IEC 61158-5-15

Edition 1.0 2007-12

INTERNATIONAL STANDARD

NORME INTERNATIONALE

Industrial communication networks – Fieldbus specifications – Part 5-15: Application layer service definition – Type 15 elements

Réseaux de communication industriels – Spécifications des bus de terrain – Partie 5-15: Définition des services de la couche application – Éléments de Type 15

IEC 61158-5-15:2007-12(en-fr)

IEC Central Office
3, rue de Varembé
CH-1211 Geneva 20
Switzerland

Tel.: +41 22 919 02 11
Fax: +41 22 919 03 00
info@iec.ch
www.iec.ch

About the IEC
The International Electrotechnical Commission (IEC) is the leading global organization that prepares and publishes International Standards for all electrical, electronic and related technologies.

About IEC publications
The technical content of IEC publications is kept under constant review by the IEC. Please make sure that you have the latest edition, a corrigenda or an amendment might have been published.

IEC Catalogue - webstore.iec.ch/catalogue
The stand-alone application for consulting the entire bibliographical information on IEC International Standards, Technical Specifications, Technical Reports and other documents. Available for PC, Mac OS, Android Tablets and iPad.

IEC publications search - www.iec.ch/searchpub
The advanced search enables to find IEC publications by a variety of criteria (reference number, text, technical committee,…). It also gives information on projects, replaced and withdrawn publications.

IEC Just Published - webstore.iec.ch/justpublished
Stay up to date on all new IEC publications. Just Published details all new publications released. Available online and also once a month by email.

Electropedia - www.electropedia.org
The world's leading online dictionary of electronic and electrical terms containing more than 30 000 terms and definitions in English and French, with equivalent terms in 14 additional languages. Also known as the International Electrotechnical Vocabulary (IEV) online.

IEC Glossary - std.iec.ch/glossary
More than 55 000 electrotechnical terminology entries in English and French extracted from the Terms and Definitions clause of IEC publications issued since 2002. Some entries have been collected from earlier publications of IEC TC 37, 77, 86 and CISPR.

IEC Customer Service Centre - webstore.iec.ch/csc
If you wish to give us your feedback on this publication or need further assistance, please contact the Customer Service Centre: csc@iec.ch.

A propos de l'IEC
La Commission Electrotechnique Internationale (IEC) est la première organisation mondiale qui élabore et publie des Normes internationales pour tout ce qui a trait à l'électricité, à l'électronique et aux technologies apparentées.

A propos des publications IEC
Le contenu technique des publications IEC est constamment revu. Veuillez vous assurer que vous possédez l'édition la plus récente, un corrigendum ou amendement peut avoir été publié.

Catalogue IEC - webstore.iec.ch/catalogue
Application autonome pour consulter tous les renseignements bibliographiques sur les Normes internationales, Spécifications techniques, Rapports techniques et autres documents de l'IEC. Disponible pour PC, Mac OS, tablettes Android et iPad.

Recherche de publications IEC - www.iec.ch/searchpub
La recherche avancée permet de trouver des publications IEC en utilisant différents critères (numéro de référence, texte, comité d'études,…). Elle donne aussi des informations sur les projets et les publications remplacées ou retirées.

IEC Just Published - webstore.iec.ch/justpublished
Restez informé sur les nouvelles publications IEC. Just Published détaille les nouvelles publications parues. Disponible en ligne et aussi une fois par mois par email.

Electropedia - www.electropedia.org
Le premier dictionnaire en ligne de termes électroniques et électriques. Il contient plus de 30 000 termes et définitions en anglais et en français, ainsi que les termes équivalents dans 14 langues additionnelles. Egalement appelé Vocabulaire Electrotechnique International (IEV) en ligne.

Glossaire IEC - std.iec.ch/glossary
Plus de 55 000 entrées terminologiques électrotechniques, en anglais et en français, extraites des articles Termes et Définitions des publications IEC parues depuis 2002. Plus certaines entrées antérieures extraites des publications des CE 37, 77, 86 et CISPR de l'IEC.

Service Clients - webstore.iec.ch/csc
Si vous désirez nous donner des commentaires sur cette publication ou si vous avez des questions contactez-nous: csc@iec.ch.

IEC 61158-5-15

Edition 1.0 2007-12

INTERNATIONAL STANDARD

NORME INTERNATIONALE

Industrial communication networks – Fieldbus specifications – Part 5-15: Application layer service definition – Type 15 elements

Réseaux de communication industriels – Spécifications des bus de terrain – Partie 5-15: Définition des services de la couche application – Éléments de Type 15

INTERNATIONAL ELECTROTECHNICAL COMMISSION

COMMISSION ELECTROTECHNIQUE INTERNATIONALE

PRICE CODE
CODE PRIX XF

ICS 25.040.40; 35.100.70

ISBN 978-2-8322-1557-9

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

INDUSTRIAL COMMUNICATION NETWORKS – FIELDBUS SPECIFICATIONS –

Part 5-15: Application layer service definition – Type 15 elements

FOREWORD

1) The International Electrotechnical Commission (IEC) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). The object of IEC is to promote international co-operation on all questions concerning standardization in the electrical and electronic fields. To this end and in addition to other activities, IEC publishes International Standards, Technical Specifications, Technical Reports, Publicly Available Specifications (PAS) and Guides (hereafter referred to as "IEC Publication(s)"). Their preparation is entrusted to technical committees; any IEC National Committee interested in the subject dealt with may participate in this preparatory work. International, governmental and non-governmental organizations liaising with the IEC also participate in this preparation. IEC collaborates closely with the International Organization for Standardization (ISO) in accordance with conditions determined by agreement between the two organizations.
2) The formal decisions or agreements of IEC on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested IEC National Committees.
3) IEC Publications have the form of recommendations for international use and are accepted by IEC National Committees in that sense. While all reasonable efforts are made to ensure that the technical content of IEC Publications is accurate, IEC cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.
4) In order to promote international uniformity, IEC National Committees undertake to apply IEC Publications transparently to the maximum extent possible in their national and regional publications. Any divergence between any IEC Publication and the corresponding national or regional publication shall be clearly indicated in the latter.
5) IEC provides no marking procedure to indicate its approval and cannot be rendered responsible for any equipment declared to be in conformity with an IEC Publication.
6) All users should ensure that they have the latest edition of this publication.
7) No liability shall attach to IEC or its directors, employees, servants or agents including individual experts and members of its technical committees and IEC National Committees for any personal injury, property damage or other damage of any nature whatsoever, whether direct or indirect, or for costs (including legal fees) and expenses arising out of the publication, use of, or reliance upon, this IEC Publication or any other IEC Publications.
8) Attention is drawn to the Normative references cited in this publication. Use of the referenced publications is indispensable for the correct application of this publication.
9) Attention is drawn to the possibility that some of the elements of this IEC Publication may be the subject of patent rights. IEC shall not be held responsible for identifying any or all such patent rights.

NOTE Use of some of the associated protocol types is restricted by their intellectual-property-right holders. In all cases, the commitment to limited release of intellectual-property-rights made by the holders of those rights permits a particular data-link layer protocol type to be used with physical layer and application layer protocols in type combinations as specified explicitly in the IEC 61784 series. Use of the various protocol types in other combinations may require permission of their respective intellectual-property-right holders.

International Standard IEC 61158-5-15 has been prepared by subcommittee 65C: Industrial networks, of IEC technical committee 65: Industrial-process measurement, control and automation.

This bilingual version (2014-06) corresponds to the English version, published in 2007-12.

This first edition and its companion parts of the IEC 61158-5 subseries cancel and replace IEC 61158-5:2003. This edition of this part constitutes a technical revision. This part and its Type 15 companion parts also cancel and replace IEC/PAS 62030, published in 2004.

This edition of IEC 61158-5 includes the following significant changes from the previous edition:

a) deletion of the former Type 6 fieldbus for lack of market relevance;

b) addition of new types of fieldbuses;

c) partition of part 5 of the third edition into multiple parts numbered -5-2, -5-3, …

The text of this standard is based on the following documents:

FDIS	Report on voting
65C/475/FDIS	65C/486/RVD

Full information on the voting for the approval of this standard can be found in the report on voting indicated in the above table.

The French version of this standard has not been voted upon.

This publication has been drafted in accordance with ISO/IEC Directives, Part 2.

The committee has decided that the contents of this publication will remain unchanged until the maintenance result date indicated on the IEC web site under http://webstore.iec.ch in the data related to the specific publication . At this date, the publication will be

- reconfirmed;
- withdrawn;
- replaced by a revised edition, or
- amended.

NOTE The revision of this standard will be synchronized with the other parts of the IEC 61158 series.

The list of all the parts of the IEC 61158 series, under the general title *Industrial communication networks – Fieldbus specifications*, can be found on the IEC web site.

INTRODUCTION

This part of IEC 61158 is one of a series produced to facilitate the interconnection of automation system components. It is related to other standards in the set as defined by the "three-layer" fieldbus reference model described in IEC/TR 61158-1.

The application service is provided by the application protocol making use of the services available from the data-link or other immediately lower layer. This standard defines the application service characteristics that fieldbus applications and/or system management may exploit.

Throughout the set of fieldbus standards, the term "service" refers to the abstract capability provided by one layer of the OSI Basic Reference Model to the layer immediately above. Thus, the application layer service defined in this standard is a conceptual architectural service, independent of administrative and implementation divisions.

INDUSTRIAL COMMUNICATION NETWORKS – FIELDBUS SPECIFICATIONS –

Part 5-15: Application layer service definition – Type 15 elements

1 Scope

1.1 Overview

In network communications, as in many fields of engineering, it is a fact that “one size does not fit all.” Engineering design is about making the right set of trade-offs, and these trade-offs must balance conflicting requirements such as simplicity, generality, ease of use, richness of features, performance, memory size and usage, scalability, determinism, and robustness. These trade-offs must be made in light of the types of information flow (e.g. periodic, one-to-many, request-reply, events), and the constraints imposed by the application and execution platforms.

The Type 15 fieldbus provides two major communication mechanisms that complement each others to satisfy communication requirements in the field of automation: the Client/Server and the Publish/Subscribe paradigms. They can be used concurrently on the same device.

Type 15 Client/Server operates in a Client/Server relationship. Its application layer service definitions and protocol specifications are independent of the underlying layers, and have been implemented on a variety of stacks and communication media, including EIA/TIA-232, EIA/TIA-422, EIA/TIA-425, HDLC (ISO 13239), fiber, TCP/IP, Wireless LANs and Radios.

Type 15 Publish/Subscribe operates in a Publish/Subscribe relationship. Its application layer service definitions and protocol specifications are independent of the underlying layers and can be configured to provide reliable behaviour and support determinism. The most common stack is UDP/IP.

The fieldbus application layer (FAL) provides user programs with a means to access the fieldbus communication environment. In this respect, the FAL can be viewed as a “window between corresponding application programs.”

This part of IEC 61158 provides common elements for basic time-critical and non-time-critical messaging communications between application programs in an automation environment and material specific to Type 15 fieldbus. The term “time-critical” is used to represent the presence of a time-window, within which one or more specified actions are required to be completed with some defined level of certainty. Failure to complete specified actions within the time window risks failure of the applications requesting the actions, with attendant risk to equipment, plant and possibly human life.

This part of IEC 61158 define in an abstract way the externally visible service provided by the Type 15 fieldbus application layer in terms of

a) an abstract model for defining application resources (objects) capable of being manipulated by users via the use of the FAL service,
b) the primitive actions and events of the service;
c) the parameters associated with each primitive action and event, and the form which they take; and
d) the interrelationship between these actions and events, and their valid sequences.

The purpose of this part of IEC 61158 is to define the services provided to

1) the FAL user at the boundary between the user and the Application Layer of the Fieldbus Reference Model, and

2) Systems Management at the boundary between the Application Layer and Systems Management of the Fieldbus Reference Model.

This part of IEC 61158 specifies the structure and services of the Type 15 IEC fieldbus Application Layer, in conformance with the OSI Basic Reference Model (ISO/IEC 7498) and the OSI Application Layer Structure (ISO/IEC 9545).

FAL services and protocols are provided by FAL application-entities (AE) contained within the application processes. The FAL AE is composed of a set of object-oriented Application Service Elements (ASEs) and a Layer Management Entity (LME) that manages the AE. The ASEs provide communication services that operate on a set of related application process object (APO) classes. One of the FAL ASEs is a management ASE that provides a common set of services for the management of the instances of FAL classes.

Although these services specify, from the perspective of applications, how request and responses are issued and delivered, they do not include a specification of what the requesting and responding applications are to do with them. That is, the behavioral aspects of the applications are not specified; only a definition of what requests and responses they can send/receive is specified. This permits greater flexibility to the FAL users in standardizing such object behavior. In addition to these services, some supporting services are also defined in this standard to provide access to the FAL to control certain aspects of its operation.

1.2 Specifications

The principal objective of this part of IEC 61158 is to specify the characteristics of conceptual application layer services suitable for time-critical communications, and thus supplement the OSI Basic Reference Model in guiding the development of application layer protocols for time-critical communications.

A secondary objective is to provide migration paths from previously-existing industrial communications protocols. It is this latter objective which gives rise to the diversity of services standardized as the various Types of IEC 61158, and the corresponding protocols standardized in subparts of IEC 61158-6.

This specification may be used as the basis for formal Application Programming-Interfaces. Nevertheless, it is not a formal programming interface, and any such interface will need to address implementation issues not covered by this specification, including

a) the sizes and octet ordering of various multi-octet service parameters, and

b) the correlation of paired request and confirm, or indication and response, primitives.

1.3 Conformance

This part of IEC 61158 does not specify individual implementations or products, nor do they constrain the implementations of application layer entities within industrial automation systems.

There is no conformance of equipment to this application layer service definition standard. Instead, conformance is achieved through implementation of conforming application layer protocols that fulfill the Type 15 application layer services as defined in this part of IEC 61158.

1.4 Type overview

In network communications, as in many fields of engineering, it is a fact that “one size does not fit all.” Engineering design is about making the right set of trade-offs, and these trade-offs must balance conflicting requirements such as simplicity, generality, ease of use, richness of features, performance, memory size and usage, scalability, determinism, and robustness. These trade-offs must be made in light of the types of information flow (e.g. periodic, one-to-

many, request-reply, events), and the constraints imposed by the application and execution platforms.

The Type 15 fieldbus provides two major communication mechanisms that complement each others to satisfy communication requirements in the field of automation: the Client/Server and the Publish/Subscribe paradigms. They can be used concurrently on the same device.

Type 15 Client/Server operates in a Client/Server relationship. Its application layer service definitions and protocol specifications are independent of the underlying layers, and have been implemented on a variety of stacks and communication media, including EIA/TIA-232, EIA/TIA-422, EIA/TIA-425, HDLC (ISO 13239), fiber, TCP/IP, Wireless LANs and Radios.

Type 15 Publish/Subscribe operates in a Publish/Subscribe relationship. Its application layer service definitions and protocol specifications are independent of the underlying layers and can be configured to provide reliable behavior and support determinism. The most common stack is UDP/IP.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

IEC/TR 61158-1 (Ed.2.0), *Industrial communication networks – Fieldbus specifications – Part 1: Overview and guidance for the IEC 61158 and IEC 61784 series*

ISO/IEC 7498-1, *Information technology – Open Systems Interconnection – Basic Reference Model – Part 1: The Basic Model*

ISO/IEC 8822, *Information technology – Open Systems Interconnection – Presentation service definition*

ISO/IEC 8824, *Information technology – Open Systems Interconnection – Specification of Abstract Syntax Notation One (ASN.1)*

ISO/IEC 9545, *Information technology – Open Systems Interconnection – Application Layer structure*

ISO/IEC 10731, *Information technology – Open Systems Interconnection – Basic Reference Model – Conventions for the definition of OSI services*

3 Terms and definitions, abbreviations, symbols and conventions

3.1 Terms and definitions

For the purposes of this document, the following terms as defined in these publications apply:

3.1.1 ISO/IEC 7498-1 terms

a) application entity
b) application process
c) application protocol data unit
d) application service element
e) application entity invocation
f) application process invocation

g) application transaction
h) real open system
i) transfer syntax

3.1.2 ISO/IEC 8822 terms

a) abstract syntax
b) presentation context

3.1.3 ISO/IEC 9545 terms

a) application-association
b) application-context
c) application context name
d) application-entity-invocation
e) application-entity-type
f) application-process-invocation
g) application-process-type
h) application-service-element
i) application control service element

3.1.4 ISO/IEC 8824 terms

a) object identifier
b) type

3.1.5 IEC/TR 61158-1 terms

The following IEC/TR 61158-1 terms apply.

3.1.5.1 application

function or data structure for which data is consumed or produced

3.1.5.2 application layer interoperability

capability of application entities to perform coordinated and cooperative operations using the services of the FAL

3.1.5.3 application object

object class that manages and provides the run time exchange of messages across the network and within the network device

NOTE: Multiple types of application object classes may be defined.

3.1.5.4 application process

part of a distributed application on a network, which is located on one device and unambiguously addressed

3.1.5.5 application process identifier

distinguishes multiple application processes used in a device

3.1.5.6 application process object

component of an application process that is identifiable and accessible through an FAL application relationship

NOTE Application process object definitions are composed of a set of values for the attributes of their class.

3.1.5.7 application process object class

class of application process objects defined in terms of the set of their network-accessible attributes and services

3.1.5.8 application relationship

cooperative association between two or more application-entity-invocations for the purpose of exchange of information and coordination of their joint operation

NOTE This relationship is activated either by the exchange of application-protocol-data-units or as a result of preconfiguration activities.

3.1.5.9 application relationship endpoint

context and behavior of an application relationship as seen and maintained by one of the application processes involved in the application relationship

NOTE Each application process involved in the application relationship maintains its own application relationship endpoint.

3.1.5.10 application service element

application-service-element that provides the exclusive means for establishing and terminating all application relationships

3.1.5.11 attribute

description of an externally visible characteristic or feature of an object

NOTE The attributes of an object contain information about variable portions of an object. Typically, they provide status information or govern the operation of an object. Attributes may also affect the behaviour of an object. Attributes are divided into class attributes and instance attributes.

3.1.5.12 behavior

indication of how the object responds to particular events

NOTE Its description includes the relationship between attribute values and services.

3.1.5.13 class

set of objects, all of which represent the same kind of system component

NOTE A class is a generalization of the object; a template for defining variables and methods. All objects in a class are identical in form and behavior, but usually contain different data in their attributes.

3.1.5.14 class attributes

attribute that is shared by all objects within the same class

3.1.5.15 class code

unique identifier assigned to each object class

3.1.5.16 class specific service

service defined by a particular object class to perform a required function which is not performed by a common service

NOTE A class specific object is unique to the object class which defines it.

3.1.5.17 client

(a) object which uses the services of another (server) object to perform a task

(b) initiator of a message to which a server reacts, such as the role of an AR endpoint in which it issues confirmed service request APDUs to a single AR endpoint acting as a server

3.1.5.18 conveyance path

unidirectional flow of APDUs across an application relationship

3.1.5.19 cyclic

term used to describe events which repeat in a regular and repetitive manner

3.1.5.20 dedicated AR

AR used directly by the FAL user

NOTE On Dedicated ARs, only the FAL Header and the user data are transferred.

3.1.5.21 device

physical hardware connection to the link

NOTE A device may contain more than one node.

3.1.5.22 device profile

collection of device dependent information and functionality providing consistency between similar devices of the same device type

3.1.5.23 dynamic AR

AR that requires the use of the AR establishment procedures to place it into an established state

3.1.5.24 endpoint

one of the communicating entities involved in a connection

3.1.5.25 error

discrepancy between a computed, observed or measured value or condition and the specified or theoretically correct value or condition

3.1.5.26 error class

general grouping for error definitions

NOTE Error codes for specific errors are defined within an error class.

3.1.5.27 error code

identification of a specific type of error within an error class

3.1.5.28 FAL subnet

networks composed of one or more data link segments

NOTE Subnets are permitted to contain bridges, but not routers. FAL subnets are identified by a subset of the network address.

3.1.5.29 logical device

FAL class that abstracts a software component or a firmware component as an autonomous self-contained facility of an automation device

3.1.5.30 management information

network-accessible information that supports managing the operation of the fieldbus system, including the application layer

NOTE Managing includes functions such as controlling, monitoring, and diagnosing.

3.1.5.31 network

series of nodes connected by some type of communication medium

NOTE The connection paths between any pair of nodes can include repeaters, routers and gateways.

3.1.5.32 peer

role of an AR endpoint in which it is capable of acting as both client and server

3.1.5.33 pre-defined AR endpoint

AR endpoint that is defined locally within a device without use of the create service

NOTE Pre-defined ARs that are not pre-established are established before being used.

3.1.5.34 pre-established AR endpoint

AR endpoint that is placed in an established state during configuration of the AEs that control its endpoints

3.1.5.35 publisher

role of an AR endpoint in which it transmits APDUs onto the fieldbus for consumption by one or more subscribers

NOTE The publisher may not be aware of the identity or the number of subscribers and it may publish its APDUs using a dedicated AR. Two types of publishers are defined by this standard, Pull Publishers and Push Publishers, each of which is defined separately.

3.1.5.36 server

a) role of an AREP in which it returns a confirmed service response APDU to the client that initiated the request

b) object which provides services to another (client) object

3.1.5.37 service

operation or function than an object and/or object class performs upon request from another object and/or object class

NOTE A set of common services is defined and provisions for the definition of object-specific services are provided. Object-specific services are those which are defined by a particular object class to perform a required function which is not performed by a common service.

3.1.5.38 subscriber

role of an AREP in which it receives APDUs produced by a publisher

NOTE Two types of subscribers are defined by this standard, pull subscribers and push subscribers, each of which is defined separately.

3.1.6 Specific definitions for client/server

3.1.6.1
coils, discrete outputs
application process object, a set of coils, characterized by the address of a coil and a quantity of coils, this set is also called discrete outputs when associated with field outputs

3.1.6.2
discrete, discrete input
application process object, addressed by an unsigned number and having a width of one bit, representing a 1-bit encoded status value, read-only, with the value '1' encoding the status ON and the value '0' encoding the status OFF, also called discrete input, especially when associated with field outputs

3.1.6.3
discrete inputs, discretes
application process object, a set of discretes, characterized by the address of a discrete and a quantity of discretes, this set is also called discrete inputs, especially when associated with field inputs

3.1.6.4
coil, discrete output
application process object, addressed by an unsigned number and having a width of one bit, representing a 1-bit encoded status value, read-write, with the value '1' encoding the status ON and the value '0' encoding the status OFF, also called discrete output when associated with field output

3.1.6.5
encapsulated interface
mechanism encapsulating a service for an interface, which is an application process object characterized by an MEI type

3.1.6.6
exception
encoding used to signal a service request failure

3.1.6.7
exception code
encoding associated with an exception, detailing the reason of a service request failure

3.1.6.8
file
application process object, an organization of records, characterized by an unsigned number

3.1.6.9
function code
encoding of a service requested to a server

3.1.6.10
holding register,output register
application process object, addressed by an unsigned number and representing values with 16 bits, read-write

3.1.6.11
holding registers, output registers
application process object, a set of holding registers, characterized by the address of a holding register and a quantity of holding registers

3.1.6.12
input register
application process object, addressed by an unsigned number and representing values with 16 bits, read-only

3.1.6.13
input registers
application process object, a set of input registers, characterized by the address of an input register and a quantity of input registers

3.1.6.14
record
application process object, a set of contiguous registers of a specified type, characterized by the address of the first register and by the quantity of registers; in the context of this definition, the registers involved have also been called references

3.1.6.15
reference
denigrated term for register

3.1.6.16
reference type
denigrated term for register type

3.1.6.17
sub-code
specialization of a function code

3.1.6.18
unit ID
logical device identifier

3.1.6.19
MEI type
type specified as an octet value, used to dispatch a service to the appropriate interface in the context of the encapsulated interface mechanism

3.1.7 Specific definitions for publish/subscribe

3.1.7.1
network object
publish/subscribe application, reader, or writer

3.1.7.2
GUID
globally unique network object identifier, used to uniquely reference an object within the network

3.1.7.3
composite state
attributes of a set of network objects

3.1.7.4
composite state transfer
Interactions between CSTWriters and interested CSTReaders with the goal to allow the CSTReaders to reconstruct the composite state of the communicating CSTWriter, without transferring the entire history that led to the current composite state of that CSTWriter.

3.1.7.5
reader
subscriber or a CSTReader

3.1.7.6
writer
publisher or a CSTWriter

3.1.7.7
CSTReader
meta-information-specialized subscriber

3.1.7.8
CSTWriter
meta-information-specialized publisher

3.1.7.9
communication actor
reader or writer

3.1.7.10
domain participant
application that uses publish/subscribe elements, also called publish/subscribe application

NOTE This terminology is adopted to avoid the overuse of the term “application”. At the same time, the term “domain” has a place within publish/subscribe. The Type extensibility allows for the concept of “domains”, or independent communication planes, effectively permitting isolation of application exchanges within domains. While OMG DDS as in “Data Distribution Service for Real-Time Systems Specification, Version 1.1, December 2005” uses this extension, the feature will not be examined further in this specification, which will consider a single domain.

3.1.7.11
manager
specialized publish/subscribe application, containing specialized publishers and subscribers, and involved in the described discovery and maintenance mechanism; not to be confused with any publishing manager

3.1.7.12
managed participant
a publish/subscribe application; the qualifier refers to its role in relation to a manager when involved in the described discovery and maintenance mechanism; not to be confused with any publishing manager subordinate

3.1.7.13
publishing manager
role of an AR endpoint in which it issues one or more confirmed service request APDUs to a publisher to request the publisher to publish a specified object. Two types of publishing managers are defined by this standard, pull publishing managers and push publishing managers, each of which is defined separately

3.1.7.14
pull publisher
type of publisher that publishes an object in response to a request received from its pull publishing manager

3.1.7.15
pull publishing manager
type of publishing manager that requests that a specified object be published in a corresponding response APDU

3.1.7.16
push publisher
type of publisher that publishes an object in an unconfirmed service request APDU

3.1.7.17
push publishing manager
type of publishing manager that requests that a specified object be published using an unconfirmed service

3.1.7.18
pull subscriber
type of subscriber that recognizes received confirmed service response APDUs as published object data

3.1.7.19
push subscriber
type of subscriber that recognizes received unconfirmed service request APDUs as published object data

3.1.7.20
sequence number
number used to uniquely identify elementary publish/subscribe messages in an ordered manner

3.2 Abbreviations and symbols

3.2.1 Common abbreviations and symbols

AE	Application Entity
AL	Application Layer
ALME	Application Layer Management Entity
ALP	Application Layer Protocol
APO	Application Object
AP	Application Process
APDU	Application Protocol Data Unit
API	Application Process Identifier
AR	Application Relationship
AREP	Application Relationship End Point
ASCII	American Standard Code for Information Interchange
ASE	Application Service Element
Cnf	Confirmation
DL-	(as a prefix) Data Link-
DLC	Data Link Connection
DLCEP	Data Link Connection End Point
DLL	Data Link Layer
DLM	Data Link-management
DLSAP	Data Link Service Access Point

DLSDU DL-service-data-unit
FAL Fieldbus Application Layer
IANA Internet Assigned Numbers Authority
ID Identifier
IEC International Electrotechnical Commission
Ind Indication
IP Internet Protocol
LME Layer Management Entity
LSB Least Significant Bit
MSB Most Significant Bit
OSI Open Systems Interconnect
QoS Quality of Service
Req Request
Rsp Response
SAP Service Access Point
SDU Service Data Unit
SMIB System Management Information Base
SMK System Management Kernel

3.2.2 Abbreviations and symbols for client/server

C/S Client/Server
FC Function Code
CAN Controller Area Network
CiA CAN in Automation
MEI Encapsulated Interface type
URL Uniform Resource Locator

3.2.3 Abbreviations and symbols for publish/subscribe

CS Composite State
CST Composite State Transfer
DCPS Data-Centric Publish-Subscribe
DDS Data Distribution Service
DLRL Data Local Reconstruction Layer
GUID Globally Unique Id
OMG Object Management Group
P/S Publish/Subscribe

3.3 Conventions

3.3.1 Overview

The FAL is defined as a set of object-oriented ASEs. Each ASE is specified in a separate subclause. Each ASE specification is composed of two parts, its class specification, and its service specification.

The class specification defines the attributes of the class. The attributes are accessible from instances of the class using the Object Management ASE services specified in Clause 5 of this standard. The service specification defines the services that are provided by the ASE.

3.3.2 General conventions

This standard uses the descriptive conventions given in ISO/IEC 10731.

3.3.3 Conventions for class definitions

Class definitions are described using templates. Each template consists of a list of attributes for the class. The general form of the template is shown below:

FAL ASE:	**ASE Name**		
CLASS:	**Class Name**		
CLASS ID:	**#**		
PARENT CLASS:		**Parent Class Name**	
ATTRIBUTES:			
1	(o)	Key Attribute:	numeric identifier
2	(o)	Key Attribute:	name
3	(m)	Attribute:	attribute name(values)
4	(m)	Attribute:	attribute name(values)
4.1	(s)	Attribute:	attribute name(values)
4.2	(s)	Attribute:	attribute name(values)
4.3	(s)	Attribute:	attribute name(values)
5.	(c)	Constraint:	constraint expression
5.1	(m)	Attribute:	attribute name(values)
5.2	(o)	Attribute:	attribute name(values)
6	(m)	Attribute:	attribute name(values)
6.1	(s)	Attribute:	attribute name(values)
6.2	(s)	Attribute:	attribute name(values)
SERVICES:			
1	(o)	OpsService:	service name
2.	(c)	Constraint:	constraint expression
2.1	(o)	OpsService:	service name
3	(m)	MgtService:	service name

(1) The "FAL ASE:" entry is the name of the FAL ASE that provides the services for the class being specified.

(2) The "CLASS:" entry is the name of the class being specified. All objects defined using this template will be an instance of this class. The class may be specified by this standard, or by a user of this standard.

(3) The "CLASS ID:" entry is a number that identifies the class being specified. This number is unique within the FAL ASE that will provide the services for this class. When qualified by the identity of its FAL ASE, it unambiguously identifies the class within the scope of the FAL. The value "NULL" indicates that the class cannot be instantiated. Class IDs between 1 and 255 are reserved by this standard to identify standardized classes. They have been assigned to maintain compatibility with existing national standards. CLASS IDs between 256 and 2048 are allocated for identifying user defined classes.

(4) The "PARENT CLASS:" entry is the name of the parent class for the class being specified. All attributes defined for the parent class and inherited by it are inherited for the class being defined, and therefore do not have to be redefined in the template for this class.

NOTE The parent-class "TOP" indicates that the class being defined is an initial class definition. The parent class TOP is used as a starting point from which all other classes are defined. The use of TOP is reserved for classes defined by this standard.

(5) The "ATTRIBUTES" label indicate that the following entries are attributes defined for the class.

a) Each of the attribute entries contains a line number in column 1, a mandatory (m) / optional (o) / conditional (c) / selector (s) indicator in column 2, an attribute type label in column 3, a name or a conditional expression in column 4, and optionally a list of

enumerated values in column 5. In the column following the list of values, the default value for the attribute may be specified.

b) Objects are normally identified by a numeric identifier or by an object name, or by both. In the class templates, these key attributes are defined under the key attribute.

c) The line number defines the sequence and the level of nesting of the line. Each nesting level is identified by period. Nesting specifies

i) fields of a structured attribute (4.1, 4.2, 4.3),

ii) attributes conditional on a constraint statement (5). Attributes may be mandatory (5.1) or optional (5.2) if the constraint is true. Not all optional attributes require constraint statements as does the attribute defined in (5.2).

iii) the selection fields of a choice type attribute (6.1 and 6.2).

(6) The "SERVICES" label indicates that the following entries are services defined for the class.

a) An (m) in column 2 indicates that the service is mandatory for the class, while an (o) indicates that it is optional. A (c) in this column indicates that the service is conditional. When all services defined for a class are defined as optional, at least one has to be selected when an instance of the class is defined.

b) The label "OpsService" designates an operational service (1).

c) The label "MgtService" designates an management service (2).

d) The line number defines the sequence and the level of nesting of the line. Each nesting level is identified by period. Nesting within the list of services specifies services conditional on a constraint statement.

3.3.4 Conventions for service definitions

3.3.4.1 General

The service model, service primitives, and time-sequence diagrams used are entirely abstract descriptions; they do not represent a specification for implementation.

3.3.4.2 Service parameters

Service primitives are used to represent service user/service provider interactions (ISO/IEC 10731). They convey parameters which indicate information available in the user/provider interaction. In any particular interface, not all parameters need be explicitly stated.

The service specifications of this standard uses a tabular format to describe the component parameters of the ASE service primitives. The parameters which apply to each group of service primitives are set out in tables. Each table consists of up to five columns for the

1) parameter name,
2) request primitive,
3) indication primitive,
4) response primitive, and
5) confirm primitive.

One parameter (or component of it) is listed in each row of each table. Under the appropriate service primitive columns, a code specifies the type of usage of the parameter on the primitive specified in the column:

M parameter is mandatory for the primitive

U parameter is a User option, and may or may not be provided depending on dynamic usage of the service user. When not provided, a default value for the parameter is assumed.

C parameter is conditional upon other parameters or upon the environment of the service user.

— (blank) parameter is never present.

S parameter is a selected item.

Some entries are further qualified by items in brackets. These may be

a) a parameter-specific constraint:

"(=)" indicates that the parameter is semantically equivalent to the parameter in the service primitive to its immediate left in the table.

b) an indication that some note applies to the entry:

"(n)" indicates that the following note "n" contains additional information pertaining to the parameter and its use.

3.3.4.3 Service procedures

The procedures are defined in terms of

- the interactions between application entities through the exchange of fieldbus Application Protocol Data Units, and
- the interactions between an application layer service provider and an application layer service user in the same system through the invocation of application layer service primitives.

These procedures are applicable to instances of communication between systems which support time-constrained communications services within the fieldbus Application Layer.

4 Concepts

4.1 Common concepts

All of IEC/TR 61158-1, Clause 9 is incorporated by reference, except as specifically overridden in 4.2 and 4.3.

4.2 Client/server specific concepts

4.2.1 Overview

Client/Server (C/S) describes a Client/Server communication mechanism between devices connected on different types of buses or networks. The client/server AL is the same irrelevant from the underlying layers. Some client/server stack configurations are illustrated in Figure 1.

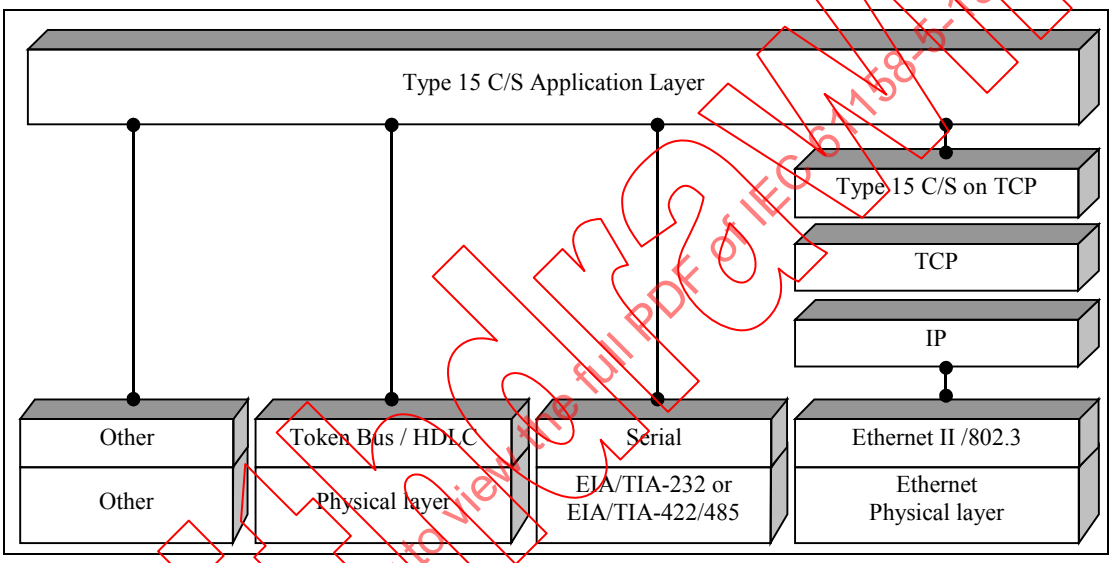


Figure 1 – Client/server stacks

The communication between devices on different types of buses or networks is made possible by the use of gateways, as illustrated in Figure 2.

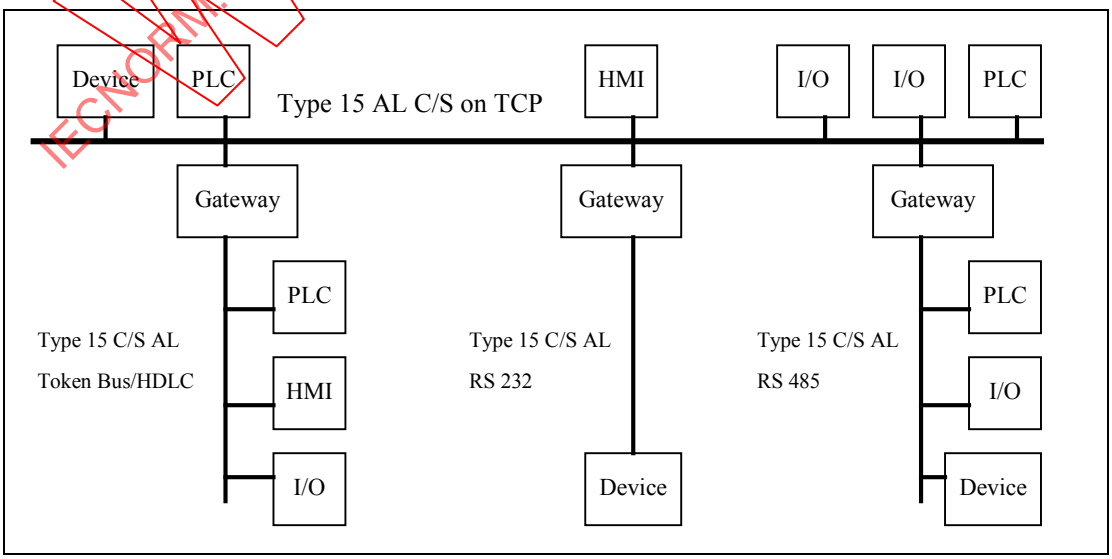


Figure 2 – Client/server communication on different buses or networks

4.2.2 Client/server APOs

4.2.2.1 General

Client/server provides application layer services specified by function codes, acting on application process objects (APOs).

An application process object is a network representation of a specific aspect of an AP. Each APO represents a specific set of information and processing capabilities of an AP that are accessible through services of the FAL. APOs are used to represent these capabilities to other APs in a fieldbus system.

From the perspective of the FAL, an APO is modeled as a network accessible object contained within an AP or within another APO (APOs may contain other APOs). APOs provide the network definition for objects contained within an AP that are remotely accessible. The definition of an APO includes an identification of the FAL services that can be used by remote APs for remote access. The FAL services, as shown in Figure 3, are provided by the FAL communications entity of the AP, known as the FAL Applications Entity (FAL AE).

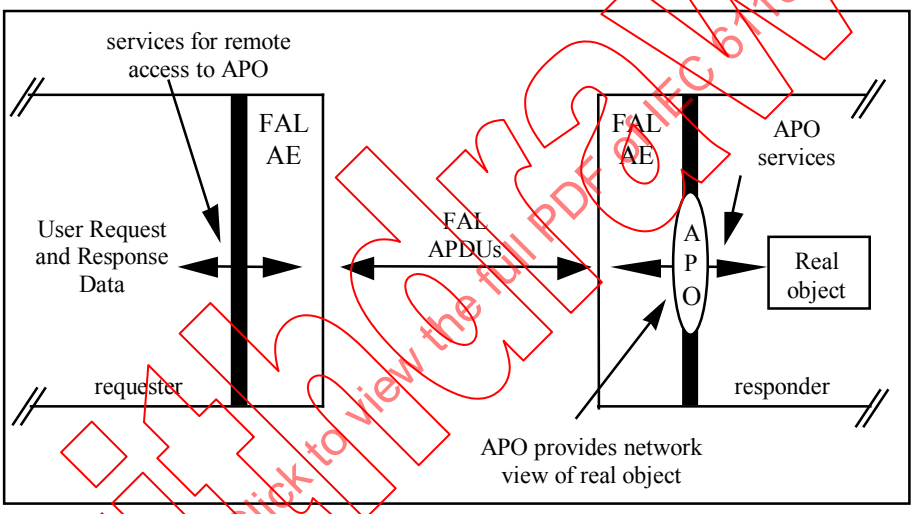


Figure 3 – Client/server APOs services conveyed by the FAL

In Figure 3, remote APs acting as clients may access the real object by sending requests through the APO that represents the real object. Local aspects of the AP convert between the network view (the APO) of the real object and the internal AP view of the real object.

For all APOs, the semantics is the one allowed by the associated FAL APO ASE.

4.2.2.2 Common APOs

Client/server bases its model on APOs that have distinguishing characteristics. Some of them, namely discretes, coils, input registers and holding registers, are often interpreted as memory tables. While this is a simple and convenient model, it is just one of many, indeed they do not even need to be seen as memory objects. It is up to the application to devise the most appropriate relation of APOs with real objects, and the subject is out of the scope of this document. Table 1 summarizes the characteristics of these four APOs.

Table 1 – Common client/server APOs

Common APOs	Object type	Comment
discrete	single bit	Read-only, its value can be provided by an I/O system. This APO is useful in the modeling of binary valued real objects that are manipulated by the server application and are supposed to be only observed by the client user. The integrity of the above contract is under control of the server AP, which can confine the exposure of the real objects to discretes.
coil	single bit	Read-write, its value can be altered by a client application program.
input register	16 bit word	Read-only, its value can be provided by an I/O system. This APO is useful in the modeling of analog valued real objects that are manipulated by the server application and are supposed to be only observed by the client user. The integrity of the above contract is under control of the server AP, which can confine the exposure of the real objects to input registers.
holding register	16 bit word	Read-write, its value can be altered by a client application program.

The distinctions between inputs and outputs, and between bit-addressable and word-addressable data items, do not imply any application behavior. It is perfectly acceptable, and customary, to regard all four tables as overlaying one another, if this is the most natural interpretation on the target machine in question.

The following Figure 4 and Figure 5, informative, give some common but by no means exhaustive interpretations, respectively as distinct memory tables and overlapping memory tables.

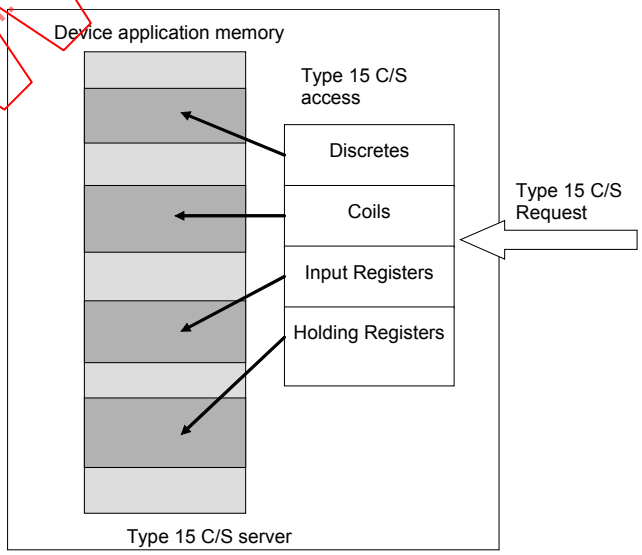


Figure 4 – [INFORMATIVE] Interpretation as distinct tables

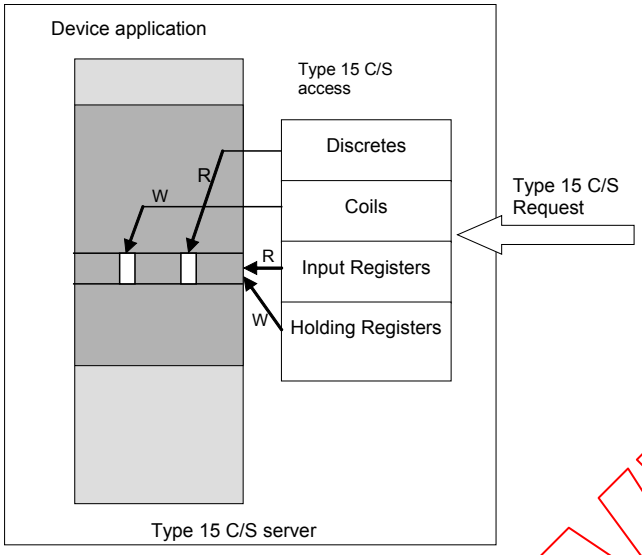


Figure 5 – [INFORMATIVE] Interpretation as overlapping tables

To emphasize what is not implied, it is worth mentioning yet another not obvious interpretation (informative) often found in the field: different requests performed on the same APO do not necessarily entail requests to the same real object. This is illustrated in Figure 6.

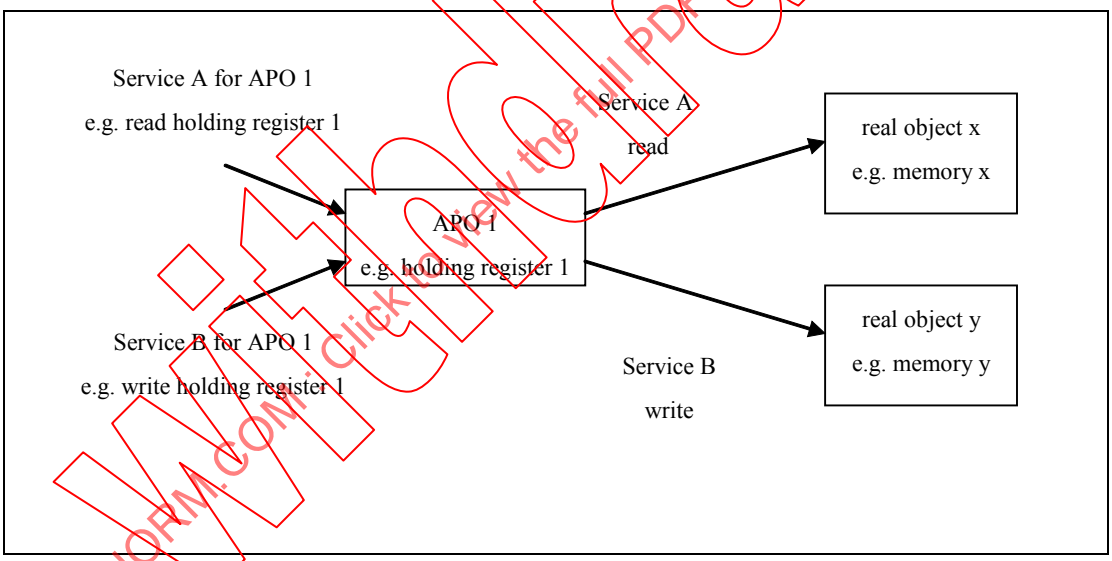


Figure 6 – [INFORMATIVE] APO and real objects, non obvious possible interpretation

Discretes, coils, input registers and holding registers are often collectively called data references or data items. For each of the above mentioned tables, the protocol allows individual selection of 65536 data items, and the operations of read or write of those items are designed to span, from an application user (client user) point of view, multiple consecutive data items up to a data size limit which is dependent on the service transaction function code.

The possible association of client/server data references (bits, registers) and actual physical storage within devices is a local issue.

The client/server logical data reference addresses, used in client/server service function codes, are unsigned integers starting at zero.

One potential source of confusion is the relationship between the data reference addresses used in client/server service function codes, and the discrete, coil or register 'user' reference

addresses, as seen e.g. in a PLC Ladder Logic program. For historical reasons user reference addresses were expressed as decimal numbers with a starting offset of 1, however client/server uses the more natural software convention starting at 0. Due to this, for example, a client/server message requesting to read a register via logical data reference at offset 0 would return what the application programmer considers register 1. There is an exception, the addressing of records via registers using client/server service function codes Read File Record and Write File Record, where record 0, a 'user' reference, is addressed specifying 0 as its first register. Details are given in the mentioned services specifications.

4.2.2.3 Record and file APOs

The record APO, from an application user (client user) point of view, is a set of contiguous registers of a specified type, characterized by the address of the first register and by the quantity of registers; in the context of this definition, the registers involved have also been called references.

The file APO is an organization of records, characterized by an unsigned number.

4.2.2.4 Interface APO

The encapsulated interface is a mechanism encapsulating a service for an interface, which is an application process object characterized by an MEI type. The MEI type, specified as an octet value, is used to dispatch the service to the appropriate interface.

While it is a simple mechanism, since it defers all the semantics to the interface, it allows maximum flexibility in what it can represent and convey.

4.2.3 Client/server function codes

Function codes are encodings of services requested to a server. These encodings are partitioned in three categories:

Publicly assigned function codes

These function codes are either assigned to a standard service or reserved for future assignment.

User definable function codes

These function codes can be used for experimentation in a controlled laboratory environment. They must not be used in an open environment.

Reserved function codes

These function codes are currently used by some companies for legacy products and are not available for public use.

NOTE 1 Function code assignments are managed by the Modbus-IDA industrial consortium.

NOTE 2 The following function codes, while publicly assigned, are not covered by this specification: FC 0x07 (Read Exception Status), FC 0x08 (Diagnostics), FC 0x0B (Get Com Event Counter), FC 0x0C (Get Com Event Log), FC 0x11 (Report Slave ID).

4.2.4 Client/server communication model overview

4.2.4.1 General

Figure 7 is a representation of the end-to-end communication model.

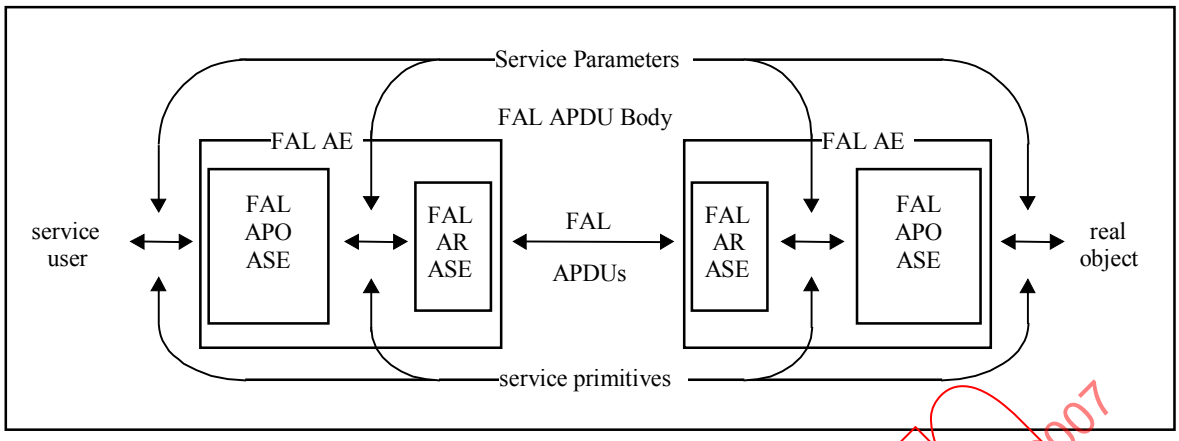


Figure 7 – ASE service conveyance

Client/server uses a Client/Server AR with confirmed interaction, and optionally when broadcast is supported, a Client/Server AR with unconfirmed interaction.

4.2.4.2 Client/server device addressing and connection

Client/server devices are addressed using a Unit ID. The Unit ID needs to be unique across all the servers addressable by a client. The set of addressable servers is determined by the underlying layer. This set is sometimes called connection.

The Unit ID assignment is outside of the scope of this specification.

The Unit ID identifies logical devices. There may be more than one logical device per physical device.

Some values of Unit ID are reserved and have particular meanings. The value 0 is reserved for broadcast.

NOTE In general the Unit ID is only required for logical devices having the role of servers. Often logical devices can have either role or multiple roles, via configuration, and their Unit ID is not used when they only perform in the client role. Depending on the underlying layers some devices can have concurrently a client and a server role on the same access point, this is the case for client/server on Token Bus/HDLC, outside the scope of this standard.

4.2.4.3 Client/server cardinality, connections and transaction objects

A client/server device may have many clients and many servers, depending on the device and on the underlying layers.

The clients and servers are associated by a connection, which in the scope of this standard is equivalent to an AR.

Clients carry a service on a connection with the help of a transaction object, managed (created and destroyed) by the client. The transaction object mechanism is exposed at the application layer due to the client/server possibility of having more than one outstanding request at a time, with the consequent need to properly associate requests and confirmations. It also controls the maximum number of such requests, which could be 1. The capabilities of a client/server application layer client depend on lower layers and on the particular implementation; these factors are captured in the configuration of the transaction object mechanism allowing programmatic adaptation.

The transaction object is instantiated for the duration of a service invocation. Every transaction object needs to be uniquely identifiable within a connection.

There can be many servers on the same connection. There can be many clients on the same connection as well, but when this is the case then only one client on that connection can be active at a time. For confirmed services, a client is active from the time it sends a request to the time it receives the corresponding confirmation, and this time is contained within the instantiation and subsequent destruction of the transaction object involved. For unconfirmed services, a client is active for an amount of time that is device and connection specific, and it must itself wait for a previous activity time to elapse before engaging in another one. The enforcement of the single active client per connection at a time is outside the scope of this specification, as is the timing of activities for unconfirmed services.

A client can issue only one service invocation per transaction object. Depending on the device there may be only one connection, or a limited number of connections. Depending on the underlying layer and local settings, a client may not be allowed to instantiate more that a limited number of transaction objects at a time.

In case of connections where multiple clients per connection are allowed, those clients can only instantiate one transaction object at a time, obtaining the above mentioned activity status when they do so. Some connections with a single client may have the same restriction.

If a client can only instantiate one transaction object at a time then, when invoking a service, it is not necessary to dynamically create a transaction object and exchange it with lower layers, since the main reason for having a transaction object is to properly couple requests with confirmations, which is automatic for these clients. In these situations a single static transaction object effectively acts like a client token, which is either taken or available, and only of interest to the client.

In all cases, the establishment of a connection and the needed configuration for the transaction object instantiation are outside the scope of this specification. Both are expected to be properly established.

4.2.4.4 Confirmed interaction

Client/server is based on request/response messages, and the model is that of a client/server interaction. With this relationship, a client issues a request as a confirmed service to a server, and the confirmation of the response from the server concludes the transaction. This interaction is shown in Figure 8.

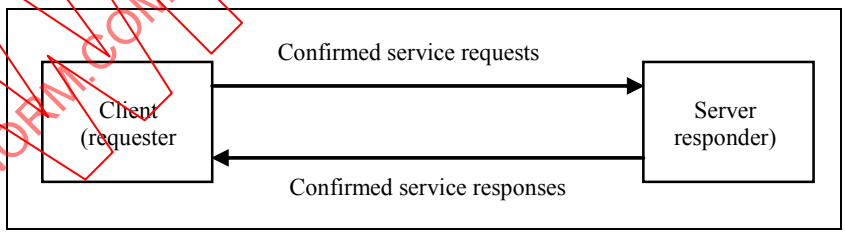


Figure 8 – Client/server confirmed interaction

This is accomplished using the Client/Server AR primitives, as shown in Figure 9 and Figure 10.

A client should start a transaction timer after issuing a confirmed service request. This timer should be set to an implementation dependent maximum time allowed for the confirmation, resetting it on receipt or reporting a service provider error on expiration.

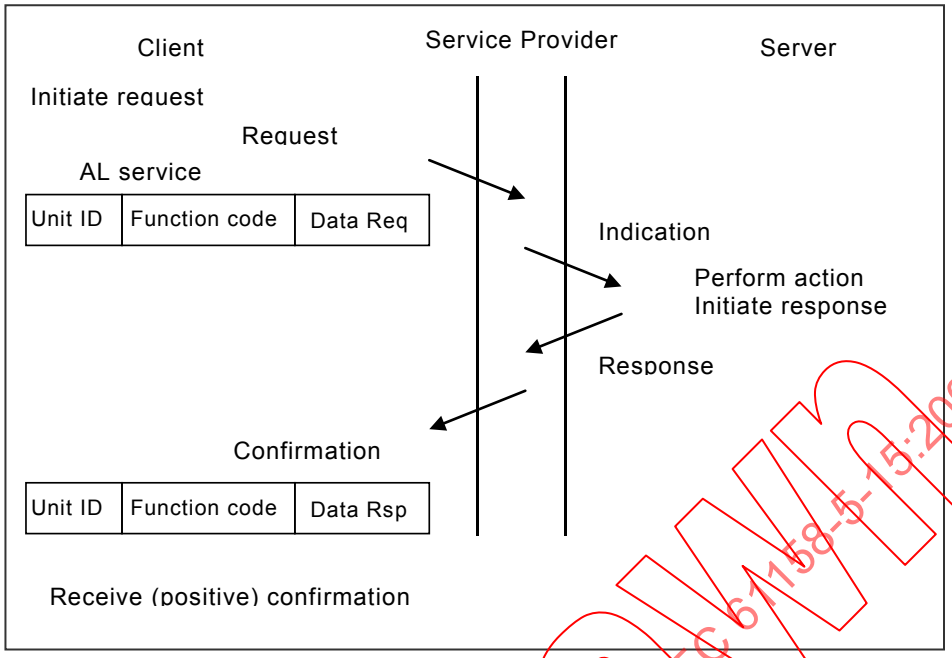


Figure 9 – Client/server AR confirmed service primitives (positive case)

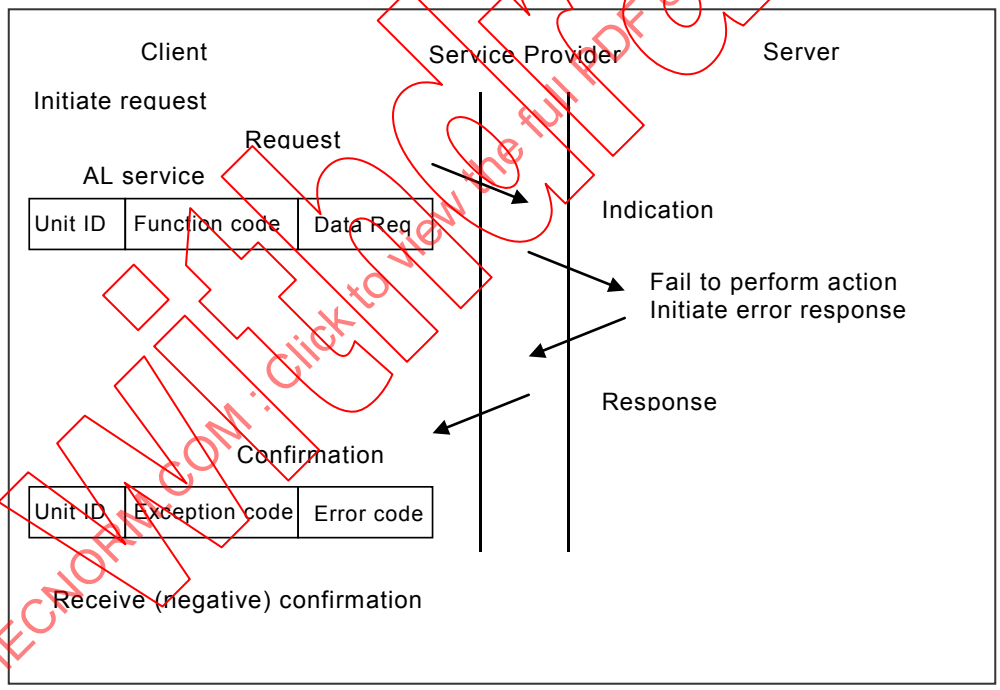


Figure 10 – Client/server AR confirmed service primitives (negative case)

4.2.4.5 Unconfirmed interaction

Optionally, clients and servers may support unconfirmed services in the form of client issued broadcast messages. This interaction is shown in Figure 11.

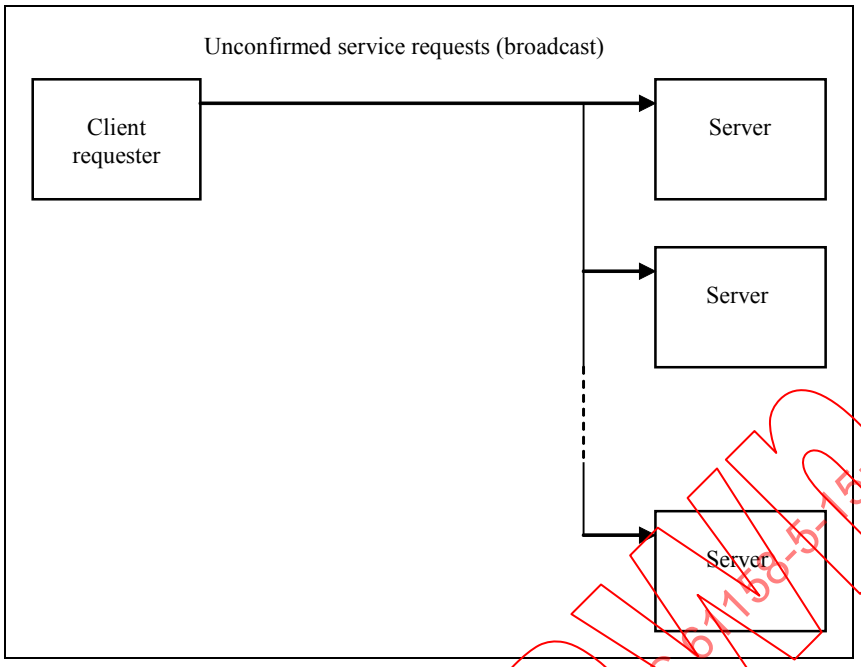


Figure 11 – Client/server unconfirmed interaction

This is accomplished using the Client/Server AR primitives, as shown in Figure 12.

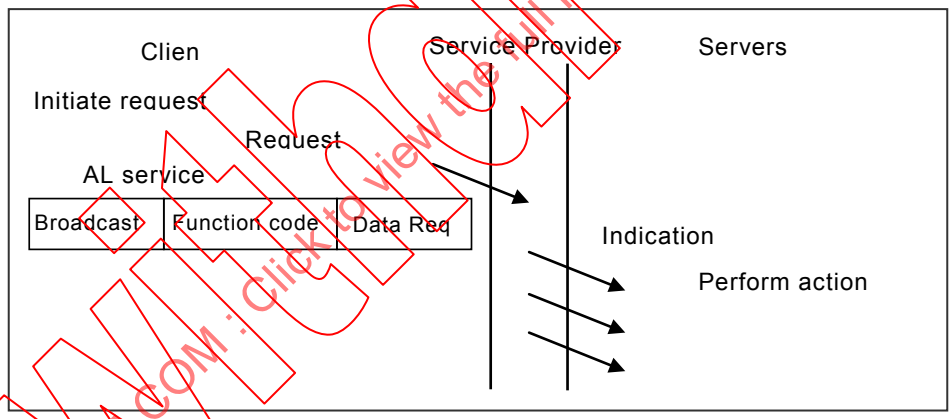


Figure 12 – Client/server AR unconfirmed service primitives

4.3 Publish/subscribe specific concepts

4.3.1 Overview

Publish/Subscribe (P/S) describes a Publish/Subscribe communication mechanism between networked devices. publish/subscribe offers a number of QoS characteristics, and it is designed with upward compatible extensibility features, so that, among other things, the QoS set can be easily augmented, using versioning or with vendor specific extensions. This specification will indicate the extension possibilities but it will not detail them, since outside the scope. It is important to realize that publish/subscribe is a wire protocol, it exposes many possibilities via configuration and extensions, and not all behaviors are documented in this specification.

NOTE A higher application services layer, dictating optimal usage and taking advantage of easy to accommodate 'designed-in' extensions for publish/subscribe, has been recently standardized by the OMG: "Data Distribution Service for Real-Time Systems Specification, Version 1.1, December 2005".

The publish/subscribe AL has minimal requirements on the underlying layers, and it could be deployed using memory offsets on shared memory or ports on UDP. publish/subscribe supports a wide variety of transports and transport QoS. The protocol is designed to be able to run on multicast and best-effort transports, such as UDP/IP, and requires only very simple services from the transport. In fact, it is sufficient that the transport offers a connectionless service capable of sending packets best-effort. That is, the transport need not guarantee each packet will reach its destination or that packets are delivered in-order. Where required, publish/subscribe implements reliability in the transfer of data and state above the transport interface. This does not preclude publish/subscribe from being implemented on top of a reliable transport. It simply makes it possible to support a wider range of transports.

The general requirements publish/subscribe asks to the underlying transport can be summarized as follows:

— the transport has a generalized notion of a unicast address (shall fit within 16 octets);

— the transport has a generalized notion of a port (shall fit within 4 octets), e.g. could be a UDP port, an offset in a shared memory segment, etc.;

— the transport can send a datagram (uninterpreted sequence of octets) to a specific address/port;

— the transport can receive a datagram at a specific address/port;

— the transport will drop messages if incomplete or corrupted during transfer (i.e. publish/subscribe assumes messages are complete and not corrupted);

— the transport provides a means to deduce the size of the received message.

A publish/subscribe stack configuration is illustrated in Figure 13.

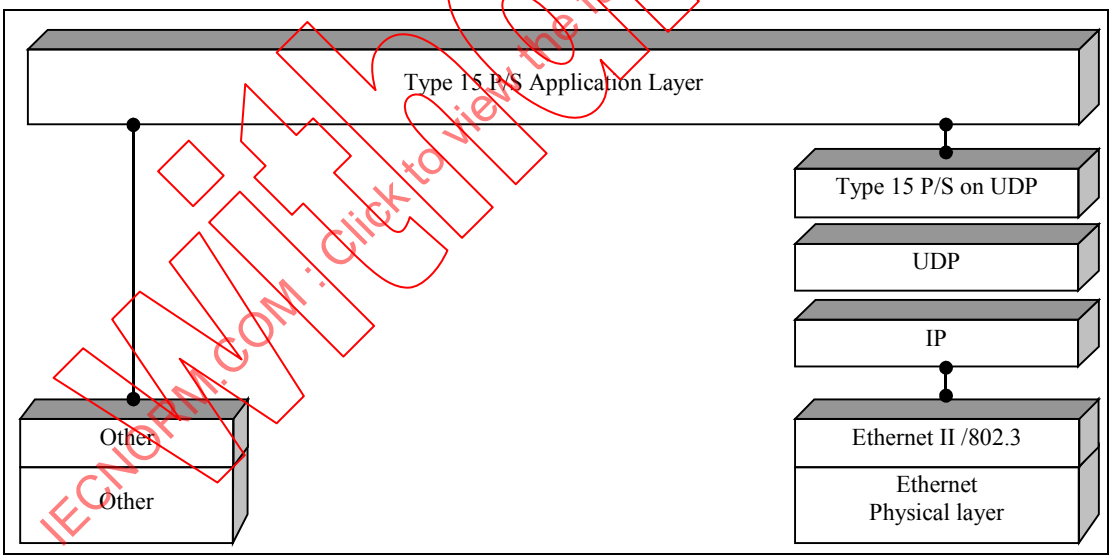


Figure 13 – Publish/subscribe communications stacks

If available, publish/subscribe can also take advantage of the multicast capabilities of the transport mechanism, where one message from a sender can reach multiple receivers.

Publish/subscribe is designed to promote determinism of the underlying communication mechanism; it provides an open trade-off between determinism and reliability. Among other features, determinism is monitored with the use of deadlines. Reliability is provided using queues, acknowledgements and retransmission. The reliability is window based, implemented natively, allowing full control and flexibility of the aforementioned trade-off.

4.3.2 Data-centric, match, decoupling and scalability

Publish/subscribe is organized around publishers and subscribers, with the additional characteristic of being data centric.

In this architecture, often called DCPS for data centric publish subscribe, the attributes of an exchange are on the data itself. In addition to the content proper, that may or may not be a discriminating factor, the operating traits of a publisher are reflected in the data it publishes, and the expectations of a subscriber are looked for in the data it subscribes to.

The above allows for a high degree of customization regarding data and its exchanges, and a match based on data, sometimes even down to parts of the content, with no direct knowledge between publishers and subscribers. This decoupling is the major reason for the scalability properties of this approach, it is easy to add and remove publishers and subscribers.

Figure 14 illustrates a scenario where applications exchange data via publishers and subscribers, unknown to each others.

The match is made by considering attributes of the data, like Topic name, Topic type, and other qualifying characteristics.

NOTE Publish/subscribe extensions allow for the concept of "domains", effectively permitting isolation of application exchanges within domains. While OMG DDS as in "Data Distribution Service for Real-Time Systems Specification, Version 1.1, December 2005" uses this extension, the feature will not be examined further in this specification, which will consider a single domain.

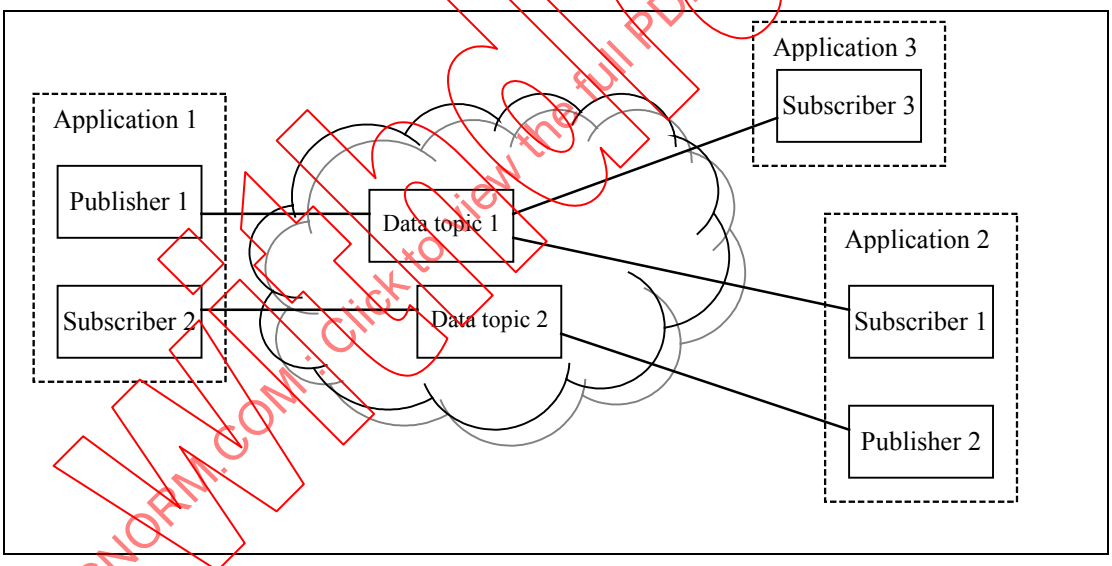


Figure 14 – Publish/subscribe data-centric exchanges between decoupled network objects

4.3.3 Publish/subscribe APOs

4.3.3.1 General

An application process object is a network representation of a specific aspect of an AP. Each APO represents a specific set of information and processing capabilities of an AP that are accessible through services of the FAL. APOs are used to represent these capabilities to other APs in a fieldbus system.

From the perspective of the FAL, an APO is modeled as a network accessible object contained within an AP or within another APO (APOs may contain other APOs). APOs provide the network definition for objects contained within an AP that are remotely accessible. The definition of an APO includes an identification of the FAL services that can be used by remote

APs for remote access. The FAL services, as shown in Figure 15, are provided by the FAL communications entity of the AP, known as the FAL Applications Entity (FAL AE).

Publish/subscribe APOs are constructed dynamically, and are exchanged from publishers to subscribers.

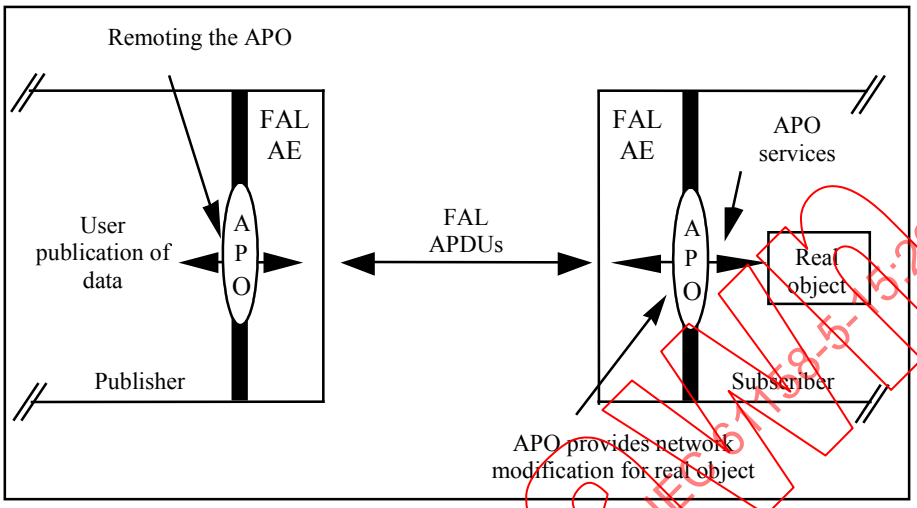


Figure 15 – Publish/subscribe APOs services conveyed by the FAL

In Figure 15, remote APs acting as publishers may modify the real object by publishing data that contributes changes to the remote APO. publish/subscribe remotes the APO itself, by publishing it. Once the APO is acquired by the subscriber, local aspects of the AP convert between the network view (the APO) of the real object and the internal AP view of the real object.

4.3.3.2 Data topic APO

The publish/subscribe APO is the data exchanged itself, the publication, here called topic to emphasize the fact that it is identifiable data.

The topic is characterized by the following:

— topic name;
— optional topic type;
— optional topic representational code.

The offered publication topic is looked at by an expected subscription topic, and a match is attempted. The matching is done on all three components, with the following considerations:

— the name components must match, but this is not required to be a literal match, as it can be done via regular expression pattern matching;
— if the topic type of one of the matching sides is the empty string, then that component is a match, and the additional meaning is that no type checking against the type information carried by this component should be done; the type information is expected to be available by other means; if the topic type of both sides is not the empty string then they must match;
— if the topic representational code of one of the matching sides is 0, then that component is a match, and the additional meaning is that there is no extra information available about the topic type and its marshalling; the type information is expected to be available by other means; if the topic representational code of both sides is not 0 then they must match.

The offered publication topics and the expected subscription topics carry with themselves additional information about the data production and consumption, respectively, via optional parameters. This is the information that enables QoS, and it is available on a publisher, topic and subscriber basis. The kind and quantity of additional information is configurable; the following (Informative) is a commonly used set:

publication topic

— topic (name, type, representational code);

— strength;

— persistence.

The strength is the precedence of the issue sent by the Publication; the persistence indicates how long the issue is valid. Strength and persistence allow the receiver to arbitrate if issues are received from several matching publications.

subscription topic

— topic (name, type, representational code);

— minimum separation;

— deadline.

The minimum separation is the minimum time between two consecutive issues received by the Subscription. It defines the maximum rate at which the Subscription is prepared to receive issues. Publications sending to this Subscription should try to send issues so that they are spaced at least this far apart.

The above configuration enables the behaviors illustrated in Figure 16.

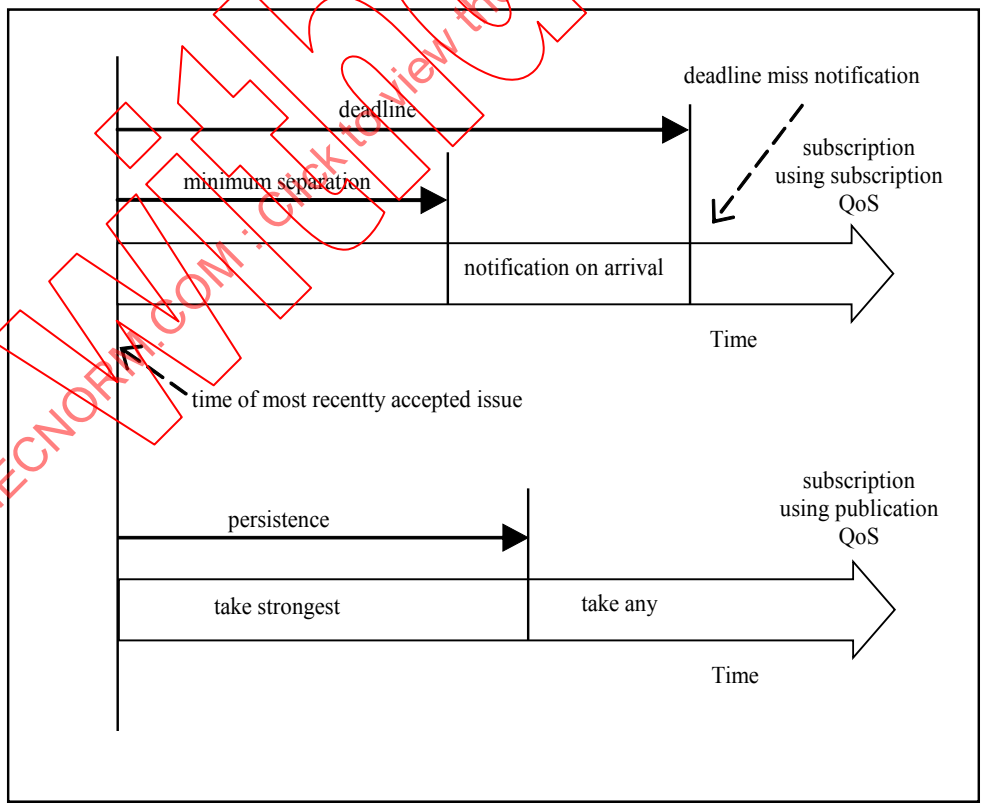


Figure 16 – [INFORMATIVE] Examples of publish/subscribe configurable behaviors via QoS

4.3.4 Cardinality and fault tolerance

Publish/subscribe exchanges can be

— one to one;

— one to many;

— many to one;

— many to many.

The “many to ..” possibilities directly support high availability with transparent failover.

4.3.5 Queues and reliability

Queues and acknowledgements configurable parameters allow an application to behave from best effort to strictly reliable.

4.3.6 Publish/subscribe communication model overview

4.3.6.1 General

Publish/subscribe interactions are based on data/events sent by one AP for use by others. This model is referred to as publisher/subscriber interactions.

The services supported by an interaction model are conveyed by application relationship endpoints (AREPs) associated with the communicating APs. The role that the AREP plays in the interaction (e.g. client, server, peer, publisher, subscriber) is defined as an attribute of the AREP.

While the general communication model architected around the APO exchanges is indeed that of publisher/subscriber interactions exchanging data topics, publish/subscribe AL supports the model using several unconfirmed services. Some publish/subscribe unconfirmed interactions do demand confirmation, but the confirmation is deferred to the AL user. From a communications perspective, there is no relationship between separate invocations of unconfirmed services as there is between the request and response of a confirmed service.

NOTE OMG DDS as in “Data Distribution Service for Real-Time Systems Specification, Version 1.1, December 2005” defines services that are here presented as responsibility of the publish/subscribe AL user instead. publish/subscribe is a wire protocol, and as such some functionality is deferred to the user application. This can be appreciated by the flexibility offered for implementations where foot-print is at a premium, and contributes to the pervasiveness of the approach.

4.3.6.2 Publisher/subscriber interactions

Common publisher/subscriber interactions involve a single publisher AP, and a group of one or more subscriber APs. This type of interaction has been defined to support variations of two models of interaction between APs, the "pull" model and the "push" model. In both models, the setup of the publishing AP is performed by management and is outside the scope of this standard.

4.3.6.2.1 Pull model interactions

In the "pull" model, the publisher receives a request to publish from a remote publishing *manager*, and broadcasts (or multicasts) its response across the network. The publishing manager is responsible only for initiating publishing by sending a request to the publisher.

Subscribers wishing to receive the published data listen for responses transmitted by the publisher. In this fashion, data is "pulled" from the publisher by requests from the publishing manager.

Confirmed FAL services are used to support this type of interaction. Two characteristics of this type of interaction differentiate it from the other types of interaction. First, a typical confirmed request/response exchange is performed between publishing manager and the publisher. However, the underlying conveyance mechanism provided by the FAL returns the response not just to the publishing manager, but also to all subscribers wishing to receive the published information. This is accomplished by having the Data Link Layer transmit the response to a group address, rather than to the individual address of the publishing manager. Therefore, the response sent by the publisher contains the published data and is multicast to the publishing manager and to all subscribers.

The second difference occurs in the behavior of the subscribers. Pull model subscribers, referred to as pull subscribers, are capable of accepting published data in confirmed service responses without having issued the corresponding request. Figure 17 illustrates these concepts.

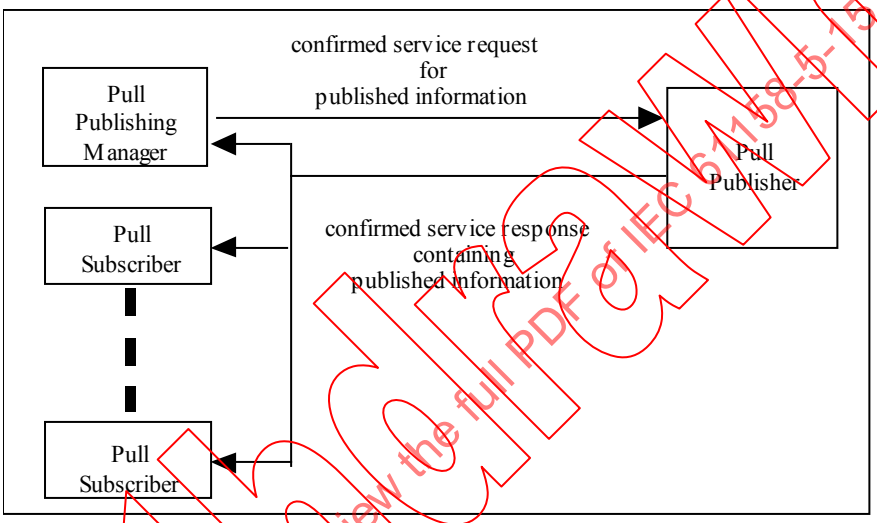


Figure 17 – Pull model interactions

4.3.6.2.2 Push model interactions

In the "push" model, two services may be used, one confirmed and one unconfirmed. The confirmed service is used by the subscriber to request to join the publishing. The response to this request is returned to the subscriber, following the client/server model of interaction. This exchange is only necessary when the subscriber and the publisher are located in different APs.

The unconfirmed service used in the Push Model is used by the publisher to distribute its information to subscribers. In this case, the publisher is responsible for invoking the correct unconfirmed service at the appropriate time and for supplying the appropriate information. In this fashion, it is configured to "push" its data onto the network.

Subscribers for the Push Model receive the published unconfirmed services distributed by publishers. Figure 18 illustrates the concept of the Push Model.

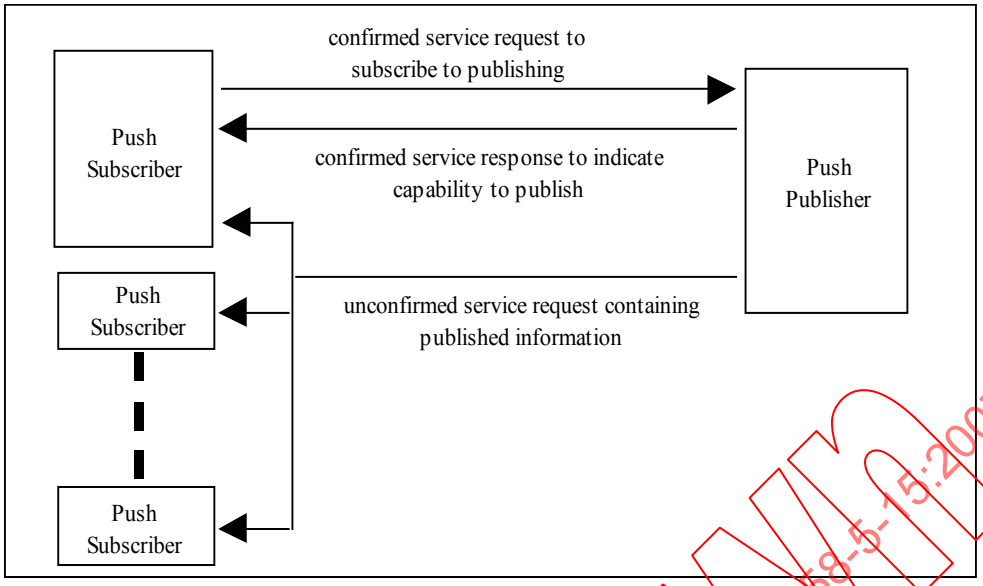


Figure 18 – Push model interactions

4.3.6.3 Publish/subscribe interactions

Publish/subscribe interactions are a variation of the push model interactions, this is illustrated via callouts in Figure 19.

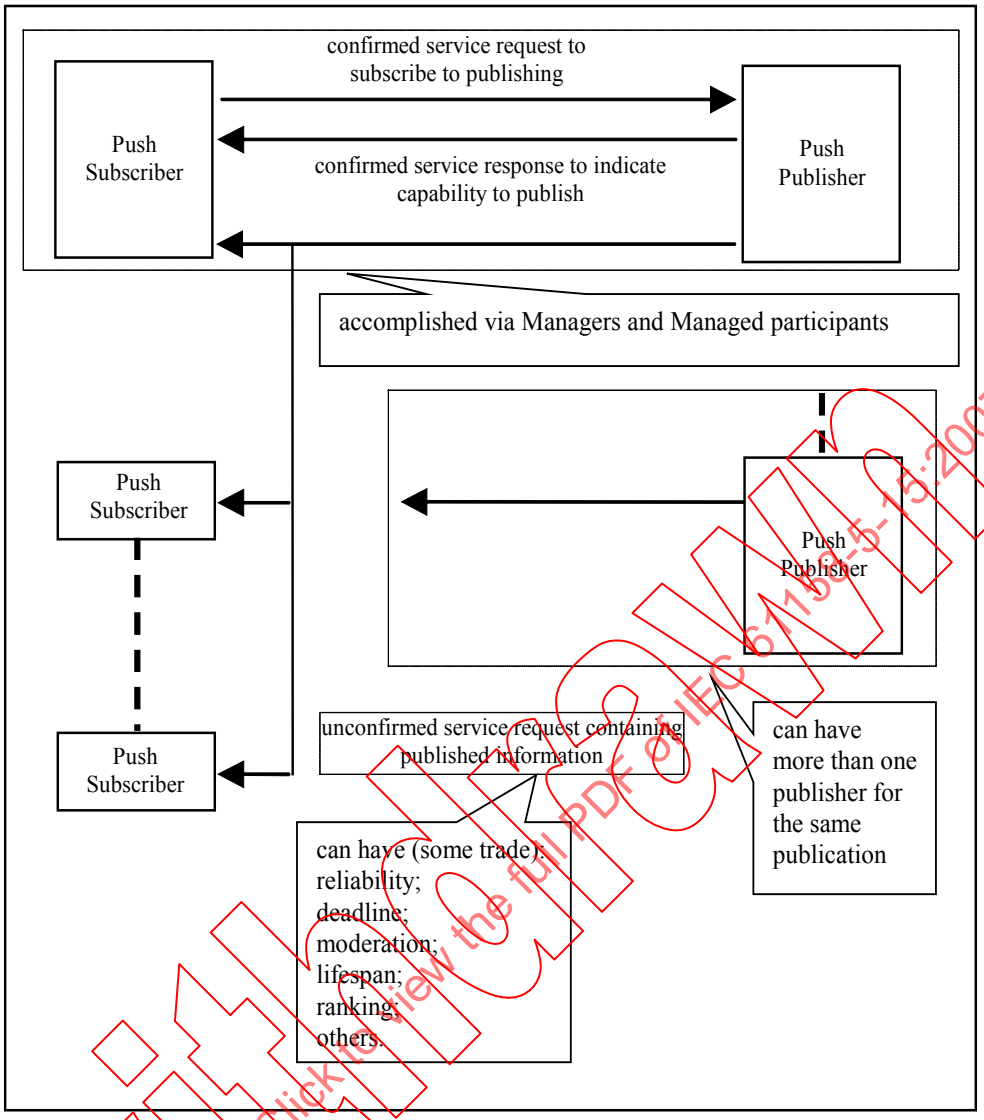


Figure 19 – Publish/subscribe model interactions

Managers and managed participants are publish/subscribe specialized applications, both containing publishers and subscribers or specialized versions of them.

The distinction between managers and managed participants is due to their role in the publish/subscribe discovery and maintenance mechanism. This mechanism allows publishing offers and subscription demands to topics to transparently match and properly behave even when publishers and subscribers come and go, dynamically.

Publish/subscribe publishers and subscribers, and their specialized versions, are collectively called communication actors. Their presence and attributes are the end result of the discovery and maintenance mechanism discussed.

It should be realized that the importance of the discovery and maintenance mechanism is in its accomplishments. Its optimality depends on the AL underlying layers, number of nodes and other hosting system features, therefore the discovery and maintenance mechanism described in this specification is just one of many, and as long as the end result is preserved, it can be replaced.

5 Data type ASE

5.1 General

All of IEC/TR 61158-1, 10.1 is incorporated by reference.

5.2 Formal definition of data type objects

All of IEC/TR 61158-1, 10.2 is incorporated by reference.

5.3 FAL defined data types

5.3.1 Common

5.3.1.1 Fixed length types

5.3.1.1.1 Boolean types

5.3.1.1.1.1 Boolean

CLASS: **Data Type**
ATTRIBUTES:

1	Data Type Numeric Identifier	=	1
2	Data Type Name	=	Boolean
3	Format	=	FIXED LENGTH
4.1	Octet Length	=	1

This data type expresses a Boolean data type with the values TRUE and FALSE.

5.3.1.1.2 Bitstring types

Common not used.

5.3.1.1.3 Currency types

Common not used.

5.3.1.1.4 Date/Time types

Common not used.

5.3.1.1.5 Enumerated types

Common not used.

5.3.1.1.6 Handle types

Common not used.

5.3.1.1.7 Numeric types

Common not used.

5.3.1.1.7.1 Floating Point types

Common not used.

5.3.1.1.7.2 Integer types

####### 5.3.1.1.7.2.1 long

This data type is the same as Integer32.

5.3.1.1.7.2.2 Integer32

CLASS: **Data Type**

ATTRIBUTES:

1	Data Type Numeric Identifier	=	4
2	Data Type Name	=	Integer32
3	Format	=	FIXED LENGTH
4.1	Octet Length	=	4

This integer type is a two's complement binary number with a length of four octets.

5.3.1.1.7.3 Unsigned types

5.3.1.1.7.3.1 Unsigned8

CLASS: **Data Type**

ATTRIBUTES:

1	Data Type Numeric Identifier	=	5
2	Data Type Name	=	Unsigned8
3	Format	=	FIXED LENGTH
4.1	Octet Length	=	1

This type is a binary number. The most significant bit of the most significant byte is always used as the most significant bit of the binary number; no sign bit is included. This type has a length of one octet.

5.3.1.1.7.3.2 USINT

This IEC 61131-3 type is the same as Unsigned8.

5.3.1.1.7.3.3 unsigned char

This data type is the same as Unsigned8.

5.3.1.1.7.3.4 Unsigned16

CLASS: **Data Type**

ATTRIBUTES:

1	Data Type Numeric Identifier	=	6
2	Data Type Name	=	Unsigned16
3	Format	=	FIXED LENGTH
4.1	Octet Length	=	2

This type is a binary number. The most significant bit of the most significant byte is always used as the most significant bit of the binary number; no sign bit is included. This unsigned type has a length of two octets.

5.3.1.1.7.3.5 UINT

This IEC 61131-3 type is the same as Unsigned16.

5.3.1.1.7.3.6 Unsigned32

CLASS: **Data Type**

ATTRIBUTES:

1	Data Type Numeric Identifier	=	7
2	Data Type Name	=	Unsigned32
3	Format	=	FIXED LENGTH
4.1	Octet Length	=	4

This type is a binary number. The most significant bit of the most significant byte is always used as the most significant bit of the binary number; no sign bit is included. This unsigned type has a length of four octets.

5.3.1.1.7.3.7 WORD

This type is used in the same way as UINT.

5.3.1.1.8 Pointer types

Common not used.

5.3.1.1.9 OctetString types

5.3.1.1.9.1 OctetString

CLASS: **Data Type**

ATTRIBUTES:

1	Data Type Numeric Identifier	=	10
2	Data Type Name	=	OctetString
3	Format	=	STRING
4.1	Octet Length	=	1 to n

An OctetString is an ordered sequence of octets, numbered from 1 to n. For the purposes of discussion, octet 1 of the sequence is referred to as the first octet. IEC 61158-6 defines the order of transmission.

5.3.1.1.10 VisibleString character types

Common not used.

5.3.1.2 String types

Common not used.

5.3.1.3 Structure types

Common not used.

5.3.2 FAL defined data types for client/server

5.3.2.1 Fixed length types

5.3.2.1.1 Boolean types

No type specific boolean types.

5.3.2.1.2 Bitstring types

5.3.2.1.2.1 Status flag

CLASS: **Data Type**

ATTRIBUTES:

1	Data Type Numeric Identifier	=	Not used
2	Data Type Name	=	Status flag
3	Format	=	FIXED LENGTH
4.1	Octet Length	=	2

This data type expresses a Status data type, with the values ON and OFF, encoded in two octets. The encoding for ON is 0xFF00 and the encoding for OFF is 0x0000.

5.3.2.1.2.2 Status bit sequence

CLASS: **Data Type**

ATTRIBUTES:

1	Data Type Numeric Identifier	=	Not used
2	Data Type Name	=	Status bit sequence
3	Format	=	STRING
4.1	Octet Length	=	1 to n

This data type expresses a Status bit sequence data type, with the values ON and OFF, encoded one value per bit. The encoding for ON is 1 and the encoding for OFF is 0. The bits are contained in octets, with the last octet 0-padded if the number of bits in the sequence is not a multiple of 8. The bits are numbered in the order as reported below, where y is a Status bit number, x is a Status bit value and – is an empty location:

	MSB							LSB
Bit	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**
Octet 1	7	6	5	4	3	2	1	0
Status	x	x	x	x	x	x	x	x
Octet 2	15	14	13	12	11	10	9	8
Status	x	x	x	x	x	x	x	x
...								
...								
Octet n	-	-	-	-	-	y	y	y
Status	0	0	0	0	0	x	x	x

5.3.2.1.3 Currency types

No type specific currency types.

5.3.2.1.4 Date/Time types

No type specific date/time types.

5.3.2.1.5 Enumerated types

No type specific enumerated types.

5.3.2.1.6 Handle types

No type specific handle types.

5.3.2.1.7 Numeric types

No type specific general numeric types.

5.3.2.1.7.1 Floating Point types

No type specific floating point types.

5.3.2.1.7.2 Integer types

No type specific integer types.

5.3.2.1.7.3 Unsigned types

No type specific unsigned types.

5.3.2.1.8 Pointer types

No type specific pointer types.

5.3.2.1.9 OctetString types

No type specific fixed length octet (byte) string types.

5.3.2.1.10 VisibleString character types

No type specific fixed length visible string types.

5.3.2.2 String types

5.3.2.2.1 ASCII string

CLASS:			**Data Type**
ATTRIBUTES:			
1	Data Type Numeric Identifier	=	Not used
2	Data Type Name	=	ASCII string
3	Format	=	STRING
4.1	Octet Length	=	1 to n

5.3.2.3 Structure types

No type specific structure types.

5.3.3 FAL defined data types for publish/subscribe

5.3.3.1 Fixed length types

5.3.3.1.1 Boolean types

No type specific boolean types.

5.3.3.1.2 Bitstring types

No type specific bitstring types.

5.3.3.1.3 Currency types

No type specific currency types.

5.3.3.1.4 Date/Time types

No type specific date/time types.

5.3.3.1.5 Enumerated types

No type specific enumerated types.

5.3.3.1.6 Handle types

No type specific handle types.

5.3.3.1.7 Numeric types

No type specific general numeric types.

5.3.3.1.7.1 Floating Point types

No type specific floating point types.

5.3.3.1.7.2 Integer types

No type specific integer types.

5.3.3.1.7.3 Unsigned types

5.3.3.1.7.3.1 IPAddress

CLASS: **Data Type**

ATTRIBUTES:

1	Data Type Numeric Identifier	=	Not used
2	Data Type Name	=	IPAddress
3	Format	=	Unsigned32

An IP address is a 4-octets unsigned number, as from RFC 791.

An IP address of zero is an invalid IP address

IPADDRESS_INVALID 0

The mapping between the dot-notation "a.b.c.d" of an IP address and its representation as an unsigned long is as follows:

IPAddress ipAddress = (((a * 256 + b) * 256) + c) * 256 + d

Example: IP address "127.0.0.1" corresponds to the unsigned long number 2130706433 or 0x7F000001.

5.3.3.1.7.3.2 Port

CLASS: **Data Type**

ATTRIBUTES:

1	Data Type Numeric Identifier	=	Not used
2	Data Type Name	=	Port
3	Format	=	Unsigned32

A port is a 4-octets unsigned number.

The port number zero is an invalid port-number

PORT_INVALID 0

If a port number represents an IPv4 UDP port, only the range of unsigned short numbers from 0x1 to 0x0000ffff is valid.

The following ports have been assigned to publish/subscribe by IANA:

— rtps-discovery 7400/tcp,udp RTPS Discovery

— rtps-dd-ut 7401/tcp,udp RTPS Data Distribution User Traffic

— rtps-dd-mt 7402/tcp,udp RTPS Data Distribution Meta Traffic

5.3.3.1.8 Pointer types

No type specific pointer types.

5.3.3.1.9 OctetString types

No type specific fixed length octet (byte) string types.

5.3.3.1.10 VisibleString character types

No type specific fixed length visible string types.

5.3.3.2 String types

No type specific string types.

5.3.3.3 Structure types

5.3.3.3.1 HostID

CLASS:			**Data Type**
ATTRIBUTES:			
1	Data Type Numeric Identifier	=	Not used
2	Data Type Name	=	HostID
3	Format	=	STRUCTURE
4	Number of Fields	=	1
5.1	Field Name	=	hostID
5.1.1	Field Data Type	=	OctetString
5.1.2	Field Length	=	4

The HostID is actually not structured at all, but just an OctetString. The distinction can be ignored for all practical purposes.

The value { 0x00, 0x00, 0x00, 0x00 } is reserved for HOSTID_UNKNOWN with the meaning of unknown HostID.

5.3.3.3.2 AppID

CLASS:			**Data Type**
ATTRIBUTES:			
1	Data Type Numeric Identifier	=	Not used
2	Data Type Name	=	AppID
3	Format	=	STRUCTURE
4	Number of Fields	=	2
5.1	Field Name	=	instanceID
5.1.1	Field Data Type	=	OctetString
5.1.2	Field Length	=	3
5.2	Field Name	=	appKind
5.2.1	Field Data Type	=	OctetString
5.2.2	Field Length	=	1
5.2.3	Field Content	=	0x01 or 0x02

An appKind value of 0x01 tags the application as a publish/subscribe managed participant.

An appKind value of 0x02 tags the application as a publish/subscribe manager.

An implementation based on this version (1.0) of the publish/subscribe will consider anything other than the above two to be an unknown class.

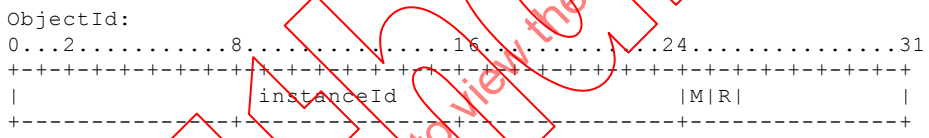
The value { 0x00, 0x00, 0x00, 0x00 } is reserved for APPID_UNKNOWN with the meaning of unknown AppID.

5.3.3.3.3 ObjectID

CLASS:			**Data Type**
ATTRIBUTES:			
1	Data Type Numeric Identifier	=	Not used
2	Data Type Name	=	ObjectID
3	Format	=	STRUCTURE
4	Number of Fields	=	2
5.1	Field Name	=	instanceID
5.1.1	Field Data Type	=	OctetString
5.1.2	Field Length	=	3
5.2	Field Name	=	objKind
5.2.1	Field Data Type	=	OctetString
5.2.2	Field Length	=	1

The value { 0x00, 0x00, 0x00, 0x00 } is reserved for OBJECTID_UNKNOWN with the meaning of unknown ObjectID.

For *objKind*, the two most significant bits indicate whether the object is meta-level or user-level (M- bit) and whether its *instanceId* is chosen or reserved (R-bit), respectively:



M=1 The network object is a meta-object: it can be reached through the meta-ports of the application to which it belongs (ports designated as meta-traffic ports).

R=1 The instanceId is reserved; it has a special meaning to the protocol, they are used by the discovery mechanism.

5.3.3.3.3.1 Reserved objects

These reserved objects are used by the discovery mechanism for the discovery of domain participants (or applications) and communication actors in the network.

Both managers and managed participants are applications, the distinction between managers and managed participants (managed applications) is due to their role in the publish/subscribe discovery and maintenance mechanism. This mechanism allows publishing offers and subscription demands to topics to transparently match and properly behave even when publishers and subscribers come and go, dynamically.

Publish/subscribe publishers and subscribers, and their specialized versions, are collectively called communication actors. Their presence and attributes are the end result of the discovery and maintenance mechanism discussed.

It should be realized that the importance of the discovery and maintenance mechanism is in its accomplishments. Its optimality depends on the AL underlying layers, number of nodes and other hosting system features, therefore the discovery and maintenance mechanism

described in this specification is just one of many, and as long as the end result is preserved, it can be replaced.

Every manager and every managed participant (or managed application) contains a number of special built-in network objects, which have reserved Object IDs.

These special objects fall into these categories:

— the domain participant (application) itself is a network object with a special GUID (the instance of the application is called *applicationSelf*). In addition, every application has a CSTWriter (*writerApplicationSelf*) that disseminates the attributes of the local application on the network;

— Several objects are dedicated to the discovery of managers and managed participants (managed applications) on the network. Every managed application has the CSTReaders *readerApplications* and *readerManagers*, through which the existence and attributes of the remote managed applications and remote managers, respectively, are obtained. Every manager has the corresponding CSTWriters *writeApplications* and *writeManagers*;

— Every managed application has, among others, two instances of a CSTReader (*readerPublications* and *readerSubscriptions*) and two instances of a CSTWriter (*writerPublications* and *writerSubscriptions*). Through the CSTReaders, the managed application can receive information about the existence and attributes of all the remote publications and subscriptions in the network. Through the CSTWriters, the managed application can send out information about its local publications and subscriptions.

5.3.3.3.3.2 Classes

The last six bits of the objectId define the class to which the object belongs (application, Publisher, Subscriber, CSTWriter, or CSTReader). Table 2 provides an overview. The meaning of the message IDs is fixed in this major version (1). New *objKinds* may be added in higher minor versions as the publish/subscribe object-model is extended with new classes.

Table 2 – Class identification

Class of object	Normal user-object	Reserved user-object	Normal meta-object	Reserved meta-object
unknown	0x00	0x40	0x80	0xc0
application	0x01	0x41	0x81	0xc1
CSTWriter	0x02	0x42	0x82	0xc2
Publisher	0x03	0x43	0x83	0xc3
Subscriber	0x04	0x44	0x84	0xc4
CSTReader	0x07	0x47	0x87	0xc7

5.3.3.3.4 GUID

CLASS: **Data Type**

ATTRIBUTES:

1	Data Type Numeric Identifier	=	Not used
2	Data Type Name	=	GUID
3	Format	=	STRUCTURE
4	Number of Fields	=	3
5.1	Field Name	=	hostID
5.1.1	Field Data Type	=	HostID
5.1.2	Field Length	=	4
5.2	Field Name	=	appID
5.2.1	Field Data Type	=	AppID
5.2.2	Field Length	=	4
5.3	Field Name	=	objectID

5.3.1	Field Data Type	=	ObjectID
5.3.2	Field Length	=	4

The GUID (Globally Unique ID) is a unique reference to a application that contains communication actors or to a communication actor on the network.

The GUID is built as a 12-octet triplet: <HostID hostId, AppID appId, ObjectID objectId>. The GUID should be a globally unique reference to one specific network object within the network.

5.3.3.3.4.1 The GUIDs of publish/subscribe applications

Every publish/subscribe application on the network has GUID <hostId, appId, OID_APP>, where the constant OID_APP is defined as follows: OID_APP {0x00,0x00,0x01,0xc1}.

The implementation is free to pick the *hostId* and *appId*, as long as the last octet of the *appId* identifies the *appKind* and as long as every application on the network has a unique GUID.

5.3.3.3.4.2 The GUIDs of communication actors within publish/subscribe applications

The communication actors that are local to the application with GUID <hostId, appId, OID_APP> have GUID <hostId, appId, *objectId*>. The *objectId* is the unique identification of the network object relative to the application. The *objectId* also encapsulates what kind of network object this is, whether the object is a user-object or a meta-object and whether the *instanceId* is freely chosen by the middleware or is a *reserved* instanceId, which has special meaning to publish/subscribe. One example of a reserved (protocol defined) objectId is OID_APP, which is used in the GUID of applications.

5.3.3.3.5 VendorID

CLASS:			**Data Type**
ATTRIBUTES:			
1	Data Type Numeric Identifier	=	Not used
2	Data Type Name	=	VendorID
3	Format	=	STRUCTURE
4	Number of Fields	=	2
5.1	Field Name	=	major
5.1.1	Field Data Type	=	OctetString
5.1.2	Field Length	=	1
5.2	Field Name	=	minor
5.2.1	Field Data Type	=	OctetString
5.2.2	Field Length	=	1

This structure identifies the vendor of the middleware implementing the publish/subscribe protocol and allows this vendor to add specific extensions to the protocol. The vendor ID does not refer to the vendor of the device or product that contains publish/subscribe middleware.

The currently assigned vendor IDs are listed in Table 3.

Table 3 – Assigned vendor IDs

Major	Minor	Name
0x00	0x00	VENDOR_ID_UNKNOWN
0x01	0x01	Real-Time Innovations, Inc., CA, USA

5.3.3.3.6 ProtocolVersion

CLASS:			**Data Type**
ATTRIBUTES:			
1	Data Type Numeric Identifier	=	Not used
2	Data Type Name	=	ProtocolVersion
3	Format	=	STRUCTURE
4	Number of Fields	=	2
5.1	Field Name	=	major
5.1.1	Field Data Type	=	OctetString
5.1.2	Field Length	=	1
5.2	Field Name	=	minor
5.2.1	Field Data Type	=	OctetString
5.2.2	Field Length	=	1

Implementations following this version of the specification implement protocol version 1.0 (major = 1, minor = 0).

5.3.3.3.7 SequenceNumber

CLASS:			**Data Type**
ATTRIBUTES:			
1	Data Type Numeric Identifier	=	Not used
2	Data Type Name	=	SequenceNumber
3	Format	=	STRUCTURE
4	Number of Fields	=	2
5.1	Field Name	=	high
5.1.1	Field Data Type	=	Integer32
5.1.2	Field Length	=	4
5.2	Field Name	=	low
5.2.1	Field Data Type	=	Unsigned32
5.2.2	Field Length	=	4

A sequence number, N, is a 64-bit signed integer, that can take values in the range: -2^63 <= N <= 2^63-1.

Using this structure, the sequence number is: *high* * 2^32 + *low*.

The sequence number is used to uniquely identify elementary publish/subscribe messages in an ordered manner.

The sequence number 0 and negative sequence numbers are used to indicate special cases:

SEQUENCE_NUMBER_NONE 0

SEQUENCE_NUMBER_UNKNOWN -1

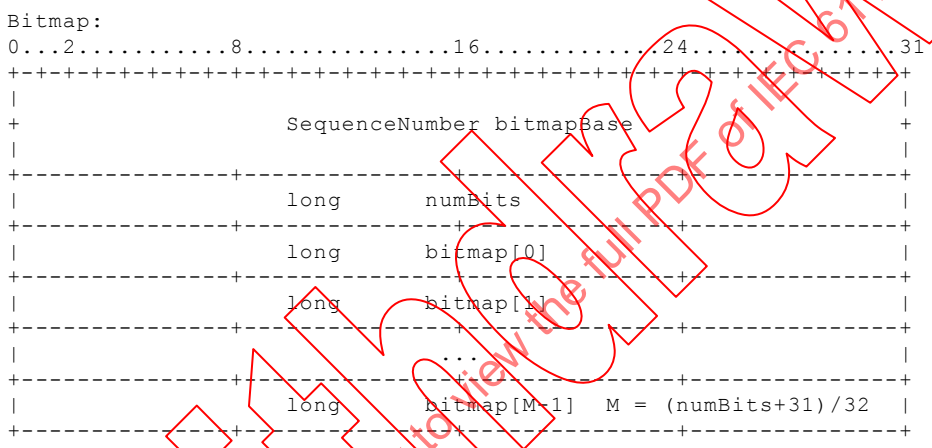
NOTE The selection of 64 bits as the representation of a sequence number ensures the sequence numbers never wrap. Even assuming an extremely fast rate of message generation for a single publish/subscribe Writer such as 100 messages per microsecond, the 64-bit integer would not roll over for approximately 3000 years of uninterrupted operation.

5.3.3.3.8 Bitmap

CLASS:			**Data Type**
ATTRIBUTES:			
1	Data Type Numeric Identifier	=	Not used

2	Data Type Name	=	Bitmap
3	Format	=	STRUCTURE
4	Number of Fields	=	3
5.1	Field Name	=	bitmapBase
5.1.1	Field Data Type	=	SequenceNumber
5.1.2	Field Length	=	8
5.2	Field Name	=	numBits
5.2.1	Field Data Type	=	long
5.2.2	Field Length	=	4
5.3	Field Name	=	bitmapArray
5.3.1	Field Data Type	=	ARRAY
5.3.2	Number of Array Elements	=	M where M = (numBits + 31)/32
5.3.3	Array Element Data Type	=	long

Bitmaps are used as parts of several messages (services) to provide binary information about individual sequence numbers within a range. The representation of the Bitmap includes the length of the Bitmap in bits and the first SequenceNumber to which the Bitmap applies.



Given a Bitmap, *bitmap*, the boolean value of the bit pertaining to SequenceNumber *N*, where *bitmapBase <= N < bitmapBase+numBits,* is:

bit(N) = bitmap[deltaN/32] & (1 << (31 - deltaN%32))

where

deltaN = N - bitmapBase

The bitmap does not indicate anything about sequence numbers outside of the range [bitmapBase, bitmapBase+numBits-1].

A valid bitmap must satisfy the following conditions:

— bitmapBase >= 1

— 0 <= numBits <= 256

— there are M=(numBits+31)/32 longs containing the pertinent bits

This document uses the following notation for a specific bitmap:

bitmapBase/numBits:bitmap

In the bitmap, the bit corresponding to sequence number *bitmapBase* is on the left. The ending "0" bits can be represented as one "0".

Example: in bitmap “1234/12:00110”, bitmapBase=1234 and numBits=12. The bits apply as follows to the sequence numbers:

Sequence	Bit
1234	0
1235	0
1236	1
1237	1
1238-1245	0

5.3.3.3.9 NtpTime

CLASS: **Data Type**

ATTRIBUTES:

1	Data Type Numeric Identifier	=	Not used
2	Data Type Name	=	NtpTime
3	Format	=	STRUCTURE
4	Number of Fields	=	2
5.1	Field Name	=	seconds
5.1.1	Field Data Type	=	long
5.1.2	Field Length	=	4
5.2	Field Name	=	fraction
5.2.1	Field Data Type	=	Unsigned32
5.2.2	Field Length	=	4

Timestamps follow the NTP standard and are represented on the wire as a pair of integers containing the high- and low-order 32 bits.

Time is expressed in seconds using the following formula:

seconds + (fraction / 2^(32))

Publish/subscribe does not require a concept of absolute time.

5.3.3.4 Self-describing types

5.3.3.4.1 ParameterSequence

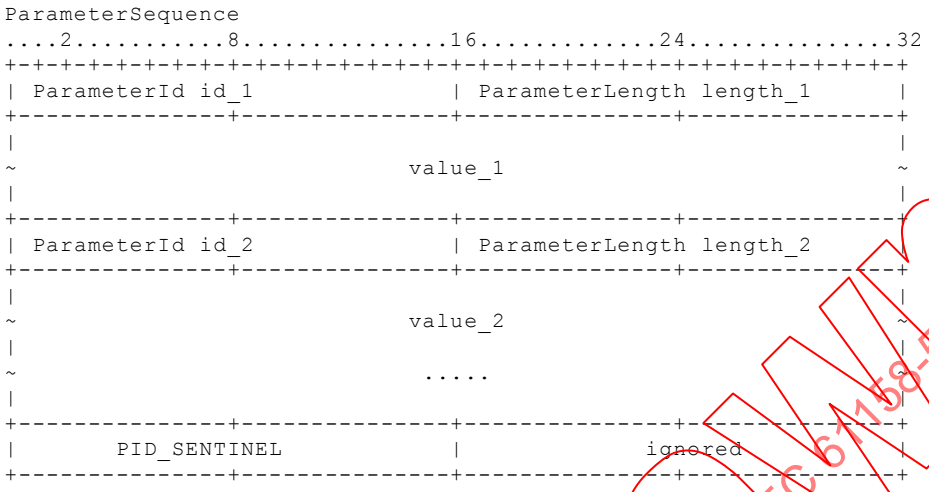
CLASS: **Data Type**

ATTRIBUTES:

1	Data Type Numeric Identifier	=	Not used
2	Data Type Name	=	ParameterSequence
3	Format	=	STRUCTURE
4	Number of Fields	=	4
5.1	Field Name	=	ParameterID | Sentinel
5.1.1	Field Data Type	=	Unsigned16
5.1.2	Field Content	=	<ParameterID content> | PID_SENTINEL
5.2	Field Name	=	ParameterLength | IGNORED
5.2.1	Field Data Type	=	Unsigned16
5.2.2	Field Content	=	n
5.3	Field Name	=	Value | EMPTY
5.3.1	Field Data Type	=	OctetString
5.3.2	Field Length	=	n

5.4 Embedded Data Type = self | EMPTY

This data type represents a variable length sequence containing parameters, the sequence is terminated by the sentinel PID_SENTINEL. The PID_SENTINEL is defined as 0x0001, and its length is ignored. Every ParameterID starts on a 4-octets boundary with respect to the start of the ParameterSequence, so ParameterLength is always a multiple of four.



5.4 Data type ASE service specification

There are no operational services defined for the type object.

6 Client/server communication model specification

6.1 ASEs

6.1.1 General

FAL ASEs are defined using a modular approach. The ASEs defined for the FAL are also object-oriented. In general, ASEs provide a set of services designed for one specific object class or for a related set of classes.

To support remote access to the AP, the Application Relationship ASE is defined. It provides services to the AP for defining and establishing communication relationships with other APs, and it provides services to the other ASEs for conveying their service requests and responses.

Only a subset of the defined FAL ASEs may be provided to meet the needs of an application.

Each FAL ASE defines a set of services, APDUs, and procedures that operate on the classes that it represents.

For a given a FAL ASE, only a subset of the ASE services may be provided to meet the needs of an application.

Profiles may be used to define subsets. Definition of profiles is beyond the scope of this standard.

APDUs are sent and received between FAL ASEs that support the same services.

Figure 20 illustrates the client/server FAL ASEs and their architectural relationships.

There is a FAL ASE for every client/server APO class. The Read Device Identification ASE and the CANopen ASE support APOs that are derived classes of the Encapsulated Interface APO class; these two ASEs are subordinate to the Encapsulated Interface ASE, and Figure 20 should be viewed with this in mind.

An additional FAL ASE, the Context Management ASE, provides a filter service and a transaction service to other ASEs.

The filter service is a provider service, as opposed to a user service: it is a front-end filter on the server. Any AL user interaction goes through it: supported services, unsupported services, and even undefined services, treated as unsupported.

The transaction service is also a provider service, it moderates the invocation of services by the client, beside being used for the management of confirmed service primitives. The Context Management ASE is an abstraction, its configuration and of its implementation are outside the scope of this specification.

A minimal client/server Server – albeit not a useful one – must implement the filter service of the Context Management ASE.

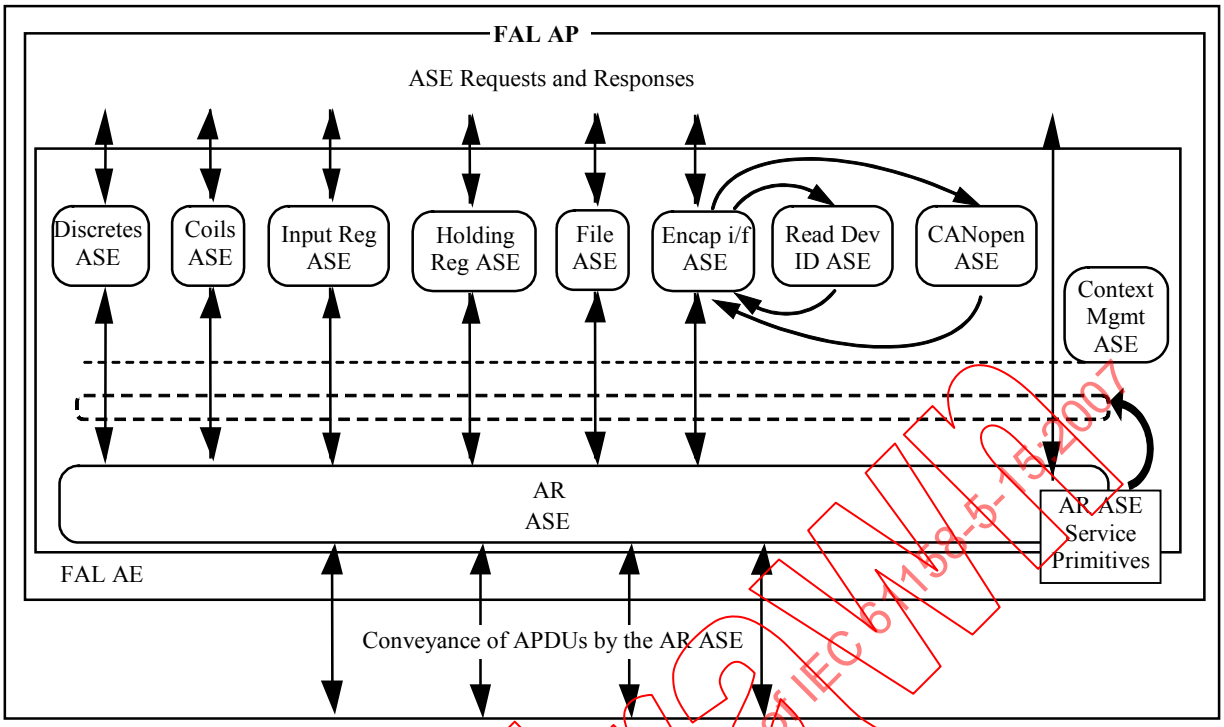


Figure 20 – FAL ASEs

6.1.2 Common parameters

Many services have the following parameters. Instead of defining them with each service, the following common definitions are provided.

Invoke ID
This parameter defines the transaction object within a connection and it is used to pair the request with the corresponding confirmation. It must be unique across all the pending transactions on the given connection. It is conditional since not needed when the client context management only allows one transaction at a time. The Result (+) and Result (-) values are the same as the parameter Invoke ID in the request.

Parameter Type: Unsigned16

Allowed values: any within the type, with the uniqueness constraint as reported above.

6.1.3 Context management ASE

6.1.3.1 Overview

The context management services allow the rejection of unsupported services, provide for the moderation of services invocation, and support the management of confirmed services primitives. A client/server Server must have the filter service of this ASE.

6.1.3.2 Context management class specifications

6.1.3.2.1 Filter class specification

6.1.3.2.1.1 Filter formal model

The Filter object is described by the following template:

ASE:			**Context Management ASE**
CLASS:			**Filter**
CLASS ID:			not used
PARENT CLASS:			TOP
ATTRIBUTES:			
1.	(m)	Key Attribute:	Implicit
2.	(m)	Attribute:	AL user service
SERVICES:			
1.	(m)	OpsService:	Filter

6.1.3.2.1.2 Attributes

Implicit
The attribute Implicit indicates that the Service filter object is implicitly addressed by the services.

AL user service
This attribute specifies the AL user service.

Attribute Type: AL user service.

6.1.3.2.1.3 Services

Filter
This service is used on the Server to discriminate between supported services and unsupported services.

6.1.3.2.2 Transaction class specification

6.1.3.2.2.1 Transaction formal model

The Transaction object is described by the following template:

ASE:			**Context Management ASE**
CLASS:			**Transaction**
CLASS ID:			not used
PARENT CLASS:			TOP
ATTRIBUTES:			
1.	(m)	Key Attribute:	Implicit
2.	(m)	Class Attribute:	Maximum number of pending transactions
2.	(m)	Attribute:	Invoke ID
3.	(m)	Attribute:	Confirmed service request info

6.1.3.2.2.2 Attributes

Implicit
The attribute Implicit indicates that the Service filter object is implicitly addressed by the services.

Maximum number of pending transactions
This class attribute specifies the maximum number of transaction objects that can be instantiated - and still pending - by a client. In practice, available resources may reduce this number. Given a client, while there is no requirement to make this value programmatically available, it must be documented.

Typical values: 1 to 16

Invoke ID
This attribute contains the transaction identifier and it must be unique across all the transaction identifiers still pending on the connection involved. The transaction object is instantiated for the duration of the service invocation, and in general it is used to couple requests and confirmations for confirmed services, and destroyed afterward. It also acts as a moderator. If a client can only instantiate one transaction object at a time then, when invoking a service, it is not necessary to dynamically create a transaction object and exchange it with

lower layers, since the main reason for having a transaction object is to properly couple requests with confirmations, which is automatic for these clients. In these situations a single static transaction object effectively acts like a client token, which is either taken or available, and only of interest to the client.

Attribute Type: Unsigned16

Confirmed service request info
This attribute contains information related to the service request, for matching purposes, including the Unit ID and the service encoding.

Attribute Type: Structure with Unsigned8 for Unit ID and service encoding type as from as described in IEC 61158-6-15.

6.1.3.3 Context management ASE service specification

6.1.3.3.1 Supported services

The Context management ASE defines the services:

Filter

6.1.3.3.2 Filter service

6.1.3.3.2.1 Service overview

The Filter service is used on the Server to discriminate between supported services and unsupported services.

6.1.3.3.2.2 Service primitives

The service parameters for this service are shown in Table 4. It is a confirmed service between ASEs.

Table 4 – Filter service parameters

Parameter name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
AL user service	M	M(=)		
Result (+)			S	S(=)
AL user service			M	M(=)
Result (-)			S	S(=)
Error info			M	M(=)

Argument
The argument shall convey the service specific parameters of the service request.

AL user service
This parameter shall be used to specify the AL user service.

Parameter Type: AL user service.

Result(+)
This selection type parameter indicates that the service request succeeded.

AL user service
In the absence of error, this parameter is identical to the correspondingly named request parameter.

Result(-)
This selection type parameter indicates that the service request failed.

Error info
This parameter contains error information: it flags the requested AL user services as unsuppoted.

Parameter Type: Unsigned8

6.1.3.3.2.3 Service procedure

This service is used on the Server to discriminate between supported services and unsupported services on that particular Server.

6.1.4 Discretes ASE

6.1.4.1 Overview

This ASE enables the interaction and modeling with binary valued real objects that are manipulated by the server application and are supposed to be only observed by the client user. The integrity of the above contract is under control of the server AP, which can confine the exposure of the real object to Discretes.

6.1.4.2 Discretes class specification

6.1.4.2.1 Formal model

The Discretes object is described by the following template:

ASE:			**Discretes ASE**
CLASS:			**Discretes**
CLASS ID:			not used
PARENT CLASS:			TOP
ATTRIBUTES:			
1.	(m)	Key Attribute:	Implicit
2.	(m)	Attribute:	Unit ID
3.	(m)	Attribute:	Address of first discrete
4.	(m)	Attribute:	Quantity of discretes
SERVICES:			
1.	(m)	OpsService:	Read Discretes

6.1.4.2.2 Attributes

Implicit
The attribute Implicit indicates that the Discretes object is implicitly addressed by the services.

Unit ID
This attribute specifies the address of the server.

Attribute Type: Unsigned8

Allowed values: 1 to 247

Address of first discrete
This attribute contains the address of the first discrete input.

Attribute Type: Unsigned16

Allowed values: 0x0000 to 0xFFFF

Quantity of discretes
The attribute contains the quantity of discretes.

Attribute Type: Unsigned16

Allowed values: 1 to 2000

6.1.4.2.3 Services

Read Discretes
This service is used to read a specified number of discrete inputs.

6.1.4.3 Discretes ASE service specification

6.1.4.3.1 Supported services

The Discretes ASE defines the services:

Read Discretes

6.1.4.3.2 Read discretes service

6.1.4.3.2.1 Service overview

The Read Discretes service is used to read a specified number of discrete inputs.

6.1.4.3.2.2 Service primitives

The service parameters for this service are shown in Table 5. It is a confirmed service.

Table 5 – Read discretes service parameters

Parameter name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
Unit ID	M	M(=)		
Invoke ID	C	C(=)		
Address of first discrete	M	M(=)		
Quantity of discretes	M	M(=)		
Result (+)			S	S(=)
Unit ID			M	M(=)
Invoke ID			C	C(=)
Data octets count			M	M(=)
Data			M	M(=)
Result (-)			S	S(=)
Unit ID			M	M(=)
Invoke ID			C	C(=)
Error info			M	M(=)

Argument
The argument conveys the service specific parameters of the service request.

Unit ID
This parameter specifies the address of the server.

Parameter Type: Unsigned8

Allowed values: 1 to 247.

Address of first discrete
This parameter specifies the address of the first discrete input.

Parameter Type: Unsigned16

Allowed values: 0x0000 to 0xFFFF

Quantity of discretes
This parameter specifies the quantity of discretes.

Parameter Type: Unsigned16

Allowed values: 1 to 2000

Result(+)
This selection type parameter indicates that the service request succeeded.

Unit ID
In the absence of error, this parameter is identical to the correspondingly named request parameter.

Data octets count
This parameter specifies the number of octets read.

Parameter Type: Unsigned16

Data
This parameter specifies the status values of the discretes read.

Parameter Type: Status bit sequence

Result(-)
This selection type parameter indicates that the service request failed.

Unit ID
In the absence of error, this parameter is identical to the correspondingly named request parameter.

Error info
This parameter specifies error information.

Parameter Type: Unsigned8

6.1.4.3.2.3 Service procedure

This service is used to read a specified number of discrete inputs.

6.1.5 Coils ASE

6.1.5.1 Overview

This ASE enables the interaction and modeling with binary valued real objects that can be manipulated by the server application and by the client user.

6.1.5.2 Coils class specification

6.1.5.2.1 Formal model

The Coils object is described by the following template:

ASE:			**Coils ASE**
CLASS:			**Coils**
CLASS ID:			not used
PARENT CLASS:			TOP
ATTRIBUTES:			
1.	(m)	Key Attribute:	Implicit
2.	(m)	Attribute:	Unit ID
3.	(m)	Attribute:	Address of first coil
4.	(c)	Constraint:	Write Single Coil | Broadcast Write Single Coil
4.1	(m)	Attribute:	Data single coil
5.	(c)	Constraint:	Read Coils | Write Multiple Coils | Broadcast Write Multiple Coils
5.1	(m)	Attribute:	Quantity of coils
5.2	(m)	Attribute:	Data octets count
5.3	(m)	Attribute:	Data
SERVICES:			
1.	(o)	OpsService:	Read Coils
2.	(o)	OpsService:	Write Single Coil
3.	(o)	OpsService:	Write Multiple Coils
4.	(c)	Constraint:	Broadcast supported
4.1	(o)	OpsService:	Broadcast Write Single Coil
4.2	(o)	OpsService:	Broadcast Write Multiple Coils

6.1.5.2.2 Attributes

Implicit
The attribute Implicit indicates that the Coils object is implicitly addressed by the services.

Unit ID
This attribute specifies the address of the server.

Attribute Type: Unsigned8

Allowed values: 1 to 247, and 0 for Broadcast when applicable.

Address of first coil
This attribute specifies the address of the first coil.

Attribute Type: Unsigned16

Allowed values: 0x0000 to 0xFFFF

Data single coil
The attribute specifies the single coil status value that has to be written.

Attribute Type: Status flag

Quantity of coils
The attribute specifies the quantity of coils.

Attribute Type: Unsigned16

Allowed values: 1 to 2000 when reading, 1 to 1968 when writing.

Data octets count
The attribute specifies the quantity of data octets.

Attribute Type: Unsigned8

Allowed values: 1 to 250 when writing, 1 to 246 when reading.

Data
The attribute specifies the coil status values that were read or to be written.

Attribute Type: Status bit sequence

6.1.5.2.3 Services

Read coils
This optional service is used to read a specified number of coils.

Write single coil
This optional service is used to write a single coil.

Write multiple coils
This optional service is used to write a specified number of coils.

Broadcast write single coil
This optional service is also conditional to the Broadcast mechanism being supported by the client, and by the servers. This service is used to write a single coil in all the Unit ID addressable servers, it is an unconfirmed service.

Broadcast write multiple coils
This optional service is also conditional to the Broadcast mechanism being supported by the client, and by the servers. This service is used to write a specified number of coils in all the Unit ID addressable servers, it is an unconfirmed service.

6.1.5.3 Coils ASE service specification

6.1.5.3.1 Supported services

This subclause specifies the definition of services that are unique to this ASE. The services defined for this ASE are:

Read Coils

Write Single Coil

Write Multiple Coils

Broadcast Write Single Coil

Broadcast Write Multiple Coils

6.1.5.3.2 Read coils service

6.1.5.3.2.1 Service overview

The Read Coils service is used to read a specified number of coils.

6.1.5.3.2.2 Service primitives

The service parameters for this service are shown in Table 6. It is a confirmed service.

Table 6 – Read coils service parameters

Parameter name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
Unit ID	M	M(=)		
Invoke ID	C	C(=)		
Address of first coil	M	M(=)		
Quantity of coils	M	M(=)		
Result (+)			S	S(=)
Unit ID			M	M(=)
Invoke ID			C	C(=)
Data octets count			M	M(=)
Data			M	M(=)
Result (-)			S	S(=)
Unit ID			M	M(=)
Invoke ID			C	C(=)
Error info			M	M(=)

Argument
The argument conveys the service specific parameters of the service request.

Unit ID
This parameter specifies the address of the server.

Parameter Type: Unsigned8

Allowed values: 1 to 247.

Address of first coil
This parameter specifies the address of the first coil.

Parameter Type: Unsigned16

Allowed values: 0x0000 to 0xFFFF

Quantity of coils
This parameter specifies the quantity of coils.

Parameter Type: Unsigned16

Allowed values: 1 to 2000

Result(+)
This selection type parameter indicates that the service request succeeded.

Unit ID
In the absence of error, this parameter is identical to the correspondingly named request parameter.

Data octets count
This parameter specifies the number of octets read.

Parameter Type: Unsigned16

Data
This parameter specifies the status values of the coils read.

Parameter Type: Status bit sequence

Result(-)
This selection type parameter indicates that the service request failed.

Unit ID
In the absence of error, this parameter is identical to the correspondingly named request parameter.

Error info
This parameter specifies error information.

Parameter Type: Unsigned8

6.1.5.3.2.3 Service procedure

This service is used to read a specified number of coils.

6.1.5.3.3 Write single coil service

6.1.5.3.3.1 Service overview

The Write Single Coil service is used to write a single coil.

6.1.5.3.3.2 Service primitives

The service parameters for this service are shown in Table 7. It is a confirmed service.

Table 7 – Write single coil service parameters

Parameter name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
Unit ID	M	M(=)		
Invoke ID	C	C(=)		
Address of first coil	M	M(=)		
Data single coil	M	M(=)		
Result (+)			S	S(=)
Unit ID			M	M(=)
Invoke ID			C	C(=)
Address of first coil			M	M(=)
Data single coil			M	M(=)
Result (-)			S	S(=)
Unit ID			M	M(=)
Invoke ID			C	C(=)
Error info			M	M(=)

Argument
The argument conveys the service specific parameters of the service request.

Unit ID
This parameter specifies the address of the server.

Parameter Type: Unsigned8

Allowed values: 1 to 247.

Address of first coil
This parameter specifies the address of the first coil (the only one in this service).

Parameter Type: Unsigned16

Allowed values: 0x0000 to 0xFFFF

Data single coil
This parameter specifies the single coil status value that has to be written.

Parameter Type: Status flag

Result(+)
This selection type parameter indicates that the service request succeeded.

Unit ID
In the absence of error, this parameter is identical to the correspondingly named request parameter.

Address of first coil
In the absence of error, this parameter is identical to the correspondingly named request parameter.

Data single coil
In the absence of error, this parameter is identical to the correspondingly named request parameter.

Result(-)
This selection type parameter indicates that the service request failed.

Unit ID
In the absence of error, this parameter is identical to the correspondingly named request parameter.

Error info
This parameter specifies error information.

Parameter Type: Unsigned8

6.1.5.3.3 Service procedure

This service is used to write a single coil.

6.1.5.3.4 Write multiple coils service

6.1.5.3.4.1 Service overview

The Write Multiple Coils service is used to write a specified number of coils.

6.1.5.3.4.2 Service primitives

The service parameters for this service are shown in Table 8. It is a confirmed service.

Table 8 – Write multiple coils service parameters

Parameter name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
Unit ID	M	M(=)		
Invoke ID	C	C(=)		
Address of first coil	M	M(=)		
Quantity of coils	M	M(=)		
Data octets count	M	M(=)		
Data	M	M(=)		
Result (+)			S	S(=)
Unit ID			M	M(=)
Invoke ID			C	C(=)
Address of first coil			M	M(=)
Quantity of coils			M	M(=)
Result (-)			S	S(=)
Unit ID			M	M(=)
Invoke ID			C	C(=)
Error info			M	M(=)

Argument
The argument conveys the service specific parameters of the service request.

Unit ID
This parameter specifies the address of the server.

Parameter Type: Unsigned8

Allowed values: 1 to 247

Address of first coil
This parameter specifies the address of the first coil.

Parameter Type: Unsigned16

Allowed values: 0x0000 to 0xFFFF

Quantity of coils
This parameter specifies the quantity of coils.

Parameter Type: Unsigned16

Allowed values: 1 to 1968, must be consistent with the Data octets count and the Data parameters.

Data octets count
This parameter specifies the number of octets carrying the coil status values to be written.

Parameter Type: Unsigned16

Allowed values: 1 to 246, must be consistent with the Quantity of coils and the Data parameters.

Data
This parameter specifies the coil status values that have to be written.

Parameter Type: Status bit sequence

Allowed values:Must be consistent with the Quantity of coils and the Data octets count parameters.

Result(+)
This selection type parameter indicates that the service request succeeded.

Unit ID
In the absence of error, this parameter is identical to the correspondingly named request parameter.

Address of first coil
In the absence of error, this parameter is identical to the correspondingly named request parameter.

Quantity of coils
In the absence of error, this parameter is identical to the correspondingly named request parameter.

Result(-)
This selection type parameter indicates that the service request failed.

Unit ID
In the absence of error, this parameter is identical to the correspondingly named request parameter.

Error info
This parameter specifies error information.

Parameter Type: Unsigned8

6.1.5.3.4.3 Service procedure

This service is used to write a specified number of coils.

6.1.5.3.5 Broadcast write single coil service

6.1.5.3.5.1 Service overview

The Broadcast Write Single Coil service is used to write a single coil in all the Unit ID addressable servers.

6.1.5.3.5.2 Service primitives

The service parameters for this service are shown in Table 9. It is an unconfirmed service.

Table 9 – Broadcast write single coil service parameters

Parameter name	Req	Ind
Argument	M	M(=)
Unit ID	M	M(=)
Address of first coil	M	M(=)
Data single coil	M	M(=)

Argument
The argument conveys the service specific parameters of the service request.

Unit ID
This parameter shall be used to flag the service as a Broadcast service.

Parameter Type: Unsigned8

Allowed values: Must be specified as 0.

Address of first coil
This parameter specifies the address of the first coil (the only one in this service).

Parameter Type: Unsigned16

Allowed values: 0x0000 to 0xFFFF

Data single coil
This parameter specifies the single coil status value that has to be written.

Parameter Type: Status flag

6.1.5.3.5.3 Service procedure

This service is used to write a single coil in all the Unit ID addressable servers.

6.1.5.3.6 Broadcast write multiple coils service

6.1.5.3.6.1 Service overview

The Broadcast Write Multiple Coils service is used to write a specified number of coils in all the Unit ID addressable servers.

6.1.5.3.6.2 Service primitives

The service parameters for this service are shown in Table 10. It is an unconfirmed service.

Table 10 – Broadcast write multiple coils service parameters

Parameter name	Req	Ind
Argument	M	M(=)
Unit ID	M	M(=)
Address of first coil	M	M(=)
Quantity of coils	M	M(=)
Data octets count	M	M(=)
Data	M	M(=)

Argument
The argument conveys the service specific parameters of the service request.

Unit ID
This parameter is used to flag the service as a Broadcast service.

Parameter Type: Unsigned8

Allowed values: Must be specified as 0.

Address of first coil
This parameter specifies the address of the first coil.

Parameter Type: Unsigned16

Allowed values: 0x0000 to 0xFFFF

Quantity of coils
This parameter specifies the quantity of coils.

Parameter Type: Unsigned16

Allowed values: 1 to 1968, must be consistent with the Data octets count and the Data parameters.

Data octets count
This parameter specifies the number of octets carrying the coil status values to be written.

Parameter Type: Unsigned16

Allowed values: 1 to 246, must be consistent with the Quantity of coils and the Data parameters.

Data
This parameter specifies the coil status values that have to be written.

Parameter Type: Status bit sequence

Allowed values:Must be consistent with the Quantity of coils and the Data octets count parameters.

6.1.5.3.6.3 Service procedure

This service is used to write a specified number of coils in all the Unit ID addressable servers.

6.1.6 Input Registers ASE

6.1.6.1 Overview

This ASE enables the interaction and modeling with analog valued real objects that are manipulated by the server application and are supposed to be only observed by the client user. The integrity of the above contract is under control of the server AP, which can confine the exposure of the real object to Input Registers.

6.1.6.2 Input registers class specification

6.1.6.2.1 Formal model

The Input Registers object is described by the following template:

ASE:			**Input Registers ASE**
CLASS:			**Input Registers**
CLASS ID:			not used
PARENT CLASS:			TOP
ATTRIBUTES:			
1.	(m)	Key Attribute:	Implicit
2.	(m)	Attribute:	Unit ID
3.	(m)	Attribute:	Address of first input register
4.	(m)	Attribute:	Quantity of input registers
SERVICES:			
1.	(m)	OpsService:	Read Input Registers

6.1.6.2.2 Attributes

Implicit
The attribute Implicit indicates that the Input Registers object is implicitly addressed by the services.

Unit ID
This attribute specifies the address of the server.

Attribute Type: Unsigned8

Allowed values: 1 to 247

Address of first input register
This attribute specifies the address of the first input register.

Attribute Type: Unsigned16

Allowed values: 0x0000 to 0xFFFF

Quantity of input registers
The attribute specifies the quantity of input registers.

Attribute Type: Unsigned16

Allowed values: 1 to 125

6.1.6.2.3 Services

Read Input Registers
This service is used to read a specified number of input registers.

6.1.6.3 Input Registers ASE service specification

6.1.6.3.1 Supported services

The Input Registers ASE defines the services:

Read Input Registers

6.1.6.3.2 Read input registers service

6.1.6.3.2.1 Service overview

The Read Input Registers service is used to read a specified number of input registers.

6.1.6.3.2.2 Service primitives

The service parameters for this service are shown in Table 11. It is a confirmed service.

Table 11 – Read input registers service parameters

Parameter name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
Unit ID	M	M(=)		
Invoke ID	C	C(=)		
Address of first input register	M	M(=)		
Quantity of input registers	M	M(=)		
Result (+)			S	S(=)
Unit ID			M	M(=)
Invoke ID			C	C(=)
Data octets count			M	M(=)
Data			M	M(=)
Result (-)			S	S(=)
Unit ID			M	M(=)
Invoke ID			C	C(=)
Error info			M	M(=)

Argument
The argument conveys the service specific parameters of the service request.

Unit ID
This parameter specifies the address of the server.

Parameter Type: Unsigned8

Allowed values: 1 to 247.

Address of first input register
This parameter specifies the address of the first input register.

Parameter Type: Unsigned16

Allowed values: 0x0000 to 0xFFFF

Quantity of input registers
This parameter specifies the quantity of input registers.

Parameter Type: Unsigned16

Allowed values: 1 to 125

Result(+)
This selection type parameter indicates that the service request succeeded.

Unit ID
In the absence of error, this parameter is identical to the correspondingly named request parameter.

Data octets count
This parameter specifies the number of octets read.

Parameter Type: Unsigned16

Data
This parameter specifies the analog values of the input registers.

Parameter Type: Array of Unsigned16, transferred using the big-endian encoding.

Result(-)
This selection type parameter indicates that the service request failed.

Unit ID
In the absence of error, this parameter is identical to the correspondingly named request parameter.

Error info
This parameter specifies error information.

Parameter Type: Unsigned8

6.1.6.3.2.3 Service procedure

This service is used to read a specified number of input registers.

6.1.7 Holding registers ASE

6.1.7.1 Overview

This ASE enables the interaction and modeling with analog valued real objects that can be manipulated by the server application and by the client user.

6.1.7.2 Holding registers class specification

6.1.7.2.1 Formal model

The Holding Registers object is described by the following template:

ASE:			**Holding Registers ASE**
CLASS:			**Holding Registers**
CLASS ID:			not used
PARENT CLASS:			TOP
ATTRIBUTES:			
1.	(m)	Key Attribute:	Implicit
2.	(m)	Attribute:	Unit ID
3.	(c)	Constraint:	Read Holding Registers
3.1	(m)	Attribute:	Address of first holding register to read
3.2	(m)	Attribute:	Quantity of holding registers to read
4.	(c)	Constraint:	Write Single Holding Register | Write Multiple Holding Registers | Broadcast Write Single Holding Register | Broadcast Write Multiple Holding Registers
4.1	(m)	Attribute:	Address of first holding register to write
4.2	(m)	Attribute:	Quantity of holding registers to write
5.	(c)	Constraint:	Read/Write Holding Registers
5.1	(m)	Attribute:	Address of first holding register to read
5.2	(m)	Attribute:	Quantity of holding registers to read
5.3	(m)	Attribute:	Address of first holding register to write
5.4	(m)	Attribute:	Quantity of holding registers to write
6.	(c)	Constraint:	Mask Write Holding Register
6.1.	(m)	Attribute:	Address of holding register to change
6.2.	(m)	Attribute:	AND Mask
6.3.	(m)	Attribute:	OR Mask
7.	(c)	Constraint:	Read FIFO
7.1	(m)	Attribute:	Address of FIFO queue
7.2	(m)	Attribute:	FIFO queue count
8.	(m)	Attribute:	Data octets count
9.	(m)	Attribute:	Data
SERVICES:			
1.	(o)	OpsService:	Read Holding Registers
2.	(o)	OpsService:	Write Single Holding Register
3.	(o)	OpsService:	Write Multiple Holding Registers
3.	(o)	OpsService:	Mask Write Holding Register
3.	(o)	OpsService:	Read/Write Holding Registers
3.	(o)	OpsService:	Read FIFO
4.	(c)	Constraint:	Broadcast supported
4.1	(o)	OpsService:	Broadcast Write Single Holding Register
4.2	(o)	OpsService:	Broadcast Write Multiple Holding Registers

6.1.7.2.2 Attributes

Implicit
The attribute Implicit indicates that the Holding Registers object is implicitly addressed by the services.

Unit ID
This attribute specifies the address of the server.

Attribute Type: Unsigned8

Allowed values: 1 to 247, and 0 for Broadcast when applicable.

Address of first holding register to read
This attribute specifies the address of the first holding register to read.

Attribute Type: Unsigned16

Allowed values: 0x0000 to 0xFFFF

Quantity of holding registers to read
The attribute specifies the quantity of holding registers to read.

Attribute Type: Unsigned16

Allowed values: 1 to 125

Address of first holding register to write
This attribute specifies the address of the first holding register to write.

Attribute Type: Unsigned16

Allowed values: 0x0000 to 0xFFFF

Quantity of holding registers to write
The attribute specifies the quantity of holding registers to write.

Attribute Type: Unsigned16

Allowed values: 1 to 123 with Write Multiple Holding Registers, 1 to 121 with Read/Write Holding Registers.

Address of first holding register to change
This attribute specifies the address of the holding register to change in-place via masks.

Attribute Type: Unsigned16

Allowed values: 0x0000 to 0xFFFF

AND Mask
The attribute specifies the binary mask AND_Mask that contributes to the new content of a register according to the equation (1):

$$new\_content = (old\_content \wedge AND\_Mask) \vee (OR\_Mask \wedge \neg AND\_Mask) \quad (1)$$

Attribute Type: Unsigned16

Allowed values: 0x0000 to 0xFFFF

OR Mask
The attribute specifies the binary mask OR_Mask that contributes to the new content of a register according to equation (1).

Attribute Type: Unsigned16

Allowed values: 0x0000 to 0xFFFF

Address of FIFO queue
This attribute specifies the address of the holding register that contains the count of the FIFO queue data holding registers to follow.

Attribute Type: Unsigned16

Allowed values: 0x0000 to 0xFFFF

FIFO queue count
The attribute specifies the quantity of FIFO queue data holding registers.

Attribute Type: Unsigned16

Allowed values: 1 to 31

Data octets count
The attribute specifies the quantity of data octets.

Attribute Type: Unsigned8

Allowed values: 1 to 250 for Read Holding Registers and as read octets count of Read/Write Holding Registers, 1 to 246 for Write Multiple Holding Registers and Broadcast Write Multiple Holding Registers, 1 to 242 as write octets count of Read/Write Holding Registers, and 1 to 64 for Read FIFO

Data
The attribute specifies the registers values that have been read or to be written.

Attribute Type: Array of Unsigned16, transferred using the big-endian encoding.

6.1.7.2.3 Services

Read holding registers
This optional service is used to read a specified number of holding registers.

Write single holding register
This optional service is used to write a single holding register.

Write multiple holding registers
This optional service is used to write a specified number of holding registers.

Mask write holding register
This optional service is used to manipulate the content of a holding register using two binary masks, the AND Mask and the OR Mask, according to equation (1).

Read/write holding registers
This optional service is used to read and write specified numbers of holding registers. The write operation is performed before the read.

Read FIFO
This optional service is used to read a bounded but otherwise unspecified number of holding registers, organized to facilitate a FIFO policy.

Broadcast write single holding register
This optional service is also conditional to the Broadcast mechanism being supported by the client, and by the servers. This service is used to write a single holding register in all the Unit ID addressable servers, it is an unconfirmed service.

Broadcast write multiple holding services
This optional service is also conditional to the Broadcast mechanism being supported by the client, and by the servers. This service is used to write a specified number of holding registers in all the Unit ID addressable servers, it is an unconfirmed service.

6.1.7.3 Holding registers ASE service specification

6.1.7.3.1 Supported services

This subclause contains the definition of services that are unique to this ASE. The services defined for this ASE are:

Read Holding Registers

Write Single Holding Register

Write Multiple Holding Registers

Mask Write Holding Register

Read/Write Holding Registers

Read FIFO

Broadcast Write Single Holding Register

Broadcast Write Multiple Holding Registers

6.1.7.3.2 Read holding registers service

6.1.7.3.2.1 Service overview

The Read Holding Registers service is used to read a specified number of holding registers.

6.1.7.3.2.2 Service primitives

The service parameters for this service are shown in Table 12. It is a confirmed service.

Table 12 – Read holding registers service parameters

Parameter name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
Unit ID	M	M(=)		
Invoke ID	C	C(=)		
Address of first holding register to read	M	M(=)		
Quantity of holding registers to read	M	M(=)		
Result (+)			S	S(=)
Unit ID			M	M(=)
Invoke ID			C	C(=)
Data octets count			M	M(=)
Data			M	M(=)
Result (-)			S	S(=)
Unit ID			M	M(=)
Invoke ID			C	C(=)
Error info			M	M(=)

Argument
The argument conveys the service specific parameters of the service request.

Unit ID
This parameter specifies the address of the server.

Parameter Type: Unsigned8

Allowed values: 1 to 247.

Address of first holding register to read
This parameter specifies the address of the first holding register to read.

Parameter Type: Unsigned16

Allowed values: 0x0000 to 0xFFFF

Quantity of holding registers to read
This parameter specifies the quantity of holding registers to read.

Parameter Type: Unsigned16

Allowed values: 1 to 125

Result(+)
This selection type parameter indicates that the service request succeeded.

Unit ID
In the absence of error, this parameter is identical to the correspondingly named request parameter.

Data octets count
This parameter specifies the number of octets read.

Parameter Type: Unsigned16

Data
This parameter specifies the analog values of the holding registers read.

Parameter Type: Array of Unsigned16, transferred using the big-endian encoding.

Result(-)
This selection type parameter indicates that the service request failed.

Unit ID
In the absence of error, this parameter is identical to the correspondingly named request parameter.

Error info
This parameter specifies error information.

Parameter Type: Unsigned8

6.1.7.3.2.3 Service procedure

This service is used to read a specified number of holding registers.

6.1.7.3.3 Write single holding register service

6.1.7.3.3.1 Service overview

The Write Single Holding Register service is used to write a single holding register.

6.1.7.3.3.2 Service primitives

The service parameters for this service are shown in Table 13. It is a confirmed service.

Table 13 – Write single holding register service parameters

Parameter name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
Unit ID	M	M(=)		
Invoke ID	C	C(=)		
Address of first holding register to write	M	M(=)		
Data	M	M(=)		
Result (+)			S	S(=)
Unit ID			M	M(=)
Invoke ID			C	C(=)
Address of first holding register to write			M	M(=)
Data			M	M(=)
Result (-)			S	S(=)
Unit ID			M	M(=)
Invoke ID			C	C(=)
Error info			M	M(=)

Argument
The argument conveys the service specific parameters of the service request.

Unit ID
This parameter specifies the address of the server.

Parameter Type: Unsigned8

Allowed values: 1 to 247.

Address of first holding register to write
This parameter specifies the address of the first holding register to write (the only one in this service).

Parameter Type: Unsigned16

Allowed values: 0x0000 to 0xFFFF

Data
This parameter specifies the holding register analog value that has to be written.

Parameter Type: Unsigned16

Result(+)
This selection type parameter indicates that the service request succeeded.

Unit ID
In the absence of error, this parameter is identical to the correspondingly named request parameter.

Address of first holding register to write
In the absence of error, this parameter is identical to the correspondingly named request parameter.

Data
In the absence of error, this parameter is identical to the correspondingly named request parameter.

Result(-)
This selection type parameter indicates that the service request failed.

Unit ID
In the absence of error, this parameter is identical to the correspondingly named request parameter.

Error info
This parameter specifies error information.

Parameter Type: Unsigned8

6.1.7.3.3.3 Service procedure

This service is used to write a single holding register.

6.1.7.3.4 Write multiple holding registers service

6.1.7.3.4.1 Service overview

The Write Multiple Holding Registers service is used to write a specified number of holding registers.

6.1.7.3.4.2 Service primitives

The service parameters for this service are shown in Table 14. It is a confirmed service.

Table 14 – Write multiple holding registers service parameters

Parameter name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
Unit ID	M	M(=)		
Invoke ID	C	C(=)		
Address of first holding register to write	M	M(=)		
Quantity of holding registers to write	M	M(=)		
Data octets count	M	M(=)		
Data	M	M(=)		
Result (+)			S	S(=)
Unit ID			M	M(=)
Invoke ID			C	C(=)
Address of first holding register to write			M	M(=)
Quantity of holding registers to write			M	M(=)
Result (-)			S	S(=)
Unit ID			M	M(=)
Invoke ID			C	C(=)
Error info			M	M(=)

Argument
The argument conveys the service specific parameters of the service request.

Unit ID
This parameter specifies the address of the server.

Parameter Type: Unsigned8

Allowed values: 1 to 247.

Address of first holding registers to write
This parameter specifies the address of the first holding registers to write.

Parameter Type: Unsigned16

Allowed values: 0x0000 to 0xFFFF

Quantity of holding registers to write
This parameter specifies the quantity of holding registers to write.

Parameter Type: Unsigned16

Allowed values: 1 to 123, must be consistent with the Data octets count and the Data parameters.

Data octets count
This parameter specifies the number of octets carrying the holding registers analog values to be written.

Parameter Type: Unsigned16

Allowed values: 1 to 246, must be consistent with the Quantity of holding registers to write and the Data parameters.

Data
This parameter specifies the holding registers analog values that have to be written.

Parameter Type: Array of Unsigned16, transferred using the big-endian encoding.

Allowed values:The number of registers must be consistent with the Quantity of holding registers to write and the Data octets count parameters.

Result(+)
This selection type parameter indicates that the service request succeeded.

Unit ID
In the absence of error, this parameter is identical to the correspondingly named request parameter.

Address of first holding register to write
In the absence of error, this parameter is identical to the correspondingly named request parameter.

Quantity of holding registers to write
In the absence of error, this parameter is identical to the correspondingly named request parameter.

Result(-)
This selection type parameter indicates that the service request failed.

Unit ID
In the absence of error, this parameter is identical to the correspondingly named request parameter.

Error info
This parameter specifies error information.

Parameter Type: Unsigned8

6.1.7.3.4.3 Service procedure

This service is used to write a specified number of holding registers.

6.1.7.3.5 Mask write holding register service

6.1.7.3.5.1 Service overview

The Mask Write Holding Register service is used to manipulate the content of a holding register using two binary masks, the AND Mask and the OR Mask, according to equation (1).

6.1.7.3.5.2 Service primitives

The service parameters for this service are shown in Table 15. It is a confirmed service.

Table 15 – Mask write holding register service parameters

Parameter name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
Unit ID	M	M(=)		
Invoke ID	C	C(=)		
Address of first holding register to write	M	M(=)		
AND Mask	M	M(=)		
OR Mask	M	M(=)		
Result (+)			S	S(=)
Unit ID			M	M(=)
Invoke ID			C	C(=)
Address of first holding register to write			M	M(=)
AND Mask			M	M(=)
OR Mask			M	M(=)
Result (-)			S	S(=)
Unit ID			M	M(=)
Invoke ID			C	C(=)
Error info			M	M(=)

Argument
The argument conveys the service specific parameters of the service request.

Unit ID
This parameter specifies the address of the server.

Parameter Type: Unsigned8

Allowed values: 1 to 247.

Address of holding register to change
This parameter specifies the address of the holding register to change using two binary masks, the AND Mask and the OR Mask, according to equation (1).

Parameter Type: Unsigned16

Allowed values: 0x0000 to 0xFFFF

AND Mask
This parameter specifies the binary mask AND_Mask that contributes to the new content of the holding register to change according to equation (1).

Parameter Type: Unsigned16

Allowed values: 0x0000 to 0xFFFF

OR Mask
This parameter specifies the binary mask OR_Mask that contributes to the new content of the holding register to change according to equation (1).

Parameter Type: Unsigned16

Allowed values: 0x0000 to 0xFFFF

Result(+)
This selection type parameter indicates that the service request succeeded.

Unit ID
In the absence of error, this parameter is identical to the correspondingly named request parameter.

Address of holding register to change
In the absence of error, this parameter is identical to the correspondingly named request parameter.

AND Mask
In the absence of error, this parameter is identical to the correspondingly named request parameter.

OR Mask
In the absence of error, this parameter is identical to the correspondingly named request parameter.

Result(-)
This selection type parameter indicates that the service request failed.

Unit ID
In the absence of error, this parameter is identical to the correspondingly named request parameter.

Error info
This parameter specify error information.

Parameter Type: Unsigned8

6.1.7.3.5.3 Service procedure

This service is used to manipulate the content of a holding register using two binary masks, the AND Mask and the OR Mask, according to equation (1).

6.1.7.3.6 Read/write holding registers service

6.1.7.3.6.1 Service overview

The Read/Write Holding Registers service is used to read and write specified numbers of holding registers.

6.1.7.3.6.2 Service primitives

The service parameters for this service are shown in Table 16. It is a confirmed service.

Table 16 – Read/write holding registers service parameters

Parameter name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
Unit ID	M	M(=)		
Invoke ID	C	C(=)		
Address of first holding register to read	M	M(=)		
Quantity of holding registers to read	M	M(=)		
Address of first holding register to write	M	M(=)		
Quantity of holding registers to write	M	M(=)		
Data octets count	M	M(=)		
Data	M	M(=)		
Result (+)			S	S(=)
Unit ID			M	M(=)
Invoke ID			C	C(=)
Data octets count			M	M(=)
Data			M	M(=)
Result (-)			S	S(=)
Unit ID			M	M(=)
Invoke ID			C	C(=)
Error info			M	M(=)

Argument
The argument conveys the service specific parameters of the service request.

Unit ID
This parameter specifies the address of the server.

Parameter Type: Unsigned8

Allowed values: 1 to 247.

Address of first holding registers to read
This parameter specifies the address of the first holding registers to read.

Parameter Type: Unsigned16

Allowed values: 0x0000 to 0xFFFF

Quantity of holding registers to read
This parameter specifies the quantity of holding registers to read.

Parameter Type: Unsigned16

Allowed values: 1 to 125

Address of first holding registers to write
This parameter specifies the address of the first holding registers to write.

Parameter Type: Unsigned16

Allowed values: 0x0000 to 0xFFFF

Quantity of holding registers to write
This parameter specifies the quantity of holding registers to write.

Parameter Type: Unsigned16

Allowed values: 1 to 121, must be consistent with the Data octets count and the Data parameters.

Data octets count
This parameter specifies the number of octets carrying the holding registers analog values to be written.

Parameter Type: Unsigned16

Allowed values: 1 to 242, must be consistent with the Quantity of holding registers to write and the Data parameters.

Data
This parameter specifies the holding registers analog values that have to be written.

Parameter Type: Array of Unsigned16, transferred using the big-endian encoding.

Allowed values:The number of registers must be consistent with the Quantity of holding registers to write and the Data octets count parameters.

Result(+)
This selection type parameter indicates that the service request succeeded.

Unit ID
In the absence of error, this parameter is identical to the correspondingly named request parameter.

Data octets count
This parameter specifies the number of octets read.

Parameter Type: Unsigned16

Data
This parameter specifies the analog values of the holding registers read.

Parameter Type: Array of Unsigned16, transferred using the big-endian encoding.

Result(-)
This selection type parameter indicates that the service request failed.

Unit ID
In the absence of error, this parameter is identical to the correspondingly named request parameter.

Error info
This parameter specifies error information.

Parameter Type: Unsigned8

6.1.7.3.6.3 Service procedure

This service is used to read and to write specified numbers of holding registers. The write operation is performed before the read.

6.1.7.3.7 Read FIFO service

6.1.7.3.7.1 Service overview

The Read FIFO service is used to read a bounded number of holding registers, organized to facilitate a FIFO policy. The bounded number is a-priori unknown, and it is part of the response. The bound is 32 registers: the register containing the above number plus up to 31 FIFO queue data holding registers to follow.

6.1.7.3.7.2 Service primitives

The service parameters for this service are shown in Table 17. It is a confirmed service.

Table 17 – Read FIFO service parameters

Parameter name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
Unit ID	M	M(=)		
Invoke ID	C	C(=)		
Address of FIFO queue	M	M(=)		
Result (+)			S	S(=)
Unit ID			M	M(=)
Invoke ID			C	C(=)
Data octets count			M	M(=)
FIFO queue count			M	M(=)
Data			M	M(=)
Result (-)			S	S(=)
Unit ID			M	M(=)
Invoke ID			C	C(=)
Error info			M	M(=)

Argument
The argument conveys the service specific parameters of the service request.

Unit ID
This parameter specifies the address of the server.

Parameter Type: Unsigned8

Allowed values: 1 to 247.

Address of FIFO queue
This parameter specifies the address of the holding register that contains the count of the FIFO queue data holding registers to follow.

Parameter Type: Unsigned16

Allowed values: 0x0000 to 0xFFFF

Result(+)
This selection type parameter indicates that the service request succeeded.

Unit ID
In the absence of error, this parameter is identical to the correspondingly named request parameter.

Data octets count
This parameter specifies the number of octets read, including the octets of the FIFO queue count.

Parameter Type: Unsigned16

FIFO queue count
This parameter specifies the number of data holding registers of the FIFO queue, read in the Data parameter. The FIFO queue count holding register itself is not included.

Parameter Type: Unsigned16

Data
This parameter specifies the analog values of the holding registers read.

Parameter Type: Array of Unsigned16, transferred using the big-endian encoding.

Result(-)
This selection type parameter indicates that the service request failed.

Unit ID
In the absence of error, this parameter is identical to the correspondingly named request parameter.

Error info
This parameter specifies error information.

Parameter Type: Unsigned8

6.1.7.3.7.3 Service procedure

The Read FIFO service is used to read a bounded number of holding registers, organized to facilitate a FIFO policy. The bounded number is a-priori unknown, and it is part of the response. The bound is 32 registers: the register containing the above number plus up to 31 FIFO queue data holding registers to follow.

6.1.7.3.8 Broadcast write single holding register service

6.1.7.3.8.1 Service overview

The Broadcast Write Single Holding Register service is used to write a single holding register in all the Unit ID addressable servers.

6.1.7.3.8.2 Service primitives

The service parameters for this service are shown in Table 18. It is an unconfirmed service.

Table 18 – Broadcast write single holding register service parameters

Parameter name	Req	Ind
Argument	M	M(=)
Unit ID	M	M(=)
Address of first holding register to write	M	M(=)
Data	M	M(=)

Argument
The argument conveys the service specific parameters of the service request.

Unit ID
This parameter is used to flag the service as a Broadcast service.

Parameter Type: Unsigned8

Allowed values: Must be specified as 0.

Address of first holding register to write
This parameter specifies the address of the first holding register to write (the only one in this service).

Parameter Type: Unsigned16

Allowed values: 0x0000 to 0xFFFF

Data
This parameter specifies the holding register analog value that has to be written.

Parameter Type: Unsigned16

6.1.7.3.8.3 Service procedure

This service is used to write a single holding register in all the Unit ID addressable servers.

6.1.7.3.9 Broadcast write multiple holding registers service

6.1.7.3.9.1 Service overview

The Broadcast Write Multiple Holding Registers service is used to write a specified number of holding registers in all the Unit ID addressable servers.

6.1.7.3.9.2 Service primitives

The service parameters for this service are shown in Table 19. It is an unconfirmed service.

Table 19 – Broadcast write multiple holding registers service parameters

Parameter name	Req	Ind
Argument	M	M(=)
Unit ID	M	M(=)
Address of first holding register to write	M	M(=)
Quantity of holding registers to write	M	M(=)
Data octets count	M	M(=)
Data	M	M(=)

Argument
The argument conveys the service specific parameters of the service request.

Unit ID
This parameter is used to flag the service as a Broadcast service.

Parameter Type: Unsigned8

Allowed values: Must be specified as 0.

Address of first holding registers to write
This parameter specifies the address of the first holding registers to write.

Parameter Type: Unsigned16

Allowed values: 0x0000 to 0xFFFF

Quantity of holding registers to write
This parameter specifies the quantity of holding registers to write.

Parameter Type: Unsigned16

Allowed values: 1 to 123, must be consistent with the Data octets count and the Data parameters.

Data octets count
This parameter specifies the number of octets carrying the holding registers analog values to be written.

Parameter Type: Unsigned16

Allowed values: 1 to 246, must be consistent with the Quantity of coils and the Data parameters.

Data
This parameter specifies the holding registers analog values that have to be written.

Parameter Type: Array of Unsigned16, transferred using the big-endian encoding.

Allowed values: The number of registers must be consistent with the Quantity of holding registers to write and the Data octets count parameters.

6.1.7.3.9.3 Service procedure

This service is used to write a specified number of holding registers in all the Unit ID addressable servers.

6.1.8 File ASE

6.1.8.1 Overview

This ASE enables the interaction and modeling with hierarchically structured real objects. The hierarchy is in three levels. The top level is made of files, and its underlying level, grouped by file, is made of records. The records themselves are sets of registers, contiguous from an application user (client user) point of view. Registers make up the third level. Since records do not have an independent naming and addressing, and are instead defined in terms of the constituent registers, the distinction between the second and third level is purely notational.

6.1.8.2 File class specification

6.1.8.2.1 Formal model

The File object is described by the following template:

ASE:			**File ASE**
CLASS:			**File**
CLASS ID:			not used
PARENT CLASS:			TOP
ATTRIBUTES:			
1.	(m)	Key Attribute:	Implicit
2.	(m)	Attribute:	Unit ID
3.	(c)	Constraint:	Read File Record
3.1	(m)	Attribute:	Read File Record sub-requests array
3.1.1	(m)	Attribute:	Sub-requests all-elements octet count
3.1.2	(m)	Attribute:	Sub-request element
3.1.2.1	(m)	Attribute:	Reference type
3.1.2.2	(m)	Attribute:	File number
3.1.2.3	(m)	Attribute:	Record number
3.1.2.4	(m)	Attribute:	Record length
3.2	(m)	Attribute:	Read File Record sub-responses array
3.2.1	(m)	Attribute:	Sub-responses all-elements octet count
3.2.2	(m)	Attribute:	Sub-response element
3.2.2.1	(m)	Attribute:	Sub-response octet count
3.2.2.2	(m)	Attribute:	Reference type
3.2.2.3	(m)	Attribute:	Address of Record data array
4.	(c)	Constraint:	Write File Record
4.1	(m)	Attribute:	Write File Record sub-requests array
4.1.1	(m)	Attribute:	Sub-requests all-elements octet count
4.1.2	(m)	Attribute:	Sub-request element
4.1.2.1	(m)	Attribute:	Reference type
4.1.2.2	(m)	Attribute:	File number
4.1.2.3	(m)	Attribute:	Record number
4.1.2.4	(m)	Attribute:	Record length
4.1.2.5	(m)	Attribute:	Address of Record data array
SERVICES:			
1.	(o)	OpsService:	Read File Record
2.	(o)	OpsService:	Write File Record

6.1.8.2.2 Attributes

6.1.8.2.2.1 Implicit

The attribute Implicit indicates that the File object is implicitly addressed by the services.

6.1.8.2.2.2 Unit ID

This attribute specifies the address of the server.

Attribute Type: Unsigned8

Allowed values: 1 to 247.

6.1.8.2.2.3 Read file record sub-requests array

This attribute is an array, and it specifies the sub-requests of the Read File Record service, one array element per sub-request.

Sub-requests all-elements octets count
This attribute is part of the Read File Record sub-requests array definition. It specifies the total count of octets of all the elements of the array (it does not include itself), each element being a Sub-request. The number of elements in the array can be obtained by dividing the Sub-requests all-elements octets count by the size in octets of one element, the Sub-request element, described below.

Attribute Type: Unsigned8

Allowed values: 7 to 245. The minimum is obtained when there is only one Sub-request element, and the maximum is obtained when there are 35 Sub-request elements. The upper bound is dictated by the maximum size of the APDU for client/server (Unit ID +

Function Code + Data = 254 octets), as described in IEC 61158-6-15. A correct Read File Record request will also have to ensure that the total size of the requested response, including all requested records, does not exceed this upper bound.

Sub-request element
This attribute is part of the Read File Record sub-requests array definition. Each element carries one sub-request, which refers to the reading of one record. Each element allows to uniquely qualify and identify a record within the information hierarchy using the Reference type, the File number, The Record number and the Record length. All is described below.

Reference type
This attribute is part of a Sub-request element, and it contributes to the qualification of the record. A record is a set of registers, in general registers have a type, and in the context of this service registers have also been called “references”, hence the term reference type. All the registers of a record are of the same type.

Attribute Type: Unsigned8

Allowed values: In the context of this service the only permitted value is 6.

File number
A file is an organization of records. This attribute is part of a Sub-request element and contributes to the identification of the record, representing the file containing the record.

Attribute Type: Unsigned16

Allowed values: The lowest File number is 1. The highest File Number should be 10.

NOTE 1 While it is allowed for the File Number to be in the range 1 to 0xFFFF, it should be noted that interoperability with legacy equipment may be compromised if the File Number is greater than 10.

Record number
A record is a set of registers. This attribute is part of a Sub-request element and contributes to the identification of a record, representing the address of the first register of the record. Records are identified using the address of their first register and their length, the latter is specified in number of registers.

Attribute Type: Unsigned16

Allowed values: Each file but the last should contain 10.000 registers, with the last file allowed to have less. This provides for records addressed from 0x0000 to 0x270F (0000 to 9999 decimal), at most. As a consequence, the Record Number should be in the range 0x0000 to 0x270F.

NOTE 2 While it is allowed for any file to have more or less than 10.000 registers, with a maximum of 65.536 (0x10000), and consequently to have records addressed from 0x0000 to 0xFFFF, it should be noted that interoperability with legacy equipment may be compromised if each file but the last does not have 10.000 registers, with the last file allowed to have less.

NOTE 3 Differently from other APOs, like Discretes, Coils, Input Registers and Holding Registers, where the lowest addressable instance is known to an application as 1-based and it is addressed in the protocol as 0-based, the lowest addressable record is record 0, known to an application as 0-based, and it is addressed in the protocol using a 0-based register address.

Record length
A record is a set of registers. This attribute is part of a Sub-request element and contributes to the identification of a record, representing the length of the record in number of registers. Records are identified using the address of their first register and their length, the latter is specified in number of registers.

Attribute Type: Unsigned16

Allowed values: For a given Record number, the Record length, in number of registers, must result in a record contained in the file. Moreover, such Record length, in combination with all the other parts of the request, must not produce a response that

exceeds the maximum size of the APDU for client/server (Unit ID + Function Code + Data = 254 octets), as described in IEC 61158-6-15.

6.1.8.2.2.4 Read file record sub-responses array

This attribute is an array, and it specifies the sub-responses of the Read File Record service, one array element per sub-response.

Sub-responses all-elements octets count
This attribute is part of the Read File Record sub-responses array definition. It contains the total count of octets of all the elements of the array (it does not inlude itself), each element being a Sub-response. The number of elements in the array can be obtained by dividing the Sub-responses all-elements octets count by the size in octets of one element, the Sub-response element, described below.

Attribute Type: Unsigned8

Allowed values:4 to 250. The minimum is obtained when there is only one Sub-response element and that is the smaller sub-response, with a record of length 1 register. The maximum can be reached in several ways, with different combinations of number of sub-responses and record lengths, for example with 34 sub-responses each with a record of length 1 register (136 octets so far), and one additional sub-response with a record of length of 56 registers (2 + (56 * 2) = 114 octets). The upper bound is dictated by the maximum size of the APDU for client/server (Unit ID + Function Code + Data = 254 octets), as described in IEC 61158-6-15. A correct Read File Record request will have ensured that the total size of the requested response, including all requested records, does not exceed this upper bound.

Sub-response element
This attribute is part of the Read File Record sub-responses array definition. Each element carries one sub-response, which refers to the reading of the record as requested in the corresponding sub-request. Each element specifies the Sub-response octet count, the Reference type and requested record data. All is described below.

Sub-response octets count
This attribute is part of a Sub-response element. It specifies the total count of octets (excluding itself) for the sub-response. The count includes the Reference type octet and the octets contained in the Record data array, all together 1 + twice the record length specified in the corresponding sub-request, which is expressed in number of registers.

NOTE 1 The Record data array is specified indirectly with the attribute Address of Record data array. The number of elements of the Record data array is in general different for each Sub-response element, since it depends on the record length of the corresponding sub-request. An embedded specification would be violating the definition of (same type) array element for the Sub-response element itself. The indirection allows a specification via arrays instead of more complex types. The encoding does embed each Record data array in-line, as described in IEC 61158-6-15.

Attribute Type: Unsigned8

Allowed values: This value shall be a minimum of 3 and a maximum of 249. The minimum is obtained when the requested record has the length of 1 register. The maximum is reached when the requested record has the length of 124 registers, and in this case this is the only sub-response in the Read File Record sub-responses array.

Reference type
This attribute is part of a Sub-response element, and it contributes to the qualification of the record. A record is a set of registers, in general registers have a type, and in the context of this service registers have also been called “references”, hence the term Reference type. All the registers of a record are of the same type.

Attribute Type: Unsigned8

Allowed values: In the context of this service the only permitted value is 6.

Address of record data array
This attribute is part of a Sub-response element and it specifies the address of the record data array, constituting the record being read.

Attribute Type: Address of array of Unsigned16

Allowed values: The number of registers in the addressed array must be consistent with the Sub-response octect count specified in the current Sub-response element.

NOTE 2 The Record data array is specified indirectly with the attribute Address of Record data array. The number of elements of the Record data array is in general different for each Sub-response element, since it depends on the record length of the corresponding sub-request. An embedded specification would be violating the definition of (same type) array element for the Sub-response element itself. The indirection allows a specification via arrays instead of more complex types. The encoding does embed each Record data array in-line, as described in IEC 61158-6-15.

6.1.8.2.2.5 Write file record sub-requests array

This attribute is an array, and it specifies the sub-requests of the Write File Record service, one array element per sub-request.

Sub-requests all-elements octets count
This attribute is part of the Write File Record sub-requests array definition. It specifies the total count of octets of all the elements of the array (it does not include itself), each element being a Sub-request. The number of elements in the array can be obtained by dividing the Sub-requests all-elements octets count by the size in octets of one element, the Sub-request element, described below.

Attribute Type: Unsigned8

Allowed values: 9 to 251. The minimum is obtained when there is only one Sub-request element, and that is the smaller sub-request, with a record of length 1 register. The maximum can be reached in several ways, with different combinations of number of sub-requests and record lengths, for example with 3 sub-requests, one with a record of length 1 register (9 octets so far), one with a record of length 113 registers (130 octets so far) and finally another one with a record length of 113 registers (for a total of 251 octets). The upper bound is dictated by the maximum size of the APDU for client/server (Unit ID + Function Code + Data = 254 octets), as described in IEC 61158-6-15. A correct Write File Record request will have a normal response that does not exceed the upper bound, since in this case, as described below, a successful Write File Record response is an exact copy of the Write File Record request.

Sub-request element
This attribute is part of the Write File Record sub-requests array definition. Each element carries one sub-request, which refers to the writing of one record. Each element allows to uniquely qualify and identify a record within the information hierarchy using the Reference type, the File number, The Record number and the Record length. In addition, since the request is a write request, each element allows for record data as well. All is described below.

Reference type
This attribute is part of a Sub-request element, and it contributes to the qualification of the record. A record is a set of registers, in general registers have a type, and in the context of this service registers have also been called “references”, hence the term Reference type. All the registers of a record are of the same type.

Attribute Type: Unsigned8

Allowed values: In the context of this service the only permited value is 6.

File number
A file is an organization of records. This attribute is part of a Sub-request element and contributes to the identification of the record, representing the file containing the record.

Attribute Type: Unsigned16

Allowed values: The lowest File number is 1. The highest File Number should be 10.

NOTE 1 While it is allowed for the File Number to be in the range 1 to 0xFFFF, it should be noted that interoperability with legacy equipment may be compromised if the File Number is greater than 10.

Record number
A record is a set of registers. This attribute is part of a Sub-request element and contributes to the identification of a record, representing the address of the first register of the record. Records are identified using the address of their first register and their length, the latter is specified in number of registers.

Attribute Type: Unsigned16

Allowed values: Each file but the last should contain 10.000 registers, with the last file allowed to have less. This provides for records addressed from 0x0000 to 0x270F (0000 to 9999 decimal), at most. As a consequence, the Record Number should be in the range 0x0000 to 0x270F.

NOTE 2 While it is allowed for any file to have more or less than 10.000 registers, with a maximum of 65.536 (0x10000), and consequently to have records addressed from 0x0000 to 0xFFFF, it should be noted that interoperability with legacy equipment may be compromised if each file but the last does not have 10.000 registers, with the last file allowed to have less.

NOTE 3 Differently from other APOs, like Discretes, Coils, Input Registers and Holding Registers, where the lowest addressable instance is known to an application as 1-based and it is addressed in the protocol as 0-based, the lowest addressable record is record 0, known to an application as 0-based, and it is addressed in the protocol using a 0-based register address.

Record length
A record is a set of registers. This attribute is part of a Sub-request element and contributes to the identification of a record, representing the length of the record in number of registers. Records are identified using the address of their first register and their length, the latter is specified in number of registers.

Attribute Type: Unsigned16

Allowed values: For a given Record number, the Record length, in number of registers, must result in a record contained in the file. Moreover, such Record length, in combination with all the other parts of the request, must not subsume a request/response that exceeds the maximum size of the APDU for client/server (Unit ID + Function Code + Data = 254 octets), as described in IEC 61158-6-15. With respect for the above, this value must be a minimum of 1 and a maximum of 122. The maximum can be specified when this is the only sub-request in the Write File Record sub-requests array.

Address of record data array
This attribute is part of a Sub-request element and it specifies the address of the record data array, constituting the record to be written.

Attribute Type: Address of array of Unsigned16

Allowed values: The number of registers in the addressed array must be consistent with the Record length specified in the current Sub-request element.

NOTE 4 The Record data array is specified indirectly with the attribute Address of Record data array. The number of elements of the Record data array is in general different for each Sub-response element, since it depends on the record length of the corresponding sub-request. An embedded specification would be violating the definition of (same type) array element for the Sub-response element itself. The indirection allows a specification via arrays instead of more complex types. The encoding does embed each Record data array in-line, as described in IEC 61158-6-15.

6.1.8.2.3 Services

Read file record
This optional service is used to read multiple records from one or more files.

Write file record
This optional service is used to write multiple records into one or more files.

6.1.8.3 File ASE service specification

6.1.8.3.1 Supported services

The File ASE defines the services:

Read File Record

Write File Record

6.1.8.3.2 Read file Record service

6.1.8.3.2.1 Service overview

The Read File Record service is used to read multiple records from one or more files.

6.1.8.3.2.2 Service primitives

6.1.8.3.2.2.1 General

The service parameters for this service are shown in Table 20. It is a confirmed service.

Table 20 – Read file service parameters

Parameter name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
Unit ID	M	M(=)		
Invoke ID	C	C(=)		
Read File Record sub-requests array	M	M(=)		
Sub-requests all-elements octets count	M	M(=)		
Sub-request element	M	M(=)		
Reference type	M	M(=)		
File number	M	M(=)		
Record number	M	M(=)		
Record length	M	M(=)		
Result (+)			S	S(=)
Unit ID			M	M(=)
Invoke ID			C	C(=)
Read File Record sub-responses array			M	M(=)
Sub-responses all-elements octets count			M	M(=)
Sub-response element			M	M(=)
Sub-response octets count			M	M(=)
Reference type			M	M(=)
Address of Record data array			M	M(=)
Result (-)			S	S(=)
Unit ID			M	M(=)
Invoke ID			C	C(=)
Error info			M	M(=)

6.1.8.3.2.2.2 Argument

The argument conveys the service specific parameters of the service request.

Unit ID
This parameter specifies the address of the server.

Parameter Type: Unsigned8

Allowed values: 1 to 247.

Read file record sub-requests array
This parameter is an array, and it specifies the sub-requests of the Read File Record service, one array element per sub-request.

Sub-requests all-elements octets count
This parameter is part of the Read File Record sub-requests array parameter. It specifies the total count of octets of all the elements of the array (it does not include itself), each element being a Sub-request. The number of elements in the array can be obtained by dividing the Sub-requests all-elements octet count by the size in octets of one element, the Sub-request element, described below.

Parameter Type: Unsigned8

Allowed values: 7 to 245. The minimum is obtained when there is only one Sub-request element, and the maximum is obtained when there are 35 Sub-request elements. The upper bound is dictated by the maximum size of the APDU for client/server (Unit ID + Function Code + Data = 254 octets), as described in IEC 61158-6-15. A correct Read File Record request will also have to ensure that the total size of the requested response, including all requested records, does not exceed this upper bound.

Sub-request element
This parameter is part of the Read File Record sub-requests array parameter. Each element carries one sub-request, which refers to the reading of one record. Each element allows to uniquely quality and identify a record within the information hierarchy using the Reference type, the File number, The Record number and the Record length. All is described below.

Reference type
This parameter is part of a Sub-request element, it specifies the reference type, that contributes to the qualification of the record. A record is a set of registers, in general registers have a type, and in the context of this service registers have also been called “references”, hence the term reference type. All the registers of a record are of the same type.

Parameter Type: Unsigned8

Allowed values: In the context of this service the only allowed value is 6.

File number
This parameter is part of a Sub-request element, it specifies the file number, that contributes to the qualification of the record. A file is an organization of records, and this parameter represents the file containing the record.

Parameter Type: Unsigned16

Allowed values: The lowest File number is 1. The highest File Number should be 10.

NOTE 1 While it is allowed for the File Number to be in the range 1 to 0xFFFF, it should be noted that interoperability with legacy equipment may be compromised if the File Number is greater than 10.

Record number
This parameter is part of a Sub-request element, it specifies the record number, that contributes to the qualification of the record. Records are identified using the

address of their first register and their length, the latter is specified in number of registers; this parameter represents the address of the first register of the record.

Parameter Type: Unsigned16

Allowed values: Each file but the last should contain 10.000 registers, with the last file allowed to have less. This provides for records addressed from 0x0000 to 0x270F (0000 to 9999 decimal), at most. As a consequence, the Record Number should be in the range 0x0000 to 0x270F.

NOTE 2 While it is allowed for any file to have more or less than 10.000 registers, with a maximum of 65.536 (0x10000), and consequently to have records addressed from 0x0000 to 0xFFFF, it should be noted that interoperability with legacy equipment may be compromised if each file but the last does not have 10.000 registers, with the last file allowed to have less.

NOTE 3 Differently from other APOs, like Discretes, Coils, Input Registers and Holding Registers, where the lowest addressable instance is known to an application as 1-based and it is addressed in the protocol as 0-based, the lowest addressable record is record 0, known to an application as 0-based, and it is addressed in the protocol using a 0-based register address.

Record length
This parameter is part of a Sub-request element, it specifies the record length, that contributes to the qualification of the record. Records are identified using the address of their first register and their length, the latter is specified in number of registers; this parameter represents the length of the record in number of registers.

Parameter Type: Unsigned16

Allowed values: For a given Record number, the Record length, in number of registers, must result in a record contained in the file. Moreover, such Record length, in combination with all the other parts of the request, must not produce a response that exceeds the maximum size of the APDU for client/server (Unit ID + Function Code + Data = 254 octets), as described in IEC 61158-6-15.

6.1.8.3.2.2.3 Result(+)

This selection type parameter indicates that the service request succeeded.

Unit ID
In the absence of error, this parameter is identical to the correspondingly named request parameter.

Read file record sub-responses array
This parameter is an array, and it specifies the sub-responses of the Read File Record service, one array element per sub-response.

Sub-responses all-elements octet count
This parameter is part of the Read File Record sub-responses array parameter. It specifies the total count of octets of all the elements of the array (it does not include itself), each element being a Sub-response. The number of elements in the array can be obtained by dividing the Sub-responses all-elements octets count by the size in octets of one element, the Sub-response element, described below.

Parameter Type: Unsigned8

Allowed values: The successful value depends on the request. For any successful request the value will be in the range 4 to 250. The minimum is obtained when there is only one Sub-response element and that is the smaller sub-response, with a record of length 1 register. The maximum can be reached in several ways, with different combinations of number of sub-responses and record lengths, for example with 34 sub-responses each with a record of length 1 register (136 octets so far), and one additional sub-response with a record of length of 56 registers (2 + (56 * 2) = 114 octets). The upper bound is dictated by the maximum size of the APDU for client/server (Unit ID + Function Code + Data = 254 octets), as described in IEC 61158-6-15. A correct Read File Record request will have ensured that the total

size of the requested response, including all requested records, does not exceed this upper bound.

Sub-response element
This parameter is part of the Read File Record sub-responses array parameter. Each element carries one sub-response, which refers to the reading of the record as requested in the corresponding sub-request. Each element specifies the Sub-response octets count, the Reference type and requested record data. All is described below.

Sub-response octets count
This parameter is part of the Sub-response element, it specifies the total count of octets (excluding itself) for the sub-response. The count includes the Reference type octet and the octets contained in the Record data array, all together 1 + twice the record length specified in the corresponding sub-request, which is expressed in number of registers.

NOTE 1 The Record data array is specified indirectly with the attribute Address of Record data array. The number of elements of the Record data array is in general different for each Sub-response element, since it depends on the record length of the corresponding sub-request. An embedded specification would be violating the definition of (same type) array element for the Sub-response element itself. The indirection allows a specification via arrays instead of more complex types. The encoding does embed each Record data array in-line, as described in IEC 61158-6-15.

Parameter Type: Unsigned8

Allowed values: The successful value depends on the request. For any successful request the value will be a minimum of 3 and a maximum of 249. The minimum is obtained when the requested record has the length of 1 register. The maximum is reached when the requested record has the length of 124 registers, and in this case this is the only sub-response in the Read File Record sub-responses array.

Reference type
This parameter shall be the same as the parameter Reference type in the request.

Parameter Type: Unsigned8

Address of Record data array
This parameter is part of the Sub-response element, it specifies the address of the record data array, constituting the record being read.

Parameter Type: Address of array of Unsigned16

Allowed values: The successful value depends on the request. For any successful request the number of registers in the addressed array will be consistent with the Sub-response octects count specified in the current Sub-response element.

NOTE 2 The Record data array is specified indirectly with the attribute Address of Record data array. The number of elements of the Record data array is in general different for each Sub-response element, since it depends on the record length of the corresponding sub-request. An embedded specification would be violating the definition of (same type) array element for the Sub-response element itself. The indirection allows a specification via arrays instead of more complex types. The encoding does embed each Record data array in-line, as described in IEC 61158-6-15.

6.1.8.3.2.2.4 Result(-)

This selection type parameter indicates that the service request failed.

Unit ID
In the absence of error, this parameter is identical to the correspondingly named request parameter.

Error info
This parameter specifies error information.

Parameter Type: Unsigned8

6.1.8.3.2.3 Service procedure

This service is used to read multiple records from one or more files.

6.1.8.3.3 Write file record service

6.1.8.3.3.1 Service overview

The Write File Record service is used to write multiple records into one or more files.

6.1.8.3.3.2 Service primitives

6.1.8.3.3.2.1 General

The service parameters for this service are shown in Table 21. It is a confirmed service.

Table 21 – Write file service parameters

Parameter name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
Unit ID	M	M(=)		
Invoke ID	C	C(=)		
Write File Record sub-requests array	M	M(=)		
Sub-requests all-elements octets count	M	M(=)		
Sub-request element	M	M(=)		
Reference type	M	M(=)		
File number	M	M(=)		
Record number	M	M(=)		
Record length	M	M(=)		
Address of Record data array	M	M(=)		
Result (+)			S	S(=)
Unit ID			M	M(=)
Invoke ID			C	C(=)
Write File Record sub-requests array			M	M(=)
Sub-requests all-elements octets count			M	M(=)
Sub-request element			M	M(=)
Reference type			M	M(=)
File number			M	M(=)
Record number			M	M(=)
Address of Record data array			M	M(=)
Result (-)			S	S(=)
Unit ID			M	M(=)
Invoke ID			C	C(=)
Error info			M	M(=)

6.1.8.3.3.2.2 Argument

The argument conveys the service specific parameters of the service request.

Unit ID
This parameter shall be used to specify the address of the server.

Parameter Type: Unsigned8

Allowed values: 1 to 247.

Write file record sub-requests array
This parameter is an array, and it specifies the sub-requests of the Write File Record service, one array element per sub-request.

Sub-requests all-elements octets count
This parameter is part of the Write File Record sub-requests array parameter. It specifies the total count of octets of all the elements of the array (it does not include itself), each element being a Sub-request. The number of elements in the array can be obtained by dividing the Sub-requests all-elements octets count by the size in octets of one element, the Sub-request element, described below.

Parameter Type: Unsigned8

Allowed values: 9 to 251. The minimum is obtained when there is only one Sub-request element, and that is the smaller sub-request, with a record of length 1 register. The maximum can be reached in several ways, with different combinations of number of sub-requests and record lengths, for example with three sub-requests, one with a record of length 1 register (9 octets so far), one with a record of length 113 registers (130 octets so far) and finally another one with a record length of 113 registers (for a total of 251 octets). The upper bound is dictated by the maximum size of the APDU for client/server (Unit ID + Function Code + Data = 254 octets), as described in IEC 61158-6-15. A correct Write File Record request will have a normal response that does not exceed the upper bound, since in this case, as described below, a successful Write File Record response is an exact copy of the Write File Record request.

Sub-request element
This parameter is part of the Write File Record sub-requests array parameter. Each element carries one sub-request, which refers to the writing of one record. Each element allows to uniquely qualify and identify a record within the information hierarchy using the Reference type, the File number, The Record number and the Record length. In addition, since the request is a write request, each element allows for record data as well. All is described below.

Reference type
This parameter is part of a Sub-request element, it specifies the reference type, that contributes to the qualification of the record. A record is a set of registers, in general registers have a type, and in the context of this service registers have also been called "references", hence the term reference type. All the registers of a record are of the same type.

Parameter Type: Unsigned8

Allowed values: In the context of this service the only permitted value is 6.

File number
This parameter is part of a Sub-request element, it specifies the file number, that contributes to the qualification of the record. A file is an organization of records, and this parameter represents the file containing the record.

Parameter Type: Unsigned16

Allowed values: The lowest File number is 1. The highest File Number should be 10.

NOTE 1 While it is allowed for the File Number to be in the range 1 to 0xFFFF, it should be noted that interoperability with legacy equipment may be compromised if the File Number is greater than 10.

Record number
This parameter is part of a Sub-request element, it specifies the record number, that contributes to the qualification of the record. Records are identified using the address of their first register and their length, the latter is specified in number of registers; this parameter represents the address of the first register of the record.

Parameter Type: Unsigned16

Allowed values: Each file but the last should contain 10.000 registers, with the last file allowed to have less. This provides for records addressed from 0x0000 to 0x270F (0000 to 9999 decimal), at most. As a consequence, the Record Number should be in the range 0x0000 to 0x270F.

NOTE 2 While it is allowed for any file to have more or less than 10.000 registers, with a maximum of 65.536 (0x10000), and consequently to have records addressed from 0x0000 to 0xFFFF, it should be noted that interoperability with legacy equipment may be compromised if each file but the last does not have 10.000 registers, with the last file allowed to have less.

NOTE 3 Differently from other APOs, like Discretes, Coils, Input Registers and Holding Registers, where the lowest addressable instance is known to an application as 1-based and it is addressed in the protocol as 0-based, the lowest addressable record is record 0, known to an application as 0-based, and it is addressed in the protocol using a 0-based register address.

Record length
This parameter is part of a Sub-request element, it specifies the record length, that contributes to the qualification of the record. Records are identified using the address of their first register and their length, the latter is specified in number of registers; this parameter represents the length of the record in number of registers.

Parameter Type: Unsigned16

Allowed values: For a given Record number, the Record length, in number of registers, must result in a record contained in the file. Moreover, such Record length, in combination with all the other parts of the request, shall not produce a response that exceeds the maximum size of the APDU for client/server (Unit ID + Function Code + Data = 254 octets), as described in IEC 61158-6-15. A correct Write File Record request will have a normal response that does not exceed the upper bound, since in this case, as described below, a successful Write File Record response is an exact copy of the Write File Record request.

Address of record data array
This parameter is part of the Sub-request element, it specifies the address of the record data array, constituting the record to be written.

Parameter Type: Address of array of Unsigned16

Allowed values: The number of registers in the addressed array must be consistent with the Record length specified in the current Sub-request element.

NOTE 5 The Record data array is specified indirectly with the attribute Address of Record data array. The number of elements of the Record data array is in general different for each Sub-response element, since it depends on the record length of the corresponding sub-request. An embedded specification would be violating the definition of (same type) array element for the Sub-response element itself. The indirection allows a specification via arrays instead of more complex types. The encoding does embed each Record data array in-line, as described in IEC 61158-6-15.

6.1.8.3.3.2.3 Result(+)

This selection type parameter indicates that the service request succeeded.

Unit ID
In the absence of error, this parameter is identical to the correspondingly named request parameter.

Write file record sub-requests array
This parameter is an array. In the absence of error, this parameter is identical to the correspondingly named request parameter.

Sub-requests all-elements octets count
In the absence of error, this parameter is identical to the correspondingly named request parameter.

Sub-request element
In the absence of error, this parameter is identical to the correspondingly named request parameter.

Reference type
In the absence of error, this parameter is identical to the correspondingly named request parameter.

File number
In the absence of error, this parameter is identical to the correspondingly named request parameter.

Record number
In the absence of error, this parameter is identical to the correspondingly named request parameter.

Record length
In the absence of error, this parameter is identical to the correspondingly named request parameter.

Address of record data array
In the absence of error, this parameter is identical to the correspondingly named request parameter.

6.1.8.3.3.2.4 Result(-)

This selection type parameter indicates that the service request failed.

Unit ID
In the absence of error, this parameter is identical to the correspondingly named request parameter.

Error info
This parameter specifies error information.

Parameter Type: Unsigned8

6.1.8.3.3.3 Service procedure

This service is used to write multiple records into one or more files.

6.1.9 Encapsulated interface ASE

6.1.9.1 Overview

The client/server Encapsulated Interface is a mechanism for tunnelling a defined interface services requests as well as their responses inside a client/server service request and response, respectively. The defined interface is characterized by an octet value called MEI type. This is illustrated in Figure 21. The MEI type is used to tag and dispatch services to the proper interface.

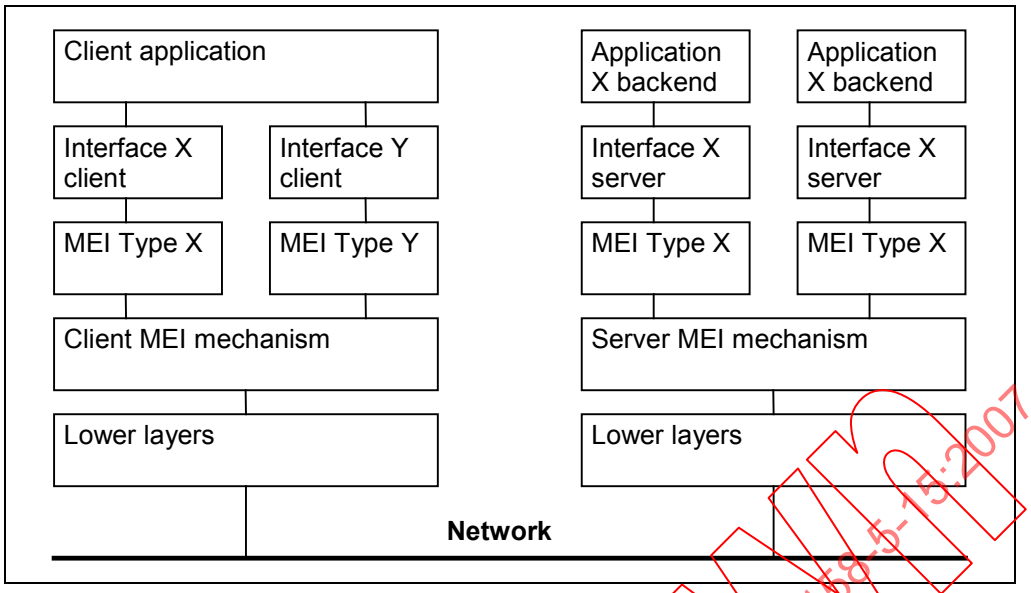


Figure 21 – Client/server encapsulated interface mechanism

6.1.9.2 Encapsulated interface class specification

6.1.9.2.1 Formal model

The Encapsulated Interface class is never instantiated and no object can be constructed with its template, it is only used for derivation and therefore it does not have a Key Attribute. It is described by the following template:

ASE: **Encapsulated Interface ASE**
CLASS: **Encapsulated Interface**
CLASS ID: NULL
PARENT CLASS: TOP
ATTRIBUTES:

1.	(m)	Attribute:	Unit ID
2.	(m)	Attribute:	MEI type
2.	(m)	Attribute:	MEI type specific data

6.1.9.2.2 Attributes

Unit ID
This attribute specifies the address of the server.

Attribute Type: Unsigned8

Allowed values: 1 to 247.

MEI type
This attribute specifies the Encapsulated Interface type.

Attribute Type: Unsigned8

Allowed values: 13 or 14; all the other values are reserved.

MEI type specific data
This attribute specifies data being transferred, and it is Encapsulated Interface specific.

Attribute Type: Array of Unsigned8

Allowed values: Encapsulated Interface specific, and a derived class will override this attribute. In any case this attribute will not have more than 251 elements. The upper bound is

dictated by the maximum size of the APDU for client/server (Unit ID + Function Code + Data = 254 octets), as described in IEC 61158-6-15.

6.1.9.2.3 Services

The Encapsulated Interface class is never instantiated and no object can be constructed with its template, it is only used for derivation. All the services are Encapsulated Interface specific, as labeled by the MEI, and therefore specified in ASEs containing Encapsulated Interface derived classes.

6.1.10 Read device identification ASE

6.1.10.1 Overview

This ASE allows the modeling of an address space populated by objects, and their retrieval. These objects are meant to provide physical and functional introspection of the remote hosting device. Each object is identified by an object ID.

The objects are partitioned in 3 categories:

Basic device identification
The objects in this category are defined by this specification, and their inclusion is mandatory. Object IDs are uniquely assigned to these objects.

Regular device identification
The objects in this category are defined by this specification, but their inclusion is optional. Only a subset of the object IDs allocated for this category is currently assigned. Reserved object IDs are kept aside for future assignments.

Extended device identification
The objects in this category are defined by an application, and their inclusion is optional. Basically an application is provided with a range of unallocated object IDs to use for the purpose of extending the identification of a device. Knowledge about these objects must be shared by client and servers by means other than this specification.

The Device Identification categories and the objects attributes are described in Table 22.

Table 22 – Device identification categories

Object IDs	Object name / Description	Type	Presence	Category
0x00	Vendor Name	ASCII string	Mandatory	Basic
0x01	Product Code	ASCII string	Mandatory	
0x02	Major Minor Revision	ASCII string	Mandatory	
0x03	Vendor URL	ASCII string	Optional	Regular
0x04	Product Name	ASCII string		
0x05	Model Name	ASCII string		
0x06	User Application Name	ASCII string		
0x07	*Reserved*			
…				
0x7F				
0x80	*Private application-defined objects. The object ID range [0x80 – 0xFF] can be used by an application to define its own objects.*	Device dependent	Optional	Extended
…				
0xFF				

In all cases any object must have an individual size consistent with the maximum size allowed by the APDU for client/server (Unit ID + Function Code + Data = 254 octets), as described in IEC 61158-6-15.

6.1.10.2 Read device identification class specification

6.1.10.2.1 Formal model

The Read Device Identification class is derived from the Encapsulated Interface class. The Read Device Identification object is described by the following template:

ASE: **Read Device Identification ASE**
CLASS: **Read Device Identification**
CLASS ID: not used
PARENT CLASS: Encapsulated Interface
ATTRIBUTES:

1.	(m)	Key Attribute:	Implicit
2.	(m)	Attribute:	Unit ID
3.	(m)	Attribute:	MEI type
4.	(m)	Attribute:	Read Device ID code
5.	(m)	Attribute:	Requested-object ID
6.	(m)	Attribute:	Conformity level
7.	(m)	Attribute:	More-available flag
8.	(m)	Attribute:	Next-object ID
9.	(m)	Attribute:	Objects array
9.1	(m)	Attribute:	Number of objects
9.2	(m)	Attribute:	Objects array element
9.2.1	(m)	Attribute:	Returned-object ID
9.2.2	(m)	Attribute:	Object length
9.2.3	(m)	Attribute:	Object value

SERVICES:

1.	(m)	OpsService:	Read Device Identification

6.1.10.2.2 Attributes

6.1.10.2.2.1 Implicit

The attribute Implicit indicates that the Read Device Identification object is implicitly addressed by the services.

6.1.10.2.2.2 Unit ID

This attribute specifies the address of the server.

Attribute Type: Unsigned8

Allowed values: 1 to 247.

6.1.10.2.2.3 MEI type

This attribute specify the Encapsulated Interface type.

Attribute Type: Unsigned8

Allowed values: 14

6.1.10.2.2.4 Read Device ID code

This attribute specifies the requested device access type, that qualifies the requested information based on device categories and, when supported, individually addressed object retrieval. This is illustrated in Table 23.

Attribute Type: Unsigned8

Allowed values: As illustrated in Table 23.

Table 23 – Read device ID code

Value	Read Device ID code
0x01	Request to retrieve the objects in the Basic Device Identification category. A stream of objects is expected.
0x02	Request to retrieve the objects in the Regular Device Identification category, if any, and this implies a request for the Basic Device Identification category as well. A stream of objects is expected, with the Basic Device Identification category returned first, and the Regular Device Identification category afterward, if any.
0x03	Request to retrieve the objects in the Extended Device Identification category, if any, and this implies a request for the Regular Device Identification category, if any, and for the Basic Device Identification category as well. A stream of objects is expected, with the Basic Device Identification category returned first, the Regular Device Identification category after that, if any, and finally the Extended Device Identification category, if any.
0x04	Request to retrieve a specific object. If this feature is supported, a single object is expected, otherwise an error response is returned.

NOTE While the returned categories are ordered as from Table 23, no assumption should be made about the order of returned objects within any category, even across service invocations. This is to avoid any extra processing load on servers that may reside in very simple devices, and to permit the best object packing when the retrieval of multiple objects requires multiple request/response transactions.

6.1.10.2.2.5 Requested-object ID

This attribute specifies the ID that identifies the first requested object, or the single requested object, according to the Read Device ID code. A response cannot exceed the maximum size of the APDU for client/server (Unit ID + Function Code + Data = 254 octets), as described in IEC 61158-6-15. An individual object size is guaranteed to fit in the maximum size by definition. If the set of returned objects requires more octets than the maximum size, then several transactions (request/response) are needed. This is an application client

responsibility. When many objects are requested, the ID of the first requested object of the first transaction must be set to 0x00. If the initial object ID is not 0x00, then in general the returned Device Identification will be incomplete. Subsequent requests within the same set of objects must set the Requested-object ID to the Next-object ID returned by the server in the previous response. If a different Requested-object ID is provided, then in general the returned Device Identification will be incomplete. When many objects are requested, if the Requested-object ID does not match any known object, the server will respond as if the object with ID 0x00 was requested, effectively restarting from the beginning if this happens in the middle of the retrieval. When the server supports the retrieval of a single object, and this is performed with a Requested-object ID that does not match any known object, the server will return an error response.

Attribute Type: Unsigned8

Allowed values: 0x00 to 0xFF.

6.1.10.2.2.6 Conformity level

This attribute specifies information about the actual object categories and object retrieval access type made available by the server. Its value is provided by the server in all the responses, irrespective of the requested Read Device ID code. Values are illustrated in Table 24. In the Read Device ID attribute description it was explained what is returned when dealing with objects unknown to the server. For known objects, if a Read Device ID code requests a category or a type of access that is not available on the server, then the returned objects are as from Table 25.

Attribute Type: Unsigned8

Expected values: As illustrated in Table 24.

Table 24 – Conformity level

Value	Read Device ID code
0x01	The device supports only the Basic Device Identification category, and only stream access.
0x02	In addition to the Basic Device Identification category, the device supports also the Regular Device Identification category, and only stream access.
0x03	In addition to the Basic Device Identification category and to the Regular Device Identification category, the device supports also the Extended Device Identification category, and only stream access.
0x81	The device supports only the Basic Device Identification category, and both stream access and individual access.
0x82	In addition to the Basic Device Identification category, the device supports also the Regular Device Identification category, and both stream access and individual access.
0x83	In addition to the Basic Device Identification category and to the Regular Device Identification category, the device supports also the Extended Device Identification category, and both stream access and individual access.

Table 25 – Requested vs. returned known objects

Read Device ID	Conformity level	Returned objects
0x01or 0x02 or 0x03	0x01	The server returns the objects in the Basic Device Identification category. Objects are returned as a stream.
0x01	0x02	The server returns the objects in the Basic Device Identification category. Objects are returned as a stream.
0x02 or 0x03	0x02	In addition to objects in the Basic Device Identification category, the server returns afterward also the objects in the Regular Device Identification category. Objects are returned as a stream.
0x01	0x03	The server returns the objects in the Basic Device Identification category. Objects are returned as a stream.
0x02	0x03	In addition to objects in the Basic Device Identification category, the server returns afterward also the objects in the Regular Device Identification category. Objects are returned as a stream.
0x03	0x03	In addition to objects in the Basic Device Identification category and after that objects in the Regular Device Identification category, the server returns afterward also the objects in the Extended Device Identification category. Objects are returned as a stream.
0x04	0x01 or 0x02 or 0x03	The server returns an illegal function error response.
0x01or 0x02 or 0x03	0x81	The server returns the objects in the Basic Device Identification category. Objects are returned as a stream.
0x01	0x82	The server returns the objects in the Basic Device Identification category. Objects are returned as a stream.
0x02 or 0x03	0x82	In addition to objects in the Basic Device Identification category, the server returns afterward also the objects in the Regular Device Identification category. Objects are returned as a stream.
0x01	0x83	The server returns the objects in the Basic Device Identification category. Objects are returned as a stream.
0x02	0x83	In addition to objects in the Basic Device Identification category, the server returns afterward also the objects in the Regular Device Identification category. Objects are returned as a stream.
0x03	0x83	In addition to objects in the Basic Device Identification category and after that objects in the Regular Device Identification category, the server returns afterward also the objects in the Extended Device Identification category. Objects are returned as a stream.
0x04	0x81 or 0x82 or 0x83	The server returns the individually requested object.

6.1.10.2.2.7 More-available flag

This attribute specifies information about having or not more objects to retrieve after a request/response. It is meaningful when the Read Device ID code is one of 0x01, 0x02, or 0x03, and the response exceed the maximum size of the APDU for client/server (Unit ID + Function Code + Data = 254 octets), as described in IEC 61158-6-15. A value of 0x00 means that there are no more objects available, while a value of 0xFF means that more requests have to be issued to retrieve the remaining objects. The meaning of this attribute is related to the one of the Next-object ID attribute.

Attribute Type: Unsigned8

Expected values: 0x00, 0xFF (boolean)

6.1.10.2.2.8 Next-object ID

This attribute specifies the ID of the object that has to be requested in a subsequent request when the More-available flag is 0xFF. This is a client's responsibility, and if the Next-object ID requested in the subsequent request does not match any known object, the server will respond as if the object with ID 0x00 was requested, effectively restarting from the beginning.

Attribute Type: Unsigned8

Allowed values: 0x00 to 0xFF.

6.1.10.2.2.9 Objects array

This attribute is an array, it specifies requested objects. Each array element holds an object, as described below. This array may or may not contain all the requested objects at once, due to size limitations as illustrated in the Requested-object ID description.

Number of objects
This attribute is part of the Read Device Identification array definition. It specifies the number of objects in the array.

Attribute Type: Unsigned8

Allowed values: As allowed by the maximum size as illustrated in the Requested-object ID description.

Objects array element
This attribute is part of the Read Device Identification array definition. Each element carries one object. Each element uniquely identifies and represents an object within the device identification address space using the Returned-object ID, the Object length and the Object value. All is described below.

Returned-object ID
This attribute is part of an Objects array element, and it uniquely identifies the object. Its description is the same as for the Requested-object ID, and it is part of the stream that initiated with the Requested-object ID.

Attribute Type: Unsigned8

Allowed values: 0x00 to 0xFF.

Object length
This attribute is part of an Objects array element. It describes the length of the object value, in octets.

Attribute Type: Unsigned8

Allowed values: As allowed by the maximum as illustrated in the Requested-object ID description.

Object value
This attribute is part of an Objects array element and contributes to the device identification.

Attribute Type: As described in Table 22.

Allowed values: As described in Table 22, with respect for the maximum size as illustrated in the Requested-object ID description.

6.1.10.2.3 Services

Read Device Identification

This service is used to retrieve the device identification objects.

6.1.10.3 Read device identification ASE service specification

6.1.10.3.1 Supported services

The File ASE defines the services:

Read Device Identification

6.1.10.3.2 Read device identification service

6.1.10.3.2.1 Service overview

The Read Device Identification service is used to retrieve the device identification objects.

6.1.10.3.2.2 Service primitives

6.1.10.3.2.2.1 General

The service parameters for this service are shown in Table 26. It is a confirmed service.

Table 26 – Read device identification service parameters

Parameter name	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
Unit ID	M	M(=)		
Invoke ID	C	C(=)		
MEI type	M	M(=)		
Read Device ID code	M	M(=)		
Requested-object ID	M	M(=)		
Result (+)			S	S(=)
Unit ID			M	M(=)
Invoke ID			C	C(=)
MEI type			M	M(=)
Read Device ID code			M	M(=)
Conformity level			M	M(=)
More-available flag			M	M(=)
Next-object ID			M	M(=)
Objects array			M	M(=)
Objects array element			M	M(=)
Returned-object ID			M	M(=)
Object length			M	M(=)
Object value			M	M(=)
Result (-)			S	S(=)
Unit ID			M	M(=)
Invoke ID			C	C(=)
Error info			M	M(=)

6.1.10.3.2.2.2 Argument

The argument conveys the service specific parameters of the service request.

Unit ID
This parameter specifies the address of the server.

Parameter Type: Unsigned8

Allowed values: 1 to 247.

MEI
This parameter specifies the Encapsulated Interface type.

Parameter Type: Unsigned8

Allowed values: 14

Read Device ID code
This parameter specifies the requested device access type, that qualifies the requested information based on device categories and, when supported, individually addressed object retrieval. This is illustrated in Table 23.

Parameter Type: Unsigned8

Allowed values: As illustrated in Table 23.

NOTE While the returned categories are ordered as from Table 23, no assumption should be made about the order of returned objects within any category, even across service invocations. This is to avoid any extra processing load on servers that may reside in very simple devices, and to permit the best object packing when the retrieval of multiple objects requires multiple request/response transactions.

Requested-object ID
This parameter specifies the first requested object, or the single requested object, according to the Read Device ID code. A response cannot exceed the maximum size of the APDU for client/server (Unit ID + Function Code + Data = 254 octets), as described in IEC 61158-6-15. An individual object size is guaranteed to fit in the maximum size by definition. If the set of returned objects requires more octets than the maximum size, then several transactions (request/response) are needed. This is an application client responsibility. When many objects are requested, the ID of the first requested object of the first transaction must be set to 0x00. If the initial object ID is not 0x00, then in general the returned Device Identification will be incomplete. Subsequent requests within the same set of objects must set the Requested-object ID to the Next-object ID returned by the server in the previous response. If a different Requested-object ID is provided, then in general the returned Device Identification will be incomplete. When many objects are requested, if the Requested-object ID does not match any known object, the server will respond as if the object with ID 0x00 was requested, effectively restarting from the beginning if this happens in the middle of the retrieval. When the server supports the retrieval of a single object, and this is performed with a Requested-object ID that does not match any known object, the server will return an error response.

Parameter Type: Unsigned8

Allowed values: 0x00 to 0xFF.

6.1.10.3.2.2.3 Result(+)

This selection type parameter indicates that the service request succeeded.

Unit ID
In the absence of error, this parameter is identical to the correspondingly named request parameter.

MEI type
In the absence of error, this parameter is identical to the correspondingly named request parametert.

Read Device ID code
In the absence of error, this parameter is identical to the correspondingly named request parametert.

Conformity level
This parameter specifies information about the actual object categories and object retrieval access type made available by the server. Its value is provided by the server in all the responses, irrespective of the requested Read Device ID code. Values are illustrated in Table 24. In the Read Device ID parameter description it was explained what

is returned when dealing with objects unknown to the server. For known objects, if a Read Device ID code requests a category or a type of access that is not available on the server, then the returned objects are as from Table 25.

Parameter Type: Unsigned8

More-available flag
This parameter specifies information about having or not more objects to retrieve after a request/response. It is meaningful when the Read Device ID code is one of 0x01, 0x02, or 0x03, and the response exceed the maximum size of the APDU for client/server (Unit ID + Function Code + Data = 254 octets), as described in IEC 61158-6-15. A value of 0x00 means that there are no more objects available, while a value of 0xFF means that more requests have to be issued to retrieve the remaining objects. The meaning of this parameter is related to the one of the Next-object ID parameter.

Parameter Type: Unsigned8

Next-object ID
This parameter specifies the ID of the object that has to be requested in a subsequent request when the More-available flag is 0xFF. This is a client's responsibility, and if the Next-object ID requested in the subsequent request does not match any known object, the server will respond as if the object with ID 0x00 was requested, effectively restarting from the beginning.

Parameter Type: Unsigned8

Object array
This parameter is an array, it specifies requested objects. Each array element holds an object, as described below. This array may or may not contain all the requested objects at once, due to size limitations as illustrated in the Requested-object ID description.

Number of objects
This parameter is part of the Read Device Identification array parameter. It specifies the number of objects in the array.

Parameter Type: Unsigned8

Objects array element
This parameter is part of the Read Device Identification array parameter. Each element carries one object. Each element uniquely identifies and represents an object within the device identification address space using the Returned-object ID, the Object length and the Object value. All is described below.

Returned-object ID
This parameter is part of the Objects array element, and it uniquely identifies the object. Its description is the same as for the Requested-object ID, and it is part of the stream that initiated with the Requested-object ID.

Attribute Type: Unsigned8

Object length
This parameter is part of the Objects array element. It describes the length of the object value, in octets.

Parameter Type: Unsigned8

Object value
This parameter is part of the Objects array element and contributes to the device identification.

Parameter Type: As described in Table 22.

6.1.10.3.2.2.4 Result(-)

This selection type parameter indicates that the service request failed.

Unit ID
In the absence of error, this parameter is identical to the correspondingly named request parameter.

Parameter Type: Unsigned8

Error info
This parameter specifies error information.

Parameter Type: Unsigned8

6.1.10.3.2.3 Service procedure

This service is used to retrieve the device identification objects.

6.1.11 CANopen general reference ASE

6.1.11.1 Overview

The CANopen General Reference ASE is an encapsulation of the services that will be used to access (read from or write to) the entries of a CAN-Open Device Object Dictionary as well as to control and monitor the CANopen system.

The MEI Type 13 is the MEI Assigned Number licensed to CiA for the CANopen General Reference.

The system is intended to work within the size limitations of existing client/server networks.

6.1.11.2 CANopen general reference class specification

6.1.11.2.1 Formal model

The CANopen General Reference class is derived from the Encapsulated Interface class. The CANopen General Reference object is described by the following template:

ASE: **CANopen General Reference ASE**
CLASS: **CANopen General Reference**
CLASS ID: not used
PARENT CLASS: Encapsulated Interface
ATTRIBUTES:

1.	(m)	Key Attribute:	Implicit
2.	(m)	Attribute:	Unit ID
3.	(m)	Attribute:	MEI type

Please refer to “CIA Draft Standard Proposal 309, Interfacing CANopen with TCP/IP, Part 2: Modbus/TCP mapping” for the MEI type 13 Encapsulated Interface derived class specific attributes.

SERVICES:

Please refer to “CIA Draft Standard Proposal 309, Interfacing CANopen with TCP/IP, Part 2: Modbus/TCP mapping” for the MEI type 13 Encapsulated Interface derived class specific services.

6.1.11.2.2 Attributes

Implicit
The attribute Implicit indicates that the Read Device Identification object is implicitly addressed by the services.

Unit ID
This attribute specifies the address of the server.

Attribute Type: Unsigned8

Allowed values: 1 to 247.

MEI type
This attribute specifies the Encapsulated Interface type.

Attribute Type: Unsigned8

Allowed values: 13

Please refer to “CIA Draft Standard Proposal 309, Interfacing CANopen with TCP/IP, Part 2: Modbus/TCP mapping” for the MEI type 13 Encapsulated Interface derived class specific attributes.

6.1.11.2.3 Services

Please refer to “CIA Draft Standard Proposal 309, Interfacing CANopen with TCP/IP, Part 2: Modbus/TCP mapping” for the MEI type 13 Encapsulated Interface derived class specific services.

6.2 ARs

6.2.1 Overview

The client/server application layer uses a user-triggered uni-directional AREP for unconfirmed services like broadcast, and a user-triggered bi-directional AREP for confirmed services.

The client/server application layer itself has no particular requirements regarding connection-oriented or connection-less AREPs. It could use, for example, the following AREPs:

Queued User-triggered Uni-directional (QUU) AR Endpoint, for broadcast services;

Queued User-triggered Bi-directional Connection-Oriented (QUB-Co) AR Endpoint for confirmed communication services, or

Queue User-triggered Bi-directional Connection-less (QUB-Cl) AR Endpoint for confirmed communication services

The queue depth depends on the configuration of the Context Management ASE.

In the next sections QUU and QUB-Co will be detailed, as one of many valid possibilities.

6.2.2 Queued user-triggered unidirectional AREP class specification

6.2.2.1 Class overview

This class is defined to support the on-demand queued distribution of unconfirmed services to one or more application processes. The behavior of this type of AR can be described as follows.

An AR ASE user wishing to convey a request APDU submits an AR ASE Service Data Unit to the sending endpoint of the AR. The AREP sending the request APDU submits it to its underlying layer for transfer. The underlying layer sends it at its next opportunity. The AREP receiving the request APDU from its underlying layer delivers it to the AR ASE user in the order that it was received.

The following summarizes the characteristics of this AREP class.

Roles: CLIENT
SERVER
Cardinality: 1-to-n
Timeliness: No

6.2.2.2 Formal model

ASE: **AR ASE**
CLASS: **Queued User-triggered Uni-directional (QUU) AREP**
CLASS ID: not used
PARENT CLASS: AR Endpoint
ATTRIBUTES:
1. (m) Attribute: Role (Client, Server)
SERVICES:
1. (m) OpsService: Unconfirmed Send

6.2.2.3 Attributes

Role
This attribute specifies possible roles of this end point. The possible values are Client and Server:

Client AREPs of this type convey their data by issuing a Data Transfer Unconfirmed PDU.

Server AREPs of this type receive data transmitted by a Client source AREP.

6.2.2.4 Services

Unconfirmed send
This service is used to send an unconfirmed service on the specified AR.

6.2.3 Queued user-triggered bidirectional connection-oriented AREP class specification

6.2.3.1 Class overview

This class is defined to support the on-demand exchange of confirmed services between two application processes. Unconfirmed services are not supported by this type of AR. It uses connection-oriented data link services for the exchanges. The behavior of these classes is described as follows.

An AR ASE user wishing to convey a request APDU submits it as an AR ASE Service Data Unit to its AREP. The AREP sending the request APDU queues it to its underlying layer for transfer at the next available opportunity.

The AREP receiving the request APDU from its underlying layer, queues it for delivery to its AR ASE user in the order that it was received.

For a confirmed service request the AREP receiving the request APDU accepts the corresponding response APDU from its AR ASE user and queues it to the underlying layer for transfer.

The AREP that issued the request APDU receives the response APDU from its underlying layer and queues it for delivery to its AR ASE user in the order that it was received. It also stops its associated service response timer.

The following summarizes the characteristics of this AREP class.

Roles: Client
Server

Cardinality: 1-to-1

Timeliness: No

6.2.3.2 Formal model

ASE: **AR ASE**
CLASS: **Queued User-triggered Bi-directional Connection=oriented (QUB-Co) AREP**
CLASS ID: not used
PARENT CLASS: AR Endpoint
ATTRIBUTES:
1. (m) Attribute: Role (Client, Server)
SERVICES:
1. (m) OpsService: Confirmed Send

6.2.3.3 Attributes

Role
This attribute specifies possible role of this end point. The possible values are Client and Server:

Client AREPs of this type issue confirmed and unconfirmed service Request-APDUs to servers and receive confirmed service Response-APDUs.

Server AREPs of this type receive confirmed and unconfirmed service Request-APDUs from clients and issue confirmed service Response-APDUs to them.

6.2.3.4 Services

Confirmed send
This service is used to send a confirmed service on the specified AR.

6.3 Summary of FAL classes

Table 27 of this subclause contains a summary of the client/server defined FAL Classes. The Class ID values have been assigned to be compatible with existing standards.

Table 27 – FAL class summary

FAL ASE	Class	Class ID
Context Management	Filter	—
	Transaction	—
Discretes	Discretes	—
Coils	Coils	—
Input Registers	Input Registers	—
Holding Registers	Holding Registers	—
File	File	—
Encapsulated Interface	Encapsulated Interface	—
Read Device Identification	Read Device Identification	—
CANopen General Reference	CANopen General Reference	—
Application Relationship	AREP	32
	QUB-Co	34
	QUU	36
	QUB-CI	45

6.4 Permitted FAL services by AREP role

Table 28 below defines the valid combinations of services and AREP roles (which service APDUs and AREP with the specified role can send or receive) for client/server. The Unc and Cnf columns indicate whether the service listed in the left-hand column is unconfirmed (Unc) or confirmed (Cnf).

Table 28 – Services by AREP role

FAL Services	Unc	Cnf	Client		Server	
			req	rcv	req	rcv
Context Mgmt ASE						
Filter		X	X			X
Transaction		X	X	X		
Discretes ASE						
Read Discretes		X	X			X
Coils ASE						
Read Coils		X	X			X
Write Single Coil		X	X			X
Write Multiple Coils		X	X			X
Broadcast Write Single Coil	X		X			X
Broadcast Write Multiple Coils	X		X			X
Input Registers ASE						
Read Input Registers		X	X			X
Holding Registers ASE						
Read Holding Registers		X	X			X
Write Single Holding Register		X	X			X
Write Multiple Holding Registers		X	X			X
Mask Write Holding Register		X	X			X
Read/Write Holding Registers		X	X			X
Read FIFO		X	X			X
Broadcast Write Single Holding Register	X		X			X
Broadcast Write Multiple Holding Registers	X		X			X
File ASE						
Read File Record		X	X			X
Write File Record		X	X			X
Encapsulated Interface ASE						
Read Device Identification ASE						
Read Device Identification		X	X			X
CANopen General Reference ASE						
Please see reference in description.						
AR ASE						
AR-Confirmed Send	X		X			X
AR-Unconfirmed Send	X		X			X

7 Publish/subscribe communication model specification

7.1 ASEs

7.1.1 General

The class hierarchy for publish/subscribe is illustrated using UML notation in Figure 22.

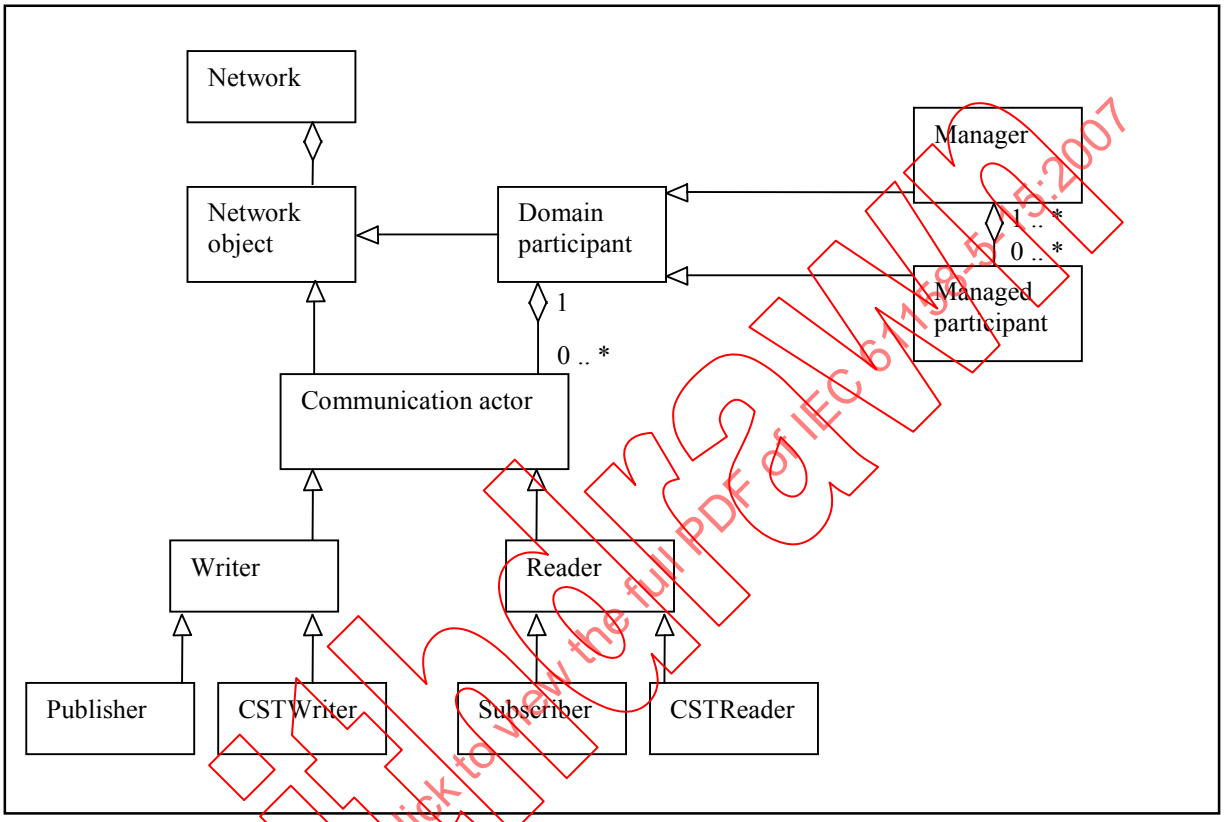


Figure 22 – Publish/subscribe class derivations and relationships

FAL ASEs are defined using a modular approach. The ASEs defined for the FAL are also object-oriented. In general, ASEs provide a set of services designed for one specific object class or for a related set of classes.

To support remote access to the AP, the Application Relationship ASE is defined. It provides services to the AP for defining and establishing communication relationships with other APs, and it provides services to the other ASEs for conveying their service requests and responses.

Each FAL ASE defines a set of services, APDUs, and procedures that operate on the classes that it represents.

Profiles may be used to define subsets. Definition of profiles is beyond the scope of this standard.

APDUs are sent and received between FAL ASEs that support the same services.

Considering the context and the attributes of the publish/subscribe classes, and the usage of ASEs to provide services, the architecture can be re-cast as in Figure 23, which illustrates the publish/subscribe FAL ASEs, its classes, and their architectural relationships.

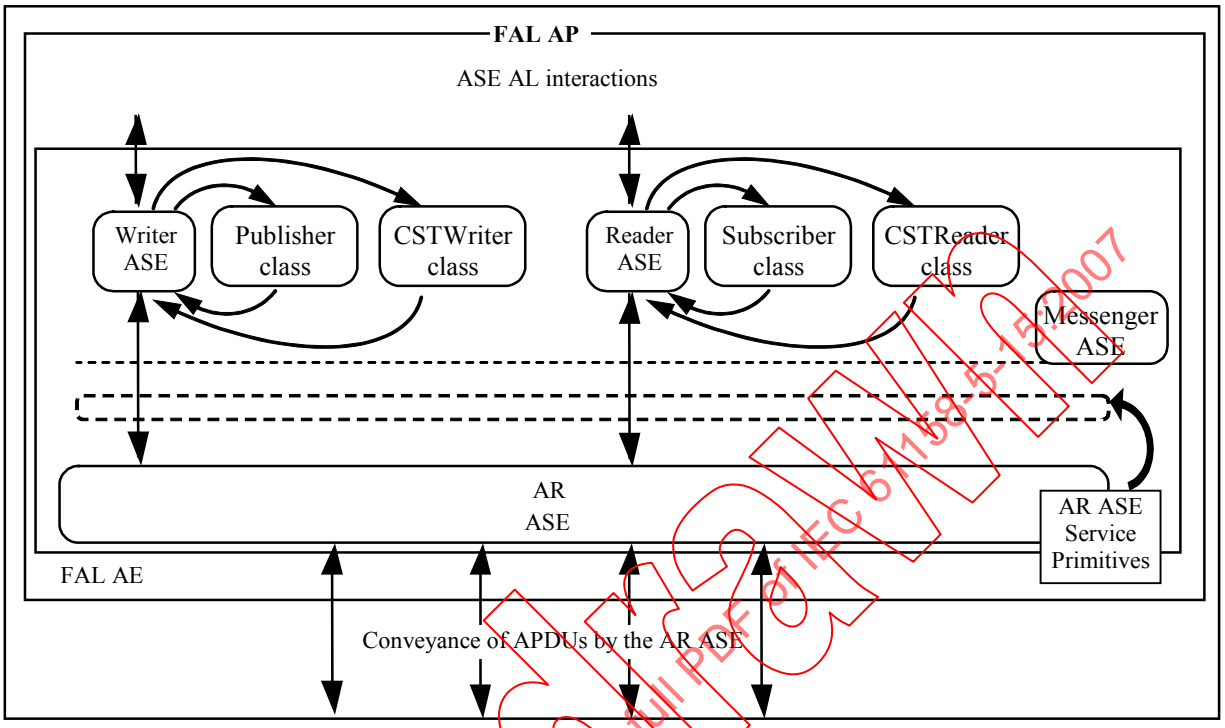


Figure 23 – FAL ASEs and classes

7.1.2 Writer ASE

7.1.2.1 Overview

The Writer services allow the distribution of data and metadata.

7.1.2.2 Writer class specifications

7.1.2.2.1 Publisher class specification

7.1.2.2.1.1 Publisher formal model

The Publisher object is described by the following template:

ASE: **Writer ASE**
CLASS: **Publisher**
CLASS ID: not used
PARENT CLASS: TOP
ATTRIBUTES:
1. (m) Key Attribute: Implicit
SERVICES:
1. (m) OpsService: Issue
2. (m) OpsService: Heartbeat

7.1.2.2.1.2 Attributes

Implicit
The attribute Implicit indicates that the Publisher object is implicitly addressed by the services.

7.1.2.2.1.3 Services

Issue
This service is used by a publisher to publish user data for one or more subscribers.

Heartbeat
This service is used to probe a reader's presence, and to inform one or more readers about a writer's available information.

7.1.2.2.2 CSTWriter class specification

7.1.2.2.2.1 CSTWriter formal model

The CSTWriter object is described by the following template:

ASE:			**Writer ASE**
CLASS:			**CSTWriter**
CLASS ID:			not used
PARENT CLASS:			TOP
ATTRIBUTES:			
1.	(m)	Key Attribute:	Implicit
SERVICES:			
1.	(m)	OpsService:	VAR
2.	(m)	OpsService:	GAP
3.	(m)	OpsService:	Heartbeat

7.1.2.2.2.2 Attributes

Implicit
The attribute Implicit indicates that the CSTWriter object is implicitly addressed by the services.

7.1.2.2.2.3 Services

VAR
This service is used by CSTWriters to publish metadata for CSTReaders, in this case information about the attributes of a network object. This information is part of a composite state.

GAP
This service is used to inform readers about the irrelevance of some data.

Heartbeat
This service is used to probe a reader's presence, and to inform one or more readers about a writer's available information via sequence numbers.

7.1.2.3 Writer ASE service specification

7.1.2.3.1 Supported services

The Writer ASE defines the services:

Issue

Heartbeat

VAR

GAP

7.1.2.3.2 Issue service

7.1.2.3.2.1 Service overview

The Issue service is used by a publisher to publish user data for one or more subscribers.

7.1.2.3.2.2 Service primitives

The service parameters for this service are shown in Table 29. It is an unconfirmed service.

Table 29 – Issue service parameters

Parameter name	Req	Ind
Argument	M	M(=)
hasParameterSequence	M	M(=)
endian-ness	M	M(=)
readerObjectID	M	M(=)
writerObjectID	M	M(=)
issueSeqNumber	M	M(=)
parameterSequence	C	C(=)
issueData	M	M(=)

Argument
The argument conveys the service specific parameters of the service request.

hasParameterSequence
This parameter specifies if there is or not a parameterSequence parameter.

Parameter Type: Boolean.

endian-ness
This parameter specifies the endian-ness of this service request and indication.

Parameter Type: Boolean.

Values: {big-endian, FALSE, 0}, {little-endian, TRUE, NOT 0}

readerObjectID
This parameter, combined with modal/state values provided by the Messenger service, specifies the subscription GUID <destHostID, destAppID, Issue.readerObjectID> for which the Issue service is meant. The Issue.readerObjectID can be OBJECTID_UNKNOWN, in which case the Issue service applies to all Subscriptions within the application <destHostId, destAppId>.

Parameter Type: ObjectID.

writerObjectID
This parameter, combined with modal/state values provided by the Messenger service, specifies the Publication GUID <sourceHostID, sourceAppID, Issue.writerObjectID> that originated the Issue.

Parameter Type: ObjectID.

issueSeqNumber
This parameter specifies the sequence number identifying the Issue.

Parameter Type: SequenceNumber.

parameterSequence
This parameter is conditional to the value of the hasParameterSequence flag; it is used to provide a variable list of operational Issue parameters, and allows for publish/subscribe extensions.

Parameter Type: ParameterSequence.

issueData
This parameter specifies the actual AI user data in this Issue.

Parameter Type: exchanged by other means.

7.1.2.3.2.3 Service procedure

This service is used by a publisher to publish user data for one or more subscribers.

7.1.2.3.3 Heartbeat service

7.1.2.3.3.1 Service overview

The Heartbeat service is used to probe a reader's presence, and to inform one or more readers about a writer's available information via sequence numbers.

7.1.2.3.3.2 Service primitives

The service parameters for this service are shown in Table 30. It is an unconfirmed service.

Table 30 – Heartbeat service parameters

Parameter name	Req	Ind
Argument	M	M(=)
IsResponseRequired	M	M(=)
endian-ness	M	M(=)
readerObjectID	M	M(=)
writerObjectID	M	M(=)
firstSeqNumber	M	M(=)
lastSeqNumber	M	M(=)

Argument
The argument conveys the service specific parameters of the service request.

isResponseRequired
This parameter specifies if the application sending the Heartbeat requires or not a response.

Parameter Type: Boolean.

endian-ness
This parameter specifies the endian-ness of this service request and indication.

Parameter Type: Boolean.

Values: {big-endian, FALSE, 0}, {little-endian, TRUE, NOT 0}

readerObjectID
This parameter, combined with modal/state values provided by the Messenger service, specifies the reader GUID <destHostID, destAppID, Heartbeat.readerObjectID> for which

the Heartbeat service is meant. The Heartbeat.readerObjectID can be OBJECTID_UNKNOWN, in which case the Heartbeat service applies to all readers within the application <destHostId, destAppId>.

Parameter Type: ObjectID.

writerObjectID
This parameter, combined with modal/state values provided by the Messenger service, specifies the writer GUID <sourceHostID, sourceAppID, Heartbeat.writerObjectID> that originated the Heartbeat.

Parameter Type: ObjectID.

firstSeqNumber
This parameter specifies the first sequence number that is still available and meaningful in the writer with writerObjectID. This field must be greater than or equal to zero. If it is equal to SEQUENCE_NUMBER_NONE, the writer has no data available.

Parameter Type: SequenceNumber.

lastSeqNumber
This parameter specifies the last sequence number that is available and in the writer with writerObjectID. This field must be greater than or equal to firstSeqNumber. If firstSeqNumber is equal to SEQUENCE_NUMBER_NONE, then lastSeqNumber must also be SEQUENCE_NUMBER_NONE.

Parameter Type: SequenceNumber.

7.1.2.3.3.3 Service procedure

This service is used by to probe a reader's presence, and to inform one or more readers about a writer's available information via sequence numbers.

7.1.2.3.4 VAR service

7.1.2.3.4.1 Service overview

The VAR service is used by CSTWriters to publish metadata for CSTReaders, in this case information about the attributes of a network object. This information is part of a composite state.

7.1.2.3.4.2 Service primitives

The service parameters for this service are shown in Table 31. It is an unconfirmed service.

Table 31 – VAR service parameters

Parameter name	Req	Ind
Argument	M	M(=)
hasHostIDandAppID	M	M(=)
alive	M	M(=)
hasParameterSequence	M	M(=)
endian-ness	M	M(=)
readerObjectID	M	M(=)
writerObjectID	M	M(=)
hostID	C	C(=)
appID	C	C(=)
objectID	M	M(=)
writerSeqNumber	M	M(=)
parameterSequence	C	C(=)

Argument
The argument conveys the service specific parameters of the service request.

hasHostIDandAppID
This parameter specifies if there are or not the hostID and the appID parameters.

Parameter Type: Boolean.

alive
Indicates to the reader whether the data-object is alive or else is not-alive (disposed).

Parameter Type: Boolean.

Values: {FALSE, 0}, {TRUE, NOT 0}

hasParameterSequence
This parameter specifies if there is or not a parameterSequence parameter.

Parameter Type: Boolean.

endian-ness
This parameter specifies the endian-ness of this service request and indication.

Parameter Type: Boolean.

Values: {big-endian, FALSE, 0}, {little-endian, TRUE, NOT 0}

readerObjectID
This parameter, combined with modal/state values provided by the Messenger service, specifies the reader GUID <destHostID, destAppID, VAR.readerObjectID> for which the VAR service is meant. The VAR.readerObjectID can be OBJECTID_UNKNOWN, in which case the VAR service applies to all readers of the writerObjectID within the application <destHostId, destAppId>.

Parameter Type: ObjectID.

writerObjectID
This parameter, combined with modal/state values provided by the Messenger service, specifies the CSTWriter GUID <sourceHostID, sourceAppID, VAR.writerObjectID> that originated the VAR.

Parameter Type: ObjectID.

hostID
This parameter is conditional to the value of the hasHostIDandAppID flag; when present, combined with the appID and objectID parameters, specifies the GUID of the object the information carried by this VAR is about.

Parameter Type: HostID.

appID
This parameter is conditional to the value of the hasHostIDandAppID flag; when present, combined with the hostID and objectID parameters, specifies the GUID of the object the information carried by this VAR is about.

Parameter Type: AppID.

objectID
This parameter specifies the GUID of the object the information carried by this VAR is about; if the parameters hostID and appID are present then the GUID is <VAR.hostID, VAR.appID, VAR.objectID> otherwise, combined with the modal/state values provided by the Messenger service, the GUID is <sourceHostID, sourceAppID, VAR.objectID>.

Parameter Type: ObjectID.

writerSeqNumber
This parameter is used to tag changes in the composite state provided by the CSTWriter; it is incremented each time a change in such composite state occurs; this should be a strictly positive number (1, 2, ...), or the special sequence number, SEQUENCE_NUMBER_UNKNOWN may be sent to indicate that the sender does not keep track of the sequence number.

Parameter Type: SequenceNumber.

parameterSequence
This parameter is conditional to the value of the hasParameterSequence flag; it is used to provide a variable list of operational Issue parameters, and allows for publish/subscribe extensions.

Parameter Type: ParameterSequence.

7.1.2.3.4.3 Service procedure

The VAR service is used by CSTWriters to publish metadata for CSTReaders, in this case information about the attributes of a network object. This information is part of a composite state.

7.1.2.3.5 GAP service

7.1.2.3.5.1 Service overview

The GAP service is sent from a CSTWriter to a CSTReader to indicate that a range of sequence numbers is no longer relevant. The set may be a contiguous range of sequence numbers or a specific set of sequence numbers.

7.1.2.3.5.2 Service primitives

The service parameters for this service are shown in Table 32. It is an unconfirmed service.

Table 32 – VAR service parameters

Parameter name	Req	Ind
Argument	M	M(=)
endian-ness	M	M(=)
readerObjectID	M	M(=)
writerObjectID	M	M(=)
firstSeqNumber	M	M(=)
bitmap	M	M(=)

Argument
The argument conveys the service specific parameters of the service request.

endian-ness
This parameter specifies the endian-ness of this service request and indication.

Parameter Type: Boolean.

Values: {big-endian, FALSE, 0}, {little-endian, TRUE, NOT 0}

readerObjectID
This parameter, combined with modal/state values provided by the Messenger service, specifies the reader GUID <destHostID, destAppID, GAP.readerObjectID> for which the GAP service is meant. The GAP.readerObjectID can be OBJECTID_UNKNOWN, in which case the GAP service applies to all readers within the application <destHostId, destAppId>.

Parameter Type: ObjectID.

writerObjectID
This parameter, combined with modal/state values provided by the Messenger service, specifies the CSTWriter GUID <sourceHostID, sourceAppID, GAP.writerObjectID> that is the subject of the GAP sequence numbers of this GAP service invocation.

Parameter Type: ObjectID.

firstSeqNumber
This parameter is used with the *bitmap* parameter to specify the sequence numbers that are no longer available in the writerObjectID network object; the list of the no longer available sequence numbers is the union of

— all the sequence numbers in the range from *GAP.firstSeqNumber* up to *GAP.bitmap.bitmapBase* – 1; this list is empty if the *firstSeqNumber* is greater than or equal to the *bitmapBase* of the *bitmap*; *GAP.firstSeqNumber* should always be greater than or equal to 1, and

— the sequence numbers that have the corresponding bit in the *bitmap* set to 1.

Parameter Type: SequenceNumber.

bitmap
This parameter is used with the *firstSeqNumber* parameter to specify the sequence numbers that are no longer available in the writerObjectID network object, as detailed in the *firstSeqNumber* parameter description above.

Parameter Type: Bitmap.

7.1.2.3.5.3 Service procedure

The GAP service is sent from a CSTWriter to a CSTReader to indicate that a range of sequence numbers is no longer relevant. The set may be a contiguous range of sequence numbers or a specific set of sequence numbers.

7.1.3 Reader ASE

7.1.3.1 Overview

The Reader services allow the acquisition and maintenance of distributed data and metadata.

7.1.3.2 Reader class specifications

7.1.3.2.1 Subscriber class specification

7.1.3.2.1.1 Subscriber formal model

The Subscriber object is described by the following template:

ASE:			**Reader ASE**
CLASS:			**Subscriber**
CLASS ID:			not used
PARENT CLASS:			TOP
ATTRIBUTES:			
1.	(m)	Key Attribute:	Implicit
SERVICES:			
1.	(m)	OpsService:	ACK

7.1.3.2.1.2 Attributes

Implicit
The attribute Implicit indicates that the Subscriber object is implicitly addressed by the services.

7.1.3.2.1.3 Services

ACK
This service is used to communicate the state of a Reader to a Writer.

7.1.3.2.2 CSTReader class specification

7.1.3.2.2.1 CSTReader formal model

The CSTReader object is described by the following template:

ASE:			**Reader ASE**
CLASS:			**CSTReader**
CLASS ID:			not used
PARENT CLASS:			TOP
ATTRIBUTES:			
1.	(m)	Key Attribute:	Implicit
SERVICES:			
1.	(m)	OpsService:	ACK

7.1.3.2.2.2 Attributes

Implicit
The attribute Implicit indicates that the CSTReader object is implicitly addressed by the services.

7.1.3.2.2.3 Services

ACK
This service is used to communicate the state of a Reader to a Writer.

7.1.3.3 Reader ASE service specification

7.1.3.3.1 Supported services

The Writer ASE defines the services:

ACK

7.1.3.3.2 ACK service

7.1.3.3.2.1 Service overview

This service is used to communicate the state of a Reader to a Writer.

7.1.3.3.2.2 Service primitives

The service parameters for this service are shown in Table 33. It is an unconfirmed service.

Table 33 – ACK service parameters

Parameter name	Req	Ind
Argument	M	M(=)
IsResponseRequired	M	M(=)
endian-ness	M	M(=)
readerObjectID	M	M(=)
writerObjectID	M	M(=)
bitmap	M	M(=)

Argument
The argument conveys the service specific parameters of the service request.

isResponseRequired
This parameter is used to specify if the application sending the Heartbeat requires or not a response.

Parameter Type: Boolean.

endian-ness
This parameter is used to specify the endian-ness of this service request and indication.

Parameter Type: Boolean.

Values: {big-endian, FALSE, 0}, {little-endian, TRUE, NOT 0}

readerObjectID
This parameter, combined with modal/state values provided by the Messenger service, specifies the GUID <sourceHostID, sourceAppID, ACK.readerObjectID> of the reader that acknowledges receipt of certain sequence numbers and/or requests to receive certain sequence numbers.

Parameter Type: ObjectID.

writerObjectID
This parameter, combined with modal/state values provided by the Messenger service, is used to specify the GUID <destHostID, deatAppID, ACK.writerObjectID> of the writer that

the reader has received these sequence numbers from and/or wants to receive these sequence numbers from.

Parameter Type: ObjectID.

bitmap
This parameter specifies the ACK botmap: a “0” in this bitmap means that the corresponding sequence-number is missing; a “1” in the bitmap conveys no information, that is, the corresponding sequence number may or may not be missing; by sending an ACK, the readerGUID object acknowledges receipt of all messages up to and including the sequence number (bitmap.bitmapBase -1).

Parameter Type: Bitmap.

7.1.3.3.2.3 Service procedure

This service is used to communicate the state of a Reader to a Writer.

7.1.4 Messenger ASE

7.1.4.1 Overview

The Messenger services allow the composition of a publish/subscribe message, and are used by the reader and writer ASEs. This is illustrated using UML notation in Figure 24.

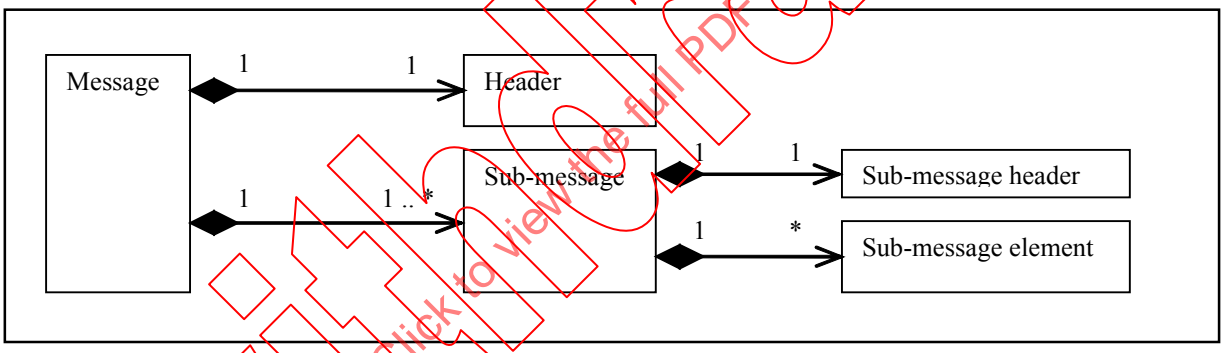


Figure 24 – Publish/subscribe service request composition

7.1.4.2 Messenger class specifications

7.1.4.2.1 Messenger class specification

7.1.4.2.1.1 Messenger formal model

The Messenger object is described by the following template:

ASE:			**Messenger ASE**
CLASS:			**Messenger**
CLASS ID:			not used
PARENT CLASS:			TOP
ATTRIBUTES:			
1.	(m)	Key Attribute:	Implicit
SERVICES:			
1.	(m)	OpsService:	Header
2.	(m)	OpsService:	INFO_DST
3.	(m)	OpsService:	INFO_REPLY
4.	(m)	OpsService:	INFO_SRC
5.	(m)	OpsService:	INFO_TS
6.	(m)	OpsService:	PAD

7.1.4.2.1.2 Attributes

Implicit
The attribute Implicit indicates that the Subscriber object is implicitly addressed by the services.

7.1.4.2.1.3 Services

Header
This service is used to communicate the state of a Reader to a Writer.

INFO_DST
This service is used to communicate the state of a Reader to a Writer.

INFO_REPLY
This service is used to communicate the state of a Reader to a Writer.

INFO_SRC
This service is used to communicate the state of a Reader to a Writer.

INFO_TS
This service is used to communicate the state of a Reader to a Writer.

PAD
This service is used to communicate the state of a Reader to a Writer.

7.1.4.3 Messenger ASE service specification

7.1.4.3.1 Supported services

The Messenger ASE defines the services:

Header
INFO_DST
INFO_REPLY
INFO_SRC
INFO_TS
PAD

7.1.4.3.2 Header service

7.1.4.3.2.1 Service overview

This service is used to assemble the header of a message that may eventually contain several requests submitted by other ASE services.

The fixed size header is used to identify the message as belonging to publish/subscribe, to identify the version of publish/subscribe used, and to provide context information that applies to the requests contained within the message as sub-messages.

7.1.4.3.2.2 Service primitives

The service parameters for this service are shown in Table 34. It is an unconfirmed service.

Table 34 – Header service parameters

Parameter name	Req	Ind
Argument	M	M(=)
protocolMarker	M	M(=)
version	M	M(=)
vendorID	M	M(=)
hostID	M	M(=)
appID	M	M(=)

Argument
The argument conveys the service specific parameters of the service request.

protocolMarker
This parameter specifies the 4 octets marker: 'R', 'T', 'P', 'S'.

Parameter Type: OctetString.

version
This parameter specifies the protocol version.

Parameter Type: ProtocolVersion.

Values: Implementations following this version of the specification implement protocol version 1.0 (major = 1, minor = 0).

vendorID
This parameter specifies the vendor of the middleware implementing the publish/subscribe protocol and allows this vendor to add specific extensions to the protocol. The vendorID does not refer to the vendor of the device or product that specifies publish/subscribe middleware.

Parameter Type: VendorID.

hostID
This parameter specifies sourceHostID for the receiver and for the reader and writer services that use sourceHostID; the value setting is modal, as specified until changed.

Parameter Type: HostID.

appID
This parameterspecifies sourceAppID for the receiver and for the reader and writer services that use sourceAppID; the value setting is modal, as specified until changed.

Parameter Type: AppID.

7.1.4.3.2.3 Service procedure

This service is used to assemble the header of a message that may eventually contain several requests submitted by other ASE services.

It has the effect of causing the following state changes for the receiver and for the reader and writer services that use the attributes:

attribute	value
sourceHostID	Header.hostID
sourceAppID	Header.appID
sourceVersion	Header.version
sourceVendorID	Header.vendorID
haveTimestamp	false

7.1.4.3.3 INFO_DST service

7.1.4.3.3.1 Service overview

This service modifies the logical destination for the service requests that follow it.

7.1.4.3.3.2 Service primitives

The service parameters for this service are shown in Table 36. It is an unconfirmed service.

Table 35 – INFO_DST service parameters

Parameter name	Req	Ind
Argument	M	M(=)
endian-ness	M	M(=)
hostID	M	M(=)
appID	M	M(=)

Argument
The argument conveys the service specific parameters of the service request.

endian-ness
This parameter specifies the endian-ness of this service request and indication.

Parameter Type: Boolean.

Values: {big-endian, FALSE, 0}, {little-endian, TRUE, NOT 0}

hostID
This parameter specifies destHostID for the reader and writer services that use destHostID; the value setting is modal, as specified until changed:

```
if (INFO_DST.hostID != HOSTID_UNKNOWN)
{destHostID = INFO_DST.hostID}
else
{destHostID = <hostID of receiving application>}
```

An INFO_DST with a HOSTID_UNKNOWN means that any host may interpret the requests following this one within the same message as if they were meant for it.

Parameter Type: HostID.

appID
This parameter specifies destAppID for the reader and writer services that use destAppID; the value setting is modal, as specified until changed:

```
if (INFO_DST.appID != APPID_UNKNOWN)
{destAppID = INFO_DST.appID}
else
{destAppID = <appID of receiving application>}
```

An INFO_DST with a APPID_UNKNOWN means that any application may interpret the requests following this one within the same message as if they were meant for it.

Parameter Type: AppID.

7.1.4.3.3.3 Service procedure

This service is used to modify the logical destination for the service requests that follow it.

7.1.4.3.4 INFO_REPLY service

7.1.4.3.4.1 Service overview

This service specify explicit information on where to send a reply to the requests that follow it within the same message.

7.1.4.3.4.2 Service primitives

The service parameters for this service are shown in Table 36. It is an unconfirmed service.

Table 36 – INFO_REPLY service parameters

Parameter name	Req	Ind
Argument	M	M(=)
hasMulticast	M	M(=)
endian-ness	M	M(=)
unicastReplyIPAddress	M	M(=)
unicastReplyPort	M	M(=)
multicastReplyIPAddress	C	C(=)
multicastReplyPort	C	C(=)

Argument
The argument conveys the service specific parameters of the service request.

hasMulticast
This parameter specifies the presence of multicast information.

Parameter Type: Boolean.

Values: {FALSE, 0}, {TRUE, NOT 0}

endian-ness
This parameter specifies the endian-ness of this service request and indication.

Parameter Type: Boolean.

Values: {big-endian, FALSE, 0}, {little-endian, TRUE, NOT 0}

unicastReplyIPAddress
This parameter specifies the IP address of where to send a reply to the requests that follow within the same message; the value setting is modal, as specified until changed:

```
If
(INFO_REPLY.unicastReplyIPAddress != IPADDRESS_INVALID)
{unicastReplyIPAddress
= INFO_REPLY.unicastReplyIPAddress}
```

Parameter Type: IPAddress.

unicastReplyPort
This parameter specifies the port of where to send a reply to the requests that follow within the same message; the value setting is modal, as specified until changed:

```
unicastReplyPort = INFO_REPLY.replyPort
```

Parameter Type: Port.

multicastReplyIPAddress
This parameter is conditional to the value of the hasMulticast parameter. It specifies the IP address of where to send a reply to the requests that follow within the same message; the value setting is modal, as specified until changed.

If this parameter is present, then:

```
multicastReplyIPAddress
= INFO_REPLY.multicastReplyIPAddress
```

If this parameter is not present, then:

```
multicastReplyIPAddress
= IPADDRESS_INVALID
```

Parameter Type: IPAddress.

multicastReplyPort
This parameter is conditional to the value of the hasMulticast parameter. It specifies the port of where to send a reply to the requests that follow within the same message; the value setting is modal, as specified until changed:

If this parameter is present, then:

```
multicastReplyPort
= INFO_REPLY.multicastReplyPort
```

If this parameter is not present, then:

```
multicastReplyPort
= PORT_INVALID
```

Parameter Type: Port.

7.1.4.3.4.3 Service procedure

This service is used to modify the logical destination for the service requests that follow it.

7.1.4.3.5 INFO_SRC service

7.1.4.3.5.1 Service overview

This service modifies the logical source for the service requests that follow it.

7.1.4.3.5.2 Service primitives

The service parameters for this service are shown in Table 37. It is an unconfirmed service.

Table 37 – INFO_SRC service parameters

Parameter name	Req	Ind
Argument	M	M(=)
endian-ness	M	M(=)
applPAddress	M	M(=)
version	M	M(=)
vendorID	M	M(=)
hostID	M	M(=)
appID	M	M(=)

Argument
The argument conveys the service specific parameters of the service request.

endian-ness
This parameter specifies the endian-ness of this service request and indication.

Parameter Type: Boolean.

Values: {big-endian, FALSE, 0}, {little-endian, TRUE, NOT 0}

applPAddress
This parameter specifies the IP address of where to send a reply to the requests that follow within the same message; the value setting is modal, as specified until changed.

Parameter Type: IPAddress.

version
This parameter specifies the protocol version.

Parameter Type: ProtocolVersion.

Values: Implementations following this version of the specification implement protocol version 1.0 (major = 1, minor = 0).

vendorID
This parameter specifies the vendor of the middleware implementing the publish/subscribe protocol and allows this vendor to add specific extensions to the protocol. The vendorID does not refer to the vendor of the device or product that specifies publish/subscribe middleware.

Parameter Type: VendorID.

hostID
This parameter specifies sourceHostID for the receiver and for the reader and writer services that use sourceHostID; the value setting is modal, as specified until changed.

Parameter Type: HostID.

appID
This parameter specifies sourceAppID for the receiver and for the reader and writer services that use sourceAppID; the value setting is modal, as specified until changed.

Parameter Type: AppID.

7.1.4.3.5.3 Service procedure

This service modifies the logical source for the service requests that follow it.

It has the effect of causing the following state changes for the receiver and for the reader and writer services that use the attributes:

attribute	value
sourceHostID	INFO_SRC.hostID
sourceAppID	INFO_SRC.appID
sourceVersion	INFO_SRC.version
sourceVendorID	INFO_SRC.vendorID
unicastReplyIPAddress	INFO_SRC.appIPAddress
unicastReplyPort	PORT_INVALID
multicastReplyIPAddress	IPADDRESS_INVALID
multicastReplyPort	PORT_INVALID
haveTimestamp	false

7.1.4.3.6 INFO_TS service

7.1.4.3.6.1 Service overview

This service is used to send a timestamp which applies to the service requests that follow it.

7.1.4.3.6.2 Service primitives

The service parameters for this service are shown in Table 38. It is an unconfirmed service.

Table 38 – INFO_TS service parameters

Parameter name	Req	Ind
Argument	M	M(=)
hasTimestamp	M	M(=)
endian-ness	M	M(=)
timestamp	C	C(=)

Argument
The argument conveys the service specific parameters of the service request.

hasTimestamp
This parameter specifies if the service request has a timestamp parameter.

Parameter Type: Boolean.

endian-ness
This parameter specifies the endian-ness of this service request and indication.

Parameter Type: Boolean.

Values: {big-endian, FALSE, 0}, {little-endian, TRUE, NOT 0}

timestamp
This parameter is conditional to the value of the hasTimestamp parameter. It specifies the timestamp for the requests that follow within the same message; the value setting is modal, as specified until changed:

If this parameter is present, then:

```
haveTimestamp = true
timestamp = INFO_TS.timestamp
```

If this parameter is not present, then:

```
haveTimestamp = false
```

Parameter Type: NtpTime.

7.1.4.3.6.3 Service procedure

This service is used to send a timestamp which applies to the service requests that follow it.

7.1.4.3.7 PAD service

7.1.4.3.7.1 Service overview

This service is used for alignment purposes within the message, whereas the receiver will skip the PAD service request length and will get to the next request. The length is specified as part of the requests formats. The service has no other meaning.

7.1.4.3.7.2 Service primitives

The service parameters for this service are shown in Table 39. It is an unconfirmed service.

Table 39 – PAD service parameters

Parameter name	Req	Ind
Argument	M	M(=)
endian-ness	M	M(=)

Argument
The argument conveys the service specific parameters of the service request.

endian-ness
This parameter specifies the endian-ness of this service request and indication.

Parameter Type: Boolean.

Values: {big-endian, FALSE, 0}, {little-endian, TRUE, NOT 0}

7.1.4.3.7.3 Service procedure

This service is used for alignment purposes within the message, whereas the receiver will skip the PAD service request length and will get to the next request. The length is specified as part of the requests formats. The service has no other meaning.

7.2 ARs

7.2.1 Overview

While the general communication model architected around the APO exchanges is indeed that of publisher/subscriber interactions exchanging data topics, publish/subscribe AL supports the model using several unconfirmed services. Some publish/subscribe unconfirmed interactions do demand confirmation, but the confirmation is deferred to the AL user. From a communications perspective, there is no relationship between separate invocations of unconfirmed services as there is between the request and response of a confirmed service.

NOTE OMG DDS as in “Data Distribution Service for Real-Time Systems Specification, Version 1.1, December 2005” defines services that are here presented as responsibility of the publish/subscribe AL user instead.

publish/subscribe is a wire protocol, and as such some functionality is deferred to the user application. This can be appreciated by the flexibility offered for implementations where foot-print is at a premium, and contributes to the pervasiveness of the approach.

The publish/subscribe application layer handles publish/subscribe interactions using an AL user-triggered uni-directional AREP for unconfirmed services.

The publish/subscribe application layer itself has no particular requirements regarding connection-oriented or connection-less AREPs, it could indeed use connection-oriented AREPs. In the next sections, the following will be detailed:

Queued User-triggered Uni-directional (QUU) AR Endpoint;

The queue depth depends on QoS configuration parameters.

7.2.2 Queued user-triggered unidirectional AREP class specification

7.2.2.1 Class overview

This class is defined to support the on-demand queued distribution of unconfirmed services to one or more application processes. The behavior of this type of AR can be described as follows.

An AR ASE user wishing to convey a request APDU submits an AR ASE Service Data Unit to the sending endpoint of the AR. The AREP sending the request APDU submits it to its underlying layer for transfer. The underlying layer sends it at its next opportunity. The AREP receiving the request APDU from its underlying layer delivers it to the AR ASE user in the order that it was received.

The following summarizes the characteristics of this AREP class.

Roles: PEER

Cardinality: n-to-n

Timeliness: Configurable

7.2.2.2 Formal model

ASE: **AR ASE**
CLASS: **Queued User-triggered Uni-directional (QUU) AREP**
CLASS ID: not used
PARENT CLASS: AR Endpoint
ATTRIBUTES:
1. (m) Attribute: Role (Peer)
SERVICES:
1. (m) OpsService: Unconfirmed Send

7.2.2.3 Attributes

Role
This attribute specifies possible roles of this end point. Both publish/subscribe readers and writers send and receive, so the appropriate role regarding the unconfirmed send is that of peer. The peer role will be further detailed when discussing services and APDUs.

Peer AREPs of this type convey their data by issuing a Data Transfer Unconfirmed PDU to peers, and receive data transmitted by peers source AREP

7.2.2.4 Services

Unconfirmed Send
This service is used to send an unconfirmed service on the specified AR.

7.3 Summary of FAL classes

Table 40 of this subclause contains a summary of the publish/subscribe defined FAL Classes. The Class ID values have been assigned to be compatible with existing standards.

Table 40 – FAL class summary

FAL ASE	Class	Class ID
Writer	Publisher	—
	CSTWriter	—
Reader	Subscriber	—
	CSTReader	—
Messenger	Messenger	—
Application Relationship	AREP	32
	QUU	36

7.4 Permitted FAL services by AREP role and sub-role

Table 41 below defines the valid combinations of services and AREP roles (which service APDUs and AREP with the specified role can send or receive) for publish/subscribe. It has already been mentioned that the role is that of peer, and here the classification will be done in terms of class sub-roles: Publisher, CSTWriter, Subscriber, CSTReader and Messenger. The Unc and Cnf columns that indicate whether the service listed in the left-hand column is unconfirmed (Unc) or confirmed (Cnf) are not needed, since all publish/subscribe services are unconfirmed from a communications perspective.

Table 41 – Services by AREP role and sub-role

	Publisher		CSTWriter		Subscriber		CSTReader	
FAL Services	**req**	**rcv**	**req**	**rcv**	**req**	**rcv**	**req**	**rcv**
Writer								
Issue	X					X		
Heartbeat	X		X			X		X
VAR			X			X		X
GAP			X					X
Reader								
ACK		X		X	X		X	
Messenger								
Header	X	X	X	X	X	X	X	X
INFO_DST	X	X	X	X	X	X	X	X
INFO_REPLY	X	X	X	X	X	X	X	X
INFO_SRC	X	X	X	X	X	X	X	X
INFO_TS	X	X	X	X	X	X	X	X
PAD	X	X	X	X	X	X	X	X
AR ASE								
AR-Unconfirmed Send	X	X	X	X	X	X	X	X

Bibliography

IEC/TR 61158-1(Ed.2.0), *Industrial communication networks – Fieldbus specifications – Part 1: Overview and guidance for the IEC 61158 and IEC 61784 series*

IEC 61158-6-15, *Industrial communication networks – Fieldbus specifications – Part 6-15: Application layer protocol specification – Type 15 elements*

IEC 61784-1 (Ed.2.0), *Industrial communication networks – Profiles – Part 1: Fieldbus profiles*

IEC 61784-2, *Industrial communication networks – Profiles – Part 2: Additional fieldbus profiles for real-time networks based on ISO/IEC 8802-3*

ISO/IEC 7498-3, *Information technology – Open Systems Interconnection – Basic Reference Model – Part 3: Naming and addressing*

OMG UML 2.0 *Superstructure Specification*, available at <http://www.omg.org>

IETF RFC 768, *User Datagram Protocol*; available at <http://www.ietf.org>

IETF RFC 791, *Internet Protocol*; available at <http://www.ietf.org>

IETF RFC 793, *Transmission Control Protocol*; available at <http://www.ietf.org>

IETF RFC 1112, *Host Extensions for IP Multicasting*; available at <http://www.ietf.org>

IETF RFC 1305, *Network Time Protocol (Version 3) Specification, Implementation and Analysis*; available at <http://www.ietf.org>

CIA Draft Standard Proposal 309, *Interfacing CANopen with TCP/IP,* Part 2: *Modbus/TCP mapping*

SOMMAIRE

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

_____

RÉSEAUX DE COMMUNICATION INDUSTRIELS – SPÉCIFICATIONS DES BUS DE TERRAIN –

Partie 5-15: Définition des services de la couche application – Éléments de Type 15

AVANT-PROPOS

1) La Commission Électrotechnique Internationale (CEI) est une organisation mondiale de normalisation composée de l'ensemble des comités électrotechniques nationaux (Comités nationaux de la CEI). La CEI a pour objet de favoriser la coopération internationale pour toutes les questions de normalisation dans les domaines de l'électricité et de l'électronique. A cet effet, la CEI – entre autres activités – publie des Normes internationales, des Spécifications techniques, des Rapports techniques, des Spécifications accessibles au public (PAS) et des Guides (ci-après dénommés "Publication(s) de la CEI"). Leur élaboration est confiée à des comités d'études, aux travaux desquels tout Comité national intéressé par le sujet traité peut participer. Les organisations internationales, gouvernementales et non gouvernementales, en liaison avec la CEI, participent également aux travaux. La CEI collabore étroitement avec l'Organisation Internationale de Normalisation (ISO), selon des conditions fixées par accord entre les deux organisations.
2) Les décisions ou accords officiels de la CEI concernant les questions techniques représentent, dans la mesure du possible, un accord international sur les sujets étudiés, étant donné que les Comités nationaux de la CEI intéressés sont représentés dans chaque comité d'études.
3) Les Publications de la CEI se présentent sous la forme de recommandations internationales et sont agréées comme telles par les Comités nationaux de la CEI. Tous les efforts raisonnables sont entrepris afin que la CEI s'assure de l'exactitude du contenu technique de ses publications; la CEI ne peut pas être tenue responsable de l'éventuelle mauvaise utilisation ou interprétation qui en est faite par un quelconque utilisateur final.
4) Dans le but d'encourager l'uniformité internationale, les Comités nationaux de la CEI s'engagent, dans toute la mesure possible, à appliquer de façon transparente les Publications de la CEI dans leurs publications nationales et régionales. Toutes divergences entre toutes Publications de la CEI et toutes publications nationales ou régionales correspondantes doivent être indiquées en termes clairs dans ces dernières.
5) La CEI n'a prévu aucune procédure de marquage valant indication d'approbation et n'engage pas sa responsabilité pour les équipements déclarés conformes à une de ses Publications.
6) Il convient que tous les utilisateurs s'assurent d'être en possession de la dernière édition de cette publication.
7) Aucune responsabilité ne doit être imputée à la CEI, à ses administrateurs, employés, auxiliaires ou mandataires, y compris ses experts particuliers et les membres de ses comités d'études et des Comités nationaux de la CEI, pour tout préjudice causé en cas de dommages corporels et matériels, ou de tout autre dommage de quelque nature que ce soit, directe ou indirecte, ou pour supporter les coûts (y compris les frais de justice) et les dépenses découlant de la publication ou de l'utilisation de cette Publication de la CEI ou de toute autre Publication de la CEI, ou au crédit qui lui est accordé.
8) L'attention est attirée sur les références normatives citées dans cette publication. L'utilisation de publications référencées est obligatoire pour une application correcte de la présente publication.
9) L'attention est attirée sur le fait que certains des éléments de la présente Publication de la CEI peuvent faire l'objet de droits de propriété intellectuelle ou de droits analogues. La CEI ne saurait être tenue pour responsable de ne pas avoir identifié de tels droits de propriété et de ne pas avoir signalé leur existence.

NOTE L'utilisation de certains des types de protocoles associés est limitée par leurs détenteurs de droits de propriété intellectuelle. Dans tous les cas, l'engagement à limiter l'attribution des droits de propriété intellectuelle effectuée par les détenteurs de ces droits permet l'utilisation d'un type de protocole de couche liaison de données particulier avec des protocoles de couche physique et de couche application en combinaisons de types, comme spécifié explicitement dans la série CEI 61784. L'utilisation des différents types de protocoles dans d'autres combinaisons peut nécessiter la permission de la part de leurs détenteurs de droits de propriété intellectuelle respectifs.

La Norme Internationale CEI 61158-5-15 a été établie par le sous-comité 65C: Réseaux de communication industriels, du comité d'études 65 de la CEI: Mesure, commande et automation dans les processus industriels.

Cette première édition et ses parties d'accompagnement de la sous-série CEI 61158-5 annulent et remplacent la CEI 61158-5:2003. La présente édition de cette partie constitue une révision technique. Cette partie et ses parties d'accompagnement pour le Type 15 annulent et remplacent également la CEI/PAS 62030, publiée en 2004.

Cette édition de la CEI 61158-5 inclut les modifications majeures suivantes par rapport à l'édition précédente:

a) suppression du précédent bus de terrain de Type 6 en raison du manque d'adéquation au marché;

b) ajout de nouveaux types de bus de terrain;

c) division de la partie 5 de la troisième édition en parties multiples numérotées -5-2,-5-3, ...

La présente version bilingue (2014-06) correspond à la version anglaise monolingue publiée en 2007-12.

Le texte anglais de cette norme est issu des documents 65C/475/FDIS et 65C/486/RVD.

Le rapport de vote 65C/486/RVD donne toute information sur le vote ayant abouti à l'approbation de cette norme.

La version française de cette norme n'a pas été soumise au vote.

Cette publication a été rédigée selon les Directives ISO/CEI, Partie 2.

Le comité a décidé que le contenu de cette publication ne sera pas modifié avant la date de résultat de maintenance indiquée sur le site web de la CEI sous "http://webstore.iec.ch" dans les données relatives à la publication recherchée. A cette date, la publication sera

- reconduite;
- supprimée;
- remplacée par une édition révisée, ou
- amendée.

NOTE La révision de la présente norme sera synchronisée avec les autres parties de la série CEI 61158.

La liste de toutes les parties de la série CEI 61158, publiées sous le titre général *Communications numériques pour les systèmes de mesure et de commande – Spécifications des bus de terrain*, peut être consultée sur le site web de la CEI.

INTRODUCTION

La présente partie de la CEI 61158 constitue l'un des éléments d'une série rédigée pour faciliter l'interconnexion des composants des systèmes d'automatisation. Elle est liée à d'autres normes de l'ensemble défini par le modèle de référence de bus de terrain dit "à trois couches", décrit dans la CEI/TR 61158-1.

Le service d'application est fourni par le protocole d'application, qui utilise les services disponibles dans la couche liaison de données ou toute autre couche immédiatement inférieure. La présente norme définit les caractéristiques des services d'application que les applications à bus de terrain et/ou la gestion de systèmes peuvent exploiter.

Dans cet ensemble de normes relatives aux bus de terrain, le terme "service" désigne la capacité abstraite fournie par une couche du modèle de référence de base OSI à la couche située juste au-dessus. Le service de couche application défini dans la présente norme est donc un service architectural conceptuel, indépendant des divisions administratives et de mise en œuvre.

RÉSEAUX DE COMMUNICATION INDUSTRIELS – SPÉCIFICATIONS DES BUS DE TERRAIN –

Partie 5-15: Définition des services de la couche application – Éléments de Type 15

1 Domaine d'application

1.1 Vue d'ensemble

Dans les communications par réseaux, de même que dans de nombreux domaines de l'ingénierie, il est un fait que "les tailles uniques ne conviennent pas à tout le monde". La conception technique a pour objet de rechercher les bons compromis, ceux-ci devant permettre de parvenir à un bon équilibre entre différentes exigences conflictuelles telles que simplicité, universalité, facilité d'utilisation, richesse des fonctions, performances, taille et usage de la mémoire, modularité, déterminisme et robustesse. Ces compromis doivent prendre en considération les types de flux d'information (par exemple périodiques, à origine unique et destinations multiples, demande-réponse, événements) et les contraintes imposées par les plates-formes d'application et d'exécution.

Le bus de terrain de type 15 fournit deux mécanismes de communication majeurs qui se complètent l'un l'autre pour satisfaire aux exigences de communication du domaine de l'automatisation: les paradigmes client/serveur et fournisseur/abonné. Ils peuvent être utilisés de manière concomitante sur le même dispositif.

Le client/serveur type 15 fonctionne dans une relation client/serveur. Les définitions des services de couche application et les spécifications de protocole le concernant sont indépendantes des couches sous-jacentes et ont été mises en œuvre sur différentes sortes de piles et de supports de communication, notamment EIA/TIA-232, EIA/TIA-422, EIA/TIA-425, HDLC (ISO 13239), la fibre, TCP/IP, les réseaux locaux sans fil et les radios.

Le fournisseur/abonné type 15 fonctionne dans une relation fournisseur/abonné. Les définitions des services de couche application et les spécifications de protocole le concernant sont indépendantes des couches sous-jacentes et peuvent être configurées pour fournir un fonctionnement fiable et supporter le déterminisme. La pile la plus commune est la pile UDP/IP.

La couche application de bus de terrain (FAL - fieldbus application layer) donne aux programmes d'utilisateur le moyen d'accéder à l'environnement de communication des bus de terrain. À cet égard, la FAL peut être considérée comme une "fenêtre entre programmes d'application correspondants".

La présente partie de la CEI 61158 fournit des éléments communs pour les communications de messagerie prioritaires et non prioritaires élémentaires entre les programmes d'application des environnements d'automatisation et le matériel spécifique au bus de terrain de type 15. On utilise le terme "prioritaire" pour traduire la présence d'une fenêtre temporelle, à l'intérieur de laquelle une ou plusieurs actions spécifiées doivent être terminées avec un niveau de certitude défini. Si les actions spécifiées ne sont pas terminées à l'intérieur de cette fenêtre temporelle, les applications qui ont demandé l'exécution de ces actions risquent de présenter des dysfonctionnements, accompagnés de risques pour les équipements, l'usine, voire les vies humaines.

La présente partie de la CEI 61158 définit d'une manière abstraite le service visible de manière externe fourni par la couche application du bus de terrain de type 15 en termes:

a) d'un modèle abstrait pour la définition des ressources d'application (objets) qui peuvent être manipulées par les utilisateurs par l'intermédiaire de l'utilisation du service FAL,

b) des actions primitives et des événements du service;

c) des paramètres associés à chaque action et événement primitif, et de la forme qu'ils peuvent prendre; et

d) des interrelations entre ces actions et événements, et de leurs séquences valides.

La présente partie de la CEI 61158 a pour objet de définir les services fournis:

1) à l'utilisateur de la FAL à la frontière entre l'utilisateur et la couche application du modèle de référence de bus de terrain, et

2) à la gestion des systèmes à la frontière entre la couche application et la gestion des systèmes du modèle de référence de bus de terrain.

La présente partie de la CEI 61158 spécifie la structure et les services de la couche application de bus de terrain CEI type 15, en conformité au modèle de référence de base OSI (ISO/CEI 7498) et à la structure de couche application OSI (ISO/CEI 9545).

Les services et les protocoles de la FAL sont fournis par des entités d'application (AE - application-entity) de FAL contenues dans les processus d'application. L'AE de FAL est composée d'un ensemble d'éléments de service d'application (ASE - Application Service Elements) orientés objet et d'une entité de gestion de couche (LME - Layer Management Entity) qui gère l'AE. Les ASE fournissent des services de communication qui fonctionnent sur un ensemble de classes d'objets de processus d'application (APO - application process object) associées. L'un des ASE de la FAL est un ASE de gestion qui fournit un ensemble commun de services pour la gestion des instances des classes FAL.

Ces services spécifient, du point de vue des applications, la façon dont les demandes et les réponses sont émises et fournies, mais ils ne spécifient pas ce que les applications émettant les demandes et les réponses doivent en faire. Autrement dit, les aspects comportementaux des applications ne sont pas spécifiés; seules sont définies les demandes et les réponses que ces applications peuvent envoyer/recevoir. Cela offre aux utilisateurs de la FAL une plus grande flexibilité pour normaliser le comportement de ces objets. Outre ces services, la présente norme définit également certains services de support permettant l'accès à la FAL pour contrôler certains aspects de son fonctionnement.

1.2 Spécifications

Le principal objectif de la présente partie de la CEI 61158 est de spécifier les caractéristiques des services conceptuels de couche application appropriés pour les communications prioritaires et, ainsi, de compléter le modèle de référence de base OSI en guidant le développement des protocoles de couche application pour les communications prioritaires.

Un objectif secondaire est de fournir des chemins de migration aux protocoles de communication industriels existants. C'est ce dernier objectif qui a donné lieu à la diversité des services normalisés reflétée par les différents types de la CEI 61158 et les protocoles correspondants normalisés des sous-parties de la CEI 61158-6.

Cette spécification peut être utilisée comme base dans les interfaces de programmation d'application formelles. Néanmoins, il ne s'agit pas d'une interface de programmation formelle, et toute interface de ce type devra résoudre les problèmes de mise en œuvre non traités par la présente spécification, notamment:

a) les tailles et l'ordre des octets de divers paramètres de service multi-octets, et

b) la corrélation des primitives appariées demande et confirmation, ou indication et réponse.

1.3 Conformité

La présente partie de la CEI 61158 ne spécifie pas de mises en œuvre individuelles ou de produits; elle n'impose pas non plus la mise en œuvre d'entités de couche application dans les systèmes d'automatisation industriels.

Il n'y a pas de conformité des équipements à la présente norme de définition des services de couche application. La conformité s'obtient au contraire par la mise en œuvre de protocoles de couche application conformes qui permettent de réaliser les services de couche application de type 15 définis dans la présente partie de la CEI 61158.

1.4 Vue d'ensemble du type

Dans les communications par réseaux, de même que dans de nombreux domaines de l'ingénierie, il est un fait que "les tailles uniques ne conviennent pas à tout le monde". La conception technique a pour objet de rechercher les bons compromis, ceux-ci devant permettre de parvenir à un bon équilibre entre différentes exigences conflictuelles telles que simplicité, universalité, facilité d'utilisation, richesse des fonctions, performances, taille et usage de la mémoire, modularité, déterminisme et robustesse. Ces compromis doivent prendre en considération les types de flux d'information (par exemple périodiques, à origine unique et destinations multiples, demande-réponse, événements) et les contraintes imposées par les plates-formes d'application et d'exécution.

Le bus de terrain de type 15 fournit deux mécanismes de communication majeurs qui se complètent l'un l'autre pour satisfaire aux exigences de communication du domaine de l'automatisation: les paradigmes client/serveur et fournisseur/abonné. Ils peuvent être utilisés de manière concomitante sur le même dispositif.

Le client/serveur type 15 fonctionne dans une relation client/serveur. Les définitions des services de couche application et les spécifications de protocole le concernant sont indépendantes des couches sous-jacentes et ont été mises en œuvre sur différentes sortes de piles et de supports de communication, notamment EIA/TIA-232, EIA/TIA-422, EIA/TIA-425, HDLC (ISO 13239), la fibre, TCP/IP, les réseaux locaux sans fil et les radios.

Le fournisseur/abonné type 15 fonctionne dans une relation fournisseur/abonné. Les définitions des services de couche application et les spécifications de protocole le concernant sont indépendantes des couches sous-jacentes et peuvent être configurées pour fournir un fonctionnement fiable et supporter le déterminisme. La pile la plus commune est la pile UDP/IP.

2 Références normatives

Les documents de référence suivants sont indispensables pour l'application du présent document. Pour les références datées, seule l'édition citée s'applique. Pour les références non datées, la dernière édition du document de référence s'applique (y compris les éventuels amendements).

CEI/TR 61158-1 (Ed.2.0), *Réseaux de communication industriels – Spécifications des bus de terrain – Partie 1: Présentation et lignes directricesSéries des séries CEI 61158 et CEI 61784*

ISO/CEI 7498-1, *Technologies de l'information – Interconnexion de systèmes ouverts (OSI) – Modèle de référence de base – Partie 1: Le modèle de base*

ISO/CEI 8822, *Systèmes de traitement de l'information – Interconnexion de systèmes ouverts – Définition du service de présentation en mode connexion*

ISO/CEI 8824, *Technologies de l'information – Interconnexion de systèmes ouverts – Spécification de la notation de syntaxe abstraite numéro un (ASN.1)*

ISO/CEI 9545, *Technologies de l'information – Interconnexion de systèmes ouverts (OSI) – Structure de la couche application*

ISO/CEI 10731, *Technologies de l'information – Interconnexion de systèmes ouverts (OSI) – Modèle de référence de base – Conventions pour la définition des services OSI*

3 Termes et définitions, abréviations, symboles et conventions

3.1 Termes et définitions

Pour les besoins du présent document, les termes suivants, tels qu'ils sont définis dans les publications indiquées, s'appliquent:

3.1.1 Termes de l'ISO/CEI 7498-1

a) entité d'application
b) processus d'application
c) unité de données de protocole d'application
d) élément de service d'application
e) appel d'entité d'application
f) appel de processus d'application
g) transaction d'application
h) système ouvert réel
i) syntaxe de transfert

3.1.2 Termes de l'ISO/CEI 8822

a) syntaxe abstraite
b) contexte de présentation

3.1.3 Termes de l'ISO/CEI 9545

a) association d'application
b) contexte d'application
c) nom de contexte d'application
d) appel d'entité d'application
e) type d'entité d'application
f) appel de processus d'application
g) type de processus d'application
h) élément de service d'application
i) élément de service de commande d'application

3.1.4 Termes de l'ISO/CEI 8824

a) identificateur d'objet
b) type

3.1.5 Termes de la CEI/TR 61158-1

Les termes suivants de la CEI/TR 61158-1 s'appliquent.

3.1.5.1 application

fonction ou structure de données pour laquelle des données sont consommées ou produites

3.1.5.2 interopérabilité de couche application

capacité qu'ont les entités d'application à effectuer des opérations coordonnées et coopératives en utilisant les services de la FAL

3.1.5.3 objet d'application

classe d'objet qui gère et fournit l'échange de messages en mode opératoire à travers le réseau et à l'intérieur du dispositif réseau

NOTE: Plusieurs types de classes d'objet d'application peuvent être définis.

3.1.5.4 processus d'application

partie d'une application distribuée sur un réseau, qui est située sur un dispositif et qui est adressée sans ambiguïté

3.1.5.5 identificateur de processus d'application

identificateur permettant de distinguer plusieurs processus d'application utilisés dans un dispositif

3.1.5.6 objet de processus d'application

composant d'un processus d'application qui est identifiable et accessible par la relation d'application de la FAL

NOTE Les définitions des objets de processus d'application sont composées d'un ensemble de valeurs données aux attributs de leur classe.

3.1.5.7 classe d'objet de processus d'application

classe d'objets de processus d'application définis par rapport à l'ensemble des attributs et services accessibles en réseau qu'ils possèdent

3.1.5.8 relation d'application

association coopérative entre deux appels d'entité d'application ou plus, destinée à l'échange d'informations et à la coordination de leur association fonctionnelle

NOTE Cette relation est activée par l'échange d'unités de données de protocole d'application ou à la suite d'activités de préconfiguration.

3.1.5.9 point final de relation d'application

contexte et comportement d'une relation d'application, vus et maintenus par l'un des processus d'application impliqués dans la relation d'application

NOTE Chaque processus d'application impliqué dans la relation d'application maintient son propre point final de relation d'application.

3.1.5.10 élément de service d'application

élément de service d'application qui fournit le moyen exclusif d'établir et de terminer toutes les relations d'application

3.1.5.11 attribut

description d'une caractéristique ou d'une particularité, visible de l'extérieur, d'un objet

NOTE Les attributs d'un objet contiennent des informations au sujet des parties variables d'un objet. Ils servent habituellement à fournir des informations de statut ou à régir le fonctionnement d'un objet. Les attributs peuvent également avoir un effet sur le fonctionnement d'un objet. Les attributs se répartissent en attributs de classe et en attributs d'instance.

3.1.5.12 comportement

indication de la façon dont l'objectif répond à des événements particuliers

NOTE Sa description comprend la relation entre les valeurs d'attribut et les services.

3.1.5.13 classe

ensemble d'objets, chacun d'eux représentant la même sorte de composant de système

NOTE Une classe constitue la généralisation de l'objet; un modèle pour définir les variables et les méthodes. Tous les objets d'une classe sont identiques en forme et en comportement, mais ils contiennent habituellement des données différentes dans leurs attributs.

3.1.5.14 attribut de classe

attribut qui est partagé par tous les objets de la même classe

3.1.5.15 code de classe

identificateur unique affecté à chaque classe d'objet

3.1.5.16 service spécifique à une classe

service défini par une classe d'objets particulière pour l'exécution d'une fonction requise qui n'est pas exécutée par un service commun

NOTE Un objet spécifique à une classe est unique pour la classe d'objets qui le définit.

3.1.5.17 client

(a) objet qui utilise les services d'un autre objet (serveur) pour exécuter une tâche

(b) initiateur d'un message auquel un serveur réagit, comme par exemple le rôle d'un point final d'AR consistant à envoyer des APDU de demande de service confirmé à un point final d'AR individuel agissant en tant que serveur

3.1.5.18 chemin de transmission

flux unidirectionnel d'APDU sur l'ensemble d'une relation d'application

3.1.5.19 cyclique

terme utilisé pour décrire des événements qui se reproduisent de manière régulière et répétitive

3.1.5.20 AR dédiée

AR utilisée directement par l'utilisateur de la FAL

NOTE Sur les AR dédiées, seuls l'en-tête de la FAL et les données de l'utilisateur sont transférés.

3.1.5.21 dispositif

connexion matérielle physique à la liaison

NOTE Un dispositif peut contenir plus d'un nœud.

3.1.5.22 profil de dispositif

collection d'informations et de fonctionnalités dépendantes du dispositif, fournissant une harmonisation entre dispositifs similaires de même type

3.1.5.23 AR dynamique

AR qui exige l'utilisation de procédures d'établissement d'AR pour pouvoir être placée dans un état établi

3.1.5.24 point final

l'une des entités communicantes impliquées dans une connexion

3.1.5.25 erreur

divergence entre une valeur ou une condition calculée, observée ou mesurée et la valeur ou la condition spécifiée ou théoriquement correcte

3.1.5.26 classe d'erreur

groupement général pour les définitions d'erreur

NOTE Les codes d'erreur correspondant à des erreurs spécifiques sont définis dans une classe d'erreur.

3.1.5.27 code d'erreur

identification d'un type d'erreur spécifique dans une classe d'erreur

3.1.5.28 sous-réseau FAL

réseaux composés d'un ou de plusieurs segments de liaison de données

NOTE Les sous-réseaux peuvent contenir des ponts, mais pas les routeurs. Les sous-réseaux FAL sont identifiés par un sous-ensemble de l'adresse réseau.

3.1.5.29 dispositif logique

classe FAL qui extrait un composant logiciel ou un composant de micrologiciel sous forme de ressource autonome indépendante d'un dispositif d'automatisation

3.1.5.30 information de gestion

information accessible en réseau utilisée pour gérer le fonctionnement du système de bus de terrain, y compris la couche application

NOTE Cette gestion comprend des fonctions telles que le contrôle, la surveillance et l'établissement de diagnostics.

3.1.5.31 réseau

série de nœuds connectés par un certain type de support de communication

NOTE Les chemins de connexion reliant n'importe quelle paire de nœuds peuvent comporter des répéteurs, des routeurs et des passerelles.

3.1.5.32 pair

rôle d'un point final d'AR consistant à être capable d'agir à la fois comme client et comme serveur

3.1.5.33 point final d'AR prédéfini

point final d'AR qui est défini localement dans un dispositif sans passer par le service de création

NOTE Les AR prédéfinis qui ne sont pas pré-établis sont établis avant d'être utilisés.

3.1.5.34 point final d'AR pré-établi

point final d'AR qui est placé dans un état établi lors de la configuration des AE qui servent à commander ses points finaux

3.1.5.35 fournisseur

rôle d'un point final d'AR consistant à émettre des APDU sur le bus de terrain en vue de leur consommation par un ou plusieurs abonnés

NOTE Le fournisseur peut ne pas avoir connaissance de l'identité des abonnés ou de leur nombre, et il peut fournir ses APDU au moyen d'une AR dédiée. Deux types de fournisseurs sont définis dans la présente norme: les fournisseurs tireurs et les fournisseurs pousseurs, chacun étant défini individuellement.

3.1.5.36 serveur

a) rôle joué par un AREP consistant à renvoyer une APDU de réponse de service confirmé au client qui a initié la demande

b) objet qui fournit des services à un autre objet (client)

3.1.5.37 service

opération ou fonction qu'un objet et/ou une classe d'objets exécute sur demande provenant d'un autre objet et/ou classe d'objets

NOTE Un ensemble de services communs est défini, et des dispositions en vue de la définition des services spécifiques aux objets sont fournies. Les services spécifiques aux objets sont les services qui sont définis par une classe d'objets particulière pour l'exécution d'une fonction requise qui n'est pas exécutée par un service commun.

3.1.5.38 abonné

rôle joué par un AREP consistant à recevoir des APDU produites par un fournisseur

NOTE Deux types d'abonnés sont définis dans la présente norme: les fournisseurs tireurs et les fournisseurs pousseurs, chacun étant défini individuellement.

3.1.6 Définitions spécifiques au client/serveur

3.1.6.1
bobines, sorties discrètes
objet de processus d'application, un ensemble de bobines, caractérisé par l'adresse d'une bobine et une quantité de bobines, cet ensemble étant également appelé sorties discrètes lorsqu'il est associé aux sorties de terrain

3.1.6.2
discret, entrée discrète
objet de processus d'application, adressé par un nombre non signé et ayant une largeur d'un bit, représentant une valeur de statut codée sur 1 bit, en lecture seule, la valeur '1' correspondant au statut actif et la valeur '0' au statut inactif, également appelé entrée discrète, en particulier lorsqu'il est associé à des sorties de terrain

3.1.6.3
entrées discrètes, discrets
objet de processus d'application, un ensemble de discrets, caractérisé par l'adresse d'un discret et une quantité de discrets, cet ensemble étant également appelé entrées discrètes, en particulier lorsqu'il est associé aux entrées de terrain

3.1.6.4
bobine, sortie discrète
objet de processus d'application, adressé par un nombre non signé et ayant une largeur d'un bit, représentant une valeur de statut codée sur 1 bit, en lecture seule, la valeur '1' correspondant au statut actif et la valeur '0' au statut inactif, également appelé sortie discrète, en particulier lorsqu'il est associé à une sortie de terrain

3.1.6.5
interface encapsulée
mécanisme servant à encapsuler un service pour une interface, ce mécanisme constituant un objet de processus d'application caractérisé par un type MEI

3.1.6.6
exception
codage utilisé pour signaler l'échec d'une demande de service

3.1.6.7
code d'exception
codage associé à une exception, détaillant les raisons pour lesquelles une demande de service a échoué

3.1.6.8
fichier
objet de processus d'application, une organisation d'enregistrements, caractérisé par un nombre non signé

3.1.6.9
code de fonction
codage d'un service demandé à un serveur

3.1.6.10
registre de maintien, registre de sortie
objet de processus d'application, adressé par un nombre non signé et représentant des valeurs de 16 bits, en lecture et écriture

3.1.6.11
registres de maintien, registres de sortie
objet de processus d'application, un ensemble de registres de maintien, caractérisé par l'adresse d'un registre de maintien et une quantité de registres de maintien

3.1.6.12
registre d'entrée
objet de processus d'application, adressé par un nombre non signé et représentant des valeurs de 16 bits, en lecture seule

3.1.6.13
registres d'entrée
objet de processus d'application, un ensemble de registres d'entrée, caractérisé par l'adresse d'un registre d'entrée et une quantité de registres d'entrée

3.1.6.14
enregistrement
objet de processus d'application, un ensemble de registres contigus d'un type spécifié, caractérisé par l'adresse du premier registre et par la quantité de registres; dans le contexte de cette définition, les registres concernés sont également appelés des références

3.1.6.15
référence
terme critiqué pour désigner un registre

3.1.6.16
type de référence
terme critiqué pour désigner un type de registre

3.1.6.17
sous-code
spécialisation d'un code de fonction

3.1.6.18
ID d'unité
identificateur de dispositif logique

3.1.6.19
type MEI
type spécifié sous forme de valeur d'octet, utilisé pour expédier un service à l'interface appropriée dans le contexte du mécanisme de l'interface encapsulée

3.1.7 Définitions spécifiques au fournisseur/abonné

3.1.7.1
objet de réseau
application fournisseur/abonné, lecteur, ou auteur

3.1.7.2
GUID
identificateur d'objet de réseau globalement unique, servant à référencer de manière unique un objet présent sur le réseau

3.1.7.3
état composite
attributs d'un ensemble d'objets de réseau

3.1.7.4
transfert d'état composite
Interactions entre CSTWriters et CSTReaders intéressés, avec pour but d'autoriser les CSTReaders à reconstruire l'état composite du CSTWriter communicant, sans transférer l'histoire complète qui a conduit à l'état composite actuel de ce CSTWriter.

3.1.7.5
lecteur
un abonné ou un CSTReader

3.1.7.6
auteur
un fournisseur ou un CSTWriter

3.1.7.7
CSTReader
abonné spécialisé en méta-informations

3.1.7.8
CSTWriter
fournisseur spécialisé en méta-informations

3.1.7.9
acteur de communication
lecteur ou auteur

3.1.7.10
participant de domaine
application qui utilise des éléments fournisseur/abonné, également appelée application fournisseur/abonné

NOTE On a adopté cette terminologie dans le but d'éviter l'usage excessif du terme "application". Parallèlement, le terme "domaine" a une place dans le mode fournisseur/abonné. L'extensibilité de type autorise le concept de "domaines", ou de plans de communication indépendants, qui permet d'isoler efficacement les échanges d'application à l'intérieur des domaines. Le DDS de l'OMG, comme dans "Data Distribution Service for Real-Time Systems Specification, Version 1.1, December 2005", utilise cette extension, mais la fonction n'est pas examinée de façon plus approfondie dans la présente spécification, laquelle ne considère qu'un domaine individuel.

3.1.7.11
gestionnaire
application fournisseur/abonné spécialisée contenant des fournisseurs et abonnés spécialisés et impliquée dans le mécanisme de découverte et maintenance décrit; à ne pas confondre avec un gestionnaire de fourniture

3.1.7.12
participant géré
application fournisseur/abonné; l'adjectif désigne son rôle par rapport à un gestionnaire lorsqu'il est impliqué dans le mécanisme de découverte et maintenance décrit; à ne pas confondre avec un gestionnaire de fourniture

3.1.7.13
gestionnaire de fourniture
rôle d'un point final d'AR consistant à émettre une ou plusieurs APDU de demande de service confirmé à un fournisseur pour demander au fournisseur de fournir un objet spécifié. Deux types de gestionnaires de fourniture sont définis dans la présente norme: les gestionnaires de fourniture tireurs et les gestionnaires de fourniture pousseurs, chacun étant défini individuellement.

3.1.7.14
fournisseur tireur
type de fournisseur qui fournit un objet en réponse à une demande reçue de son gestionnaire de fourniture tireur

3.1.7.15
gestionnaire de fourniture tireur
type de gestionnaire de fourniture qui demande qu'un objet spécifié soit fourni dans une APDU de réponse correspondante

3.1.7.16
fournisseur pousseur
type de fournisseur qui fournit un objet dans une APDU de demande de service non confirmée

3.1.7.17
gestionnaire de fourniture pousseur
type de gestionnaire de fourniture qui demande qu'un objet spécifié soit fourni au moyen d'un service non confirmé

3.1.7.18
abonné tireur
type d'abonné qui reconnaît les APDU de réponse de service confirmé reçues en tant que données d'objet fournies

3.1.7.19
abonné pousseur
type d'abonné qui reconnaît les APDU de demande de service non confirmé reçues en tant que données d'objet fournies

3.1.7.20
numéro de séquence
numéro utilisé pour identifier de manière unique des messages fournisseur/abonné élémentaires d'une manière ordonnée

3.2 Abréviations et symboles

3.2.1 Abréviations et symboles courants

AE Application Entity (Entité d'application)
AL Application Layer (Couche application)
ALME Application Layer Management Entity (Entité de gestion de couche application)
ALP Application Layer Protocol (Protocole de couche application)
APO Application Object (Objet d'application)
AP Application Process (Processus d'application)
APDU Application Protocol Data Unit (Unité de données de protocole d'application)
API Application Process Identifier (Identificateur de processus d'application)
AR Application Relationship (Relation d'application)
AREP Application Relationship End Point (point final de relation d'application)
ASCII American Standard Code for Information Interchange (Code normalisé américain pour l'échange d'informations)
ASE Application Service Element (Élément de service d'application)
Cnf Confirmation
DL- (préfixe) Data Link- (Liaison de données)
DLC Data Link Connection (Connexion de liaison de données)
DLCEP Data Link Connection End Point (Point final de connexion de liaison de données)
DLL Data Link Layer (Couche liaison de données)
DLM Data Link-management (Gestion de liaison de données)
DLSAP Data Link Service Access Point (Point d'accès au service de liaison de données)
DLSDU DL-service-data-unit (Unité de données de service de liaison de données)
FAL Fieldbus Application Layer (Couche application de bus de terrain)
IANA Internet Assigned Numbers Authority (Autorité de gestion des numéros attribués sur Internet)
ID Identificateur
CEI Commission Electrotechnique Internationale
Ind Indication
IP Internet Protocol (Protocole Internet)
LME Layer Management Entity (Entité de gestion de couche)
LSB Least Significant Bit (Bit de poids faible)
MSB Most Significant Bit (Bit de poids fort)
OSI Open Systems Interconnection (Interconnexion de systèmes ouverts)
QoS Quality of Service (Qualité de service)
Req Request (Demande)
Rsp Response (Réponse)

SAP Service Access Point (Point d'accès au service)

SDU Service Data Unit (Unité de données de service)

SMIB System Management Information Base (Base d'information de gestion de système)

SMK System Management Kernel (Noyau de gestion de système)

3.2.2 Abréviations et symboles pour le mode client/serveur

C/S Client/serveur

FC Function Code (Code de fonction)

CAN Controller Area Network (Réseau local de contrôleurs)

CiA CAN en automatisation

MEI Type interface encapsulée

URL Uniform Resource Locator, adresse

3.2.3 Abréviations et symboles pour le mode fournisseur/abonné

CS Composite State (État composite)

CST Composite State Transfer (Transfert d'état composite)

DCPS Data-Centric Publish-Subscribe (Fournisseur/abonné centré sur les données)

DDS Data Distribution Service (Service de distribution de données)

DLRL Data Local Reconstruction Layer (Couche de reconstruction locale de données)

GUID Globally Unique Id (Identificateur globalement unique)

OMG Object Management Group (Groupe de gestion d'objets)

P/S Publish/Subscribe (Fournisseur/abonné)

3.3 Conventions

3.3.1 Vue d'ensemble

La FAL est définie comme étant un ensemble d'ASE orientés objet. Chaque ASE est spécifié dans un paragraphe qui lui est propre. Chaque spécification d'ASE comprend deux parties: la spécification de classe et la spécification de service.

La spécification de classe définit les attributs de la classe. Les attributs sont accessibles dans les instances de la classe au moyen des services ASE de gestion d'objets spécifiés à l'Article 5 de la présente norme. La spécification de service définit les services qui sont fournis par l'ASE.

3.3.2 Conventions générales

La présente norme utilise les conventions descriptives données dans l'ISO/CEI 10731.

3.3.3 Conventions pour les définitions de classe

Les définitions de classe sont décrites au moyen de modèles. Chaque modèle consiste en une liste d'attributs destinés à la classe. La forme générale du modèle est représentée ci-dessous:

ASE de FAL:	**Nom de l'ASE**		
CLASSE:	**Nom de la classe**		
ID CLASSE:	**#**		
CLASSE PARENTE:		**Nom de la classe parente**	
ATTRIBUTS:			
1	(o)	Attribut clé:	identificateur numérique
2	(o)	Attribut clé:	nom
3	(m)	Attribut:	nom de l'attribut(valeurs)
4	(m)	Attribut:	nom de l'attribut(valeurs)
4.1	(s)	Attribut:	nom de l'attribut(valeurs)
4.2	(s)	Attribut:	nom de l'attribut(valeurs)
4.3	(s)	Attribut:	nom de l'attribut(valeurs)
5.	(c)	Contrainte:	expression de la contrainte
5.1	(m)	Attribut:	nom de l'attribut(valeurs)
5.2	(o)	Attribut:	nom de l'attribut(valeurs)
6	(m)	Attribut:	nom de l'attribut(valeurs)
6.1	(s)	Attribut:	nom de l'attribut(valeurs)
6.2	(s)	Attribut:	nom de l'attribut(valeurs)
SERVICES:			
1	(o)	OpsService:	nom du service:
2.	(c)	Contrainte:	expression de la contrainte
2.1	(o)	OpsService:	nom du service:
3	(m)	MgtService:	nom du service:

(1) L'entrée "ASE de FAL:" est le nom de l'ASE de FAL qui fournit les services correspondant à la classe qui est en train d'être spécifiée.

(2) L'entrée "CLASSE:" est le nom de la classe qui est en train d'être spécifiée. Tout objet défini au moyen de ce modèle constitue une instance de cette classe. La classe peut être spécifiée au moyen de la présente norme ou par un utilisateur de la présente norme.

(3) L'entrée "ID CLASSE:" est le numéro qui identifie la classe qui est en train d'être spécifiée. Ce numéro est unique au sein de l'ASE de FAL qui fournit les services correspondant à cette classe. Lorsqu'on lui adjoint l'identité de son ASE de FAL, il identifie sans ambiguïté la classe dans le domaine d'application de la FAL. La valeur "NULL" indique que la classe ne peut pas être instanciée. Les ID de classe compris entre 1 et 255 sont réservés par la présente norme pour l'identification des classes normalisées. Ils ont été définis ainsi pour maintenir la compatibilité avec les normes nationales existantes. Les ID de classe compris entre 256 et 2048 sont alloués pour l'identification des classes définies par les utilisateurs.

(4) L'entrée "CLASSE PARENTE:" est le nom de la classe parente de la classe qui est en train d'être spécifiée. La classe en train d'être définie hérite de tous les attributs définis pour la classe parente et dont celle-ci a hérité, aussi ces attributs ne doivent pas être redéfinis dans le modèle correspondant à cette classe.

NOTE La classe parente "TOP" indique que la classe en train d'être définie est une définition de classe initiale. La classe parente TOP est utilisée comme point de départ à partir duquel toutes les autres classes sont définies. L'utilisation de TOP est réservée aux classes définies par la présente norme.

(5) L'étiquette "ATTRIBUTS" indique que les entrées suivantes sont des attributs définis pour la classe.

a) Chacune des entrées d'attribut est composée d'un numéro de ligne dans la colonne 1, d'un indicateur obligatoire (mandatory) (m) / optionnel (o) / conditionnel (c) / sélecteur (s) dans la colonne 2, d'une étiquette de type d'attribut dans la colonne 3, d'un nom ou d'une expression conditionnelle dans la colonne 4 et, en option, d'une liste de valeurs énumérées dans la colonne 5. Dans la colonne qui suit la liste de valeurs, on peut spécifier la valeur par défaut de l'attribut.

b) Les objets sont normalement identifiés par un identificateur numérique ou par un nom d'objet, ou les deux. Dans les modèles de classe, ces attributs clés sont définis sous l'attribut clé.

c) Le numéro de ligne identifie la séquence et le niveau d'emboîtement de la ligne. Chaque niveau d'emboîtement est identifié par période. L'emboîtement spécifie

i) les champs d'un attribut structuré (4.1, 4.2, 4.3),

ii) les attributs dépendant d'une déclaration de contrainte (5). Les attributs peuvent être obligatoires (5.1) ou optionnels (5.2) si la contrainte est vraie. Contrairement à l'attribut défini en (5.2), les attributs optionnels ne nécessitent pas systématiquement de déclaration de contrainte.

iii) les champs de sélection d'un attribut de type de choix (6.1 et 6.2).

(6) L'étiquette "SERVICES" indique que les entrées suivantes sont des services définis pour la classe.

a) Un (m) dans la colonne 2 indique le service est obligatoire (*mandatory*) pour la classe, le (o) indiquant qu'il est optionnel. Un (c) dans cette colonne indique que le service est conditionnel. Lorsque tous les services définis pour une classe sont définis comme étant optionnels, au moins un service doit être sélectionné lorsque l'on définit une instance de la classe.

b) L'étiquette "OpsService" désigne un service opérationnel (1).

c) L'étiquette "MgtService" désigne un service de gestion (2).

d) Le numéro de ligne identifie la séquence et le niveau d'emboîtement de la ligne. Chaque niveau d'emboîtement est identifié par période. L'emboîtement dans la liste de services spécifie les services dépendant d'une déclaration de contrainte.

3.3.4 Conventions pour les définitions de service

3.3.4.1 Généralités

Le modèle de service, les primitives de service et les schémas de séquence de temps utilisés sont des descriptions entièrement abstraites; elles ne correspondent pas à une spécification de mise en œuvre.

3.3.4.2 Paramètres de service

Les primitives de service sont utilisées pour représenter les interactions entre l'utilisateur du service et le fournisseur du service (ISO/CEI 10731). Elles transmettent des paramètres qui indiquent les informations disponibles dans l'interaction utilisateur/fournisseur. Dans une interface particulière, il n'est pas nécessaire de déclarer explicitement tous les paramètres.

Les spécifications de service de la présente norme décrivent les paramètres qui composent les primitives de service ASE au moyen d'un format tabulaire. Les paramètres qui s'appliquent à chaque groupe des primitives de service sont présentés dans des tableaux. Chaque tableau comprend jusqu'à cinq colonnes donnant:

1) le nom du paramètre,
2) la primitive de demande,
3) la primitive d'indication,
4) la primitive de réponse et
5) la primitive de confirmation.

Un paramètre (ou un composant de celui-ci) est répertorié sur chaque ligne de chacun des tableaux. Dans les colonnes des primitives de service appropriées, un code indique le type d'utilisation du paramètre pour la primitive spécifiée dans la colonne:

M Paramètre obligatoire pour la primitive.

U Option de l'utilisateur; ce paramètre peut ou non être indiqué, en fonction de l'utilisation dynamique par l'utilisateur du service. Si ce paramètre n'est pas indiqué, il peut prendre une valeur par défaut.

C Paramètre dépendant d'autres paramètres ou de l'environnement de l'utilisateur du service.

— (vide) Paramètre jamais spécifié.

S Paramètre qui correspond à un élément sélectionné.

Certaines entrées sont également qualifiées par des éléments entre parenthèses. Il peut s'agir

a) d'une contrainte spécifique à un paramètre:

"(=)" indique que le paramètre équivaut sur le plan sémantique au paramètre de la primitive de service se trouvant immédiatement à sa gauche dans le tableau.

b) d'une indication qu'une note s'applique à l'entrée:

"(n)" indique que la note "n" qui suit contient des informations supplémentaires concernant le paramètre et son utilisation.

3.3.4.3 Procédures de service

Les procédures sont définies en termes:

- des interactions entre entités d'application par échange d'unités de données de protocole d'application de bus de terrain, et
- des interactions entre le fournisseur de service de couche application et un utilisateur de service de couche application dans le même système par appel de primitives de service de couche.

Ces procédures sont applicables aux instances de communication entre systèmes supportant les services de communication à contrainte de temps propres à la couche application de bus de terrain.

4 Concepts

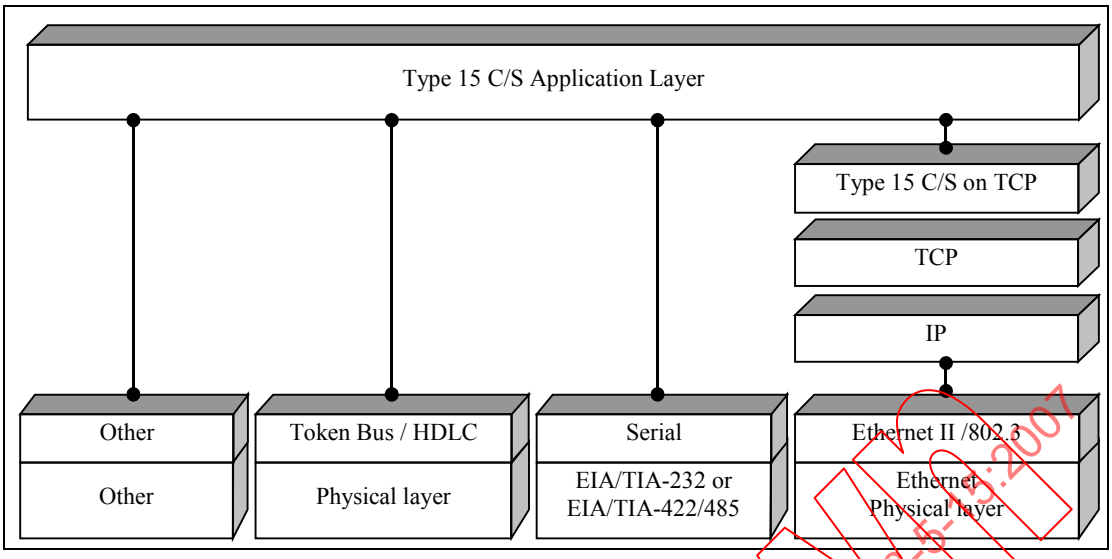
4.1 Concepts communs

La totalité de l'Article 9 de la CEI/TR 61158-1 est incorporée par référence, sauf mentions contraires spécifiques en 4.2 et 4.3.

4.2 Concepts spécifiques au client/serveur

4.2.1 Vue d'ensemble

Client/serveur (C/S) désigne un mécanisme de communication client/serveur entre les dispositifs raccordés à différents types de bus ou de réseaux. La couche application client/serveur est la même et ne dépend pas des couches sous-jacentes. Certaines configurations de piles client/serveur sont illustrées sur Figure 1.

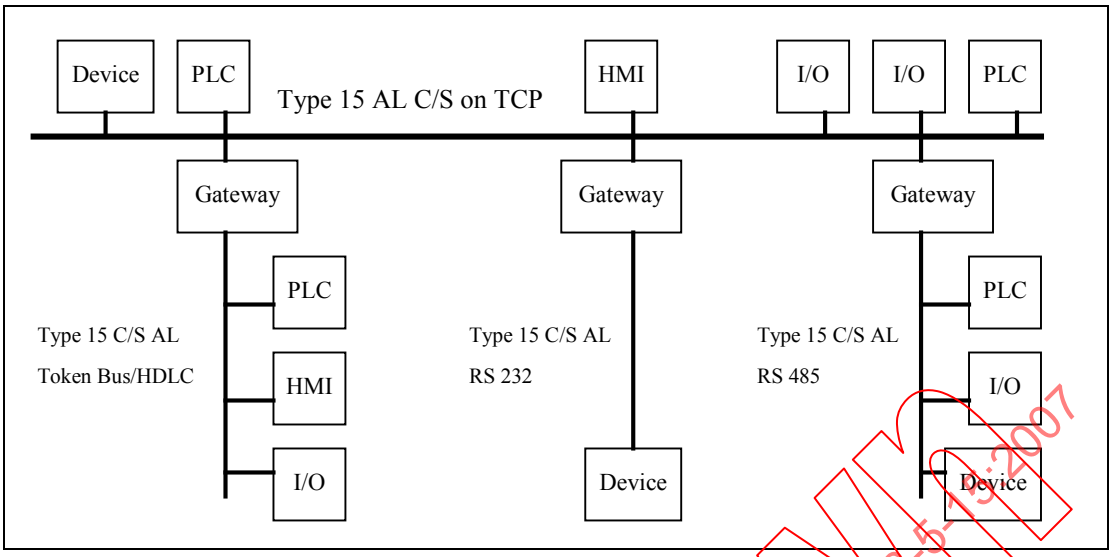


Légende

Anglais	Français
Type 15 C/S Application Layer	Couche application C/S de type 15
Type 15 C/S on TCP	C/S de type 15 sur TCP
TCP	TCP
IP	IP
Other	Autre
Token Bus / HDLC	Bus à jetons / HDLC
Physical layer	Couche physique
Serial	Série
EIA/TIA-232 or EIA/TIA-422/485	EIA/TIA-232 ou EIA/TIA-422/485
Ethernet II /802.3	Ethernet II /802.3
Ethernet Physical layer	Couche physique Ethernet

Figure 1 – Piles client/serveur

La communication entre dispositifs sur différents types de bus ou de réseaux est rendue possible par l'utilisation de passerelles, comme illustré sur la Figure 2.



Légende

Anglais	Français
Device	Dispositif
PLC	Automate programmable
Type 15 AL C/S on TCP	C/S AL de type 15 sur TCP
HMI	HMI
I/O	E/S
Gateway	Passerelle
Type 15 C/S AL	AL C/S de type 15
Token Bus/HDLC	Bus à jetons / HDLC
RS 232	RS 232
RS 485	RS 485

Figure 2 – Communication client/serveur sur différents bus ou réseaux

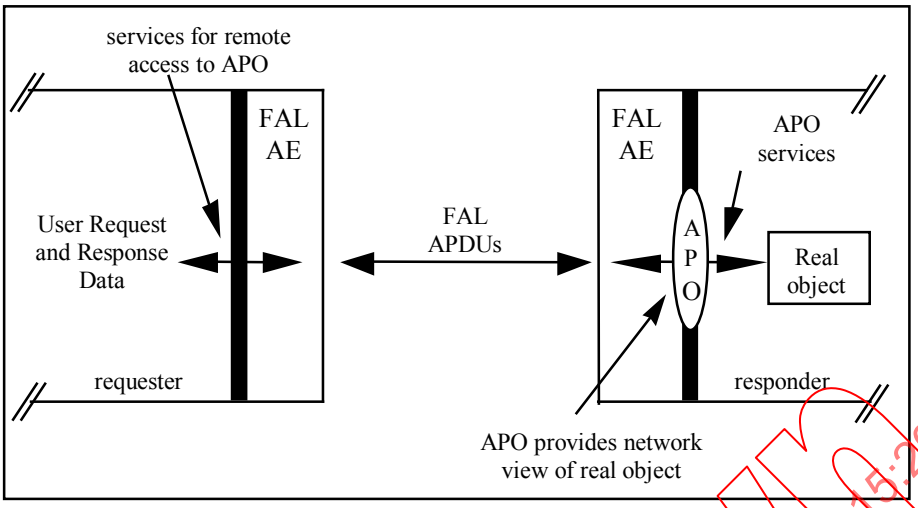
4.2.2 APO client/serveur

4.2.2.1 Généralités

Le mode client/serveur fournit des services de couche application spécifiés par des codes de fonction, ce qui permet d'agir sur des objets de processus d'application (APO - application process objects).

Un objet de processus d'application est une représentation réseau d'un aspect spécifique d'un AP. Chaque APO représente un ensemble spécifique d'informations et de capacités de traitement d'un AP, qui sont accessibles par les services de la FAL. On utilise les APO pour représenter ces capacités pour d'autres AP d'un système de bus de terrain.

Vu de la FAL, un APO est modélisé sous forme d'objet accessible en réseau contenu dans un AP ou dans un autre APO (les APO pouvant contenir d'autres APO). Les APO fournissent la définition réseau des objets contenus dans un AP qui sont accessibles à distance. La définition d'un APO comprend une identification des services de la FAL qui peuvent être utilisés par des AP distants pour l'accès distant. Les services de la FAL, représentés sur la Figure 3, sont fournis par l'entité de communication de FAL de l'AP, appelée entité d'application de FAL (AE de FAL).



Légende

Anglais	Français
service for remote access to APO	service pour l'accès distant à l'APO
FAL AE	AE de FAL
User Request and Response Data	Demande utilisateur et données de réponse
requester	demandeur
FAL APDUs	APDU de FAL
APO services	Services d'APO
APO	APO
Real object	Objet réel
responder	répondeur
APO provides network view of real object	L'APO fournit une vue réseau de l'objet réel

Figure 3 – Services d'APO client/serveur transmis par la FAL

Sur la Figure 3, les AP distants agissant comme clients peuvent accéder à l'objet réel en envoyant des demandes par l'intermédiaire de l'APO qui représente l'objet réel. Les aspects locaux de l'AP font la conversion entre la vue réseau (l'APO) de l'objet réel et la vue d'AP interne de l'objet réel.

Pour tous les APO, la sémantique est celle autorisée par l'ASE d'APO de la FAL.

4.2.2.2 APO communs

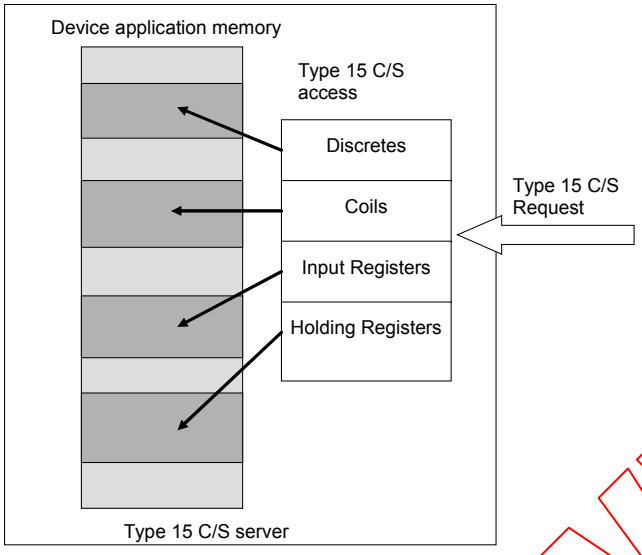
Le mode client/serveur base son modèle sur des APO qui ont des caractéristiques distinctives. Certains d'entre eux, à savoir les discrets, les bobines, les registres d'entrée et les registres de maintien, sont souvent interprétés comme des tableaux de mémoire. Ce modèle, simple et commode, n'est qu'un modèle parmi tant d'autres, et il n'est même pas nécessaire que ces objets soient vus comme des objets de mémoire. C'est au concepteur de l'application qu'il incombe de concevoir la relation la plus appropriée entre les APO et les objets réels, ce sujet ne faisant pas partie du domaine d'application du présent document. Le Tableau 1 résume les caractéristiques de ces quatre APO.

Tableau 1 – APO client/serveur communs

APO communs	Type d'objet	Note
discret	bit unique	En lecture seule; sa valeur peut être fournie par un système d'E/S. Cet APO est utile pour la modélisation d'objets réels à valeurs binaires qui sont manipulés par l'application serveur et ne sont censés être observés que par l'utilisateur client. L'intégrité du contrat ci-dessus est sous le contrôle de l'AP serveur, qui peut limiter l'exposition des objets réels à des discrets.
bobine	bit unique	Lecture-écriture; sa valeur peut être modifiée par un programme d'application client
registre d'entrée	mot de 16 bits	En lecture seule; sa valeur peut être fournie par un système d'E/S. Cet APO est utile pour la modélisation d'objets réels à valeurs analogiques qui sont manipulés par l'application serveur et ne sont censés être observés que par l'utilisateur client. L'intégrité du contrat ci-dessus est sous le contrôle de l'AP serveur, qui peut limiter l'exposition des objets réels à des registres d'entrée.
registre de maintien	mot de 16 bits	Lecture-écriture; sa valeur peut être modifiée par un programme d'application client

Les distinctions entre les entrées et les sorties, ainsi qu'entre les éléments de données adressables par bits ou adressables par mots, n'impliquent pas de comportement d'application. Il est parfaitement acceptable, et même habituel, de considérer que les quatre tableaux se chevauchent les uns les autres, s'il s'agit là de leur interprétation la plus naturelle sur la machine cible en question.

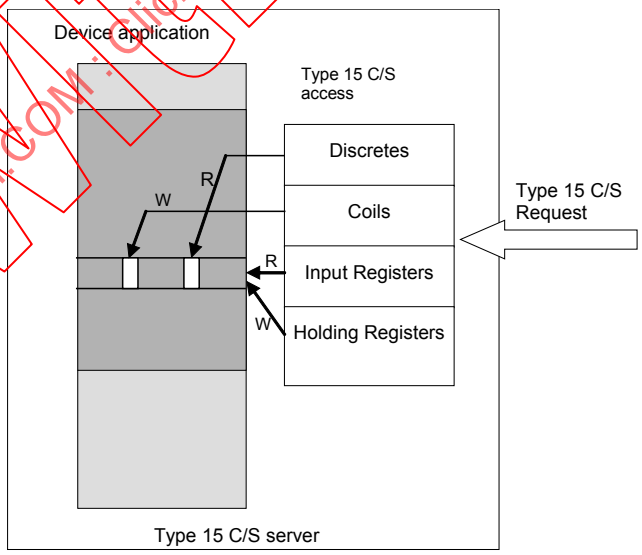
La Figure 4 et la Figure 5 qui suivent, qui sont des figures informatives, donnent certaines interprétations communes, bien qu'en aucun cas exhaustives, en tant que tableaux de mémoire distincts et tableaux de mémoire chevauchants.



Légende

Anglais	Français
Device application memory	Mémoire d'application du dispositif
Type 15 C/S access	Accès C/S de type 15
Discretes	Discrets
Coils	Bobines
Input Registers	Registres d'entrée
Holding Registers	Registres de maintien
Type 15 C/S server	Serveur C/S de type 15
Type 15 C/S Request	Demande C/S de type 15

Figure 4 – [INFORMATIVE] Interprétation en tant que tableaux distincts

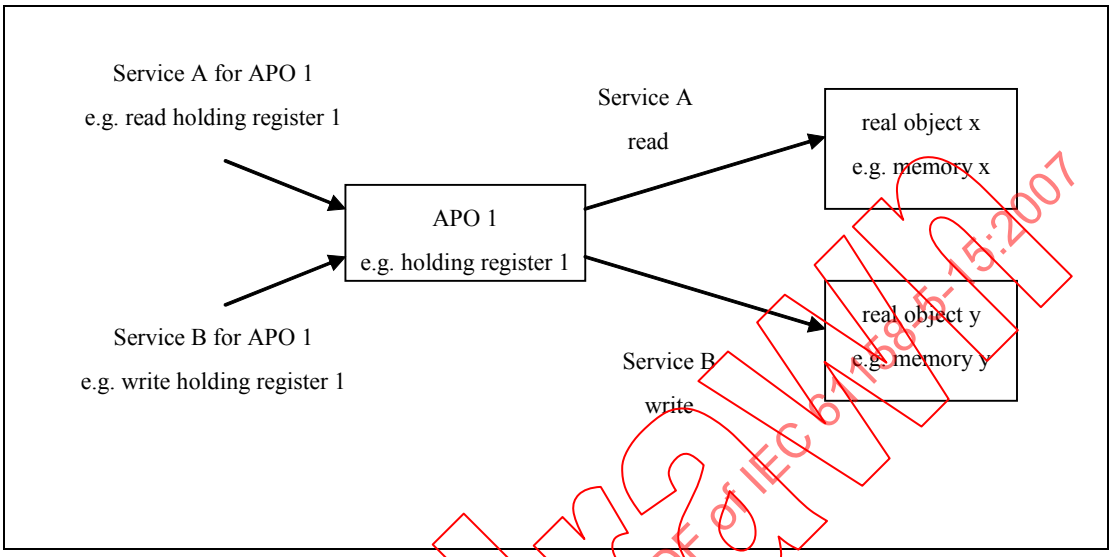


Légende

Anglais	Français
R	Lecture
W	Écriture

Figure 5 – [INFORMATIVE] Interprétation en tant que tableaux chevauchants

Pour souligner ce qui n'est pas implicite, il convient de mentionner une autre interprétation (informative) qui n'a rien d'évident, mais que l'on rencontre fréquemment sur le terrain: différentes demandes effectuées sur le même APO n'entraînent pas nécessairement des demandes vers le même objet réel. Cela est illustré sur la Figure 6.



Légende

Anglais	Français
Service A for APO 1 e.g. read holding register 1	Service A pour APO 1 par ex. lecture du registre de maintien 1
Service B for APO 1 e.g. write holding register 1	Service B pour APO 1 par ex. écriture du registre de maintien 1
APO 1 e.g. holding register 1	APO 1 par ex. registre de maintien 1
Service A	Service A
read	lecture
Service B	Service B
write	écriture
real object x e.g. memory x	objet réel x par ex. mémoire x
real object y e.g. memory y	objet réel y par ex. mémoire y

Figure 6 – [INFORMATIVE] APO et objets réels, interprétation possible non évidente

Les discrets, les bobines, les registres d'entrée et les registres de maintien sont souvent appelés collectivement des références de données ou des éléments de données. Pour chacun des tableaux mentionnés ci-dessus, le protocole autorise la sélection individuelle de 65536 éléments de données, et les opérations de lecture et d'écriture de ces éléments sont conçues pour s'étendre, du point de vue de l'utilisateur de l'application (utilisateur client), sur plusieurs éléments de données consécutifs, jusqu'à une limite de taille de données qui dépend du code de fonction des transactions de service.

L'association possible des références de données client/serveur (bits, registres) et le stockage physique réel dans les dispositifs est un problème local.

Les adresses des références de données logiques client/serveur, utilisées dans les codes de fonction de service client/serveur, sont des entiers non signés commençant à zéro.

Une source potentielle de confusion est la relation entre les adresses des références de données utilisées dans les codes de fonction client/serveur et les adresses des références "utilisateur" de type discret, bobine ou registre, telles que celles que l'on trouve, par exemple, dans les programmes de logique en échelle (ladder) des automates programmables. Pour des raisons historiques, les adresses des références utilisateur étaient exprimées en nombres décimaux avec un décalage de départ de 1, tandis que le client/serveur utilise la convention logicielle naturelle du point de départ à 0. De ce fait, par exemple, un message client/serveur demandant de lire un registre par l'intermédiaire d'une référence de données logique au point de décalage 0 renverra ce que le programmeur de l'application considère comme étant le registre 1. Il existe une exception, l'adressage des enregistrements par l'intermédiaire des registres au moyen des codes de fonction de service client/serveur Read File Record et Write File Record, dans lequel l'enregistrement 0 est une référence "utilisateur" dont l'adresse est spécifiée avec 0 comme premier registre. Les détails sont disponibles dans les spécifications des services mentionnées.

4.2.2.3 APO d'enregistrement et de fichier

L'APO d'enregistrement, du point de vue de l'utilisateur de l'application (utilisateur client), est un ensemble de registres contigus d'un type spécifié, caractérisé par l'adresse du premier registre et la quantité de registres; dans le contexte de cette définition, les registres concernés sont également appelés des références.

L'APO de fichier est une organisation d'enregistrements caractérisée par un nombre non signé.

4.2.2.4 APO d'interface

L'interface encapsulée est un mécanisme servant à encapsuler un service pour une interface, ce mécanisme constituant un objet de processus d'application caractérisé par un type MEI. Le type MEI, spécifié sous forme de valeur d'octet, sert à expédier le service à l'interface appropriée.

C'est un mécanisme simple, puisqu'il remet toute la sémantique à l'interface, mais il offre une flexibilité maximale dans ce qu'il peut représenter et transmettre.

4.2.3 Codes de fonction client/serveur

Les codes de fonction sont des codages de services demandés à un serveur. Ces codages se répartissent en trois catégories:

Codes de fonction attribués publiquement

Ces codes de fonction sont soit affectés à un service normalisé, soit réservés à une affectation future.

Codes de fonction définis par l'utilisateur

Ces codes de fonction peuvent être utilisés à des fins d'expérimentation dans un environnement de laboratoire contrôlé. Il ne faut pas les utiliser dans un environnement ouvert.

Codes de fonction réservés

Ces codes de fonction sont actuellement utilisés par certaines entreprises pour les produits patrimoniaux; ils ne sont pas disponibles pour une utilisation publique.

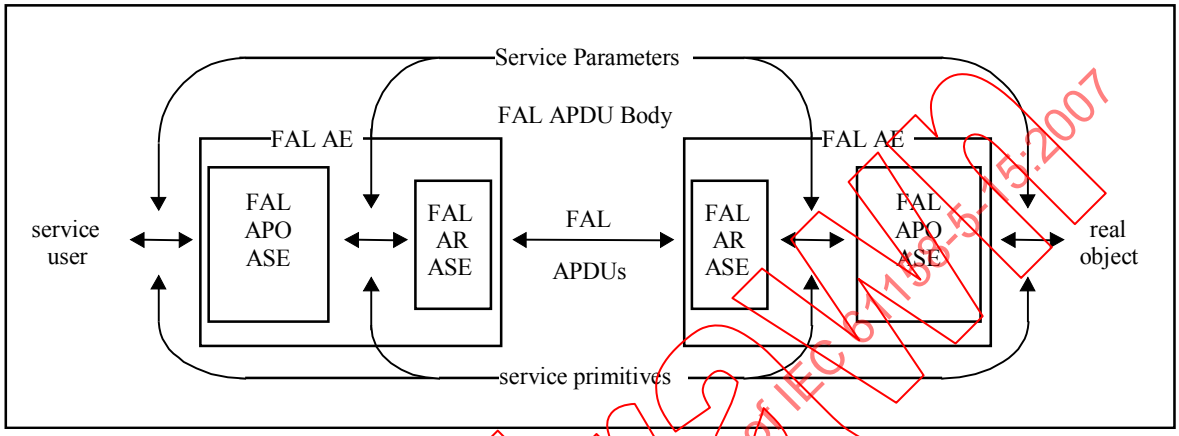
NOTE 1 Les affectations de code de fonction sont gérées par le consortium industriel Modbus-IDA.

NOTE 2 Les codes de fonction suivants, bien qu'affectés publiquement, ne sont pas couverts par la présente spécification: FC 0x07 (Read Exception Status – lire statut d'exception), FC 0x08 (Diagnostics), FC 0x0B (Get Com Event Counter – obtenir compteur d'événements de communication), FC 0x0C (Get Com Event Log – obtenir journal d'événements de communication), FC 0x11 (Report Slave ID – rapporter ID esclave).

4.2.4 Présentation du modèle de communication client/serveur

4.2.4.1 Généralités

Figure 7 est une représentation du modèle de communication de bout en bout.



Légende

Anglais	Français
Service Parameters	Paramètres des services
service user	utilisateur de service
FAL AE	AE de FAL
FAL APDU Body	Corps d'APDU de FAL
FAL APO ASE	ASE d'APO de FAL
FAL AR ASE	ASE d'AR de FAL
FAL APDUs	APDU de FAL
service primitives	primitives de service
real object	objet réel

Figure 7 – Transmission des services d'ASE

Le client/serveur utilise un AR client/serveur avec interaction confirmée et, en option si la diffusion est supportée, un AR client/serveur avec interaction non confirmée.

4.2.4.2 Adressage et connexion de dispositif client/serveur

Les dispositifs client/serveur sont adressés au moyen d'un ID d'unité. L'ID d'unité doit être unique parmi tous les serveurs adressables par un client. L'ensemble des serveurs adressables est déterminé par la couche sous-jacente. On appelle parfois cet ensemble une connexion.

L'affectation des ID d'unité ne fait pas partie du domaine d'application de la présente spécification.

L'ID d'unité identifie les dispositifs logiques. Il peut exister plus d'un dispositif logique par dispositif physique.

Certaines valeurs d'ID d'unité sont réservées et ont des significations particulières. La valeur 0 est réservée pour la diffusion.

NOTE En général, l'ID d'unité n'est obligatoire que pour les dispositifs logiques jouant le rôle de serveurs. Les dispositifs logiques peuvent souvent avoir l'un des deux rôles ou plusieurs rôles, définis par une configuration, leur ID d'unité n'étant pas utilisé s'ils jouent uniquement le rôle de client. Selon les couches sous-jacentes, certains dispositifs peuvent avoir simultanément un rôle de client et un rôle de serveur sur le même point d'accès; c'est le cas du client/serveur sur bus à jetons/HDLC, qui ne fait pas partie du domaine d'application de la présente norme.

4.2.4.3 Cardinalité, connexions et objets de transaction en client/serveur

Un dispositif client/serveur peut comporter de nombreux clients et de nombreux serveurs, selon le dispositif et les couches sous-jacentes.

Les clients et les serveurs sont associés par une connexion qui, dans le domaine d'application de la présente norme, est l'équivalent d'une AR.

Les clients portent un service sur une connexion avec l'aide d'un objet de transaction, géré (créé et détruit) par le client. Le mécanisme d'objet de transaction est exposé au niveau de la couche application, du fait qu'en client/serveur, plusieurs demandes peuvent être actives à la fois, ce qui rend nécessaire d'associer correctement les demandes et les confirmations. Il contrôle également le nombre maximum de ces demandes, qui peut être égal à 1. Les capacités d'une couche application client/serveur dépendent des couches inférieures et de la mise en œuvre utilisée; ces facteurs sont capturés dans la configuration du mécanisme d'objet de transaction pour permettre l'adaptation par programme.

L'objet de transaction est instancié pour la durée d'un appel de service. Chaque objet de transaction doit être identifiable de manière unique dans une connexion.

Il peut y avoir de nombreux serveurs sur la même connexion. Il peut y avoir également de nombreux clients sur la même connexion, mais dans ce cas il ne peut y avoir qu'un seul client actif à la fois au niveau de cette connexion. Pour les services confirmés, un client est actif entre l'instant où il envoie une demande et l'instant où il reçoit la confirmation correspondante, et ce temps est contenu dans l'instanciation et la destruction consécutive de l'objet de transaction concerné. Pour les services non confirmés, un client est actif pendant une durée qui est spécifique au dispositif et à la connexion, et il faut qu'il attende de lui-même l'expiration de la durée de l'activité précédente avant de s'engager dans une nouvelle activité. Ni la limitation à un seul client actif à la fois dans une connexion, ni l'ordonnancement des activités des services non confirmés ne font partie du domaine d'application de la présente spécification.

Un client ne peut émettre qu'un seul appel de service par objet de transaction. En fonction du dispositif, il peut n'exister qu'une seule connexion ou un nombre limité de connexions. Selon la couche sous-jacente et les paramètres locaux, un client peut ne pas être autorisé à instancier plus d'un certain nombre d'objets de transaction à la fois.

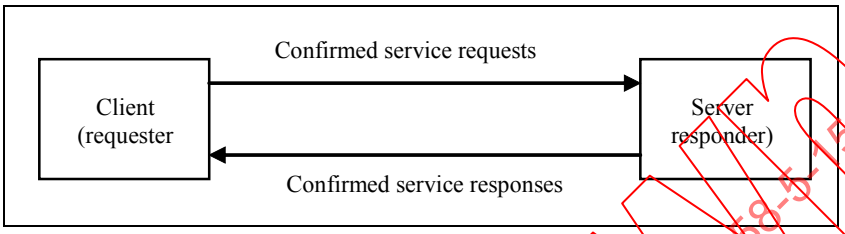
Dans le cas de connexions où il peut y avoir plusieurs clients par connexion, ces clients ne peuvent instancier qu'un objet de transaction à la fois; ils obtiennent le statut d'activité mentionné ci-dessus lorsqu'ils le font. Certaines connexions avec un seul client peuvent présenter les mêmes restrictions.

Si un client ne peut instancier qu'un seul objet de transaction à la fois, alors, lorsqu'il appelle un service, il n'est pas nécessaire de créer dynamiquement un objet de transaction et de l'échanger avec les couches inférieures, puisque la raison principale de l'existence de l'objet de transaction est de permettre le couplage correct des demandes et les confirmations, ce qui est automatique pour ces clients. Dans ce type de situation, un objet de transaction statique individuel joue effectivement le rôle de jeton client, lequel est soit pris soit disponible, et n'intéresse que le client.

Dans tous les cas, l'établissement d'une connexion et la configuration nécessaire à l'instanciation de l'objet de transaction ne font pas partie du domaine d'application de la présente spécification. Tous deux sont cependant censés être établis correctement.

4.2.4.4 Interaction confirmée

Le client/serveur repose sur des messages demande/réponse, le modèle étant celui d'une interaction client/serveur. Dans cette relation, un client envoie à un serveur une demande constituant un service confirmé, et la confirmation de réponse reçue du serveur met fin à la transaction. Cette interaction est représentée à la Figure 8.



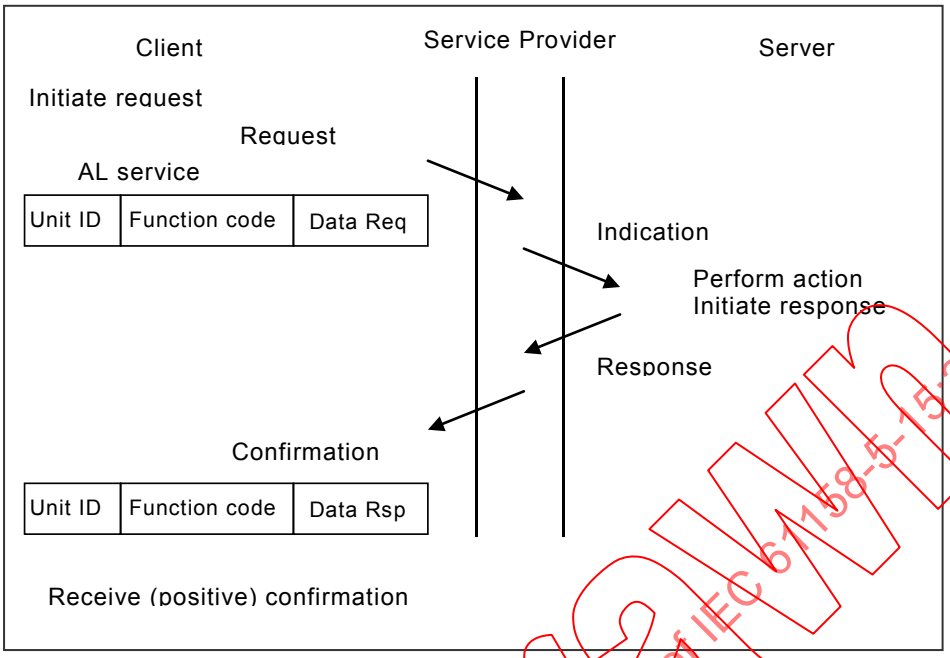
Légende

Anglais	Français
Confirmed service requests	Demandes de service confirmé
Client (requester	Client (demandeur
Server responder)	Serveur répondeur)
Confirmed service responses	Réponses de service confirmé

Figure 8 – Interaction confirmée client/serveur

Elle s'accomplit au moyen des primitives d'AR client/serveur, comme représenté sur la Figure 9 et la Figure 10.

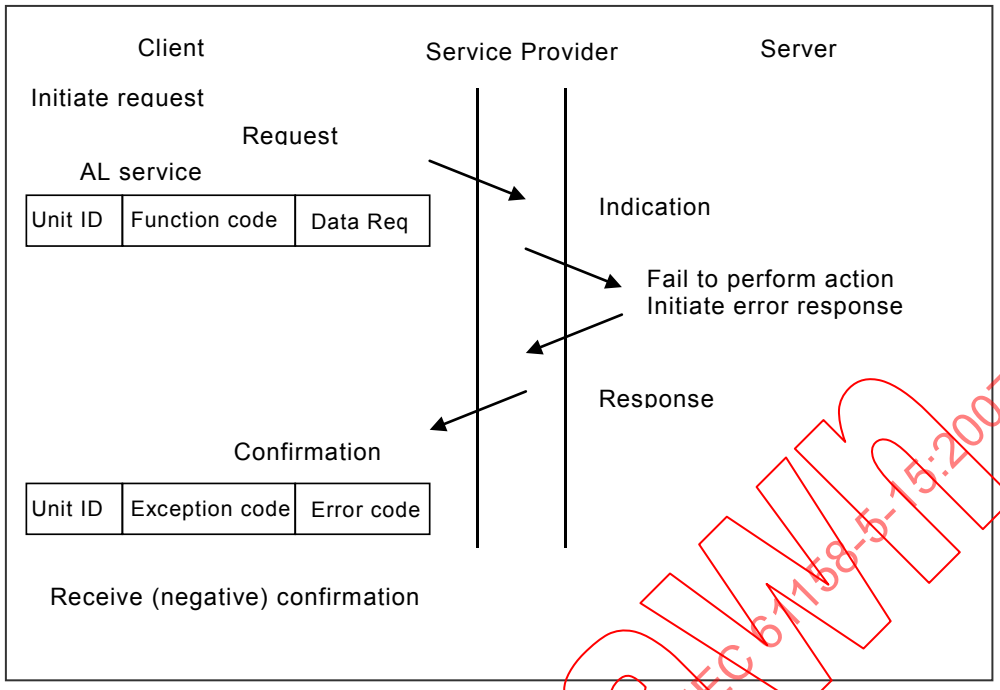
Il convient que le client lance un temporisateur de transaction après avoir émis une demande de service confirmé. Il convient que ce temporisateur soit réglé sur la durée maximale autorisée pour la confirmation, cette durée dépendant de la mise en œuvre, et que ce temporisateur se remette à zéro lors de la réception ou qu'il envoie une erreur de fournisseur de service en cas d'expiration.



Légende

Anglais	Français
Client	Client
Service Provider	Fournisseur de service
Server	Serveur
Initiate request	Initiation de la demande
Request	Demande
AL service	Service d'AL
Unit ID	ID d'unité
Function code	Code de fonction
Data Req	Demande de données
Indication	Indication
Perform action	Exécution de l'action
Initiate response	Initiation de la réponse
Response	Réponse
Confirmation	Confirmation
Data Rsp	Réponse de données
Receive (positive) confirmation	Réception de confirmation (positive)

Figure 9 – Primitives de service confirmé d'AR client/serveur (cas positif)



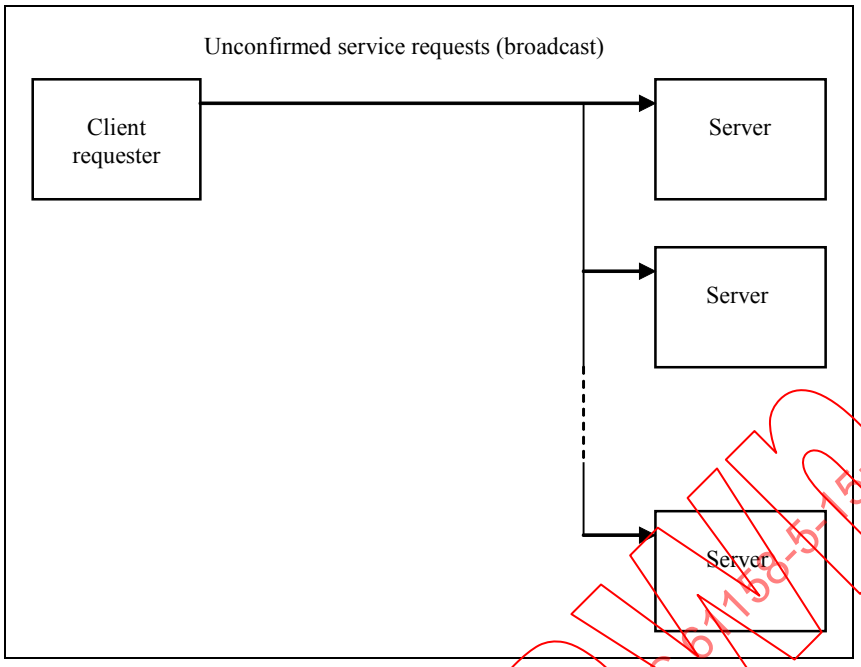
Légende

Anglais	Français
Exception code	Code d'exception
Error code	Code d'erreur
Fail to perform action	Échec de l'exécution de l'action
Initiate error response	Initiation d'une réponse d'erreur
Receive (negative) confirmation	Réception de confirmation (négative)

Figure 10 – Primitives de service confirmé d'AR client/serveur (cas négatif)

4.2.4.5 Interaction non confirmée

En option, les clients et les serveurs peuvent supporter des services non confirmés sous forme de messages de diffusion émis par les clients. Cette interaction est représentée à la Figure 11.

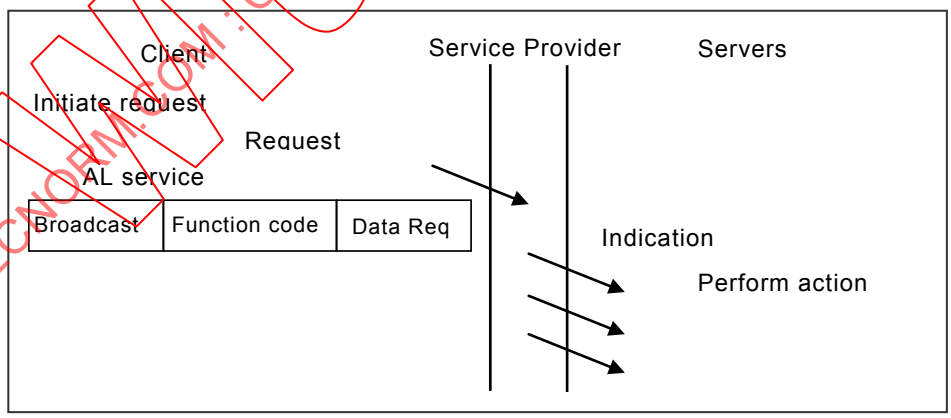


Légende

Anglais	Français
Unconfirmed service requests (broadcast)	Demande de service non confirmé (diffusion)
Client requester	Demandeur client
Server	Serveur

Figure 11 – Interaction non confirmée client/serveur

Elle s'accomplit au moyen des primitives d'AR client/serveur, comme représenté sur la Figure 12.



Légende

Anglais	Français
Client	Client
Service Provider	Fournisseur de service
Servers	Serveurs
Initiate request	Initiation de la demande
Request	Demande

Anglais	Français
AL service	Service d'AL
Broadcast	Diffusion
Function code	Code de fonction
Data Req	Demande de données
Indication	Indication
Perform action	Exécution de l'action

Figure 12 – Primitives de service non confirmé d'AR client/serveur

4.3 Concepts spécifiques au fournisseur/abonné

4.3.1 Vue d'ensemble

Le mode fournisseur/abonné (P/S - Publish/Subscribe) correspond à un mécanisme de communication fournisseur/abonné entre dispositifs en réseau. Ce mode présente un certain nombre de caractéristiques de QoS et est conçu avec des fonctions d'extensibilité à compatibilité ascendante, de telle sorte que, entre autres choses, l'ensemble QoS peut être facilement augmenté au moyen du contrôle de version ou d'extensions spécifiques au vendeur. La présente spécification indique les possibilités d'extension existantes, mais sans les détailler car elles ne font pas partie du domaine d'application du présent document. Il est important de comprendre que le fournisseur/abonné est un protocole point à point. Par configuration et en utilisant les extensions, on peut en exploiter les nombreuses possibilités, et tous les comportements ne sont pas documentés dans la présente spécification.

NOTE Une couche de services d'application d'un niveau plus élevé, dictant l'utilisation optimale et tirant parti d'extensions "intégrées" faciles à mettre en place pour le fournisseur/abonné, a été normalisée récemment par l'OMG: "Data Distribution Service for Real-Time Systems Specification, Version 1.1, décembre 2005".

L'AL fournisseur/abonné n'impose que des exigences minimes aux couches sous-jacentes, et on peut la déployer en utilisant des décalages de mémoire sur la mémoire partagée ou les ports de l'UDP. Le fournisseur/abonné supporte une grande variété de transports et de QoS de transport. Le protocole est conçu pour pouvoir fonctionner sur les transports multidiffusion et de meilleur effort, tels que UDP/IP, et n'exige du transport que des services très simples. Il suffit en réalité que le transport fournisse un service sans connexion, capable d'envoyer des paquets de meilleur effort. Autrement dit, le transport peut ne pas garantir que chaque paquet va atteindre sa destination ou que les paquets vont être délivrés dans l'ordre. Si nécessaire, le fournisseur/abonné met en œuvre la fiabilité dans le transfert des données et de l'état au-dessus de l'interface de transport. Cela n'empêche pas le fournisseur/abonné d'être mis en œuvre au-dessus d'un transport fiable. Cela permet simplement de supporter une gamme de transports plus étendue.

Les exigences générales que le fournisseur/abonné demande au transport sous-jacent peuvent se résumer de la façon suivante:

— le transport possède une notion généralisée d'adresse de diffusion individuelle (qui doit tenir sur 16 octets);

— le transport possède une notion généralisée de port (qui doit tenir sur 4 octets); il peut s'agir par exemple d'un port UDP, d'un décalage dans un segment de mémoire partagée, etc.;

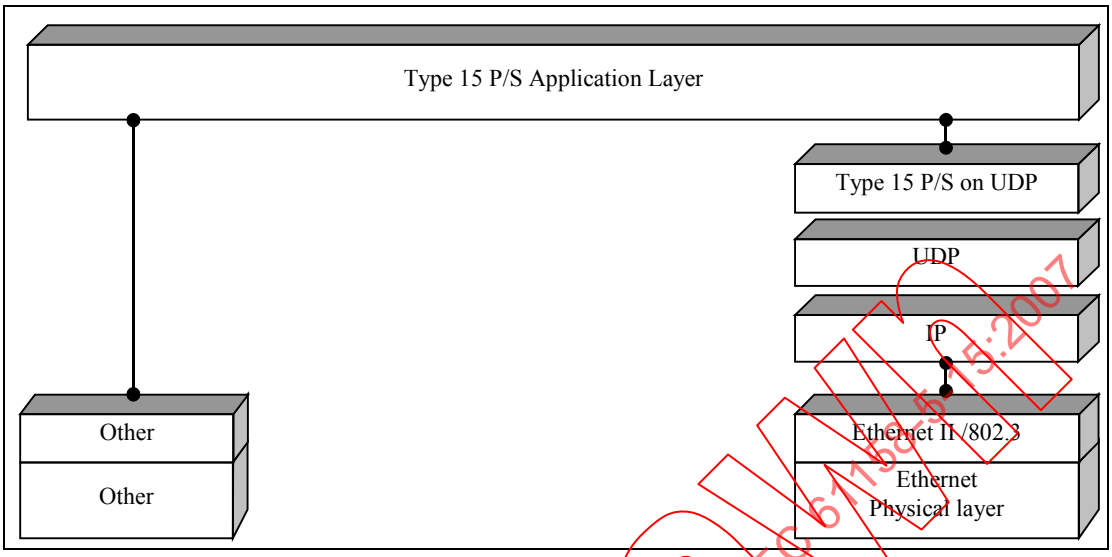
— le transport peut envoyer un datagramme (séquence d'octets non interprétée) à une adresse ou un port spécifique;

— le transport peut recevoir un datagramme à une adresse ou un port spécifique;

— le transport abandonne les messages s'ils sont incomplets ou ont été corrompus durant le transfert (le fournisseur/abonné part du principe que les messages sont complets et non corrompus);

— le transport fournit un moyen de déduire la taille du message reçu.

Une configuration de pile fournisseur/abonné est illustrée à la Figure 13.



Légende

Anglais	Français
Type 15 P/S Application Layer	Couche application P/S de type 15
Type 15 P/S on UDP	P/S de type 15 sur UDP
UDP	UDP
IP	IP
Other	Autre
Ethernet II /802.3	Ethernet II /802.3
Ethernet Physical layer	Couche physique Ethernet

Figure 13 – Piles de communication fournisseur/abonné

Le fournisseur/abonné peut également tirer parti des fonctionnalités multidiffusion du mécanisme de transport, grâce auxquelles un message en provenance d'un envoyeur peut atteindre plusieurs destinataires.

Le fournisseur/abonné est conçu pour favoriser le déterminisme du mécanisme de communication sous-jacent; il offre un compromis ouvert entre déterminisme et fiabilité. Comme d'autres fonctions, le déterminisme est surveillé au moyen d'échéances. La fiabilité est fournie au moyen de files d'attente, d'accusés de réception et de retransmissions. La fiabilité est basée sur les fenêtres; elle est mise en œuvre de façon native et offre un contrôle complet et entièrement flexible du compromis mentionné ci-dessus.

4.3.2 Centrage sur les données, correspondance, découplage et modularité

Le fournisseur/abonné est organisé autour de fournisseurs et d'abonnés, avec la propriété additionnelle d'être centré sur les données.

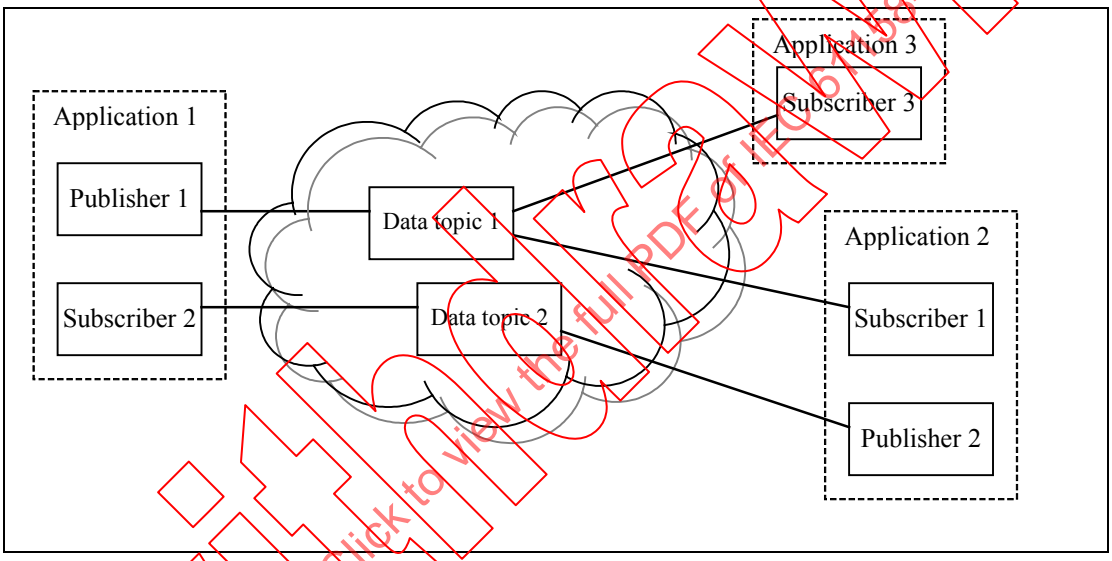
Dans cette architecture, souvent appelée DCPS (data centric publish subscribe - fournisseur/abonné centré sur les données), les attributs d'un échange se trouvent sur les données elles-mêmes. Outre le contenu proprement dit, qui peut être ou non un facteur discriminant, les particularités fonctionnelles d'un fournisseur sont reflétées par les données qu'il fournit, et les attentes de l'abonné sont exprimées par les données auxquelles il s'abonne.

Les considérations ci-dessus conduisent à un degré élevé de personnalisation concernant les données et leur échange, et à une correspondance basée sur les données, parfois même jusqu'aux parties du contenu, sans que les fournisseurs et les abonnés ne se connaissent directement. Ce découplage est la principale raison de l'existence des propriétés de modularité de cette approche, puisqu'il est facile d'ajouter et d'enlever des fournisseurs et des abonnés.

La Figure 14 illustre un scénario dans lequel les applications échangent des données par l'intermédiaire de fournisseurs et d'abonnés, qui ne se connaissent pas les uns les autres.

La correspondance s'effectue en considérant les attributs des données, comme le nom du sujet, le type de sujet et d'autres caractéristiques qualificatives.

NOTE Les extensions fournisseur/abonné permettent de faire appel au concept de "domaines", qui permet d'isoler efficacement les échanges d'applications à l'intérieur des domaines. Le DDS de l'OMG, comme dans "Data Distribution Service for Real-Time Systems Specification, Version 1.1, December 2005", utilise cette extension, mais la fonction n'est pas examinée de façon plus approfondie dans la présente spécification, laquelle ne considère qu'un domaine individuel.



Légende

Anglais	Français
Application	Application
Publisher	Fournisseur
Subscriber	Abonné
Data Topic	Sujet de données

Figure 14 – Échanges fournisseur/abonné centrés sur les données entre objets de réseau découplés

4.3.3 APO fournisseur/abonné

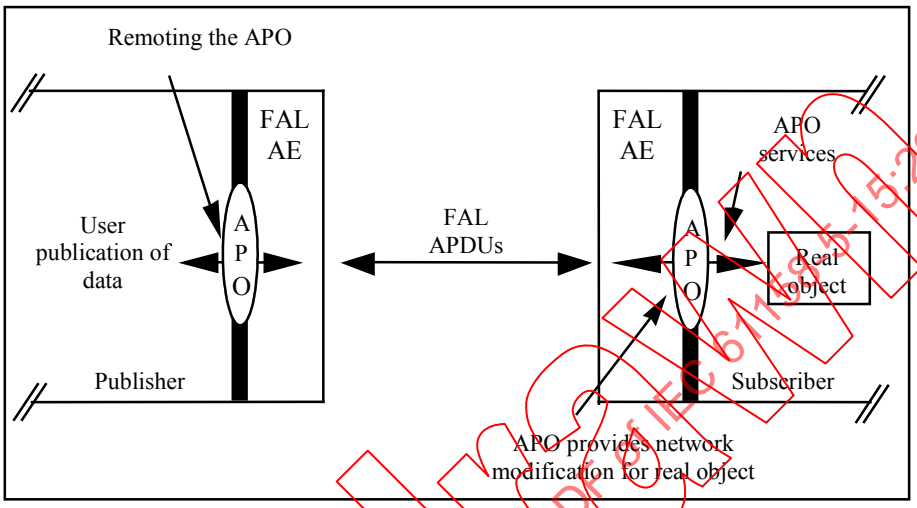
4.3.3.1 Généralités

Un objet de processus d'application est une représentation réseau d'un aspect spécifique d'un AP. Chaque APO représente un ensemble spécifique d'informations et de capacités de traitement d'un AP, qui sont accessibles par les services de la FAL. On utilise les APO pour représenter ces capacités pour d'autres AP d'un système de bus de terrain.

Vu de la FAL, un APO est modélisé sous forme d'objet accessible en réseau contenu dans un AP ou dans un autre APO (les APO pouvant contenir d'autres APO). Les APO fournissent la

définition réseau des objets contenus dans un AP qui sont accessibles à distance. La définition d'un APO comprend une identification des services de la FAL qui peuvent être utilisés par des AP distants pour l'accès distant. Les services de la FAL, représentés sur la Figure 15, sont fournis par l'entité de communication de FAL de l'AP, appelée entité d'application de FAL (AE de FAL).

Les APO fournisseur/abonné sont construits dynamiquement et sont échangés en partant des fournisseurs pour aller vers les abonnés.



Légende

Anglais	Français
Remoting the APO	Éloignement de l'APO
User publication of data	Fourniture de données pour l'utilisateur
FAL AE	AE de FAL
APO	APO
Publisher	Fournisseur
FAL APDUs	APDU de FAL
APO services	Services d'APO
Real object	Objet réel
Subscriber	Abonné
APO provides network modification for real object	L'APO fournit une modification en réseau pour l'objet réel

Figure 15 – Services d'APO fournisseur/abonné transmis par la FAL

Sur la Figure 15, des AP distants agissant comme des fournisseurs peuvent modifier l'objet réel en fournissant des données qui contribuent à modifier l'APO distant. Le fournisseur/abonné éloigne l'APO lui-même en le fournissant. Une fois l'APO acquis par l'abonné, les aspects locaux de l'AP se convertissent par le passage de la vue réseau (l'APO) de l'objet réel à la vue AP interne de l'objet réel.

4.3.3.2 APO du sujet de données

L'APO fournisseur/abonné est constitué des données échangées elles-mêmes, la publication, appelée ici sujet pour souligner le fait qu'il s'agit de données identifiables.

Le sujet est caractérisé par les éléments suivants:

— nom du sujet;

— type de sujet, optionnel;

— code de représentation de sujet, optionnel.

Le sujet de publication proposé est examiné par un sujet d'abonnement attendu, et une tentative de mise en correspondance a lieu. La correspondance est effectuée de la façon suivante sur chacun des trois composants:

— les composants de type nom doivent correspondre, mais aucune correspondance littérale n'est exigée, car on peut faire correspondre des modèles d'expressions habituelles;

— si le type de sujet d'un des côtés mis en correspondance est la chaîne vide, ce composant est une correspondance, la signification additionnelle étant qu'il convient de n'effectuer aucune vérification de type sur l'information de type transportée par ce composant; il est attendu que l'information de type soit disponible par d'autres moyens; si le type de sujet n'est la chaîne vide ni d'un côté ni de l'autre, il doit y avoir correspondance;

— si le code de représentation de sujet d'un des deux côtés mis en correspondance est 0, ce composant est une correspondance, la signification additionnelle étant qu'il n'y a pas d'information supplémentaire disponible à propos du type de sujet et de sa fonction de mise en ordre; il est attendu que l'information de type soit disponible par d'autres moyens; si le code de représentation de sujet n'est égal à 0 ni d'un côté ni de l'autre, il doit y avoir correspondance.

Les sujets de publication proposés et les sujets d'abonnement attendus transportent avec eux des informations additionnelles sur la production et la consommation des données, respectivement, définies par des paramètres optionnels. Ce sont des informations qui permettent de déterminer la QoS et qui sont disponibles au niveau du fournisseur, du sujet et de l'abonné. La nature et la quantité de ces informations additionnelles sont configurables; les informations suivantes (informatives) sont un ensemble d'informations couramment utilisées:

sujet de publication

— sujet (nom, type, code de représentation);

— force;

— persistance.

La force est la priorité de l'information envoyée par la publication; la persistance est la durée pendant laquelle l'information est valide. La force et la persistance permettent au destinataire de déterminer si des informations ont été émises par plusieurs publications en correspondance.

sujet d'abonnement

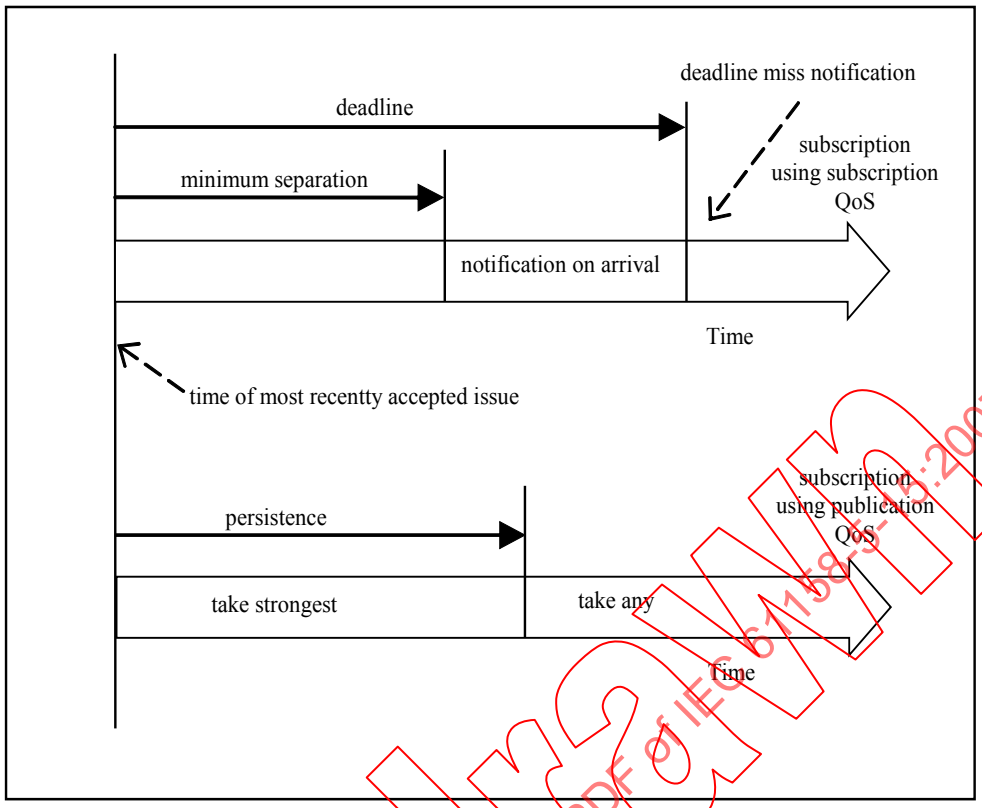
— sujet (nom, type, code de représentation);

— séparation minimum;

— échéance.

La séparation minimum est le temps minimum entre deux informations consécutives reçues par l'abonnement. Elle définit la vitesse maximale à laquelle l'abonnement est capable de recevoir les informations. Il convient que les publications qui envoient des informations à cet abonnement fassent en sorte de le faire de façon à les espacer d'au moins ce temps minimum.

La configuration ci-dessus active les comportements illustrés sur la Figure 16.



Légende

Anglais	Français
deadline	échéance
deadline miss notification	notification d'échéance manquée
minimum separation	séparation minimum
notification on arrival	notification à l'arrivée
subscription using subscription QoS	abonnement utilisant la QoS d'abonnement
Time	Temps
time of most recentty accepted issue	temps des informations les plus récemment acceptées
persistence	persistance
subscription using publication QoS	abonnement utilisant la QoS de fourniture
take strongest	prendre les plus fortes
take any	prendre n'importe lesquelles

Figure 16 – [INFORMATIVE] Exemples de comportements configurables fournisseur/abonné utilisant la QoS

4.3.4 Cardinalité et tolérance des défauts

Les échanges fournisseur/abonné peuvent s'effectuer:

— entre une origine et une destination;

— entre une origine et de nombreuses destinations;

— entre de nombreuses origines et une destination;

— entre de nombreuses origines et de nombreuses destinations;

Les possibilités "entre de nombreuses origines et .." supportent directement une disponibilité élevée avec basculement transparent.

4.3.5 Files d'attente et fiabilité

Les paramètres configurables relatifs aux files d'attente et aux accusés de réception permettent à une application d'avoir un comportement situé entre meilleur effort et fiabilité stricte.

4.3.6 Présentation du modèle de communication fournisseur/abonné

4.3.6.1 Généralités

Les interactions fournisseur/abonné sont basées sur des données/événements envoyés par un AP et destinés à être utilisés par d'autres. Ce modèle constitue ce que l'on appelle les interactions entre fournisseur et abonné.

Les services supportés par un modèle d'interaction sont transmis par des points finaux de relation d'application (des AREP) associés aux AP communicants. Le rôle joué par l'AREP dans l'interaction (client, serveur, pair, fournisseur, abonné) est défini en tant qu'attribut de l'AREP.

Le modèle de communication général architecturé autour des échanges d'APO est en fait celui des interactions entre fournisseur et abonné échangeant des sujets de données, mais l'AL fournisseur/abonné supporte le modèle en utilisant plusieurs services non confirmés. Certaines interactions non confirmées fournisseur/abonné demandent effectivement une confirmation, mais la confirmation est remise à l'utilisateur de l'AL. Du point de vue des communications, il n'existe pas de relation entre appels séparés de service non confirmé, comme il en existe entre la demande et la réponse d'un service confirmé.

NOTE Le DDS de l'OMG, comme dans "Data Distribution Service for Real-Time Systems Specification, Version 1.1, décembre 2005", définit des services qui sont présentés ici comme relevant plutôt de la responsabilité de l'utilisateur de l'AL fournisseur/abonné. Le fournisseur/abonné est un protocole point à point, et en tant que tel une partie des fonctionnalités est remise à l'application de l'utilisateur. On peut le constater à la flexibilité apportée aux mises en œuvre dans lesquelles l'encombrement est une préoccupation majeure, qui contribue à l'omniprésence de cette approche.

4.3.6.2 Interactions entre fournisseur et abonné

Les interactions entre fournisseur et abonné communes impliquent un AP de fournisseur unique et un groupe constitué d'un ou de plusieurs AP d'abonné. Ce type d'interaction a été défini pour supporter des variations des deux modèles d'interaction entre AP, le modèle "tireur" et le modèle "pousseur". Dans ces deux modèles, la configuration de l'AP fournisseur est effectuée par des actions de gestion et ne fait pas partie du domaine d'application de la présente norme.

4.3.6.2.1 Interactions du modèle tireur

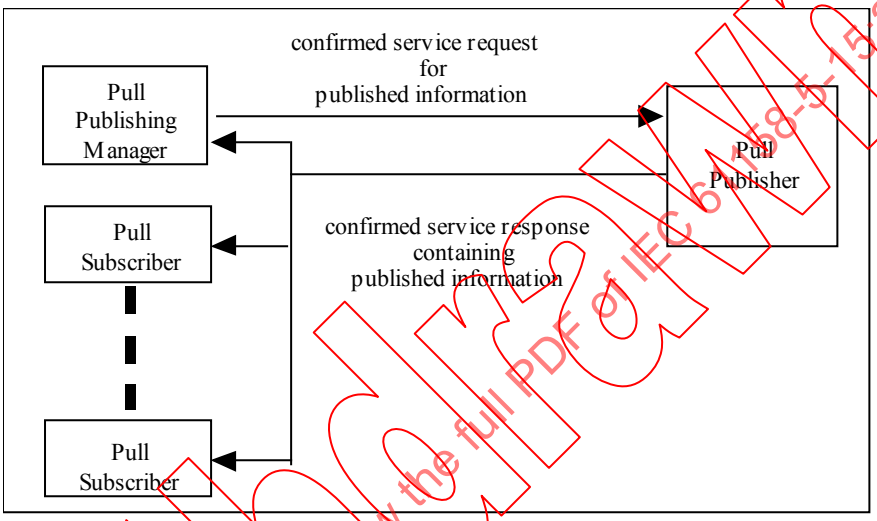
Dans le modèle "tireur", le fournisseur reçoit d'un *gestionnaire* de fourniture distant une demande de fournir, et diffuse (ou multidiffuse) sa réponse à travers le réseau. Le gestionnaire de fourniture a pour seule responsabilité celle d'initier la fourniture en envoyant une demande au fournisseur.

Les abonnés souhaitant recevoir les données fournies écoutent les réponses émises par le fournisseur. C'est ainsi que les données sont "tirées" au fournisseur par des demandes émanant du gestionnaire de fourniture.

Les services de FAL confirmés sont utilisés pour supporter ce type d'interaction. Ce type d'interaction se distingue des autres types d'interaction par deux caractéristiques.

Premièrement, un échange demande/réponse confirmé type a lieu entre le gestionnaire de fourniture et le fournisseur. Cependant, le mécanisme de transmission sous-jacent fourni par la FAL renvoie la réponse non seulement au gestionnaire de fourniture, mais aussi à tous les abonnés souhaitant recevoir les informations fournies. On accomplit cela en faisant en sorte que la couche liaison de données émette la réponse à un groupe d'adresses plutôt qu'à l'adresse individuelle du gestionnaire de fourniture. Par conséquent, la réponse envoyée par le fournisseur contient les données fournies et est multidiffusée au gestionnaire de fourniture et à tous les abonnés.

La deuxième différence concerne le comportement des abonnés. Les abonnés du modèle tireur, appelés abonnés tireurs, sont capables d'accepter les données fournies contenues dans les réponses de service confirmé sans avoir émis la demande correspondante. La Figure 17 illustre ces concepts.



Légende

Anglais	Français
confirmed service request for published information	demande de service confirmé pour l'obtention d'informations fournies
Pull Publishing Manager	Gestionnaire de fourniture tireur
Pull Subscriber	Abonné tireur
Pull Publisher	Fournisseur tireur
confirmed service response containing published information	réponse de service confirmé contenant des informations fournies

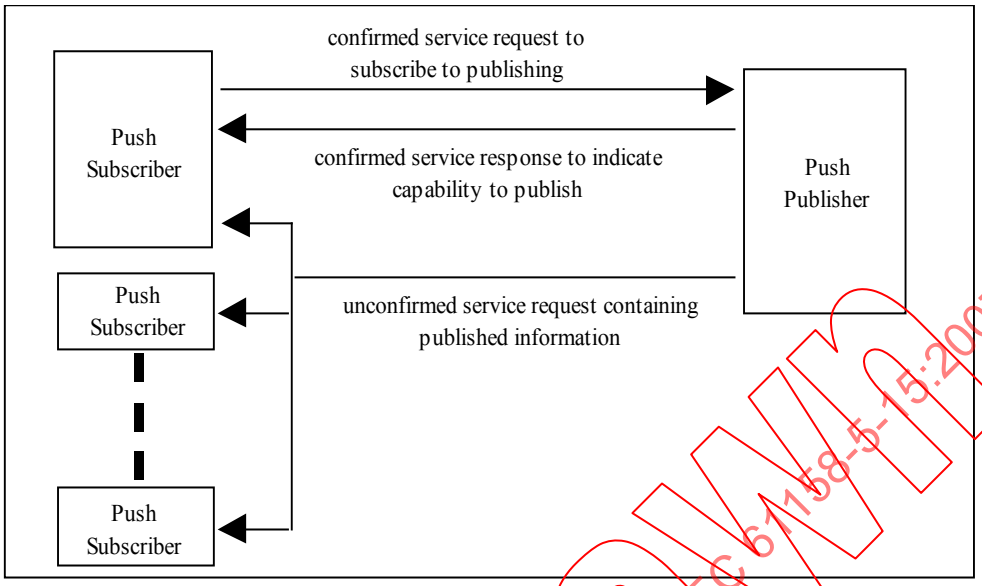
Figure 17 – Interactions du modèle tireur

4.3.6.2.2 Interactions du modèle pousseur

Dans le modèle "pousseur", deux services peuvent être utilisés, le service confirmé et le service non confirmé. L'abonné utilise le service confirmé pour demander à joindre la fourniture. La réponse à cette demande est renvoyée à l'abonné en suivant le modèle d'interaction client/serveur. Cet échange est seulement nécessaire lorsque l'abonné et le fournisseur sont situés dans des AP différents.

Le service non confirmé utilisé dans le modèle pousseur est celui qu'utilise le fournisseur pour distribuer ses informations aux abonnés. Dans ce cas, le fournisseur a la responsabilité d'appeler le service non confirmé correct au moment approprié et de fournir les informations appropriées. Dans ce mode, il est configuré pour "pousser" ses données sur le réseau.

Les abonnés du modèle pousseur reçoivent les services non confirmés fournis, distribués par les fournisseurs. La Figure 18 illustre le concept de modèle pousseur.



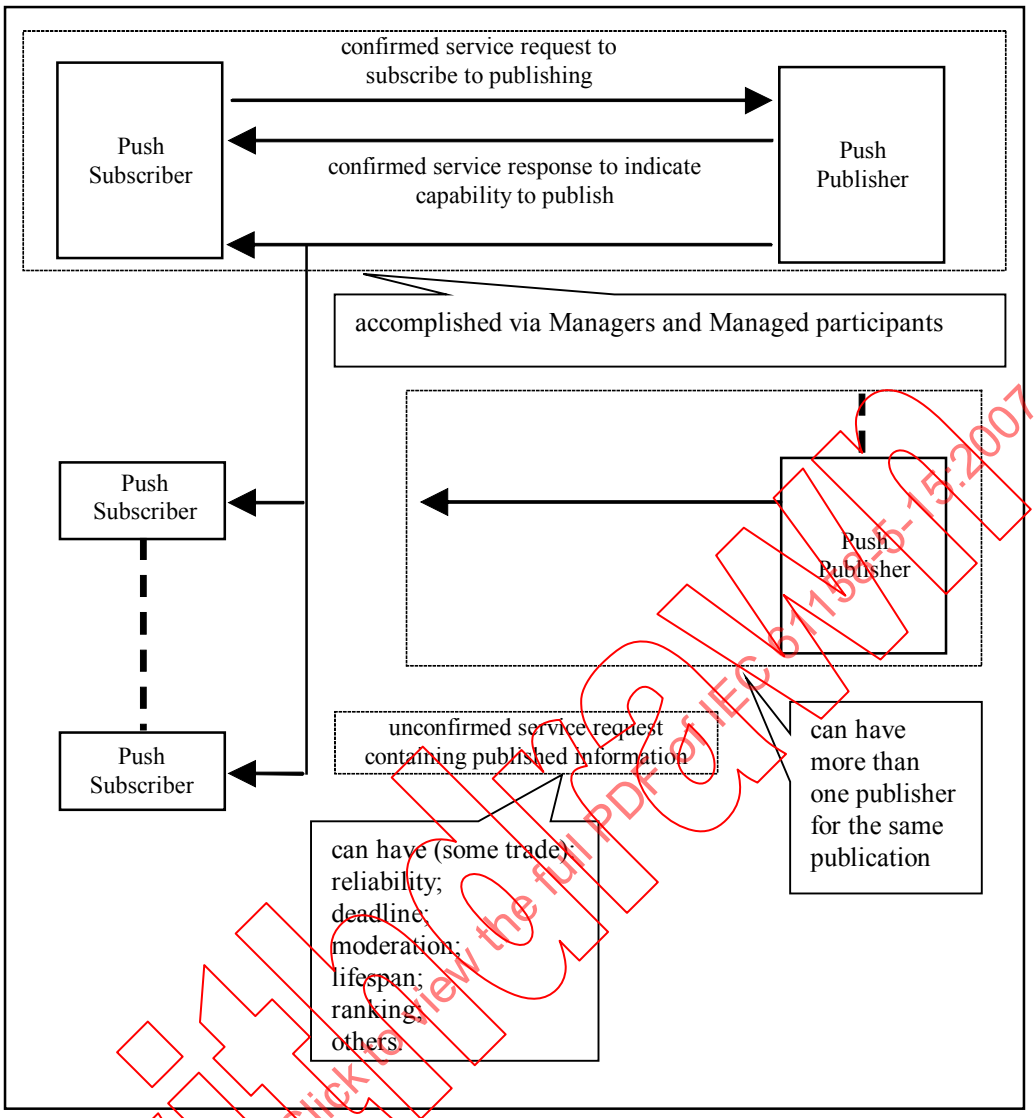
Légende

Anglais	Français
confirmed service request to subscribe to publishing	demande de service confirmé pour s'abonner à la fourniture
Push Subscriber	Abonné pousseur
confirmed service response to indicate capability to publish	réponse de service confirmé pour indiquer la capacité à fournir
Push Publisher	Fournisseur pousseur
unconfirmed service request containing published information	demande de service non confirmé contenant des informations fournies

Figure 18 – Interactions du modèle pousseur

4.3.6.3 Interactions fournisseur/abonné

Les interactions fournisseur/abonné constituent une variante des interactions du modèle pousseur; elles sont illustrées par les légendes de la Figure 19.



Légende

Anglais	Français
confirmed service request to subscribe to publishing	demande de service confirmé pour s'abonner à la fourniture
Push Subscriber	Abonné pousseur
confirmed service response to indicate capability to publish	réponse de service confirmé pour indiquer la capacité à fournir
Push Publisher	Fournisseur pousseur
accomplished via Managers and Managed participants	accompli par les gestionnaires et les participants gérés
unconfirmed service request containing published information	demande de service non confirmé contenant des informations fournies
can have more than one publisher for the same publication	il peut y avoir plus d'un fournisseur pour la même fourniture
can have (some trade): reliability; deadline; moderation; lifespan; ranking; others.	il peut y avoir (pour certains domaines): fiabilité; échéance; modération; durée de vie; attribution de rang; autres.

Figure 19 – Interactions du modèle fournisseur/abonné

Les gestionnaires et les participants gérés sont des applications spécialisées fournisseur/abonné, contenant toutes deux des fournisseurs et des abonnés ou des versions spécialisées de ceux-ci.

La distinction entre gestionnaires et participants gérés est due à leur rôle dans le mécanisme de découverte et maintenance fournisseur/abonné. Ce mécanisme permet de fournir des offres de fourniture et des demandes d'abonnement à des sujets afin d'assurer une correspondance transparente et un comportement approprié, même lorsque les fournisseurs et les abonnés vont et viennent, de manière dynamique.

Les fournisseurs et les abonnés du modèle fournisseur/abonné, ainsi que leurs versions spécialisées, sont appelés collectivement des acteurs de communication. Leur présence et leurs attributs sont le résultat final du mécanisme de découverte et maintenance décrit.

Il convient de réaliser que l'importance du mécanisme de découverte et maintenance réside dans ce qu'il permet d'accomplir. Son caractère optimal dépend des couches sous-jacentes à la couche application, du nombre de nœuds et d'autres caractéristiques du système hôte. Par conséquent le mécanisme de découverte et maintenance décrit dans la présente spécification n'est qu'un mécanisme parmi tant d'autres; aussi, tant que le résultat final est préservé, peut-on substituer les différents mécanismes les uns aux autres.

5 ASE de type de données

5.1 Généralités

La totalité de la CEI/TR 61158-1, 10.1 est incorporée par référence.

5.2 Définition formelle des objets de type de données

La totalité de la CEI/TR 61158-1, 10.2 est incorporée par référence.

5.3 Types de données définis dans la FAL

5.3.1 Communs

5.3.1.1 Types de longueur fixe

5.3.1.1.1 Types booléens

5.3.1.1.1.1 Booléen

CLASSE: **Type de données**
ATTRIBUTS:

1	Identificateur numérique de type de données	=	1
2	Nom de type de données	=	Booléen
3	Format	=	LONGUEUR FIXE
4.1	Longueur en octets	=	1

Ce type de données exprime un type de données booléen prenant les valeurs VRAI et FAUX.

5.3.1.1.2 Types chaîne de bits

Communs non utilisés.

5.3.1.1.3 Types monétaires

Communs non utilisés.

5.3.1.1.4 Type date/heure

Communs non utilisés.

5.3.1.1.5 Types énumérés

Communs non utilisés.

5.3.1.1.6 Types handle

Communs non utilisés.

5.3.1.1.7 Types numériques

Communs non utilisés.

5.3.1.1.7.1 Type virgule flottante

Communs non utilisés.

5.3.1.1.7.2 Types entiers

5.3.1.1.7.2.1 long

Ce type de données est le même que le type Integer32.

5.3.1.1.7.2.2 Integer32

CLASSE: **Type de données**

ATTRIBUTS:

1	Identificateur numérique de type de données	=	4
2	Nom de type de données	=	Integer32
3	Format	=	LONGUEUR FIXE
4.1	Longueur en octets	=	4

Ce type d'entier est un nombre binaire en complément à deux d'une longueur de quatre octets.

5.3.1.1.7.3 Types non signés

5.3.1.1.7.3.1 Unsigned8

CLASSE: **Type de données**

ATTRIBUTS:

1	Identificateur numérique de type de données	=	5
2	Nom de type de données	=	Unsigned8
3	Format	=	LONGUEUR FIXE
4.1	Longueur en octets	=	1

Ce type correspond à un nombre binaire. Le bit de poids fort de l'octet de poids fort est toujours utilisé comme bit poids fort du nombre binaire; aucun bit de signe n'est inclus. Ce type a une longueur d'un octet.

5.3.1.1.7.3.2 USINT

Ce type de la CEI 61131-3 est le même que le type Unsigned8.

5.3.1.1.7.3.3 caractère non signé

Ce type de données est le même que le type Unsigned8.

5.3.1.1.7.3.4 Unsigned16

CLASSE: **Type de données**

ATTRIBUTS:

1	Identificateur numérique de type de données	=	6
2	Nom de type de données	=	Unsigned16
3	Format	=	LONGUEUR FIXE
4.1	Longueur en octets	=	2

Ce type correspond à un nombre binaire. Le bit de poids fort de l'octet de poids fort est toujours utilisé comme bit poids fort du nombre binaire; aucun bit de signe n'est inclus. Ce type non signé a une longueur de deux octets.

5.3.1.1.7.3.5 UINT

Ce type de la CEI 61131-3 est le même que le type Unsigned16.

5.3.1.1.7.3.6 Unsigned32

CLASSE: **Type de données**

ATTRIBUTS:

1	Identificateur numérique de type de données	=	7
2	Nom de type de données	=	Unsigned32
3	Format	=	LONGUEUR FIXE
4.1	Longueur en octets	=	4

Ce type correspond à un nombre binaire. Le bit de poids fort de l'octet de poids fort est toujours utilisé comme bit poids fort du nombre binaire; aucun bit de signe n'est inclus. Ce type non signé a une longueur de quatre octets.

5.3.1.1.7.3.7 MOT

Ce type est utilisé de la même façon que UINT.

5.3.1.1.8 Types de pointeur

Communs non utilisés.

5.3.1.1.9 Types d'OctetString

5.3.1.1.9.1 OctetString

CLASSE: **Type de données**

ATTRIBUTS:

1	Identificateur numérique de type de données	=	10
2	Nom de type de données	=	OctetString
3	Format	=	CHAÎNE
4.1	Longueur en octets	=	1 à n

Une OctetString est une séquence d'octets ordonnés, numérotés de 1 à n. Pour les besoins de cette description, l'octet 1 de la séquence est désigné comme étant le premier octet. La CEI 61158-6 définit l'ordre de transmission.

5.3.1.1.10 Types de caractère chaîne visible

Communs non utilisés.

5.3.1.2 Types chaîne

Communs non utilisés.

5.3.1.3 Types de structure

Communs non utilisés.

5.3.2 Types de données définis par la FAL pour le client/serveur

5.3.2.1 Types de longueur fixe

5.3.2.1.1 Types booléens

Pas de types booléens spécifiques au type.

5.3.2.1.2 Types chaîne de bits

5.3.2.1.2.1 Indicateur de statut

CLASSE: **Type de données**

ATTRIBUTS:

1	Identificateur numérique de type de données	=	Non utilisé
2	Nom de type de données	=	Indicateur de statut
3	Format	=	LONGUEUR FIXE
4.1	Longueur en octets	=	2

Ce type de données exprime un type de données de statut, prenant les valeurs ON (Actif) et OFF (Inactif), codé au moyen de deux octets. La valeur de codage de ON est 0xFF00 et la valeur de codage de OFF est 0x0000.

5.3.2.1.2.2 Séquence de bits de statut

CLASSE: **Type de données**

ATTRIBUTS:

1	Identificateur numérique de type de données	=	Non utilisé
2	Nom de type de données	=	Séquence de bits de statut
3	Format	=	CHAÎNE
4.1	Longueur en octets	=	1 à n

Ce type de données exprime un type de données de séquence de bits de statut, prenant les valeurs ON et OFF, codé au moyen d'une valeur par bit. La valeur de codage de ON est 1 et la valeur de codage de OFF est 0. Les bits sont contenus dans des octets, le dernier octet étant rempli par des 0 si le nombre de bits de la séquence n'est pas un multiple de 8. Les bits sont numérotés dans l'ordre indiqué ci-dessous, où y est un nombre de bits de statut, x est une valeur binaire de statut et – est un emplacement vide:

	MSB							LSB
Bit	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**
Octet 1	7	6	5	4	3	2	1	0
Statut	x	x	x	x	x	x	x	x
Octet 2	15	14	13	12	11	10	9	8
Statut	x	x	x	x	x	x	x	x
...								
...								
Octet n	-	-	-	-	-	y	y	y
Statut	0	0	0	0	0	x	x	x

5.3.2.1.3 Types monétaires

Pas de types monétaires spécifiques au type.

5.3.2.1.4 Type date/heure

Pas de types date/heure spécifiques au type.

5.3.2.1.5 Types énumérés

Pas de types énumérés spécifiques au type.

5.3.2.1.6 Types descripteur

Pas de types descripteur spécifiques au type.

5.3.2.1.7 Types numériques

Pas de types numériques généraux spécifiques au type.

5.3.2.1.7.1 Type virgule flottante

Pas de types virgule flottante spécifiques au type.

5.3.2.1.7.2 Types entiers

Pas de types entiers spécifiques au type.

5.3.2.1.7.3 Types non signés

Pas de types non signés spécifiques au type.

5.3.2.1.8 Types de pointeur

Pas de types de pointeur spécifiques au type.

5.3.2.1.9 Types d'OctetString

Pas de types de chaîne d'octet de longueur fixe spécifiques au type.

5.3.2.1.10 Types de caractère chaîne visible

Pas de types de chaîne visible de longueur fixe spécifiques au type.

5.3.2.2 Types chaîne

5.3.2.2.1 Chaîne ASCII

CLASSE:			**Type de données**
ATTRIBUTS:			
1	Identificateur numérique de type de données	=	Non utilisé
2	Nom de type de données	=	Chaîne ASCII
3	Format	=	CHAÎNE
4.1	Longueur en octets	=	1 à n

5.3.2.3 Types de structure

Pas de types de structure spécifiques au type.

5.3.3 Types de données définis par la FAL pour le fournisseur/abonné

5.3.3.1 Types de longueur fixe

5.3.3.1.1 Types booléens

Pas de types booléens spécifiques au type.

5.3.3.1.2 Types chaîne de bits

Pas de types chaîne de bits spécifiques au type.

5.3.3.1.3 Types monétaires

Pas de types monétaires spécifiques au type.

5.3.3.1.4 Type date/heure

Pas de types date/heure spécifiques au type.

5.3.3.1.5 Types énumérés

Pas de types énumérés spécifiques au type.

5.3.3.1.6 Types descripteur

Pas de types descripteur spécifiques au type.

5.3.3.1.7 Types numériques

Pas de types numériques généraux spécifiques au type.

5.3.3.1.7.1 Type virgule flottante

Pas de types virgule flottante spécifiques au type.

5.3.3.1.7.2 Types entiers

Pas de types entiers spécifiques au type.

5.3.3.1.7.3 Types non signés

5.3.3.1.7.3.1 IPAddress

CLASSE: **Type de données**

ATTRIBUTS:

1	Identificateur numérique de type de données	=	Non utilisé
2	Nom de type de données	=	IPAddress
3	Format	=	Unsigned32

Une adresse IP est un nombre non signé de 4 octets, comme indiqué dans la RFC 791.

Une adresse IP égale à zéro est une adresse IP invalide

IPADDRESS_INVALID 0

La correspondance entre la notation à points "a.b.c.d" d'une adresse IP et sa représentation sous forme de nombre long non signé est la suivante:

IPAddress ipAddress = (((a * 256 + b) * 256) + c) * 256 + d

Exemple: L'adresse IP "127.0.0.1" correspond au nombre long non signé 2130706433 ou 0x7F000001.

5.3.3.1.7.3.2 Port

CLASSE: **Type de données**

ATTRIBUTS:

1	Identificateur numérique de type de données	=	Non utilisé
2	Nom de type de données	=	Port
3	Format	=	Unsigned32

Un port est un nombre non signé de 4 octets.

Le numéro de port zéro est un numéro de port invalide.

PORT_INVALID 0

Si un numéro de port représente un port UDP IPv4, seule la plage des nombres courts non signés allant de 0x1 à 0x0000ffff est valide.

Les ports suivants ont été affectés au fournisseur/abonné par l'IANA:

— rtps-discovery 7400/tcp,udp RTPS Découverte
— rtps-dd-ut 7401/tcp,udp RTPS Trafic utilisateur de distribution de données
— rtps-dd-mt 7402/tcp,udp RTPS Méta trafic de distribution de données

5.3.3.1.8 Types de pointeur

Pas de types de pointeur spécifiques au type.

5.3.3.1.9 Types d'OctetString

Pas de types de chaîne d'octet de longueur fixe spécifiques au type.

5.3.3.1.10 Types de caractère chaîne visible

Pas de types de chaîne visible de longueur fixe spécifiques au type.

5.3.3.2 Types chaîne

Pas de types de chaîne spécifiques au type.

5.3.3.3 Types de structure

5.3.3.3.1 HostID

CLASSE: **Type de données**

ATTRIBUTS:

1	Identificateur numérique de type de données	=	Non utilisé
2	Nom de type de données	=	HostID
3	Format	=	STRUCTURE
4	Nombre de champs	=	1
5.1	Nom de champ	=	HostID
5.1.1	Type de données de champ	=	OctetString
5.1.2	Longueur de champ	=	4

L'HostID n'est en réalité pas du tout structuré; il s'agit simplement d'une OctetString. La distinction peut être ignorée dans toutes les applications pratiques.

La valeur { 0x00, 0x00, 0x00, 0x00 } est réservée à HOSTID_UNKNOWN, avec la signification d'un HOSTID_UNKNOWN.

5.3.3.3.2 AppID

CLASSE: **Type de données**

ATTRIBUTS:

1	Identificateur numérique de type de données	=	Non utilisé
2	Nom de type de données	=	AppID
3	Format	=	STRUCTURE
4	Nombre de champs	=	2
5.1	Nom de champ	=	InstanceID
5.1.1	Type de données de champ	=	OctetString
5.1.2	Longueur de champ	=	3
5.2	Nom de champ	=	AppKind
5.2.1	Type de données de champ	=	OctetString
5.2.2	Longueur de champ	=	1
5.2.3	Contenu de champ	=	0x01 ou 0x02

Une valeur AppKind de 0x01 donne à l'application l'étiquette de participant géré fournisseur/abonné.

Une valeur AppKind de 0x02 donne à l'application l'étiquette de gestionnaire fournisseur/abonné.

Les mises en œuvre basées sur cette version (1.0) du fournisseur/abonné considèrent toute autre valeur que l'une des deux ci-dessus comme une classe inconnue.

La valeur { 0x00, 0x00, 0x00, 0x00 } est réservée à APPID_UNKNOWN, avec la signification d'un AppID inconnu.

5.3.3.3.3 ID d'objet

CLASSE: **Type de données**

ATTRIBUTS:

1	Identificateur numérique de type de données	=	Non utilisé
2	Nom de type de données	=	ObjectID
3	Format	=	STRUCTURE
4	Nombre de champs	=	2
5.1	Nom de champ	=	InstanceID
5.1.1	Type de données de champ	=	OctetString
5.1.2	Longueur de champ	=	3
5.2	Nom de champ	=	ObjKind
5.2.1	Type de données de champ	=	OctetString
5.2.2	Longueur de champ	=	1

La valeur { 0x00, 0x00, 0x00, 0x00 } est réservée à OBJECTID_UNKNOWN, avec la signification d'un ObjectID inconnu.

Pour *ObjKind*, les deux bits de poids le plus fort indiquent si l'objet est du niveau méta ou du niveau utilisateur (bit M) et si son *instanceID* est choisi ou réservé (bit R), respectivement:

```
ObjectId:
0...2...........8...............16.............24..............31
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                  instanceId                   |M|R|             |
+---------------+---------------+---------------+---------------+
```

M=1 L'objet réseau est un méta-objet: il peut être atteint par les méta-ports de l'application à laquelle il appartient (ports désignés comme étant des ports de méta-trafic).

R=1 L'InstanceID est réservé; il a une signification spéciale pour le protocole, il est utilisé par le mécanisme de découverte.

5.3.3.3.3.1 Objets réservés

Ces objets réservés sont utilisés par le mécanisme de découverte pour la découverte des participants (ou applications) du domaine et des acteurs de communication du réseau.

Les gestionnaires et les participants gérés sont des applications; la différence entre un gestionnaire et un participant géré (application gérée) réside dans leur rôle dans le mécanisme de découverte et maintenance fournisseur/abonné. Ce mécanisme permet de fournir des offres de fourniture et des demandes d'abonnement à des sujets afin d'assurer une correspondance transparente et un comportement approprié, même lorsque les fournisseurs et les abonnés vont et viennent, de manière dynamique.

Les fournisseurs et les abonnés du modèle fournisseur/abonné, ainsi que leurs versions spécialisées, sont appelés collectivement des acteurs de communication. Leur présence et leurs attributs sont le résultat final du mécanisme de découverte et maintenance décrit.

Il convient de réaliser que l'importance du mécanisme de découverte et maintenance réside dans ce qu'il permet d'accomplir. Son caractère optimal dépend des couches sous-jacentes à la couche application, du nombre de nœuds et d'autres caractéristiques du système hôte. Par conséquent le mécanisme de découverte et maintenance décrit dans la présente spécification n'est qu'un mécanisme parmi tant d'autres; aussi, tant que le résultat final est préservé, peut-on substituer les différents mécanismes les uns aux autres.

Chaque gestionnaire et chaque participant géré (ou application gérée) contient un certain nombre d'objets de réseau intégré, qui possèdent des ID d'objet réservés.

Ces objets spéciaux se répartissent en trois catégories:

— le participant (application) du domaine lui-même est un objet de réseau avec un GUID spécial (l'instance de l'application est appelée *applicationSelf*). De plus, toute application a un CSTWriter (*writerApplicationSelf*) qui dissémine les attributs de l'application locale sur le réseau;

— Plusieurs objets sont dédiés à la découverte de gestionnaires et de participants gérés (applications gérées) sur le réseau. Chaque application gérée possède les CSTReaders *readerApplications* et *readerManagers*, grâce auxquels l'existence et les attributs des applications gérées distantes et des gestionnaires distants, respectivement, sont obtenus. Chaque gestionnaire possède les CSTWriters *writeApplications* et *writeManagers* correspondants;

— Chaque application gérée possède, entre autres, deux instances d'un CSTReader (*readerPublications* et *readerSubscriptions*) et deux instances d'un CSTWriter (*writerPublications* et *writerSubscriptions*). Au moyen des CSTReaders, l'application gérée peut recevoir des informations à propos de l'existence et des attributs de toutes les fournitures distantes et de tous les abonnements distants du réseau. Au moyen des CSTWriters, l'application gérée peut envoyer des informations à propos de ses publications locales et de ses abonnements locaux.

5.3.3.3.3.2 Classes

Les six derniers bits de l'objectID définissent la classe à laquelle l'objet appartient (application, fournisseur, abonné, CSTWriter ou CSTReader). Le Tableau 2 en offre une vue d'ensemble. La signification des ID de message est fixe dans cette version majeure (1). De nouveaux *objKinds* peuvent être ajoutés dans des versions mineures supérieures à mesure que le modèle d'objet fournisseur/abonné est étendu par de nouvelles classes.

Tableau 2 – Identification des classes

Classe d'objet	Objet utilisateur normal	Objet utilisateur réservé	Méta-objet normal	Méta-objet réservé
inconnu	0x00	0x40	0x80	0xc0
application	0x01	0x41	0x81	0xc1
CSTWriter	0x02	0x42	0x82	0xc2
Fournisseur	0x03	0x43	0x83	0xc3
Abonné	0x04	0x44	0x84	0xc4
CSTReader	0x07	0x47	0x87	0xc7

5.3.3.3.4 GUID

CLASSE: **Type de données**

ATTRIBUTS:

1 Identificateur numérique de type de données = Non utilisé
2 Nom de type de données = GUID
3 Format = STRUCTURE
4 Nombre de champs = 3
5.1 Nom de champ = hostID
5.1.1 Type de données de champ = hostID
5.1.2 Longueur de champ = 4
5.2 Nom de champ = appID
5.2.1 Type de données de champ = appID
5.2.2 Longueur de champ = 4
5.3 Nom de champ = objectID
5.3.1 Type de données de champ = ObjectID
5.3.2 Longueur de champ = 4

Le GUID (Globally Unique ID - identificateur globalement unique) est une référence unique à une application contenant des acteurs de communication ou à un acteur de communication du réseau.

Le GUID est constitué d'un triplet de 12 octets: <HostID hostId, AppID appId, ObjectID objectId>. Il convient que le GUID soit une référence globalement unique à un objet réseau spécifique se trouvant sur le réseau.

5.3.3.3.4.1 Les GUID des applications fournisseur/abonné

Chaque application fournisseur/abonné du réseau a pour GUID <hostId, appId, OID_APP>, où la constante OID_APP est définie de la façon suivante: OID_APP {0x00,0x00,0x01,0xc1}.

La mise en œuvre peut choisir librement l'*HostID* et l'*AppID*, à partir du moment où le dernier octet de l'*AppID* identifie l'*AppKind* et à partir du moment où chaque application du réseau possède un GUID unique.

5.3.3.3.4.2 Les GUID des acteurs de communication des applications fournisseur/abonné

Les acteurs de communication qui appartiennent à l'application de GUID <hostId, appId, OID_APP> ont pour GUID <hostId, appId, *objectId*>. L'*ObjectID* est l'identification unique de l'objet réseau relatif à l'application. L'*ObjectID* contient également la nature de l'objet réseau, c'est-à-dire si l'objet est un objet utilisateur ou un méta-objet, et si l'*InstanceID* est librement choisi par l'intergiciel (*middleware* en anglais) ou s'il s'agit d'un InstanceID *réservé*, ce qui a une signification spéciale pour le fournisseur/abonné. Un exemple d'ObjectID réservé (défini par protocole) est OID_APP, qui est utilisé dans le GUID des applications.

5.3.3.3.5 VendorID

CLASSE: **Type de données**

ATTRIBUTS:

1	Identificateur numérique de type de données	=	Non utilisé
2	Nom de type de données	=	VendorID
3	Format	=	STRUCTURE
4	Nombre de champs	=	2
5.1	Nom de champ	=	majeur
5.1.1	Type de données de champ	=	OctetString
5.1.2	Longueur de champ	=	1
5.2	Nom de champ	=	mineur
5.2.1	Type de données de champ	=	OctetString
5.2.2	Longueur de champ	=	1

Cette structure identifie le vendeur de l'intergiciel servant à mettre en œuvre le protocole fournisseur/abonné et permet à ce vendeur d'ajouter des extensions spécifiques au protocole. L'ID du vendeur ne désigne pas le vendeur du dispositif ou du produit qui contient l'intergiciel fournisseur/abonné.

Les ID de vendeur actuellement affectés sont répertoriés dans le Tableau 3.

Tableau 3 – ID de vendeur affectés

Majeur	Mineur	Nom
0x00	0x00	VENDOR_ID_UNKNOWN
0x01	0x01	Real-Time Innovations, Inc., CA, USA

5.3.3.3.6 ProtocolVersion

CLASSE: **Type de données**

ATTRIBUTS:

1	Identificateur numérique de type de données	=	Non utilisé
2	Nom de type de données	=	ProtocolVersion
3	Format	=	STRUCTURE
4	Nombre de champs	=	2
5.1	Nom de champ	=	majeur
5.1.1	Type de données de champ	=	OctetString
5.1.2	Longueur de champ	=	1
5.2	Nom de champ	=	mineur
5.2.1	Type de données de champ	=	OctetString
5.2.2	Longueur de champ	=	1

Les mises en œuvre qui suivent cette version de la spécification mettent en œuvre la version de protocole 1.0 (majeur = 1, mineur = 0).

5.3.3.3.7 SequenceNumber

CLASSE: **Type de données**

ATTRIBUTS:

1	Identificateur numérique de type de données	=	Non utilisé
2	Nom de type de données	=	SequenceNumber
3	Format	=	STRUCTURE

4	Nombre de champs	=	2
5.1	Nom de champ	=	haut
5.1.1	Type de données de champ	=	Integer32
5.1.2	Longueur de champ	=	4
5.2	Nom de champ	=	bas
5.2.1	Type de données de champ	=	Unsigned32
5.2.2	Longueur de champ	=	4

Un numéro de séquence, N, est un entier signé de 64 bits qui peut prendre des valeurs dans la plage:
-2^63 <= N <= 2^63-1.

Avec cette structure, le numéro de séquence est: *haut* * 2^32 + *bas*.

Le numéro de séquence sert à identifier de manière unique des messages fournisseur/abonné élémentaires d'une manière ordonnée.

Le numéro de séquence 0 et les numéros de séquence négatifs servent à désigner les cas spéciaux:

SEQUENCE_NUMBER_NONE 0

SEQUENCE_NUMBER_UNKNOWN -1

NOTE Le choix de 64 bits pour représenter les numéros de séquence offre la garantie que les numéros de séquence ne recommencent jamais à zéro. Même dans le cas d'un débit de génération de messages extrêmement rapide de la part d'un auteur fournisseur/abonné tel que 100 messages par microseconde, l'entier de 64 bits ne recommencerait à zéro qu'au bout d'environ 3000 ans de fonctionnement ininterrompu.

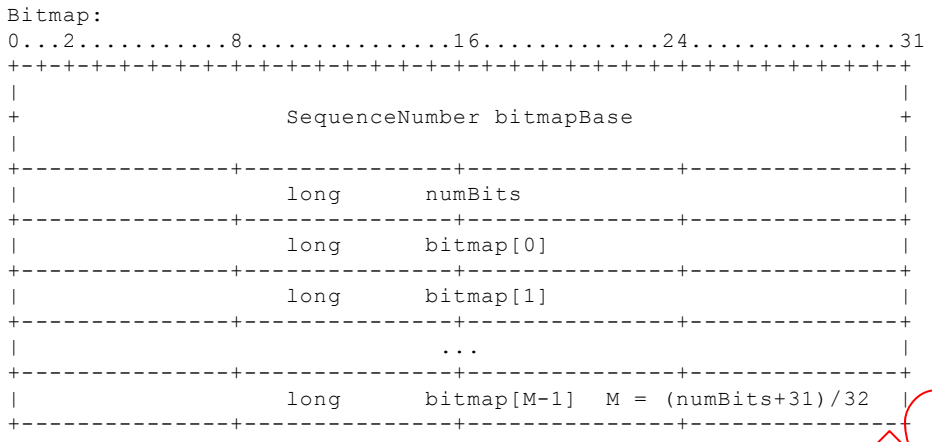
5.3.3.3.8 Bitmap

CLASSE: **Type de données**

ATTRIBUTS:

1	Identificateur numérique de type de données	=	Non utilisé
2	Nom de type de données	=	Bitmap
3	Format	=	STRUCTURE
4	Nombre de champs	=	3
5.1	Nom de champ	=	bitmapBase
5.1.1	Type de données de champ	=	SequenceNumber
5.1.2	Longueur de champ	=	8
5.2	Nom de champ	=	numBits
5.2.1	Type de données de champ	=	long
5.2.2	Longueur de champ	=	4
5.3	Nom de champ	=	BitmapArray
5.3.1	Type de données de champ	=	MATRICE
5.3.2	Nombre d'éléments de matrice	=	M, où M = (numBits + 31)/32
5.3.3	Type de données d'élément de matrice	=	long

Les bitmaps sont utilisés en tant que parties de plusieurs messages (services) pour fournir des informations binaires au sujet des numéros de séquence individuels d'une plage. La représentation du bitmap comprend la longueur du bitmap en bits et le premier SequenceNumber auquel le bitmap s'applique.



Pour un bitmap, *bitmap*, la valeur booléenne du bit appartenant au SequenceNumber *N*, où *bitmapBase* <= *N* < *bitmapBase+numBits,* est:

bit(N) = bitmap[deltaN/32] & (1 << (31 - deltaN%32))

où:

deltaN = N – bitmapBase

Le bitmap n'indique rien au sujet des numéros de séquence situés en dehors de la plage [bitmapBase, bitmapBase + numBits -1].

Un bitmap valide doit satisfaire aux conditions suivantes:

— bitmapBase >= 1

— 0 <= numBits <= 256

— il y a M=(numBits +31)/32 longs contenant les bits pertinents

Le présent document utilise la notation suivante pour un bitmap spécifique:

bitmapBase/numBits:bitmap

Dans le bitmap, le bit correspondant au numéro de séquence *bitmapBase* se trouve sur la gauche. Les bits "0" de fin peuvent être représentés sous la forme d'un "0" unique.

Exemple: Dans le bitmap "1234/12:00110", bitmapBase = 1234 et numBits = 12. Les bits s'appliquent comme suit aux numéros de séquence:

Séquence	Bit
1234	0
1235	0
1236	1
1237	1
1238-1245	0

5.3.3.3.9 NtpTime

CLASSE: **Type de données**

ATTRIBUTS:

1 Identificateur numérique de type de données = Non utilisé

2	Nom de type de données	=	NtpTime
3	Format	=	STRUCTURE
4	Nombre de champs	=	2
5.1	Nom de champ	=	secondes
5.1.1	Type de données de champ	=	long
5.1.2	Longueur de champ	=	4
5.2	Nom de champ	=	fraction
5.2.1	Type de données de champ	=	Unsigned32
5.2.2	Longueur de champ	=	4

Les horodatages suivent la norme NTP et sont représentés sur le câble sous la forme d'une paire d'entiers contenant les 32 bits d'ordre supérieur et d'ordre inférieur.

Le temps est exprimé en secondes au moyen de la formule suivant:

secondes + (fraction / 2^(32))

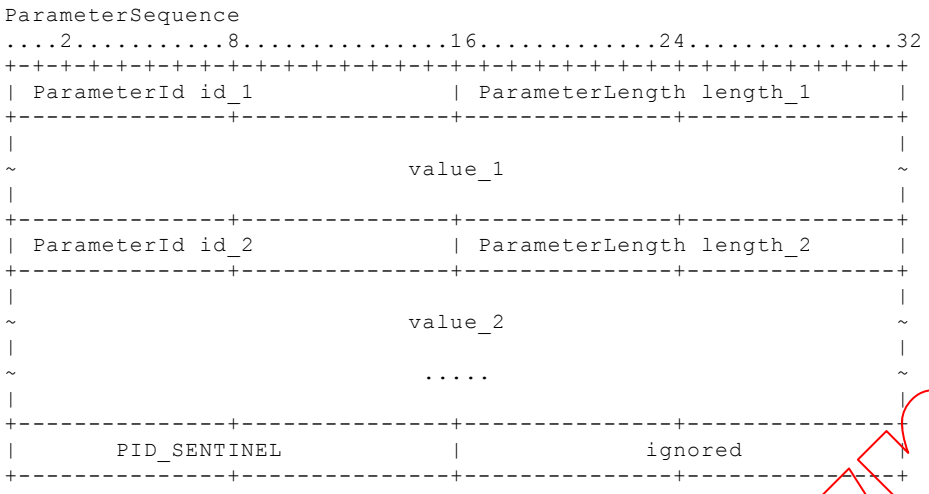
Le fournisseur/abonné ne nécessite pas de concept de temps absolu.

5.3.3.4 Types auto-descripteurs

5.3.3.4.1 Séquence de paramètres

CLASSE:			**Type de données**
ATTRIBUTS:			
1	Identificateur numérique de type de données	=	Non utilisé
2	Nom de type de données	=	ParameterSequence
3	Format	=	STRUCTURE
4	Nombre de champs	=	4
5.1	Nom de champ	=	ParameterID | Sentinelle
5.1.1	Type de données de champ	=	Unsigned16
5.1.2	Contenu de champ	=	bitmapArray | PID_SENTINEL
5.2	Nom de champ	=	ParameterLength | IGNORÉ
5.2.1	Type de données de champ	=	Unsigned16
5.2.2	Contenu de champ	=	n
5.3	Nom de champ	=	Valeur | VIDE
5.3.1	Type de données de c hamp	=	OctetString
5.3.2	Longueur de champ	=	n
5.4	Type de données intégrées	=	self | VIDE

Ce type de données représente une séquence de longueur variable contenant des paramètres, la séquence se terminant par la sentinelle PID_SENTINEL. PID_SENTINEL est défini avec la valeur 0x0001, sa longueur étant ignorée. Chaque ParameterId commence par une frontière de 4 octets avec le début de la séquence de paramètres, de sorte que ParameterLength est toujours un multiple de quatre.



5.4 Spécification des services des ASE de type de données

Il n'existe pas de services opérationnels définis pour l'objet type.

6 Spécification du modèle de communication client/serveur

6.1 ASE

6.1.1 Généralités

Les ASE de la FAL sont définis au moyen d'une approche modulaire. Les ASE définis pour la FAL sont également orientés objet. En général, les ASE fournissent un ensemble de services conçu pour une classe d'objets spécifique ou pour un ensemble de classes associé.

Pour supporter l'accès distant à l'AP, on définit l'ASE de relation d'application. Il fournit des services à l'AP pour définir et établir des relations de communication avec les autres AP, et il fournit des services aux autres ASE pour transmettre leurs demandes de service et leurs réponses.

Seul un sous-ensemble des ASE de FAL définis peut être fourni pour satisfaire aux besoins de l'application.

Chaque ASE de FAL définit un ensemble de services, les APDU, et les procédures qui fonctionnent sur les classes qu'il représente.

Pour un ASE de FAL donné, seul un sous-ensemble des services d'ASE peut être fourni pour satisfaire aux besoins d'une application.

Les sous-ensembles peuvent être définis au moyen de profils. La définition des profils ne fait pas partie du domaine d'application de la présente norme.

Les APDU sont envoyés et reçus entre les ASE de FAL qui supportent les mêmes services.

La Figure 20 illustre les ASE de FAL client/serveur et leurs relations architecturales.

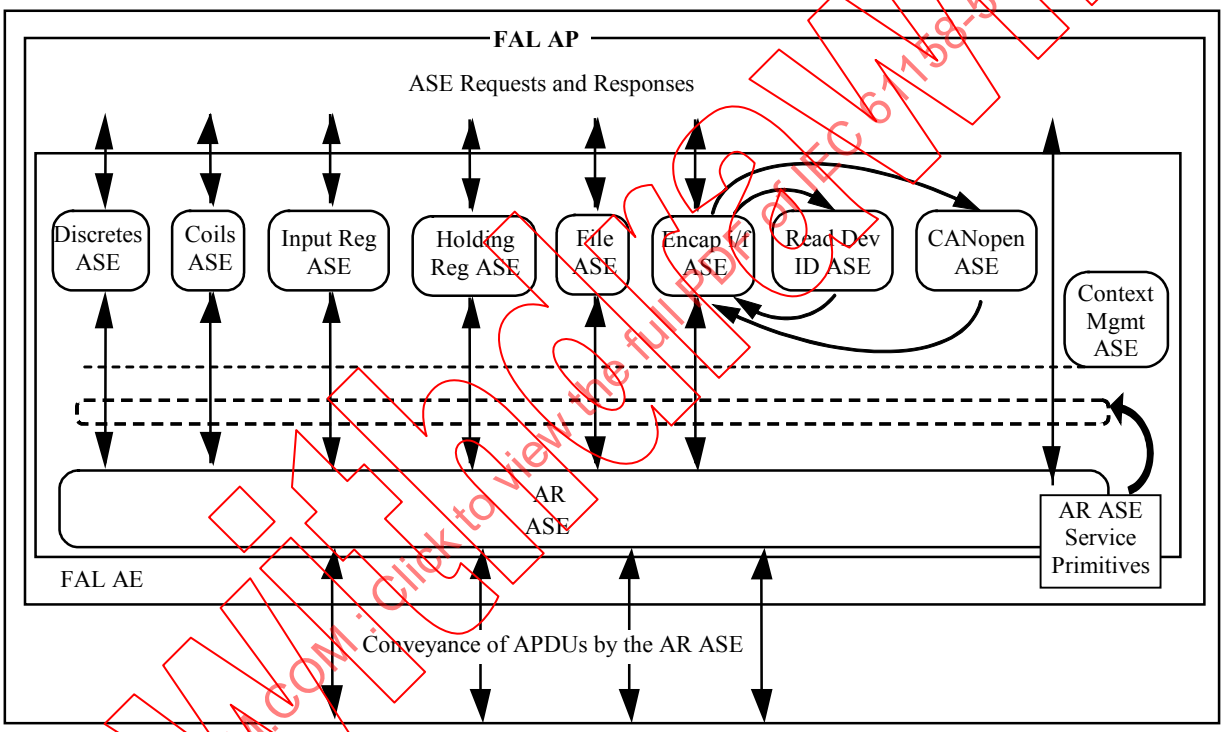
Il existe un ASE de FAL pour chaque classe d'APO client/serveur. L'ASE Lire identification de dispositif et l'ASE CANopen supportent des APO qui sont des classes dérivées de la classe d'APO d'interface encapsulée; ces deux ASE sont subordonnés à l'ASE d'interface encapsulée, aussi convient-il d'examiner la Figure 20 en gardant cela à l'esprit.

Un ASE de FAL additionnel, l'ASE de gestion de contexte, fournit un service de filtre et un service de transaction aux autres ASE.

Le service de filtre est un service fournisseur, non un service utilisateur: il s'agit d'un filtre frontal sur le serveur. Toutes les interactions d'utilisateur d'AL le traversent: les services supportés, les services non supportés, et même les services non définis, traités comme des services non supportés.

Le service de transaction est également un service fournisseur; il modère l'appel des services par le client, en plus d'être utilisé pour la gestion des primitives de service confirmé. L'ASE de gestion de contexte est une abstraction; sa configuration et sa mise en œuvre ne font pas partie de la présente spécification.

Un serveur client/serveur minimal, bien que ne présentant pas d'utilité particulière, doit mettre en œuvre le service de filtre de l'ASE de gestion de contexte.



Légende

Anglais	Français
FAL AP	AP de FAL:
ASE Requests and Responses	Demandes et réponses ASE
Discretes ASE	ASE des discrets
Coils ASE	ASE des bobines
Input Reg ASE	ASE des registres d'entrée
Holding Reg ASE	ASE des registres de maintien
File ASE	ASE de fichier
Encap i/f ASE	ASE d'interface encapsul.
Read Dev ID ASE	ASE de lecture d'ID disp.
CANopen ASE	ASE CANopen
Context Mgmt ASE	ASE de gestion de contexte
FAL AE	AE de FAL
AR ASE	ASE d'AR

Anglais	Français
AR ASE Service Primitives	Primitives des services d'ASE d'AR
Conveyance of APDUs by the AR ASE	Transmission d'APDU par l'ASE d'AR

Figure 20 – Les ASE de la FAL

6.1.2 Paramètres communs

De nombreux services possèdent les paramètres suivants. Ils ne sont pas définis pour chaque service, mais au moyen des définitions communes données ci-dessous.

ID appel
Ce paramètre définit l'objet de transaction à l'intérieur d'une connexion; il sert à apparier la demande avec la confirmation correspondante. Il faut qu'il soit unique parmi toutes les transactions en cours sur la connexion en question. Il est conditionnel, car il n'est pas exigé lorsque la gestion de contexte client n'autorise qu'une seule transaction à la fois. Les valeurs Résultat (+) et Résultat (-) sont les mêmes que le paramètre ID appel de la demande.

Type de paramètre: Unsigned16.

Valeurs autorisées: n'importe quelle valeur appartenant au type, dans le respect de la contrainte d'unicité indiquée ci-dessus.

6.1.3 ASE de gestion de contexte

6.1.3.1 Vue d'ensemble

Les services de gestion de contexte permettent de rejeter les services non supportés, assurent la modération des appels de services et supportent la gestion des primitives des services confirmés. Il faut qu'un serveur client/serveur possède le service de filtre de cet ASE.

6.1.3.2 Spécifications de la classe de gestion de contexte

6.1.3.2.1 Spécification de la classe des filtres

6.1.3.2.1.1 Modèle formel de filtre

L'objet filtre est décrit par le modèle suivant:

ASE: **ASE de gestion de contexte**
CLASSE: **Filtre**
ID CLASSE: non utilisé
CLASSE PARENTE: TOP
ATTRIBUTS:
1. (m) Attribut clé: Implicite
2. (m) Attribut: Service utilisateur d'AL
SERVICES:
1. (m) OpsService: Filtre

6.1.3.2.1.2 Attributs

Implicite
L'attribut Implicite indique que l'objet filtre de service est adressé implicitement par les services.

Service utilisateur d'AL
Cet attribut spécifie le service utilisateur d'AL.

Type d'attribut: Service utilisateur d'AL.

6.1.3.2.1.3 Services

Filtre
Ce service est utilisé sur le serveur pour permettre de distinguer les services supportés des services non supportés.

6.1.3.2.2 Spécification de la classe des transactions

6.1.3.2.2.1 Modèle formel de transaction

L'objet transaction est décrit par le modèle suivant:

ASE:			**ASE de gestion de contexte**
CLASSE:			**Transaction**
ID CLASSE:			non utilisé
CLASSE PARENTE:			TOP
ATTRIBUTS:			
1.	(m)	Attribut clé:	Implicite
2.	(m)	Attribut de classe:	Nombre maximum de transactions en cours
2.	(m)	Attribut:	ID appel
3.	(m)	Attribut:	Info de demande de service confirmé

6.1.3.2.2.2 Attributs

Implicite
L'attribut Implicite indique que l'objet filtre de service est adressé implicitement par les services.

Nombre maximum de transactions en cours
Cet attribut de classe spécifie le nombre maximum d'objets de transaction qui peuvent être instanciés – tout en étant toujours en cours – par un client. En pratique, les ressources disponibles peuvent diminuer ce nombre. Pour un client donné, même en l'absence d'exigence de rendre cette valeur disponible par programme, il faut qu'elle soit documentée.

Valeurs types: 1 à 16

ID appel
Cet attribut contient l'identificateur de transaction, qui doit être unique parmi tous les identificateurs de transaction toujours en cours sur la connexion en question. L'objet de transaction est instancié pour la durée de l'appel de service; il est généralement utilisé pour coupler les demandes et les confirmations pour les services confirmés, puis est détruit. Il sert également de modérateur. Si un client ne peut instancier qu'un seul objet de transaction à la fois, alors, lorsqu'il appelle un service, il n'est pas nécessaire de créer dynamiquement un objet de transaction et de l'échanger avec les couches inférieures, puisque la raison principale de l'existence de l'objet de transaction est de permettre le couplage correct des demandes et les confirmations, ce qui est automatique pour ces clients. Dans ce type de situation, un objet de transaction statique individuel joue effectivement le rôle de jeton client, lequel est soit pris soit disponible, et n'intéresse que le client.

Type d'attribut: Unsigned16

Info de demande de service confirmé
Cet attribut contient des informations liées à la demande de service, à des fins de mise en correspondance, comprenant l'ID d'unité et le codage du service.

Type d'attribut: Structure en Unsigned8 pour l'ID d'unité et le type de codage de service, comme décrit dans la CEI 61158-6-15.

6.1.3.3 Spécification des services des ASE de gestion de contexte

6.1.3.3.1 Services supportés

L'ASE de gestion de contexte définit les services:

Filtre

6.1.3.3.2 Service de filtre

6.1.3.3.2.1 Vue d'ensemble du service

Le service de filtre est utilisé sur le serveur pour permettre de distinguer les services supportés des services non supportés.

6.1.3.3.2.2 Primitives de service

Les paramètres de service de ce service sont indiqués dans le Tableau 4. Il s'agit d'un service confirmé entre ASE.

Tableau 4 – Paramètres de service de filtre

Nom de paramètre	Dem.	Ind.	Rép.	Conf.
Argument	M	M(=)		
Service utilisateur d'AL	M	M(=)		
Résultat (+)			S	S(=)
Service utilisateur d'AL			M	M(=)
Résultat (-)			S	S(=)
Info. erreur			M	M(=)

Argument
L'argument doit transmettre les paramètres de service spécifiques de la demande de service.

Service utilisateur d'AL
Ce paramètre doit être utilisé pour spécifier le service utilisateur d'AL.

Type de paramètre: Service utilisateur d'AL.

Résultat (+)
Ce paramètre de type de sélection indique que la demande de service a réussi.

Service utilisateur d'AL
En l'absence d'erreur, ce paramètre est identique au paramètre de demande portant le nom correspondant.

Résultat (-)
Ce paramètre de type de sélection indique que la demande de service a échoué.

Info. erreur
Ce paramètre contient les informations d'erreur: il donne aux services utilisateur d'AL demandés l'indicateur "non supporté".

Type de paramètre: Unsigned8.

6.1.3.3.2.3 Procédure de service

Ce service est utilisé sur le serveur pour permettre de distinguer les services supportés des services non supportés sur ce serveur particulier.

6.1.4 ASE des discrets

6.1.4.1 Vue d'ensemble

Cet ASE permet l'interaction et la modélisation par des objets réels à valeurs binaires qui sont manipulés par l'application serveur et ne sont censés être observés que par l'utilisateur

client. L'intégrité du contrat ci-dessus est sous le contrôle de l'AP serveur, qui peut limiter l'exposition de l'objet réel à des discrets.

6.1.4.2 Spécification de la classe des discrets

6.1.4.2.1 Modèle formel

L'objet Discrets est décrit par le modèle suivant:

ASE:			**ASE des discrets**
CLASSE:			**Discrets**
ID CLASSE:			non utilisé
CLASSE PARENTE:			TOP
ATTRIBUTS:			
1.	(m)	Attribut clé:	Implicite
2.	(m)	Attribut:	ID d'unité
3.	(m)	Attribut:	Adresse du premier discret
4.	(m)	Attribut:	Quantité de discrets
SERVICES:			
1.	(m)	OpsService:	Lire discrets

6.1.4.2.2 Attributs

Implicite
L'attribut Implicite indique que l'objet Discrets est adressé implicitement par les services.

ID d'unité
Cet attribut spécifie l'adresse du serveur.

Type d'attribut: Unsigned8

Valeurs autorisées: 1 à 247

Adresse du premier discret
Cet attribut contient l'adresse de la première entrée de discret.

Type d'attribut: Unsigned16

Valeurs autorisées: 0x0000 à 0xFFFF

Quantité de discrets
L'attribut contient la quantité de discrets.

Type d'attribut: Unsigned16

Valeurs autorisées: 1 à 2000

6.1.4.2.3 Services

Lire discrets
Ce service permet de lire un nombre spécifié d'entrées discrètes.

6.1.4.3 Spécification des services des ASE des discrets

6.1.4.3.1 Services supportés

L'ASE des discrets définit les services:

Lire discrets

6.1.4.3.2 Service Lire discrets

6.1.4.3.2.1 Vue d'ensemble du service

Le service Lire discrets permet de lire un nombre spécifié d'entrées discrètes.

6.1.4.3.2.2 Primitives de service

Les paramètres de service de ce service sont indiqués dans le Tableau 5. Il s'agit d'un service confirmé.

Tableau 5 – Paramètres du service Lire discrets

Nom de paramètre	Dem.	Ind.	Rép.	Conf.
Argument	M	M(=)		
ID d'unité	M	M(=)		
ID appel	C	C(=)		
Adresse du premier discret	M	M(=)		
Quantité de discrets	M	M(=)		
Résultat (+)			S	S(=)
ID d'unité			M	M(=)
ID appel			C	C(=)
Nombre d'octets de données			M	M(=)
Données			M	M(=)
Résultat (-)			S	S(=)
ID d'unité			M	M(=)
ID appel			C	C(=)
Info. erreur			M	M(=)

Argument
L'argument transmet les paramètres de service spécifiques de la demande de service.

ID d'unité
Ce paramètre spécifie l'adresse du serveur.

Type de paramètre: Unsigned8.

Valeurs autorisées: 1 à 247.

Adresse du premier discret
Ce paramètre spécifie l'adresse de la première entrée de discret.

Type de paramètre: Unsigned16.

Valeurs autorisées: 0x0000 à 0xFFFF

Quantité de discrets
Ce paramètre spécifie la quantité de discrets.

Type de paramètre: Unsigned16.

Valeurs autorisées: 1 à 2000

Résultat (+)
Ce paramètre de type de sélection indique que la demande de service a réussi.

ID d'unité
En l'absence d'erreur, ce paramètre est identique au paramètre de demande portant le nom correspondant.

Nombre d'octets de données
Ce paramètre spécifie le nombre d'octets lus.

Type de paramètre: Unsigned16.

Données
Ce paramètre spécifie les valeurs de statut des discrets lus.

Type de paramètre: Séquence de bits de statut.

Résultat (-)
Ce paramètre de type de sélection indique que la demande de service a échoué.

ID d'unité
En l'absence d'erreur, ce paramètre est identique au paramètre de demande portant le nom correspondant.

Info. erreur
Ce paramètre spécifie les informations d'erreur.

Type de paramètre: Unsigned8.

6.1.4.3.2.3 Procédure de service

Ce service permet de lire un nombre spécifié d'entrées discrètes.

6.1.5 ASE des bobines

6.1.5.1 Vue d'ensemble

Cet ASE permet l'interaction et la modélisation par des objets réels à valeurs binaires qui peuvent être manipulés par l'application serveur et par l'utilisateur client.

6.1.5.2 Spécification de la classe des bobines

6.1.5.2.1 Modèle formel

L'objet Bobines est décrit par le modèle suivant:

ASE:			**ASE des bobines**
CLASSE:			**Bobines**
ID CLASSE:			non utilisé
CLASSE PARENTE:			TOP
ATTRIBUTS:			
1.	(m)	Attribut clé:	Implicite
2.	(m)	Attribut:	ID d'unité
3.	(m)	Attribut:	Adresse de la première bobine
4.	(c)	Contrainte:	Écrire bobine individuelle | Écrire en diffusion bobine individuelle
4.1	(m)	Attribut:	Bobine individuelle de données
5.	(c)	Contrainte:	Lire bobines | Écrire bobines multiples | Écrire en diffusion bobines multiples
5.1	(m)	Attribut:	Quantité de bobines
5.2	(m)	Attribut:	Nombre d'octets de données
5.3	(m)	Attribut:	Données
SERVICES:			
1.	(o)	OpsService:	Lire bobines
2.	(o)	OpsService:	Écrire bobine individuelle
3.	(o)	OpsService:	Écrire bobines multiples
4.	(c)	Contrainte:	Diffusion supportée
4.1	(o)	OpsService:	Écrire en diffusion bobine individuelle
4.2	(o)	OpsService:	Écrire en diffusion bobines multiples

6.1.5.2.2 Attributs

Implicite
L'attribut Implicite indique que l'objet Bobines est adressé implicitement par les services.

ID d'unité
Cet attribut spécifie l'adresse du serveur.

Type d'attribut: Unsigned8

Valeurs autorisées: 1 à 247, et 0 pour la diffusion si cela s'applique.

Adresse de la première bobine
Cet attribut spécifie l'adresse de la première bobine.

Type d'attribut: Unsigned16

Valeurs autorisées: 0x0000 à 0xFFFF

Bobine individuelle de données
L'attribut spécifie la valeur de statut de bobine individuelle qui doit être écrite.

Type d'attribut: Indicateur de statut

Quantité de bobines
L'attribut spécifie la quantité de bobines.

Type d'attribut: Unsigned16

Valeurs autorisées: 1 à 2000 pendant la lecture; 1 à 1968 pendant l'écriture.

Nombre d'octets de données
L'attribut spécifie la quantité d'octets de données.

Type d'attribut: Unsigned8

Valeurs autorisées: 1 à 250 pendant l'écriture; 1 à 246 pendant la lecture.

Données
L'attribut spécifie les valeurs de statut de bobine qui ont été lues ou qui doivent être écrites.

Type d'attribut: Séquence de bits de statut

6.1.5.2.3 Services

Lire bobines
Ce service optionnel permet de lire un nombre spécifié de bobines.

Écrire bobine individuelle
Ce service optionnel permet d'écrire une bobine individuelle.

Écrire bobines multiples
Ce service optionnel permet d'écrire un nombre spécifié de bobines.

Écrire en diffusion bobine individuelle
Ce service optionnel est également conditionnel pour le mécanisme de diffusion supporté par le client et par les serveurs. Ce service permet d'écrire une bobine individuelle dans tous les serveurs adressables par l'ID d'unité; il s'agit d'un service non confirmé.

Écrire en diffusion bobines multiples
Ce service optionnel est également conditionnel pour le mécanisme de diffusion supporté par le client et par les serveurs. Ce service permet d'écrire un nombre spécifié de bobines dans tous les serveurs adressables par l'ID d'unité; il s'agit d'un service non confirmé.

6.1.5.3 Spécification des services des ASE des bobines

6.1.5.3.1 Services supportés

Le présent paragraphe spécifie la définition de services qui sont uniques pour cet ASE. Les services définis pour cet ASE sont:

Lire bobines

Écrire bobine individuelle

Écrire bobines multiples

Écrire en diffusion bobine individuelle

Écrire en diffusion bobines multiples

6.1.5.3.2 Service Lire bobines

6.1.5.3.2.1 Vue d'ensemble du service

Le service Lire bobines permet de lire un nombre spécifié de bobines.

6.1.5.3.2.2 Primitives de service

Les paramètres de service de ce service sont indiqués dans le Tableau 6. Il s'agit d'un service confirmé.

Tableau 6 – Paramètres du service Lire bobines

Nom de paramètre	Dem.	Ind.	Rép.	Conf.
Argument	M	M(=)		
ID d'unité	M	M(=)		
ID appel	C	C(=)		
Adresse de la première bobine	M	M(=)		
Quantité de bobines	M	M(=)		
Résultat (+)			S	S(=)
ID d'unité			M	M(=)
ID appel			C	C(=)
Nombre d'octets de données			M	M(=)
Données			M	M(=)
Résultat (-)			S	S(=)
ID d'unité			M	M(=)
ID appel			C	C(=)
Info. erreur			M	M(=)

Argument
L'argument transmet les paramètres de service spécifiques de la demande de service.

ID d'unité
Ce paramètre spécifie l'adresse du serveur.

Type de paramètre: Unsigned8.

Valeurs autorisées: 1 à 247.

Adresse de la première bobine
Ce paramètre spécifie l'adresse de la première bobine.

Type de paramètre: Unsigned16.

Valeurs autorisées: 0x0000 à 0xFFFF

Quantité de bobines
Ce paramètre spécifie la quantité de bobines.

Type de paramètre: Unsigned16.

Valeurs autorisées: 1 à 2000

Résultat (+)
Ce paramètre de type de sélection indique que la demande de service a réussi.

ID d'unité
En l'absence d'erreur, ce paramètre est identique au paramètre de demande portant le nom correspondant.

Nombre d'octets de données
Ce paramètre spécifie le nombre d'octets lus.

Type de paramètre: Unsigned16.

Données
Ce paramètre spécifie les valeurs de statut des bobines lues.

Type de paramètre: Séquence de bits de statut.

Résultat (-)
Ce paramètre de type de sélection indique que la demande de service a échoué.

ID d'unité
En l'absence d'erreur, ce paramètre est identique au paramètre de demande portant le nom correspondant.

Info. erreur
Ce paramètre spécifie les informations d'erreur.

Type de paramètre: Unsigned8.

6.1.5.3.2.3 Procédure de service

Ce service permet de lire un nombre spécifié de bobines.

6.1.5.3.3 Service Écrire bobine individuelle

6.1.5.3.3.1 Vue d'ensemble du service

Le service Écrire bobine individuelle permet d'écrire une bobine individuelle.

6.1.5.3.3.2 Primitives de service

Les paramètres de service de ce service sont indiqués dans le Tableau 7. Il s'agit d'un service confirmé.

Tableau 7 – Paramètres du service Écrire bobine individuelle

Nom de paramètre	Dem.	Ind.	Rép.	Conf.
Argument	M	M(=)		
ID d'unité	M	M(=)		
ID appel	C	C(=)		
Adresse de la première bobine	M	M(=)		
Bobine individuelle de données	M	M(=)		
Résultat (+)			S	S(=)
ID d'unité			M	M(=)
ID appel			C	C(=)
Adresse de la première bobine			M	M(=)
Bobine individuelle de données			M	M(=)
Résultat (-)			S	S(=)
ID d'unité			M	M(=)
ID appel			C	C(=)
Info. erreur			M	M(=)

Argument
L'argument transmet les paramètres de service spécifiques de la demande de service.

ID d'unité
Ce paramètre spécifie l'adresse du serveur.

Type de paramètre: Unsigned8.

Valeurs autorisées: 1 à 247.

Adresse de la première bobine
Ce paramètre spécifie l'adresse de la première bobine (la seule dans ce service).

Type de paramètre: Unsigned16.

Valeurs autorisées: 0x0000 à 0xFFFF

Bobine individuelle de données
Ce paramètre spécifie la valeur de statut de bobine individuelle qui doit être écrite.

Type de paramètre: Indicateur de statut.

Résultat (+)
Ce paramètre de type de sélection indique que la demande de service a réussi.

ID d'unité
En l'absence d'erreur, ce paramètre est identique au paramètre de demande portant le nom correspondant.

Adresse de la première bobine
En l'absence d'erreur, ce paramètre est identique au paramètre de demande portant le nom correspondant.

Bobine individuelle de données
En l'absence d'erreur, ce paramètre est identique au paramètre de demande portant le nom correspondant.

Résultat (-)
Ce paramètre de type de sélection indique que la demande de service a échoué.

ID d'unité
En l'absence d'erreur, ce paramètre est identique au paramètre de demande portant le nom correspondant.

Info. erreur
Ce paramètre spécifie les informations d'erreur.

Type de paramètre: Unsigned8.

6.1.5.3.3.3 Procédure de service

Ce service permet d'écrire une bobine individuelle.

6.1.5.3.4 Service Écrire bobines multiples

6.1.5.3.4.1 Vue d'ensemble du service

Le service Écrire bobines multiples permet d'écrire un nombre spécifié de bobines.

6.1.5.3.4.2 Primitives de service

Les paramètres de service de ce service sont indiqués dans le Tableau 8. Il s'agit d'un service confirmé.

Tableau 8 – Paramètres du service Écrire bobines multiples

Nom de paramètre	Dem.	Ind.	Rép.	Conf.
Argument	M	M(=)		
ID d'unité	M	M(=)		
ID appel	C	C(=)		
Adresse de la première bobine	M	M(=)		
Quantité de bobines	M	M(=)		
Nombre d'octets de données	M	M(=)		
Données	M	M(=)		
Résultat (+)			S	S(=)
ID d'unité			M	M(=)
ID appel			C	C(=)
Adresse de la première bobine			M	M(=)
Quantité de bobines			M	M(=)
Résultat (-)			S	S(=)
ID d'unité			M	M(=)
ID appel			C	C(=)
Info. erreur			M	M(=)

Argument
L'argument transmet les paramètres de service spécifiques de la demande de service.

ID d'unité
Ce paramètre spécifie l'adresse du serveur.

Type de paramètre: Unsigned8.

Valeurs autorisées: 1 à 247.

Adresse de la première bobine
Ce paramètre spécifie l'adresse de la première bobine.

Type de paramètre: Unsigned16.

Valeurs autorisées: 0x0000 à 0xFFFF

Quantité de bobines
Ce paramètre spécifie la quantité de bobines.

Type de paramètre: Unsigned16.

Valeurs autorisées: 1 à 1968, doivent être compatibles avec le nombre d'octets de données et les paramètres de données.

Nombre d'octets de données
Ce paramètre spécifie le nombre d'octets transportant les valeurs de statut de bobine à écrire.

Type de paramètre: Unsigned16.

Valeurs autorisées: 1 à 246, doivent être compatibles avec la quantité de bobines et les paramètres de données.

Données
Ce paramètre spécifie les valeurs de statut de bobine qui doivent être écrites.

Type de paramètre: Séquence de bits de statut.

Valeurs autorisées: Elles doivent être compatibles avec la quantité de bobines et les paramètres de nombre d'octets de données.

Résultat (+)
Ce paramètre de type de sélection indique que la demande de service a réussi.

ID d'unité
En l'absence d'erreur, ce paramètre est identique au paramètre de demande portant le nom correspondant.

Adresse de la première bobine
En l'absence d'erreur, ce paramètre est identique au paramètre de demande portant le nom correspondant.

Quantité de bobines
En l'absence d'erreur, ce paramètre est identique au paramètre de demande portant le nom correspondant.

Résultat (-)
Ce paramètre de type de sélection indique que la demande de service a échoué.

ID d'unité
En l'absence d'erreur, ce paramètre est identique au paramètre de demande portant le nom correspondant.

Info. erreur
Ce paramètre spécifie les informations d'erreur.

Type de paramètre: Unsigned8.

6.1.5.3.4.3 Procédure de service

Ce service permet d'écrire un nombre spécifié de bobines.

6.1.5.3.5 Service Écrire en diffusion bobine individuelle

6.1.5.3.5.1 Vue d'ensemble du service

Le service Écrire en diffusion bobine individuelle permet d'écrire une bobine individuelle dans tous les serveurs adressables par l'ID d'unité.

6.1.5.3.5.2 Primitives de service

Les paramètres de service de ce service sont indiqués dans le Tableau 9. Il s'agit d'un service non confirmé.

Tableau 9 – Paramètres du service Écrire en diffusion bobine individuelle

Nom de paramètre	Dem.	Ind.
Argument	M	M(=)
ID d'unité	M	M(=)
Adresse de la première bobine	M	M(=)
Bobine individuelle de données	M	M(=)

Argument
L'argument transmet les paramètres de service spécifiques de la demande de service.

ID d'unité
Ce paramètre doit être utilisé pour donner à ce service l'indicateur de service en diffusion.

Type de paramètre: Unsigned8.

Valeurs autorisées: La valeur qu'il faut spécifier est 0.

Adresse de la première bobine
Ce paramètre spécifie l'adresse de la première bobine (la seule dans ce service).

Type de paramètre: Unsigned16.

Valeurs autorisées: 0x0000 à 0xFFFF

Bobine individuelle de données
Ce paramètre spécifie la valeur de statut de bobine individuelle qui doit être écrite.

Type de paramètre: Indicateur de statut.

6.1.5.3.5.3 Procédure de service

Ce service permet d'écrire une bobine individuelle dans tous les serveurs adressables par l'ID d'unité.

6.1.5.3.6 Écrire en diffusion bobines multiples

6.1.5.3.6.1 Vue d'ensemble du service

Le service Écrire en diffusion bobines multiples permet d'écrire un nombre spécifié de bobines dans tous les serveurs adressables par l'ID d'unité.

6.1.5.3.6.2 Primitives de service

Les paramètres de service de ce service sont indiqués dans le Tableau 10. Il s'agit d'un service non confirmé.

Tableau 10 – Paramètres du service Écrire en diffusion bobines multiples

Nom de paramètre	Dem.	Ind.
Argument	M	M(=)
ID d'unité	M	M(=)
Adresse de la première bobine	M	M(=)
Quantité de bobines	M	M(=)
Nombre d'octets de données	M	M(=)
Données	M	M(=)

Argument
L'argument transmet les paramètres de service spécifiques de la demande de service.

ID d'unité
Ce paramètre est utilisé pour donner à ce service l'indicateur de service en diffusion.

Type de paramètre: Unsigned8.

Valeurs autorisées: La valeur qu'il faut spécifier est 0.

Adresse de la première bobine
Ce paramètre spécifie l'adresse de la première bobine.

Type de paramètre: Unsigned16.

Valeurs autorisées: 0x0000 à 0xFFFF

Quantité de bobines
Ce paramètre spécifie la quantité de bobines.

Type de paramètre: Unsigned16.

Valeurs autorisées: 1 à 1968, qui doivent être compatibles avec le nombre d'octets de données et les paramètres de données.

Nombre d'octets de données
Ce paramètre spécifie le nombre d'octets transportant les valeurs de statut de bobine à écrire.

Type de paramètre: Unsigned16.

Valeurs autorisées: 1 à 246, qui doivent être compatibles avec la quantité de bobines et les paramètres de données.

Données
Ce paramètre spécifie les valeurs de statut de bobine qui doivent être écrites.

Type de paramètre: Séquence de bits de statut.

Valeurs autorisées: Elles doivent être compatibles avec la quantité de bobines et les paramètres de nombre d'octets de données.

6.1.5.3.6.3 Procédure de service

Ce service permet d'écrire un nombre spécifié de bobines dans tous les serveurs adressables par l'ID d'unité.

6.1.6 ASE de registres d'entrée

6.1.6.1 Vue d'ensemble

Cet ASE permet l'interaction et la modélisation par des objets réels à valeurs analogiques qui sont manipulés par l'application serveur et ne sont censés être observés que par l'utilisateur client. L'intégrité du contrat ci-dessus est sous le contrôle de l'AP serveur, qui peut limiter l'exposition de l'objet réel aux registres d'entrée.

6.1.6.2 Spécification de la classe des registres d'entrée

6.1.6.2.1 Modèle formel

L'objet Registres d'entrée est décrit par le modèle suivant:

ASE:			**ASE de registres d'entrée**
CLASSE:			**Registres d'entrée**
ID CLASSE:			non utilisé
CLASSE PARENTE:			TOP
ATTRIBUTS:			
1.	(m)	Attribut clé:	Implicite
2.	(m)	Attribut:	ID d'unité
3.	(m)	Attribut:	Adresse du premier registre d'entrée
4.	(m)	Attribut:	Quantité de registres d'entrée
SERVICES:			
1.	(m)	OpsService:	Lire registres d'entrée

6.1.6.2.2 Attributs

Implicite
L'attribut Implicite indique que l'objet Registres d'entrée est adressé implicitement par les services.

ID d'unité
Cet attribut spécifie l'adresse du serveur.

Type d'attribut: Unsigned8

Valeurs autorisées: 1 à 247

Adresse du premier registre d'entrée
Cet attribut spécifie l'adresse du premier registre d'entrée.

Type d'attribut: Unsigned16

Valeurs autorisées: 0x0000 à 0xFFFF

Quantité de registres d'entrée
L'attribut spécifie la quantité de registres d'entrée.

Type d'attribut: Unsigned16

Valeurs autorisées: 1 à 125

6.1.6.2.3 Services

Lire registres d'entrée
Ce service permet de lire un nombre spécifié de registres d'entrée.

6.1.6.3 Spécification des services des ASE de registres d'entrée

6.1.6.3.1 Services supportés

L'ASE des registres d'entrée définit les services:

Lire registres d'entrée

6.1.6.3.2 Service Lire registres d'entrée

6.1.6.3.2.1 Vue d'ensemble du service

Le service Lire registres d'entrée permet de lire un nombre spécifié de registres d'entrée.

6.1.6.3.2.2 Primitives de service

Les paramètres de service de ce service sont indiqués dans le Tableau 11. Il s'agit d'un service confirmé.

Tableau 11 – Paramètres du service Lire registres d'entrée

Nom de paramètre	Dem.	Ind.	Rép.	Conf.
Argument	M	M(=)		
ID d'unité	M	M(=)		
ID appel	C	C(=)		
Adresse du premier registre d'entrée	M	M(=)		
Quantité de registres d'entrée	M	M(=)		
Résultat (+)			S	S(=)
ID d'unité			M	M(=)
ID appel			C	C(=)
Nombre d'octets de données			M	M(=)
Données			M	M(=)
Résultat (-)			S	S(=)
ID d'unité			M	M(=)
ID appel			C	C(=)
Info. erreur			M	M(=)

Argument
L'argument transmet les paramètres de service spécifiques de la demande de service.

ID d'unité
Ce paramètre spécifie l'adresse du serveur.

Type de paramètre: Unsigned8.

Valeurs autorisées: 1 à 247.

Adresse du premier registre d'entrée
Ce paramètre spécifie l'adresse du premier registre d'entrée.

Type de paramètre: Unsigned16.

Valeurs autorisées: 0x0000 à 0xFFFF

Quantité de registres d'entrée
Ce paramètre spécifie la quantité de registres d'entrée.

Type de paramètre: Unsigned16.

Valeurs autorisées: 1 à 125

Résultat (+)
Ce paramètre de type de sélection indique que la demande de service a réussi.

ID d'unité
En l'absence d'erreur, ce paramètre est identique au paramètre de demande portant le nom correspondant.

Nombre d'octets de données
Ce paramètre spécifie le nombre d'octets lus.

Type de paramètre: Unsigned16.

Données
Ce paramètre spécifie les valeurs analogiques des registres d'entrée.

Type de paramètre: Matrice de Unsigned16, transférée au moyen du codage gros-boutiste.

Résultat (-)
Ce paramètre de type de sélection indique que la demande de service a échoué.

ID d'unité
En l'absence d'erreur, ce paramètre est identique au paramètre de demande portant le nom correspondant.

Info. erreur
Ce paramètre spécifie les informations d'erreur.

Type de paramètre: Unsigned8.

6.1.6.3.2.3 Procédure de service

Ce service permet de lire un nombre spécifié de registres d'entrée.

6.1.7 ASE des registres de maintien

6.1.7.1 Vue d'ensemble

Cet ASE permet l'interaction et la modélisation par des objets réels à valeurs analogiques qui peuvent être manipulés par l'application serveur et par l'utilisateur client.

6.1.7.2 Spécification de la classe des registres de maintien

6.1.7.2.1 Modèle formel

L'objet Registres de maintien est décrit par le modèle suivant:

ASE: **ASE des registres de maintien**
CLASSE: **Registres de maintien**
ID CLASSE: non utilisé
CLASSE PARENTE: TOP
ATTRIBUTS:

1.	(m)	Attribut clé:	Implicite
2.	(m)	Attribut:	ID d'unité
3.	(c)	Contrainte:	Lire registres de maintien
3.1	(m)	Attribut:	Adresse du premier registre de maintien à lire
3.2	(m)	Attribut:	Quantité de registres de maintien à lire
4.	(c)	Contrainte:	Écrire registre de maintien individuel | Écrire registres de maintien multiples | Écrire en diffusion registre de maintien individuel | Écrire en diffusion registres de maintien multiples
4.1	(m)	Attribut:	Adresse du premier registre de maintien à écrire
4.2	(m)	Attribut:	Quantité de registres de maintien à écrire
5.	(c)	Contrainte:	Lire/écrire registres de maintien
5.1	(m)	Attribut:	Adresse du premier registre de maintien à lire
5.2	(m)	Attribut:	Quantité de registres de maintien à lire
5.3	(m)	Attribut:	Adresse du premier registre de maintien à écrire
5.4	(m)	Attribut:	Quantité de registres de maintien à écrire
6.	(c)	Contrainte:	Écrire avec masque registre de maintien
6.1.	(m)	Attribut:	Adresse du registre de maintien à modifier
6.2.	(m)	Attribut:	Masque ET
6.3.	(m)	Attribut:	Masque OU
7.	(c)	Contrainte:	Lire FIFO
7.1	(m)	Attribut:	Adresse de file d'attente FIFO
7.2	(m)	Attribut:	Comptage file d'attente FIFO
8.	(m)	Attribut:	Nombre d'octets de données
9.	(m)	Attribut:	Données

SERVICES:

1.	(o)	OpsService:	Lire registres de maintien
2.	(o)	OpsService:	Écrire registre de maintien individuel
3.	(o)	OpsService:	Écrire registres de maintien multiples
3.	(o)	OpsService:	Écrire avec masque registre de maintien
3.	(o)	OpsService:	Lire/écrire registres de maintien
3.	(o)	OpsService:	Lire FIFO
4.	(c)	Contrainte:	Diffusion supportée
4.1	(o)	OpsService:	Écrire en diffusion registre de maintien individuel
4.2	(o)	OpsService:	Écrire en diffusion registres de maintien multiples

6.1.7.2.2 Attributs

Implicite
L'attribut Implicite indique que l'objet Registres de maintien est adressé implicitement par les services.

ID d'unité
Cet attribut spécifie l'adresse du serveur.

Type d'attribut: Unsigned8

Valeurs autorisées: 1 à 247, et 0 pour la diffusion si cela s'applique.

Adresse du premier registre de maintien à lire
Cet attribut spécifie l'adresse du premier registre de maintien à lire.

Type d'attribut: Unsigned16

Valeurs autorisées: 0x0000 à 0xFFFF

Quantité de registres de maintien à lire
L'attribut spécifie la quantité de registres de maintien à lire.

Type d'attribut: Unsigned16

Valeurs autorisées: 1 à 125

Adresse du premier registre de maintien à écrire
Cet attribut spécifie l'adresse du premier registre de maintien à écrire.

Type d'attribut: Unsigned16

Valeurs autorisées: 0x0000 à 0xFFFF

Quantité de registres de maintien à écrire
L'attribut spécifie la quantité de registres de maintien à écrire.

Type d'attribut: Unsigned16

Valeurs autorisées: 1 à 123 avec Écrire registres de maintien multiples, 1 à 121 avec Lire/écrire registres de maintien.

Adresse du premier registre de maintien à modifier
Cet attribut spécifie l'adresse du registre de maintien à modifier à sa place par l'intermédiaire de masques.

Type d'attribut: Unsigned16

Valeurs autorisées: 0x0000 à 0xFFFF

Masque ET
L'attribut spécifie le masque binaire AND_Mask qui produit le nouveau contenu d'un registre d'après l'équation (1):

$$new\_content = (old\_content \wedge AND\_Mask) \vee (OR\_Mask \wedge \neg AND\_Mask) \quad (1)$$

Type d'attribut: Unsigned16

Valeurs autorisées: 0x0000 à 0xFFFF

Masque OU
L'attribut spécifie le masque binaire OR_Mask qui produit le nouveau contenu d'un registre d'après l'équation (1).L'attribut spécifie le masque binaire AND_Mask qui produit le nouveau contenu d'un registre d'après l'équation (1):

Type d'attribut: Unsigned16

Valeurs autorisées: 0x0000 à 0xFFFF

Adresse de file d'attente FIFO
Cet attribut spécifie l'adresse du registre de maintien qui contient le nombre de registres de maintien de données de file d'attente FIFO qui suivent.

Type d'attribut: Unsigned16

Valeurs autorisées: 0x0000 à 0xFFFF

Comptage file d'attente FIFO
L'attribut spécifie la quantité de registres de maintien de données de file d'attente FIFO.

Type d'attribut: Unsigned16

Valeurs autorisées: 1 à 31

Nombre d'octets de données
L'attribut spécifie la quantité d'octets de données.

Type d'attribut: Unsigned8

Valeurs autorisées: 1 à 250 pour Lire registres de maintien et comme lecture de nombre d'octets pour Lire/écrire registres de maintien, 1 à 246 pour Écrire registres de maintien multiples et Écrire en diffusion registres de maintien multiples, 1 à 242 comme écriture de nombre d'octets de Lire/écrire registres de maintien, et 1 à 64 pour Lire FIFO

Données
L'attribut spécifie les valeurs de registres qui ont été lues ou qui doivent être écrites.

Type d'attribut: Matrice d'Unsigned16, transférée au moyen du codage gros-boutiste.

6.1.7.2.3 Services

Lire registres de maintien
Ce service optionnel permet de lire un nombre spécifié de registres de maintien.

Écrire registre de maintien individuel
Ce service optionnel permet d'écrire un registre de maintien individuel.

Écrire registres de maintien multiples
Ce service optionnel permet d'écrire un nombre spécifié de registres de maintien.

Écrire avec masque registre de maintien
Ce service optionnel permet de manipuler le contenu d'un registre de maintien au moyen de deux masques binaires, le masque ET et le masque OU, d'après l'équation (1).

Lire/écrire registres de maintien
Ce service optionnel permet de lire et d'écrire des nombres spécifiés de registres de maintien. L'opération d'écriture est effectuée avant la lecture.

Lire FIFO
Ce service optionnel permet de lire un nombre limité mais non spécifié de registres de maintien, organisés pour faciliter une politique FIFO.

Écrire en diffusion registre de maintien individuel
Ce service optionnel est également conditionnel pour le mécanisme de diffusion supporté par le client et par les serveurs. Ce service permet d'écrire un registre de maintien individuel dans tous les serveurs adressables par l'ID d'unité; il s'agit d'un service non confirmé.

Écrire en diffusion services de maintien multiples
Ce service optionnel est également conditionnel pour le mécanisme de diffusion supporté par le client et par les serveurs. Ce service permet d'écrire un nombre spécifié de registres de maintien dans tous les serveurs adressables par l'ID d'unité; il s'agit d'un service non confirmé.

6.1.7.3 Spécification des services des ASE de registres de maintien

6.1.7.3.1 Services supportés

Le présent paragraphe contient la définition de services qui sont uniques pour cet ASE. Les services définis pour cet ASE sont:

Lire registres de maintien

Écrire registre de maintien individuel

Écrire registres de maintien multiples

Écrire avec masque registre de maintien

Lire/écrire registres de maintien

Lire FIFO

Écrire en diffusion registre de maintien individuel

Écrire en diffusion registres de maintien multiples

6.1.7.3.2 Service Lire registres de maintien

6.1.7.3.2.1 Vue d'ensemble du service

Le service Lire registres de maintien permet de lire un nombre spécifié de registres de maintien.

6.1.7.3.2.2 Primitives de service

Les paramètres de service de ce service sont indiqués dans le Tableau 12. Il s'agit d'un service confirmé.

Tableau 12 – Paramètres du service Lire registres de maintien

Nom de paramètre	Dem.	Ind.	Rép.	Conf.
Argument	M	M(=)		
ID d'unité	M	M(=)		
ID appel	C	C(=)		
Adresse du premier registre de maintien à lire	M	M(=)		
Quantité de registres de maintien à lire	M	M(=)		
Résultat (+)			S	S(=)
ID d'unité			M	M(=)
ID appel			C	C(=)
Nombre d'octets de données			M	M(=)
Données			M	M(=)
Résultat (-)			S	S(=)
ID d'unité			M	M(=)
ID appel			C	C(=)
Info. erreur			M	M(=)

Argument
L'argument transmet les paramètres de service spécifiques de la demande de service.

ID d'unité
Ce paramètre spécifie l'adresse du serveur.

Type de paramètre: Unsigned8.

Valeurs autorisées: 1 à 247.

Adresse du premier registre de maintien à lire
Ce paramètre spécifie l'adresse du premier registre de maintien à lire.

Type de paramètre: Unsigned16.

Valeurs autorisées: 0x0000 à 0xFFFF

Quantité de registres de maintien à lire
Ce paramètre spécifie la quantité de registres de maintien à lire.

Type de paramètre: Unsigned16.

Valeurs autorisées: 1 à 125

Résultat (+)
Ce paramètre de type de sélection indique que la demande de service a réussi.

ID d'unité
En l'absence d'erreur, ce paramètre est identique au paramètre de demande portant le nom correspondant.

Nombre d'octets de données
Ce paramètre spécifie le nombre d'octets lus.

Type de paramètre: Unsigned16.

Données
Ce paramètre spécifie les valeurs analogiques des registres de maintien lus.

Type de paramètre: Matrice de Unsigned16, transférée au moyen du codage gros-boutiste.

Résultat (-)
Ce paramètre de type de sélection indique que la demande de service a échoué.

ID d'unité
En l'absence d'erreur, ce paramètre est identique au paramètre de demande portant le nom correspondant.

Info. erreur
Ce paramètre spécifie les informations d'erreur.

Type de paramètre: Unsigned8.

6.1.7.3.2.3 Procédure de service

Ce service permet de lire un nombre spécifié de registres de maintien.

6.1.7.3.3 Service Écrire registre de maintien individuel

6.1.7.3.3.1 Vue d'ensemble du service

Le service Écrire registre de maintien individuel permet d'écrire un registre de maintien individuel.

6.1.7.3.3.2 Primitives de service

Les paramètres de service de ce service sont indiqués dans le Tableau 13. Il s'agit d'un service confirmé.

Tableau 13 – Paramètres du service Écrire registre de maintien individuel

Nom de paramètre	Dem.	Ind.	Rép.	Conf.
Argument	M	M(=)		
ID d'unité	M	M(=)		
ID appel	C	C(=)		
Adresse du premier registre de maintien à écrire	M	M(=)		
Données	M	M(=)		
Résultat (+)			S	S(=)
ID d'unité			M	M(=)
ID appel			C	C(=)
Adresse du premier registre de maintien à écrire			M	M(=)
Données			M	M(=)
Résultat (-)			S	S(=)
ID d'unité			M	M(=)
ID appel			C	C(=)
Info. erreur			M	M(=)

Argument
L'argument transmet les paramètres de service spécifiques de la demande de service.

ID d'unité
Ce paramètre spécifie l'adresse du serveur.

Type de paramètre: Unsigned8.

Valeurs autorisées: 1 à 247.

Adresse du premier registre de maintien à écrire
Ce paramètre spécifie l'adresse du premier registre de maintien à écrire (le seul dans ce service).

Type de paramètre: Unsigned16.

Valeurs autorisées: 0x0000 à 0xFFFF

Données
Ce paramètre spécifie la valeur analogique de registre de maintien qui doit être écrite.

Type de paramètre: Unsigned16.

Résultat (+)
Ce paramètre de type de sélection indique que la demande de service a réussi.

ID d'unité
En l'absence d'erreur, ce paramètre est identique au paramètre de demande portant le nom correspondant.

Adresse du premier registre de maintien à écrire
En l'absence d'erreur, ce paramètre est identique au paramètre de demande portant le nom correspondant.

Données
En l'absence d'erreur, ce paramètre est identique au paramètre de demande portant le nom correspondant.

Résultat (-)
Ce paramètre de type de sélection indique que la demande de service a échoué.

ID d'unité
En l'absence d'erreur, ce paramètre est identique au paramètre de demande portant le nom correspondant.

Info. erreur
Ce paramètre spécifie les informations d'erreur.

Type de paramètre: Unsigned8.

6.1.7.3.3.3 Procédure de service

Ce service permet d'écrire un registre de maintien individuel.

6.1.7.3.4 Service Écrire registres de maintien multiples

6.1.7.3.4.1 Vue d'ensemble du service

Le service Écrire registres de maintien multiples permet d'écrire un nombre spécifié de registres de maintien.

6.1.7.3.4.2 Primitives de service

Les paramètres de service de ce service sont indiqués dans le Tableau 14. Il s'agit d'un service confirmé.

Tableau 14 – Paramètres du service Écrire registres de maintien multiples

Nom de paramètre	Dem.	Ind.	Rép.	Conf.
Argument	M	M(=)		
ID d'unité	M	M(=)		
ID appel	C	C(=)		
Adresse du premier registre de maintien à écrire	M	M(=)		
Quantité de registres de maintien à écrire	M	M(=)		
Nombre d'octets de données	M	M(=)		
Données	M	M(=)		
Résultat (+)			S	S(=)
ID d'unité			M	M(=)
ID appel			C	C(=)
Adresse du premier registre de maintien à écrire			M	M(=)
Quantité de registres de maintien à écrire			M	M(=)
Résultat (-)			S	S(=)
ID d'unité			M	M(=)
ID appel			C	C(=)
Info. erreur			M	M(=)

Argument
L'argument transmet les paramètres de service spécifiques de la demande de service.

ID d'unité
Ce paramètre spécifie l'adresse du serveur.

Type de paramètre: Unsigned8.

Valeurs autorisées: 1 à 247.

Adresse des premiers registres de maintien à écrire
Ce paramètre spécifie l'adresse des premiers registres de maintien à écrire.

Type de paramètre: Unsigned16.

Valeurs autorisées: 0x0000 à 0xFFFF

Quantité de registres de maintien à écrire
Ce paramètre spécifie la quantité de registres de maintien à écrire.

Type de paramètre: Unsigned16.

Valeurs autorisées: 1 à 123, doivent être compatibles avec le nombre d'octets de données et les paramètres de données.

Nombre d'octets de données
Ce paramètre spécifie le nombre d'octets transportant les valeurs analogiques de registres de maintien à écrire.

Type de paramètre: Unsigned16.

Valeurs autorisées: 1 à 246, doivent être compatibles avec la quantité de registres de maintien à écrire et les paramètres de données.

Données
Ce paramètre spécifie les valeurs analogiques de registres de maintien qui doivent être écrites.

Type de paramètre: Matrice de Unsigned16, transférée au moyen du codage gros-boutiste.

Valeurs autorisées: le nombre de registres doit être compatible avec la quantité de registres de maintien à écrire et les paramètres de nombre d'octets de données.

Résultat (+)
Ce paramètre de type de sélection indique que la demande de service a réussi.

ID d'unité
En l'absence d'erreur, ce paramètre est identique au paramètre de demande portant le nom correspondant.

Adresse du premier registre de maintien à écrire
En l'absence d'erreur, ce paramètre est identique au paramètre de demande portant le nom correspondant.

Quantité de registres de maintien à écrire
En l'absence d'erreur, ce paramètre est identique au paramètre de demande portant le nom correspondant.

Résultat (-)
Ce paramètre de type de sélection indique que la demande de service a échoué.

ID d'unité
En l'absence d'erreur, ce paramètre est identique au paramètre de demande portant le nom correspondant.

Info. erreur
Ce paramètre spécifie les informations d'erreur.

Type de paramètre: Unsigned8.

6.1.7.3.4.3 Procédure de service

Ce service permet d'écrire un nombre spécifié de registres de maintien.

6.1.7.3.5 Service Écrire avec masque registre de maintien

6.1.7.3.5.1 Vue d'ensemble du service

Le service Écrire avec masque registre de maintien permet de manipuler le contenu d'un registre de maintien au moyen de deux masques binaires, le masque ET et le masque OU, d'après l'équation (1).

6.1.7.3.5.2 Primitives de service

Les paramètres de service de ce service sont indiqués dans le Tableau 15. Il s'agit d'un service confirmé.

Tableau 15 – Paramètres du service Écrire avec masque registre de maintien

Nom de paramètre	Dem.	Ind.	Rép.	Conf.
Argument	M	M(=)		
ID d'unité	M	M(=)		
ID appel	C	C(=)		
Adresse du premier registre de maintien à écrire	M	M(=)		
Masque ET	M	M(=)		
Masque OU	M	M(=)		
Résultat (+)			S	S(=)
ID d'unité			M	M(=)
ID appel			C	C(=)
Adresse du premier registre de maintien à écrire			M	M(=)
Masque ET			M	M(=)
Masque OU			M	M(=)
Résultat (-)			S	S(=)
ID d'unité			M	M(=)
ID appel			C	C(=)
Info. erreur			M	M(=)

Argument
L'argument transmet les paramètres de service spécifiques de la demande de service.

ID d'unité
Ce paramètre spécifie l'adresse du serveur.

Type de paramètre: Unsigned8.

Valeurs autorisées: 1 à 247.

Adresse du registre de maintien à modifier
Ce paramètre spécifie l'adresse du registre de maintien à modifier au moyen de deux masques binaires, le masque ET et le masque OU, d'après l'équation (1).

Type de paramètre: Unsigned16.

Valeurs autorisées: 0x0000 à 0xFFFF

Masque ET
Ce paramètre spécifie le masque binaire AND_Mask qui produit le nouveau contenu du registre de maintien à modifier d'après l'équation (1).

Type de paramètre: Unsigned16.

Valeurs autorisées: 0x0000 à 0xFFFF

Masque OU
Ce paramètre spécifie le masque binaire OR_Mask qui produit le nouveau contenu du registre de maintien à modifier d'après l'équation (1).

Type de paramètre: Unsigned16.

Valeurs autorisées: 0x0000 à 0xFFFF

Résultat (+)
Ce paramètre de type de sélection indique que la demande de service a réussi.

ID d'unité
En l'absence d'erreur, ce paramètre est identique au paramètre de demande portant le nom correspondant.

Adresse du registre de maintien à modifier
En l'absence d'erreur, ce paramètre est identique au paramètre de demande portant le nom correspondant.

Masque ET
En l'absence d'erreur, ce paramètre est identique au paramètre de demande portant le nom correspondant.

Masque OU
En l'absence d'erreur, ce paramètre est identique au paramètre de demande portant le nom correspondant.

Résultat (-)
Ce paramètre de type de sélection indique que la demande de service a échoué.

ID d'unité
En l'absence d'erreur, ce paramètre est identique au paramètre de demande portant le nom correspondant.

Info. erreur
Ce paramètre spécifie les informations d'erreur.

Type de paramètre: Unsigned8.

6.1.7.3.5.3 Procédure de service

Ce service permet de manipuler le contenu d'un registre de maintien au moyen de deux masques binaires, le masque ET et le masque OU, d'après l'équation (1).

6.1.7.3.6 Service Lire/écrire registres de maintien

6.1.7.3.6.1 Vue d'ensemble du service

Le service Lire/écrire registres de maintien permet de lire et d'écrire des nombres spécifiés de registres de maintien.

6.1.7.3.6.2 Primitives de service

Les paramètres de service de ce service sont indiqués dans le Tableau 16. Il s'agit d'un service confirmé.

Tableau 16 – Paramètres du service Lire/écrire registres de maintien

Nom de paramètre	Dem.	Ind.	Rép.	Conf.
Argument	M	M(=)		
ID d'unité	M	M(=)		
ID appel	C	C(=)		
Adresse du premier registre de maintien à lire	M	M(=)		
Quantité de registres de maintien à lire	M	M(=)		
Adresse du premier registre de maintien à écrire	M	M(=)		
Quantité de registres de maintien à écrire	M	M(=)		
Nombre d'octets de données	M	M(=)		
Données	M	M(=)		
Résultat (+)			S	S(=)
ID d'unité			M	M(=)
ID appel			C	C(=)
Nombre d'octets de données			M	M(=)
Données			M	M(=)
Résultat (-)			S	S(=)
ID d'unité			M	M(=)
ID appel			C	C(=)
Info. erreur			M	M(=)

Argument
L'argument transmet les paramètres de service spécifiques de la demande de service.

ID d'unité
Ce paramètre spécifie l'adresse du serveur.

Type de paramètre: Unsigned8.

Valeurs autorisées: 1 à 247.

Adresse des premiers registres de maintien à lire
Ce paramètre spécifie l'adresse des premiers registres de maintien à lire.

Type de paramètre: Unsigned16.

Valeurs autorisées: 0x0000 à 0xFFFF

Quantité de registres de maintien à lire
Ce paramètre spécifie la quantité de registres de maintien à lire.

Type de paramètre: Unsigned16.

Valeurs autorisées: 1 à 125

Adresse des premiers registres de maintien à écrire
Ce paramètre spécifie l'adresse des premiers registres de maintien à écrire.

Type de paramètre: Unsigned16.

Valeurs autorisées: 0x0000 à 0xFFFF

Quantité de registres de maintien à écrire
Ce paramètre spécifie la quantité de registres de maintien à écrire.

Type de paramètre: Unsigned16.

Valeurs autorisées: 1 à 121, doivent être compatibles avec le nombre d'octets de données et les paramètres de données.

Nombre d'octets de données
Ce paramètre spécifie le nombre d'octets transportant les valeurs analogiques de registres de maintien à écrire.

Type de paramètre: Unsigned16.

Valeurs autorisées: 1 à 242, doivent être compatibles avec la quantité de registres de maintien à écrire et les paramètres de données.

Données
Ce paramètre spécifie les valeurs analogiques de registres de maintien qui doivent être écrites.

Type de paramètre: Matrice de Unsigned16, transférée au moyen du codage gros-boutiste.

Valeurs autorisées: il faut que le nombre de registres soit compatible avec la quantité de registres de maintien à écrire et les paramètres de nombre d'octets de données.

Résultat (+)
Ce paramètre de type de sélection indique que la demande de service a réussi.

ID d'unité
En l'absence d'erreur, ce paramètre est identique au paramètre de demande portant le nom correspondant.

Nombre d'octets de données
Ce paramètre spécifie le nombre d'octets lus.

Type de paramètre: Unsigned16.

Données
Ce paramètre spécifie les valeurs analogiques des registres de maintien lus.

Type de paramètre: Matrice de Unsigned16, transférée au moyen du codage gros-boutiste.

Résultat (-)
Ce paramètre de type de sélection indique que la demande de service a échoué.

ID d'unité
En l'absence d'erreur, ce paramètre est identique au paramètre de demande portant le nom correspondant.

Info. erreur
Ce paramètre spécifie les informations d'erreur.

Type de paramètre: Unsigned8.

6.1.7.3.6.3 Procédure de service

Ce service permet de lire et d'écrire des nombres spécifiés de registres de maintien. L'opération d'écriture est effectuée avant la lecture.

6.1.7.3.7 Service Lire FIFO

6.1.7.3.7.1 Vue d'ensemble du service

Le service Lire FIFO permet de lire un nombre limité de registres de maintien, organisés pour faciliter une politique FIFO. Le nombre limité est a priori inconnu et fait partie de la réponse. La limite est de 32 registres: le registre contenant le nombre ci-dessus plus jusqu'à 31 registres de maintien de file d'attente FIFO venant ensuite.

6.1.7.3.7.2 Primitives de service

Les paramètres de service de ce service sont indiqués dans le Tableau 17. Il s'agit d'un service confirmé.

Tableau 17 – Paramètres du service Lire FIFO

Nom de paramètre	Dem.	Ind.	Rép.	Conf.
Argument	M	M(=)		
ID d'unité	M	M(=)		
ID appel	C	C(=)		
Adresse de file d'attente FIFO	M	M(=)		
Résultat (+)			S	S(=)
ID d'unité			M	M(=)
ID appel			C	C(=)
Nombre d'octets de données			M	M(=)
Comptage file d'attente FIFO			M	M(=)
Données			M	M(=)
Résultat (-)			S	S(=)
ID d'unité			M	M(=)
ID appel			C	C(=)
Info. erreur			M	M(=)

Argument
L'argument transmet les paramètres de service spécifiques de la demande de service.

ID d'unité
Ce paramètre spécifie l'adresse du serveur.

Type de paramètre: Unsigned8.

Valeurs autorisées: 1 à 247.

Adresse de file d'attente FIFO
Ce paramètre spécifie l'adresse du registre de maintien qui contient le nombre de registres de maintien de données de file d'attente FIFO qui viennent ensuite.

Type de paramètre: Unsigned16.

Valeurs autorisées: 0x0000 à 0xFFFF

Résultat (+)
Ce paramètre de type de sélection indique que la demande de service a réussi.

ID d'unité
En l'absence d'erreur, ce paramètre est identique au paramètre de demande portant le nom correspondant.

Nombre d'octets de données
Ce paramètre spécifie le nombre d'octets lus, y compris les octets du comptage de file d'attente FIFO.

Type de paramètre: Unsigned16.

Comptage file d'attente FIFO
Ce paramètre spécifie le nombre de registres de maintien de données de la file d'attente FIFO, lus dans le paramètre de données. Le registre de maintien du comptage de file d'attente FIFO n'en fait pas partie.

Type de paramètre: Unsigned16.

Données
Ce paramètre spécifie les valeurs analogiques des registres de maintien lus.

Type de paramètre: Matrice de Unsigned16, transférée au moyen du codage gros-boutiste.

Résultat (-)
Ce paramètre de type de sélection indique que la demande de service a échoué.

ID d'unité
En l'absence d'erreur, ce paramètre est identique au paramètre de demande portant le nom correspondant.

Info. erreur
Ce paramètre spécifie les informations d'erreur.

Type de paramètre: Unsigned8.

6.1.7.3.7.3 Procédure de service

Le service Lire FIFO permet de lire un nombre limité de registres de maintien, organisés pour faciliter une politique FIFO. Le nombre limité est a priori inconnu et fait partie de la réponse. La limite est de 32 registres: le registre contenant le nombre ci-dessus plus jusqu'à 31 registres de maintien de file d'attente FIFO venant ensuite.

6.1.7.3.8 Service Écrire en diffusion registre de maintien individuel

6.1.7.3.8.1 Vue d'ensemble du service

Le service Écrire en diffusion registre de maintien individuel permet d'écrire un registre de maintien individuel dans tous les serveurs adressables par l'ID d'unité.

6.1.7.3.8.2 Primitives de service

Les paramètres de service de ce service sont indiqués dans le Tableau 18. Il s'agit d'un service non confirmé.

Tableau 18 – Paramètres du service Écrire en diffusion registre de maintien individuel

Nom de paramètre	Dem.	Ind.
Argument	M	M(=)
ID d'unité	M	M(=)
Adresse du premier registre de maintien à écrire	M	M(=)
Données	M	M(=)

Argument
L'argument transmet les paramètres de service spécifiques de la demande de service.

ID d'unité
Ce paramètre est utilisé pour donner à ce service l'indicateur de service en diffusion.

Type de paramètre: Unsigned8.

Valeurs autorisées: La valeur qu'il faut spécifier est 0.

Adresse du premier registre de maintien à écrire
Ce paramètre spécifie l'adresse du premier registre de maintien à écrire (le seul dans ce service).

Type de paramètre: Unsigned16.

Valeurs autorisées: 0x0000 à 0xFFFF

Données
Ce paramètre spécifie la valeur analogique de registre de maintien qui doit être écrite.

Type de paramètre: Unsigned16.

6.1.7.3.8.3 Procédure de service

Ce service permet d'écrire un registre de maintien individuel dans tous les serveurs adressables par l'ID d'unité.

6.1.7.3.9 Service Écrire en diffusion registres de maintien multiples

6.1.7.3.9.1 Vue d'ensemble du service

Le service Écrire en diffusion registres de maintien multiples permet d'écrire un nombre spécifié de registres de maintien dans tous les serveurs adressables par l'ID d'unité.

6.1.7.3.9.2 Primitives de service

Les paramètres de service de ce service sont indiqués dans le Tableau 19. Il s'agit d'un service non confirmé.

Tableau 19 – Paramètres du service Écrire en diffusion registres de maintien multiples

Nom de paramètre	Dem.	Ind.
Argument	M	M(=)
ID d'unité	M	M(=)
Adresse du premier registre de maintien à écrire	M	M(=)
Quantité de registres de maintien à écrire	M	M(=)
Nombre d'octets de données	M	M(=)
Données	M	M(=)

Argument
L'argument transmet les paramètres de service spécifiques de la demande de service.

ID d'unité
Ce paramètre est utilisé pour donner à ce service l'indicateur de service en diffusion.

Type de paramètre: Unsigned8.

Valeurs autorisées: La valeur qu'il faut spécifier est 0.

Adresse des premiers registres de maintien à écrire
Ce paramètre spécifie l'adresse des premiers registres de maintien à écrire.

Type de paramètre: Unsigned16.

Valeurs autorisées: 0x0000 à 0xFFFF

Quantité de registres de maintien à écrire
Ce paramètre spécifie la quantité de registres de maintien à écrire.

Type de paramètre: Unsigned16.

Valeurs autorisées: 1 à 123, doivent être compatibles avec le nombre d'octets de données et les paramètres de données.

Nombre d'octets de données
Ce paramètre spécifie le nombre d'octets transportant les valeurs analogiques de registres de maintien à écrire.

Type de paramètre: Unsigned16.

Valeurs autorisées: 1 à 246, doivent être compatibles avec la quantité de bobines et les paramètres de données.

Données
Ce paramètre spécifie les valeurs analogiques de registres de maintien qui doivent être écrites.

Type de paramètre: Matrice de Unsigned16, transférée au moyen du codage gros-boutiste.

Valeurs autorisées: Il faut que le nombre de registres soit compatible avec la quantité de registres de maintien à écrire et les paramètres de nombre d'octets de données.

6.1.7.3.9.3 Procédure de service

Ce service permet d'écrire un nombre spécifié de registres de maintien dans tous les serveurs adressables par l'ID d'unité.

6.1.8 ASE de fichier

6.1.8.1 Vue d'ensemble

Cet ASE permet l'interaction et la modélisation par des objets réels structurés hiérarchiquement. La hiérarchie comporte trois niveaux. Le niveau supérieur est constitué de fichiers, et son niveau sous-jacent, groupé par fichiers, est constitué d'enregistrements. Les enregistrements eux-mêmes sont des ensembles de registres, contigus du point de vue d'un utilisateur d'application (utilisateur client). Les registres constituent le troisième niveau. Du fait que les enregistrements ne possèdent pas de système de nommage et d'adressage indépendant, étant au contraire définis par les registres dont ils sont constitués, la distinction entre le deuxième niveau et le troisième est purement une affaire de notation.

6.1.8.2 Spécification de la classe des fichiers

6.1.8.2.1 Modèle formel

L'objet fichier est décrit par le modèle suivant: