements (List<AbstractBaseType>)
  - element [0] - ConcreteTypeA
  - element [1] - ConcreteTypeB
  - element [2] - ConcreteTypeA
  - element [3] – ConcreteTypeC

If an abstract list is specified, some Encoders may also have to carry the 'Short Form Part' for each of the elements in that list in order to be able to decode them. This is, however, Encoder-specific.

4.1.7 REPRESENTING OBJECT REFERENCES

An object reference is a reference to an instance of an MO Object. Object references use the ObjectRef Attribute type as defined in 4.3.19. When specifying an object reference, the reference shall point to an MO Object (which is defined using MAL::Object). Object references are represented in the service specifications as below:

ObjectRef<ExampleStructure>

The object reference is shown as having a 'type' of ExampleStructure and is therefore a reference to an MO Object of ExampleStructure type.

In case the object reference type is unknown, the representation shall use the MAL base type Element as follows:

ObjectRef<Element>

4.1.8 REPRESENTING NULL

In some message structures it may be required to have optional components. To this end it is required to be able to represent, for each type of component, the concept of NULL. This is separate and completely different from the concept of empty. For example, an empty string ("") is different from a NULL string which has no value. Language mappings must have the ability to represent NULL in messages, and transport mappings must have the ability to transport NULL.

Each composite structure definition contains an entry for each field denoting whether that field shall be allowed to contain the NULL value.

4.1.9 MESSAGE BODIES

When the message body of an operation is specified using the MAL data type specification, it is permitted to use the concept of multiple part messages, where the body of a message is specified using more than one type (see 3.4.2.2).

Any type in a message body may be an abstract type.

~~The last type in a message body may be an abstract type; all preceding types must be concrete types.~~

~~It is permitted to specify the final part of the message to be a list of an abstract type. When the message body is encoded, any list of an abstract type shall be replaced with a list of a derived concrete type.~~

~~When the final type of a PUBSUB operation is an abstract type, all of the update values in a single notify or publish message shall be of the same concrete type and contained in a concrete update list.~~

~~When the final type of a PUBSUB operation is an abstract type, if updates of different concrete types (that extend the abstract type) are being published, then separate publish/notify messages shall be used, one for each concrete type. The use of the Transport TRANSMITMULTIPLE (see 3.7.5) can be used to minimise the overhead in this case.~~

4.2 FUNDAMENTALS

4.2.1 ELEMENT

Element is the base type of all data constructs. All types that make up the MAL data model are derived from it.

Name	Element
Extends	
Abstract	

4.2.2 ATTRIBUTE

Attribute is the base type of all attributes of the MAL data model. Attributes are contained within Composites and are used to build complex structures that make the data model.

Name	Attribute
Extends	Element
Abstract	

4.2.3 COMPOSITE

Composite is the base structure for composite structures that contain a set of elements.

Name	Composite
Extends	Element
Abstract	

4.2.4 OBJECT

Object is the base structure for composite structures that contain a set of elements for MO Objects.

Name	Object
Extends	Composite
Abstract	

4.3 ATTRIBUTES

4.3.1 BLOB

The Blob structure is used to ~~store~~hold binary object attributes. It is a variable-length, unbounded, octet array. The distinction between this type and a list of Octet attributes is that this type may allow language mappings and encodings to use more efficient or appropriate representations.

Name	Blob
Extends	Attribute
Short Form Part	1

4.3.2 BOOLEAN

The Boolean structure is used to ~~store~~hold Boolean attributes. Possible values are ‘True’ or ‘False’.

Name	Boolean
Extends	Attribute
Short Form Part	2

4.3.3 DURATION

The Duration structure is used to ~~store~~hold Duration attributes. It represents a length of time in seconds. It may contain a fractional component.

Name	Duration
Extends	Attribute
Short Form Part	3

4.3.4 FLOAT

The Float structure is used to ~~store~~hold floating point attributes using the IEEE 754 32-bit range.

Three special values exist for this type: POSITIVE_INFINITY, NEGATIVE_INFINITY, and NaN (Not A Number).

Name	Float
Extends	Attribute
Short Form Part	4

4.3.5 DOUBLE

The Double structure is used to ~~store~~hold floating point attributes using the IEEE 754 64-bit range.

Three special values exist for this type: POSITIVE_INFINITY, NEGATIVE_INFINITY, and NaN (Not A Number).

Name	Double
Extends	Attribute
Short Form Part	5

4.3.6 IDENTIFIER

The Identifier structure is used to ~~store~~hold an identifier and can be used for indexing. It is a variable-length, unbounded, Unicode string. For some encoding/decoding bindings, the use of a numeric value might be appropriate for this Attribute, for example, via a dictionary.

Name	Identifier
Extends	Attribute
Short Form Part	6

4.3.7 OCTET

The Octet structure is used to ~~store~~hold 8-bit signed attributes. The permitted range is -128 to 127.

Name	Octet
Extends	Attribute
Short Form Part	7

4.3.8 UOCTET

The UOctet structure is used to ~~store~~hold 8-bit unsigned attributes. The permitted range is 0 to 255.

Name	UOctet
Extends	Attribute
Short Form Part	8

4.3.9 SHORT

The Short structure is used to ~~store~~hold 16-bit signed attributes. The permitted range is -32768 to 32767.

Name	Short
Extends	Attribute
Short Form Part	9

4.3.10 USHORT

The UShort structure is used to ~~store~~hold 16-bit unsigned attributes. The permitted range is 0 to 65535.

Name	UShort
Extends	Attribute
Short Form Part	10

4.3.11 INTEGER

The Integer structure is used to ~~store~~hold 32-bit signed attributes. The permitted range is -2147483648 to 2147483647.

Name	Integer
Extends	Attribute
Short Form Part	11

4.3.12 UINTEGER

The UInteger structure is used to ~~store~~hold 32-bit unsigned attributes. The permitted range is 0 to 4294967295.

Name	UInteger
Extends	Attribute
Short Form Part	12

4.3.13 LONG

The Long structure is used to ~~store~~hold 64-bit signed attributes. The permitted range is -9223372036854775808 to 9223372036854775807.

Name	Long
Extends	Attribute
Short Form Part	13

4.3.14 ULONG

The ULong structure is used to ~~store~~hold 64-bit unsigned attributes. The permitted range is 0 to 18446744073709551615.

Name	ULong
Extends	Attribute
Short Form Part	14

4.3.15 STRING

The String structure is used to ~~store~~hold String attributes. It is a variable-length, unbounded, Unicode string.

Name	String
Extends	Attribute
Short Form Part	15

4.3.16 TIME

The Time structure is used to ~~store~~hold absolute time attributes. It represents an absolute date and time to millisecond resolution.

Name	Time
Extends	Attribute
Short Form Part	16

4.3.17 FineTime

The FineTime structure is used to ~~store~~hold high-resolution absolute time attributes. It represents an absolute date and time to ~~picosecond~~nanosecond resolution.

Name	FineTime
Extends	Attribute
Short Form Part	17

4.3.18 URI

The URI structure is used to ~~store~~hold URI addresses. It is a variable-length, unbounded, Unicode string.

Name	URI
Extends	Attribute
Short Form Part	18

4.3.19 ObjectRef

The ObjectRef structure is used to hold references to MO Objects. The ObjectRef type shall be represented in specifications as ObjectRef<T> as presented in 4.1.6.3.

Name	ObjectRef
Extends	Attribute
Short Form Part	19

4.4 DATA STRUCTURES

4.4.1 InteractionType ENUMERATION

InteractionType is an enumeration holding the possible interaction pattern types.

Name	InteractionType	
Short Form Part	~~19~~101	
Enumeration Value	Numeric Value	Comment
SEND	1	Used for Send interactions.
SUBMIT	2	Used for Submit interactions.
REQUEST	3	Used for Request interactions.
INVOKE	4	Used for Invoke interactions.
PROGRESS	5	Used for Progress interactions.
PUBSUB	6	Used for Publish/Subscribe interactions.

4.4.2 SessionType ENUMERATION

SessionType is an enumeration holding the session types.

Name	SessionType	
Short Form Part	~~20~~102	
Enumeration Value	Numeric Value	Comment
LIVE	1	Used for Live sessions.
SIMULATION	2	Used for Simulation sessions.
REPLAY	3	Used for Replay sessions.

4.4.3 QoSLevel ENUMERATION

QoSLevel is an enumeration holding the possible QoS levels.

Name	QoSLevel	
Short Form Part	~~21~~103	
Enumeration Value	Numeric Value	Comment
BESTEFFORT	1	Used for Best Effort QoS Level.
ASSURED	2	Used for Assured QoS Level.
QUEUED	3	Used for Queued QoS Level.
TIMELY	4	Used for Timely QoS Level.

~~**4.4.4 UPDATETYPE ENUMERATION**~~

~~UpdateType is an enumeration holding the possible Update types.~~

~~Name~~	~~UpdateType~~	
~~Short Form Part~~	~~22~~	
~~Enumeration Value~~	~~Numeric Value~~	~~Comment~~
~~CREATION~~	~~1~~	~~Update is notification of the creation of the item.~~
~~UPDATE~~	~~2~~	~~Update is just a periodic update of the item and has not changed its value.~~
~~MODIFICATION~~	~~3~~	~~Update is for a changed value or modification of the item.~~
~~DELETION~~	~~4~~	~~Update is notification of the removal of the item.~~

~~4.4.5 SUBSCRIPTION~~

~~The Subscription structure is used when subscribing for updates using the PUBSUB interaction pattern. It contains a single identifier that identifies the subscription being defined and a set of entities being requested.~~

~~Name~~	~~Subscription~~		
~~Extends~~	~~Composite~~		
~~Short Form Part~~	~~23~~		
~~Field~~	~~Type~~	~~Nullable~~	~~Comment~~
~~subscriptionId~~	~~Identifier~~	~~No~~	~~The identifier of this subscription.~~
~~entities~~	~~List<EntityRequest>~~	~~No~~	~~The list of entities that are being subscribed for by this identified subscription.~~

4.4.6 ENTITYREQUEST

The EntityRequest structure is used when subscribing for updates using the PUBSUB interaction pattern.

Name	EntityRequest		
Extends	Composite		
Short Form Part	24		
Field	**Type**	**Nullable**	**Comment**
subDomain	List<Identifier>	Yes	Optional subdomain identifier that is appended to the Message Header domain identifier when requesting entities in a subdomain of this domain.
allAreas	Boolean	No	If set to True, then all updates regardless of Area shall be sent.
allServices	Boolean	No	If set to True, then all updates regardless of Service shall be sent.
allOperations	Boolean	No	If set to True, then all updates regardless of Operation shall be sent.
onlyOnChange	Boolean	No	The Boolean denotes that only change updates are to be sent rather than all updates.
entityKeys	List<EntityKey>	No	The list of entities to be monitored.

4.4.7 ENTITYKEY

The EntityKey structure is used to identify an entity in the PUBSUB interaction pattern.

Name	EntityKey		
Extends	Composite		
Short Form Part	25		
Field	**Type**	**Nullable**	**Comment**
firstSubKey	Identifier	Yes	The first sub-key of the key.
secondSubKey	Long	Yes	The second sub-key of the key.
thirdSubKey	Long	Yes	The third sub-key of the key.
fourthSubKey	Long	Yes	The fourth sub-key of the key.

4.4.8 UPDATEHEADER

The UpdateHeader structure is used by updates using the PUBSUB interaction pattern. It holds information that identifies a single update.

Name	UpdateHeader		
Extends	Composite		
Short Form Part	26		
Field	**Type**	**Nullable**	**Comment**
timestamp	Time	No	Creation timestamp of the update.
sourceURI	URI	No	URI of the source of the update, usually a PUBSUB provider.
updateType	UpdateType	No	Type of update being reported.
key	EntityKey	No	The key of the entity; shall not contain the wildcard value.

4.4.4 AttributeType ENUMERATION

AttributeType is an enumeration holding the defined MAL Attribute types.

Name	AttributeType	
Short Form Part	104	
Enumeration Value	Numerical Value	Comment
BLOB	1	Blob type.
BOOLEAN	2	Boolean type.
DURATION	3	Duration type.
FLOAT	4	Float type.
DOUBLE	5	Double type.
IDENTIFIER	6	Identifier type.
OCTET	7	Octet type.
UOCTET	8	UOctet type.
SHORT	9	Short type.
USHORT	10	UShort type.
INTEGER	11	Integer type.
UINTEGER	12	UInteger type.
LONG	13	Long type.
ULONG	14	ULong type.
STRING	15	String type.
TIME	16	Time type.
FINETIME	17	FineTime type.
URI	18	URI type.
OBJECTREF	19	ObjectRef type.

4.4.5 AreaNumber

AreaNumber is an enumeration holding the known existing area numbers in use. The table follows the corresponding MAL Area numbers available on the SANA registry.

Name	AreaNumber	
Short Form Part	105	
Enumeration Value	Numeric Value	Comment
MAL	1	The MAL area number.
COM	2	The COM area number. The COM is deprecated; therefore this area number is reserved for backward compatibility.
COMMON	3	The Common area number.
MC	4	The Monitor and Control area number.
MPS	5	The Mission Planning and Scheduling area number.
SM	7	The Software Management area number.
MDPD	9	The Mission Data Product Distribution area number.

4.4.6 SUBSCRIPTION

The Subscription structure is used when subscribing for updates using the PUBSUB interaction pattern. It contains a single identifier that identifies the subscription being defined and a set of entities being requested.

Name	Subscription		
Extends	Composite		
Short Form Part	1001		
Field	**Type**	**Nullable**	**Comment**
subscriptionId	Identifier	No	The identifier of this subscription.
domain	List<Identifier>	Yes	Optional domain identifier. If NULL, the subscription shall match with any domain.
filters	List<SubscriptionFilter>	Yes	The list of filters that are being selected for this subscription. The list of filters must be ANDed together. If NULL, the subscription will not filter specific keys.

4.4.7 SubscriptionFilter

The SubscriptionFilter structure is used when subscribing for updates using the PUBSUB interaction pattern. It contains a single identifier that identifies the subscription key name and the set of values to be registered for the defined key name.

Name	SubscriptionFilter		
Extends	Composite		
Short Form Part	1002		
Field	Type	Nullable	Comment
name	Identifier	No	The identifier name of the key.
values	List<MAL::Attribute>	No	The list of values that are being subscribed for this key. These shall be ORed together.

4.4.8 UpdateHeader

The UpdateHeader structure is used by updates using the PUBSUB interaction pattern. It holds information that identifies a single update.

Name	UpdateHeader		
Extends	Composite		
Short Form Part	1003		
Field	Type	Nullable	Comment
source	Identifier	Yes	The source of the update, usually a PUBSUB provider.
keyValues	List<Attribute>	Yes	The values for the PUBSUB keys. The values shall be ordered according to the defined keys.

4.4.9 IdBooleanPair

IdBooleanPair is a simple pair type of an identifier and Boolean value.

Name	IdBooleanPair		
Extends	Composite		
Short Form Part	~~27~~1004		
Field	Type	Nullable	Comment
id	Identifier	Yes	The Identifier value.
value	Boolean	Yes	The Boolean value.

4.4.10 PAIR

Pair is a simple composite structure for holding pairs. The pairs can be user-defined attributes.

Name	Pair		
Extends	Composite		
Short Form Part	~~28~~1005		
Field	Type	Nullable	Comment
first	Attribute	Yes	The attribute value for the first element of this pair.
second	Attribute	Yes	The attribute value for the second element of this pair.

4.4.11 NamedValue

The NamedValue structure represents a simple pair type of an identifier and abstract attribute value.

Name	NamedValue		
Extends	Composite		
Short Form Part	~~29~~1006		
Field	Type	Nullable	Comment
name	Identifier	Yes	The Identifier value.
value	Attribute	Yes	The Attribute value.

4.4.12 FILE

The File structure represents a File and holds details about a File. It can also, optionally, hold a BLOB of the file data. The file type is denoted using the internet MIME media types; the list of official MIME types is held in reference [2].

Name	File		
Extends	Composite		
Short Form Part	~~30~~1007		
Field	Type	Nullable	Comment
name	Identifier	No	The file name.
mimeType	String	Yes	The MIME type of the file, NULL if not known.
creationDate	Time	Yes	The creation timestamp of the file, NULL if not known.
modificationDate	Time	Yes	The last modification timestamp of the file, NULL if not known.
size	ULong	Yes	The size of the file, NULL if not known.
content	Blob	Yes	The contents of the file, NULL if not supplied.
metaData	List<NamedValue>	Yes	A list of extra metadata for the file.

4.4.13 ObjectIdentity

The ObjectIdentity structure represents the object identity of an MO Object.

Name	ObjectIdentity		
Extends	Composite		
Short Form Part	1008		
Field	**Type**	**Nullable**	**Comment**
domain	List<Identifier>	No	The domain of the MO Object being referenced.
area	Identifier	No	The area of the MO Object being referenced.
type	Identifier	No	The type of the MO Object being referenced.
key	Identifier	No	The key of the MO Object being referenced.
version	UInteger	No	The version of the MO Object being referenced.

4.4.14 ServiceId

The ServiceId composite represents a specific service in MO.

Name	ServiceId		
Extends	Composite		
Short Form Part	1009		
Field	**Type**	**Nullable**	**Comment**
serviceArea	UOctet	No	The area of this service taken from the numeric Area identifier of the service specification.
serviceNumber	UShort	No	The service taken from the numeric Service identifier of the service specification.
keyServiceVersion	UOctet	No	The version of this service taken from the Service Version of the service specification.

5 MAL ERRORS

Each operation shall list any errors, specific to that operation over and above any standard errors, that can be raised by an implementation. Error codes occupy the UInteger range and for an operation should start at zero ‘0’ and increment; standard errors start at 65536, 0x10000 in hex, and increment; operation-specific errors shall therefore remain inside the inclusive range of 0 to 65535.

The following table lists the standard errors:

Table 5-1: Standard MAL Error Codes

Error	Error #	Comment
DELIVERY_FAILED	65536	Confirmed communication error.
DELIVERY_TIMEDOUT	65537	Unconfirmed communication error.
DELIVERY_DELAYED	65538	Message queued somewhere awaiting contact.
DESTINATION_UNKNOWN	65539	Destination cannot be contacted.
DESTINATION_TRANSIENT	65540	Destination middleware reports destination application does not exist.
DESTINATION_LOST	65541	Destination lost halfway through conversation.
AUTHENTICATION_FAIL	65542	A failure to authenticate the message correctly.
AUTHORISATION_FAIL	65543	A failure in the MAL to authorise the message.
ENCRYPTION_FAIL	65544	A failure in the MAL to encrypt/decrypt the message.
UNSUPPORTED_AREA	65545	The destination does not support the service area.
UNSUPPORTED_OPERATION	65546	The destination does not support the operation.
UNSUPPORTED_VERSION	65547	The destination does not support the ~~area~~service version.
BAD_ENCODING	65548	The destination was unable to decode the message.
INTERNAL	65549	An internal error has occurred.
UNKNOWN	65550	Operation specific.
INCORRECT_STATE	65551	The destination was not in the correct state for the received message.
TOO_MANY	65552	Maximum number of subscriptions or providers of a broker has been exceeded.
SHUTDOWN	65553	The component is being shutdown.
TRANSACTION_TIMEOUT	65554	The interaction exceeded the defined timeout duration.

Only two errors are possible from the MAL and below with regards to Authentication and Authorisation. They are concerned with message-level issues and do not cover login issues such as an incorrect username/password combination.

NOTE – The authentication and authorisation failure messages are very generic in nature; it is possible that a weakness in a specific protocol or authentication mechanism could be exploited if too much information is returned about the specific nature of an authentication or authorisation failure.

6 SERVICE SPECIFICATION XML

This specification defines a normative XML schema for validating MO service specifications. The use of XML for service specification provides a machine readable format rather than the text based document format. The published specifications and XML Schema are held in an online SANA registry, located:

http://sanaregistry.org/r/moschemas/

The XML Schema that is used to validate the actual XML service specifications is located:

http://sanaregistry.org/r/moschemas/ServiceSchema-v.v.xsd

The normative XML for the MAL specification, validated against the XML schema, is located:

http://sanaregistry.org/r/moschemas/ServiceDefMAL-v.v.xml

Where the ‘v.v’ part is replaced with the issue number of the corresponding document.

The latest version of any specification shall always be directly available by removing the ‘-v.v’ part from the address, for example:

http://sanaregistry.org/r/moschemas/ServiceDefMAL.xml

7 CONFORMANCE MATRIX

This section provides the Conformance Matrix for implementations of the MAL. A MAL will be considered to be 'conformant' if the mandatory elements identified in the matrix are implemented as described in this Recommended Standard.

Table 7-1: Conformance Matrix

Interaction Pattern	Optional/Mandatory
SEND	Mandatory
SUBMIT	Mandatory
REQUEST	Mandatory
INVOKE	Mandatory
PROGRESS	Mandatory
PUBSUB	Mandatory

ANNEX A

PROTOCOL IMPLEMENTATION CONFORMANCE STATEMENT (PICS) PROFORMA

(NORMATIVE)

A1 INTRODUCTION

A1.1 OVERVIEW

This annex provides the Protocol Implementation Conformance Statement (PICS) Requirements List (RL) for an implementation of the Mission Operations Message Abstraction Layer (MAL) Recommended Standard. The PICS for an implementation is generated by completing the RL in accordance with the instructions below. An implementation claiming conformance must satisfy the mandatory requirements referenced in the RL.

An implementation's completed RL is called the PICS. The PICS states which protocol features have been implemented. The following entities can use the PICS:

- the protocol implementer, as a checklist to reduce the risk of failure to conform to the Recommended Standard through oversight;
- the supplier and acquirer or potential acquirer of the implementation, as a detailed indication of the capabilities of the implementation, stated relative to the common basis for understanding provided by the standard PICS proforma;
- the user or potential user of the implementation, as a basis for initially checking the possibility of interworking with another implementation (while interworking can never be guaranteed, failure to interwork can often be predicted from incompatible PICSes);
- a protocol tester, as the basis for selecting appropriate tests against which to assess the claim for conformance of the implementation..

A1.2 NOTATION

A1.2.1 Status Column Symbols

The following are used in the RL to indicate the status of features:

Symbol	Meaning
M	Mandatory
O	Optional

Support Column Symbols

The support of every item as claimed by the implementer is stated by entering the appropriate answer (Y, N, or N/A) in the support column.

Symbol	Meaning
Y	Yes, supported by the implementation
N	No, not supported by the implementation
N/A	Not applicable

A2 GENERAL INFORMATION

A2.1 IDENTIFICATION OF PICS

Ref	Question	Response
1	Date of Statement (DD/MM/YYYY)	
2	CCSDS document number containing the PICS	
3	Date of CCSDS document containing the PICS	

A2.2 IDENTIFICATION OF IMPLEMENTATION UNDER TEST (IUT)

Ref	Question	Response
1	Implementation name	
2	Implementation version	
3	Machine name	
4	Machine version	
5	Operating System name	
6	Operating System version	
7	Special Configuration	
8	Other Information	

A2.3 USER IDENTIFICATION

Supplier	
Contact Point for Queries	
Implementation name(s) and Versions	
Other Information Necessary for full identification, for example, name(s) and version(s) for machines and/or operating systems; System Name(s)	

A2.4 INSTRUCTIONS FOR COMPLETING THE RL

An implementer shows the extent of compliance to the protocol by completing the RL; the resulting completed RL is called a PICS.

A3 MAL PICS

There are six separate Interaction Patterns. All Interaction Patterns are mandatory and must be implemented according to the specifications defined in this Recommended Standard. This section must be completed.

Interaction Pattern	Optional/Mandatory	Support
SEND	Mandatory	
SUBMIT	Mandatory	
REQUEST	Mandatory	
INVOKE	Mandatory	
PROGRESS	Mandatory	
PUBSUB	Mandatory	

ANNEX B

SECURITY, SANA AND PATENT CONSIDERATIONS

(NORMATIVE)

B1 SECURITY CONSIDERATIONS

B1.1 ANALYSIS OF SECURITY CONSIDERATIONS

This annex presents the results of an analysis of security considerations applied to the technologies specified in this Recommended Standard.

B1.2 CONSEQUENCES OF NOT APPLYING SECURITY TO THE TECHNOLOGY

The consequences of not applying security to the systems and networks on which this Recommended Standard is implemented could include potential loss, corruption, and theft of data. Because these messages are capable of carrying any class of mission information, the consequences of not applying security to the systems and networks on which this Recommended Standard is implemented could include compromise or loss of the mission if malicious tampering of a particularly severe nature occurs.

B1.3 POTENTIAL THREATS AND ATTACK SCENARIOS

Potential threats or attack scenarios include, but are not limited to, (a) unauthorized access to the programs/processes that generate and interpret the messages, and (b) unauthorized access to the messages during transmission between exchange partners. Protection from unauthorized access during transmission is especially important if the mission utilizes open ground networks such as the Internet to provide ground station connectivity for the exchange of data formatted in compliance with this Recommended Standard. It is strongly recommended that potential threats or attack scenarios applicable to the systems and networks on which this Recommended Standard is implemented be addressed by the management of those systems and networks.

B1.4 DATA PRIVACY

Privacy of data formatted in compliance with the specifications of this Recommended Standard should be assured by the systems and networks on which this Recommended Standard is implemented.

B1.5 DATA INTEGRITY

Integrity of data formatted in compliance with the specifications of this Recommended Standard should be assured by the systems and networks on which this Recommended Standard is implemented.

B1.6 AUTHENTICATION OF COMMUNICATING ENTITIES

Authentication of communicating entities involved in the transport of data which complies with the specifications of this Recommended Standard should be provided by the systems and networks on which this Recommended Standard is implemented.

B1.7 DATA TRANSFER BETWEEN COMMUNICATING ENTITIES

The transfer of data formatted in compliance with this Recommended Standard between communicating entities should be accomplished via secure mechanisms approved by the IT Security functionaries of exchange participants.

B1.8 CONTROL OF ACCESS TO RESOURCES

Control of access to resources should be managed by the systems upon which originator formatting and recipient processing are performed.

B1.9 AUDITING OF RESOURCE USAGE

Auditing of resource usage should be handled by the management of systems and networks on which this Recommended Standard is implemented.

B1.10 UNAUTHORIZED ACCESS

Unauthorized access to the programs/processes that generate and interpret the messages should be prohibited in order to minimize potential threats and attack scenarios.

B1.11 DATA SECURITY IMPLEMENTATION SPECIFICS

Specific information-security interoperability provisions that may apply between agencies and other independent users involved in an exchange of data formatted in compliance with this Recommended Standard should be specified in an ICD.

B2 SANA CONSIDERATIONS

The recommendations of this document request SANA to create a registry named “Mission Operations Service XML” that consists of a set of XML Schemas and XML service specifications.

The registration rule for change to this registry requires an engineering review by a designated expert. The expert shall be assigned by the working group chair, or in absence, Area Director.

B3 PATENT CONSIDERATIONS

The recommendations of this document have no patent issues.

ANNEX C

DEFINITION OF ACRONYMS

(INFORMATIVE)

API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
BLOB	Binary Large Object
CCSDS	Consultative Committee for Space Data Standards
IEEE	Institute of Electrical and Electronics Engineers
MAL	Message Abstract Layer
MO	Mission Operations
NaN	Not A Number
QoS	Quality of Service
RPC	Remote Procedure Call
SM&C	CCSDS Spacecraft Monitor & Control
URI	Universal Resource Identifier
XML	eXtensible Markup Language

ANNEX D

INFORMATIVE REFERENCES

(INFORMATIVE)

[D1] *Mission Operations Services Concept*. Issue 3. Report Concerning Space Data System Standards (Green Book), CCSDS 520.0-G-3. Washington, D.C.: CCSDS, December 2010.

NOTE – Normative references are listed in 1.6.